МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Государственное образовательное учреждение высшего профессионального образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В.В. Соколова

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета

Издательство
Томского политехнического университета
2011

УДК 621.396.21(074.8)
ББК 32.884.1я73
С594

Соколова В.В.
С594 Разработка мобильных приложений: учебное пособие / В.В. Соколова; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2011. – 175 с.

В данном учебном пособии рассматриваются современные мобильные технологии и платформы разработки мобильных приложений. Представлена классификация мобильных устройств и приложений. Подробно рассматриваются вопросы разработки мобильных приложений с использованием языка программирования Java для плат-

форм J2ME и Android.

Пособие подготовлено на кафедре «Оптимизации систем управления» ТПУ и предназначено для студентов направления 230100 «Информатика и вычислительная техника», магистерской программы 230113 «Сети ЭВМ и телекоммуникации».

УДК 621.396.21(074.8)
ББК 32.884.1я73

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современное общество постепенно переходит от электронного бизнеса к мобильному. И если электронные технологии обеспечивают доступность и полноту информации, то мобильные технологии своей первоочередной задачей считают обеспечение своевременности информации и ее релевантность. Мобильные технологии в настоящее время активно развиваются. При этом можно выделить три тенденции в этом процессе. Во-первых, в мобильные телефоны, которые приобрели большую популярность у пользователей, добавляется все больше возможностей выполнять вычисления (смартфоны), что позволяют использовать их не только для голосовой связи, но и в качестве небольших компьютеров. Во-вторых, миниатюризация компьютеров привела к созданию карманных персональных компьютеров (КПК), которые пользователь может носить с собой и в автономном режиме использовать нужные ему программы. И, в-третьих, активно развиваются технологии и инфраструктура беспроводной связи, которые позволяют обмениваться данными между карманными персональными компьютерами и стационарными информационно-вычислительными сетями как в рамках локальных (например, технология Wi-Fi), так и глобальной (технологии GPRS, UMTS, WiMax) компьютерных сетей. Все это происходит на фоне снижения стоимости, а значит повышения доступности мобильных вычислительных устройств.

Такое быстрое развитие мобильных устройств и беспроводных технологий связи создает возможность предоставления их пользователям информационных и вычислительных услуг в любом месте, где имеется потребность в их результатах. Появляется возможность реализовать предоставление информационных сервисов любым пользователям, в любое время.

В настоящее время в РФ доля мобильных устройств, поддерживающих платформу J2ME составляет около 70 %. Но, в тоже время, интенсивно развиваются новые мобильные платформы (BlackBerry, Android, iOS и т.п.), ориентированные на более мощные многофункциональные устройства, такие как смартфоны и коммуникаторы. Данные устройства могут использоваться не только в целях коммуникации и развлечений, но и для выполнения профессиональных задач.

В зависимости от типа мобильного устройства и сферы применения, могут быть разработаны различные приложения. Например, функциональные возможности простых телефонов позволяют организовать

эффективную обратную связь с потребителем с использованием SMS-рассылки. Приложения, ориентированные на смартфоны, могут выступать как мобильные клиенты корпоративных сетей. Нет никаких сомне-

ний, что потребность в различных категориях мобильных приложений будет постоянно увеличиваться, а функциональность мобильных устройств – расширяться. Поэтому разработка приложений для мобильных устройств ещѐ долгое время будет перспективным направлением развития информационных технологий.

В связи с развитием мобильного бизнеса и сетевых сервисов актуальным становится обучение специалистов по проектированию и разработке мобильных приложений. Целью данного пособия и является систематизация, описание и обучение студентов разработке мобильных приложений. В пособии рассмотрены вопросы разработки мобильных приложений на языке Java для простых телефонов, поддерживающих платформу J2ME, и смартфонов ориентированных на платформу Android.

1. МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ И ТЕХНОЛОГИИ

Мобильными устройствами считаются компактные устройства, которые обеспечивают подключение к локальным и глобальным компьютерным сетям с помощью беспроводных технологий или технологий цифровой связи.

В настоящее время пользователям доступен широкий спектр мобильных устройств для решения разных задач. Он варьируются от очень недорогих мобильных телефонов до коммуникаторов и смартфонов. Наряду с форм-фактором имеются различия в функциональности и производительности, которые эти устройства обеспечивают.

1.1. Классификация мобильных устройств

Из всего разнообразия МВУ можно выделить следующие основные категории:

- сотовые телефоны;
- пейджеры;
- смартфоны;
- карманные персональные компьютеры (КПК);
- коммуникаторы.

Сотовым телефоном называется портативное устройство, основным назначением которого является предоставление услуг голосовой связи посредством сотовой сети. Сотовые телефоны управляются операционной системой, которая не предназначена для запуска прикладных

программ (приложений).

В настоящее время сотовые телефоны являются беспроводными устройствами, которые чаще всего используются на рынке мобильных устройств. В некоторых европейских странах более 75 % населения имеет мобильный телефон. В большинстве случаев используется голосовая связь, но с появлением беспроводного протокола передачи данных (WAP) и возможности обмена текстовыми сообщениями стали более распространенными и другие приложения по обработке и передаче данных.

Обычно такой телефон имеет небольшой по размеру дисплей (как правило, от 4 до 12 строк текста) с типовой 12-кнопочной клавиатурой для ввода данных. Эти ограничения делают сотовые телефоны неподходящими для веб-поиска, т.к. объемы данных, которые могут быть получены или выведены на экран, весьма ограничены. Основным преимуще-

ством сотовых телефонов является их широкое использование, что делает их очень подходящими для разработки конкретных приложений, ориентированных на пользователей. Например, такими приложениями могут быть: просмотр котировок акций, отчеты о трафике, покупка билета и чтение новостей. Во всех этих приложениях требуется ограниченный набор данных, необходимых для получения ожидаемого ответа. Однако, если вы планируете внедрить корпоративные приложения, такие как продвижение продаж и автоматизация обслуживания, лучшим выбором будут более мощные устройства.

Сотовые телефоны имеют встроенные беспроводные модемы, что позволяет пользователям легко подключиться к беспроводным сетям. После подключения пользователь может использовать модем как для голосовой связи, так и для приложений, или, используя беспроводную телефонную связь (WTA), поддерживаемую в WAP, одновременно передавать данные и голосовую связь. Кроме того, сотовые телефоны идеально подходят для обмена текстовыми сообщениями. Поскольку длина этих сообщений не превышает 160 символов, то возможности ввода этих устройств являются достаточными. С другой стороны преимуществом является длительное время работы батареи телефона. Ограниченная мощность сотовых телефонов позволяет экономить заряд аккумулятора (батареи), что приводит к более длительной работе по сравнению с более сложными устройствами, такими как смартфоны и карманные персональные компьютеры (КПК). Основной целью использования этих устройств является голосовая связь, поэтому качество голосовой связи, подключения к сети и передачи пакетов имеют приоритет при выборе лучших устройств.

Пейджеры нужны для обмена короткими текстовыми сообщениями между пользователями. Очень часто эти приложения загружаются производителями и не модернизируются. Эти устройства выполняют большую работу по подкачке данных, но часто не подходят для более сложных приложений. Это утверждение верно для большинства пейджеров, но есть некоторые исключения. Например, RIM и Motorola предлагают более продвинутые устройства, которые предоставляют возможности, подобные КПК. Они оба начинали с использования собственных операционных систем, но с появлением платформы J2ME они перешли на неѐ. Были расширены основные возможности подкачки данных устройств RIM Blackberry и пейджеров Motorola Timeport с приложениями для управления персональной информацией и беспроводными Интернет-приложениями. Кроме того, пользователям предлагаются продвинутые возможности ввода с использованием стандартной клавиатуры QWERTY, поддерживающей ввод большими пальцами рук.

Для этих устройств доступно несколько микробраузеров, которые часто предоставляются компанией, обеспечивающей беспроводное подключение. Очень часто эти браузеры основаны на WAP с использованием в качестве языка разметки WML, хотя также доступны некоторые HTML-браузеры. Поскольку эти устройства имеют удобное средство для ввода данных, они способны запускать беспроводные приложения, направленные на сбор данных.

Одной из самых удобных функций пейджеров является то, что они всегда подключены к беспроводной сети. Пользователям не обязательно нужно быть подключенным к беспроводной сети, когда они хотят получить доступ к приложению. Достаточно начать использовать приложение, которое будет передавать данные по мере необходимости. Это возможно, поскольку устройства используют сети с коммутацией пакетов при явной передаче данных, а не ежеминутную передачу, как в сетях с коммутацией каналов. Эта особенность требуется при получении страниц, электронной почты и других видов сообщений, а также является удобной при использовании беспроводных интернет-приложений. Кроме того, учитывая, что эти устройства всегда подключены к сети, батареи могут работать нередко в течение недели без подзарядки.

Простые смартфоны называются так потому, что они обеспечивают возможность запуска локальных приложений, а также совершения звонков. Как и веб-телефоны, это, прежде всего, голосовые устройства, поэтому выбор должен быть основан на возможностях голосовой связи. Поддержка приложений несколько ограничена из-за ограниченного объема хранения, мощности обработки и размеров отображения. Они похожи на веб-устройства по форме и размеру дисплея, но позволяют

приложениям работать локально, не требуя подключения к сети.

Одной из главных особенностей смартфонов является их простота. Пользователь может использовать устройство в течение нескольких минут, не беспокоясь о расширении конфигурации. Кроме того, поскольку устройства имеют очень небольшие процессоры и мало памяти, они могут работать несколько дней на одном заряде аккумулятора. Так как смартфоны поддерживают возможности мобильных телефонов, а также поддерживают приложения, это делает их идеальным выбором для тех, кто хочет ограничить набор устройств.

Пользователи могут расширить возможности устройства путем загрузки новых приложений «из воздуха». Это называется OTA-резервы. Производители мобильных телефонов и беспроводных устройств создают электронные витрины, на которых разработчики могут представить свои приложения для загрузки. Пользователи загружают интересные для них приложения за символическую плату (обычно от $ 1 до

$ 5). Прибыль делится между разработчиком и поставщиком магазина. С целью создания приложений для этих устройств в основном используется платформа Java 2 Micro Edition (J2ME). Этот факт представляет очень большие возможности программистам для разработки приложений на платформе J2ME.

На рынке беспроводных приложений идет движение в сторону более совместимых устройств в виде высококачественных профессиональных смартфонов. Ведущие производители сотовых телефонов, включая HTC, Nokia, Sony Ericsson, и Motorola, производят устройства, которые ориентированы на корпоративный рынок. Эти устройства обеспечивают возможность для голосовой связи, а также для интеллектуальных клиентских приложений. Это делает их хорошим выбором для людей, которые не хотят носить с собой несколько устройств, но которым все ещѐ нужна поддержка различных приложений.

Форма смартфона попадает между сотовым телефоном и карманным персональным компьютером. Обычно они имеют механизм флип-топ для отображения на весь экран и клавиатуру. В закрытом виде они выглядят как большой сотовый телефон с типовой 12-значной клавиатурой и небольшим экраном. В открытом виде имеют экран размером от 640×200 до 320×240. Они также часто предоставляют клавиатуру для ввода данных. Процессоры в этих устройствах достаточно мощные, чтобы запустить сложные локальные приложения, а также продвинутые беспроводные интернет-приложения. Браузеры для этих устройств часто имеют поддержку цветной графики и мультимедиа с использованием либо WML или HTML языков разметки.

Смартфоны, также как пейджеры и сотовые телефоны, могут ис-

пользоваться в течение нескольких дней на одном заряде аккумулятора. До недавнего времени услуги, предоставляемые для смартфонов, были малонастраиваемыми. Пользователь не мог добавить приложения или изменить содержание, доступное на устройстве. В настоящее время последние модели смартфонов предлагают полноценную операционную систему с доступной памятью для приложений сторонних разработчиков. Наиболее распространенные операционные системы в смартфонах – Android, Symbian OS, Palm OS, Pocket PC Phone Edition и Windows Mobile 7. Поддержка J2ME является обычным явлением.

Карманный персональный компьютер (КПК) – это портативное устройство, на котором предустановлена операционная система, позволяющая устанавливать и выполнять прикладные программы.

КПК, размером с ладонь (наладонные), в настоящее время занимают основное место на рынке мобильных устройств. Это устройство предоставляет пользователю возможность сенсорного ввода. По форме

КПК находятся между карманными компьютерами и смартфонами, как правило, с цветным VGA-экраном (320×240). Наладонные КПК имеют возможность мгновенно включаться/выключаться, т.к. устройства не перезагружаются перед использованием. Большинство последних устройств также имеют встроенные беспроводные модемы для беспроводной связи. Устройства без встроенного модема, как правило, имеют слот, в который беспроводной модем может быть подключен.

Наиболее распространенными устройствами на этом сегменте являются Palm и Pocket PC. В настоящее время Palm-устройства являются лидерами рынка в этой категории, хотя их доля на рынке сокращается, поскольку в устройства Pocket PC добавили больше возможностей, которые хорошо подходят для корпоративных пользователей. Оба устройства имеют сенсорные экраны со встроенной поддержкой распознавания символов. Palm-устройства доступны как в монохромном, так и в цветном видах, в то время как КПК обычно предлагают цветные экраны. Palm-устройства, как правило, несколько меньше, чем карманные ПК, что позволяет их легко носить с собой, поскольку они могут быть прикреплены к поясу или находиться в портфеле. Как правило, они весят около 225 граммов.

Хотя по форме Palm-устройства могут быть привлекательными, их производительность и возможности хранения данных часто не на должном уровне. Многие Palm-устройства базируются на операционной системе (OS) от 8 до 32 мегабайт общего объема памяти и с процессорами, работающими на 33 МГц или меньше. Поскольку эти приложения становятся все более совершенными, они требуют бóльшего хранилища данных и вычислительной мощности для удовлетворения требований приложений. Кроме того, поскольку Java-приложения развертываются на этих устройствах, они также требуют большей мощности для обра-

ботки требований виртуальной машины Java (JVM) и связанных с ней приложений. Новое поколение Palm-устройств, работающее на операционной системе OS 5.0, преодолело эти трудности с помощью перехода на ARM-процессоры, работающие на 206 МГц. Это делает их более равными на рынке с КПК.

Большинство устройств на базе Pocket PC имеют лучшие характеристики. Последние устройства на рынке имеют от 32 МБ до 64 МБ оперативной памяти вместе с X-Scale процессорами, работающими на частоте 400 МГц. Дополнительные вычислительные мощности требуются для запуска операционной системы Windows CE, которая мощнее, чем Palm OS. Одним из недостатков повышенной мощности является его влияние на потребление энергии аккумулятора. Palm-устройства

часто могут работать несколько дней на одном заряде батареи, в то время как устройства Pocket PC обычно должны заряжаться ежедневно.

Первоначально рынок этих устройств была ориентирован на создание персонального цифрового помощника, отсюда и название КПК. Palm захватил огромный рынок КПК, предлагая интуитивно понятный пользовательский интерфейс для доступа к общим приложениям, таким как календари, списки контактов и списки дел. Такой широкий спектр применения, наряду с относительно низкой стоимостью, продолжает делать Palm OS устройства привлекательными для многих пользователей КПК.

Устройства Pocket PC пришли на рынок на несколько лет позже устройств Palm. Эти устройства предназначены для аудитории, которая ищет богатый пользовательский интерфейс с мультимедийными возможностями. Кроме того, устройства также нацелены на тех, кто хочет иметь возможность использовать пакеты Microsoft Office, такие как Microsoft Word и Excel. Для достижения этих целей устройства Pocket PC должны иметь больше возможностей, что также влияет на цену. В целом, большинство устройств Pocket PC на 30–50 % дороже, чем устройства Palm OS.

Чтобы более четко различать их потребительские и корпоративные предложения, Palm создали два суббренда устройств: Tungsten и Zire. Семейство Tungsten продукции Palm предназначено для специалистов и работы на предприятиях. Эти решения сочетают мощные аппаратные и программные возможности для разрешения сложных проблем ИТ-специалистами. Семейство Zire продукции Palm направлено на потребительский рынок, предлагая низкие цены на устройства с простыми в использовании конструкциями.

Когда речь идет о беспроводной связи, большинству КПК требует-

ся модем как дополнительный компонент. Обычно это достигается с помощью слота расширения. Использование модема добавляет дополнительную стоимость и сложность для структуры КПК. В большинстве случаев пользователи должны установить подключение к беспроводной сети, когда данные должны быть переданы, и отключить, когда они закончили передачу. Это означает, что они не всегда подключены к сети, так как запуск данных в таких устройствах является более сложным.

Ручное подключение к беспроводным сетям изменилось с интеграцией комплексных беспроводных устройств, таких как Palm i705. Эти устройства поставляются со встроенными модемами для быстрого и легкого доступа к данным. Интегрированные устройства становятся все более популярными как на потребительском, так и на бизнес-рынках. Простота в использовании и онлайн-подключение к сети делают их

очень привлекательными при выборе различных мобильных приложений.

Производители делают устройства, направленные на корпоративных пользователей, которым требуются высокопроизводительные устройства для развертывания более сложных приложений. Именно на этом рынке многие продавцы ориентированы на клиентские приложения. С появлением Windows CE. NET и Palm OS 5.0 стало ясно, что этот рынок будет оставаться конкурентоспособным в течение долгого времени и это будет только на пользу потребителю, т.к. постоянно будут появляться новые возможности, привлекающие внимание.

Следует отметить, что современные сотовые телефоны также позволяют выполнять прикладные программы (приложения). Разница состоит в том, что в случае с сотовым телефоном можно говорить лишь о запуске программ с ограниченной функциональностью, запущенных в специально выделенной изолированной среде (так называемой «песочнице»). В случае со смартфонами и коммуникаторами функциональные возможности программ практически не ограничены, например в платформе Android, при желании, можно заменить даже программный модуль сотового телефона.

Если разделение на КПК и телефоны является очевидным, то различие между смартфонами и коммуникаторами медленно уменьшается. Само же разделение произошло, исходя из того, на какой основе возникли те или иные устройства:

- Коммуникаторы – это КПК, в которые была добавлена поддержка сотовых сетей. Примером коммуникаторов могут служить устройства под управлением операционной системы Windows Mobile.
- Смартфоны – это сотовые телефоны с операционной системой, приспособленной для запуска сторонних приложений. Классическим

примером смартфонов являются устройства на основе операционной системы Symbian OS.

На рис. 1.1 графически показано, как соотносятся эти категории мобильных устройств.

Основным фактором, который в настоящий момент позволяет разделить коммуникаторы и смартфоны, является различие в принципах ввода данных. Стандартным средством ввода данных для КПК и, соответственно, коммуникаторов, является сенсорный экран. В смартфонах сенсорные экраны не используются, а исторически применяется клавиатурный ввод в качестве основного.

Рис. 1.1. Категории мобильных устройств

Однако даже эта разница медленно, но верно исчезает: появившийся в 2007 году Apple iPhone, который на момент выпуска не поддерживал возможности установки дополнительных программ и, соответственно, мог быть отнесен к категории «телефоны», имел поддержку сенсорного ввода. Популярность этого телефона привела к тому, что крупнейшие компании, создающие смартфоны, внедрили возможность сенсорного ввода для своих платформ.

1.2. Технические характеристики мобильных устройств

1.2.1. Конструкция мобильных устройств

Чаще всего конструкцию мобильного устройства называют «форм-фактором». Под форм-фактором понимается совокупность элементов для ввода и вывода информации. В качестве примера можно рассмотреть, из каких элементов состоит коммуникатор Treo 600, приведенный на рис. 1.2 и 1.3:

1) индикатор статуса;
2) динамик коммуникатора;
3) цветной экран;
4) кнопка ответа;
5) четырехпозиционная кнопка навигации;
6) кнопка включения/выключения коммуникатора и завершения разговора;
7) кнопка доступа к функциям телефона;
8) кнопка вызова стандартного приложения (календаря);

9) кнопка вызова приложения сообщений;

10) кнопка выхода в главное меню;
11) QWERTY-клавиатура;
12) клавиатура набора номера;
13) стилус;
14) зеркало для автопортрета;
15) цифровая камера;
16) внешний динамик;
17) слот для SIM-карты;
18) съемный аккумулятор;
19) кнопка регулирования громкости;
20) настраиваемая кнопка;
21) слот расширения Secure Digital;
22) отключение звука звонка;
23) гнездо для аудио-гарнитуры;
24) мультиконнектор;
25) микрофон.

Рис. 1.2. Внешний вид коммуникатора Treo 600

Рис. 1.3. Конструкция коммуникатора Treo 600

Если устройство, например коммуникатор Treo 600, не имеет никаких подвижных частей, то его обычно называют «моноблоком». Коммуникатор Treo 600 имеет форм-фактор «моноблок с qwerty-клавиатурой». Впрочем, приведенный пример является далеко не единственным: среди смартфонов и коммуникаторов встречаются и «раскладушки» – когда аппарат может раскладываться на две части, «слайдеры» (причем как вертикальные, так и горизонтальные) – когда одна часть устройства сдвигается относительно другой.

Все мобильные устройства включают встроенный компьютер со всеми присущими ему составными частями, такими как процессор и оперативная память.

1.2.2. Процессоры мобильных устройств

Ключевой процессорной архитектурой, использующейся в мобильных устройствах, стала архитектура Advanced RISC Machine (ARM). В настоящий момент по данной архитектуре производится около 75 % из всех встраиваемых 32-битных RISC-процессоров, что делает ее одной из наиболее широкоиспользуемых архитектур в мире. Необходимо отметить, что процессоры ARM используются не только в портативных устройствах, но и в компьютерной периферии – от жестких дисков до маршрутизаторов (роутеров).

Разработкой данной архитектуры занимается компания ARM Limited. В ее задачи входит создание научных разработок и продажа лицензий сторонним фирмам. Такие процессоры, как Intel XScale или Texas Instruments OMAP, являются реализациями архитектуры ARM.

В настоящее время разработано шесть основных семейств процессоров: ARM7 (рис. 1.4), ARM9, ARM9E, ARM10, ARM11 и SecurCore. Также совместно с компанией Intel разработаны семейства XScale и StrongARM.

Как дополнение к ARM-архитектуре могут интегрироваться несколько расширений:

- Набор инструкций Thumb, позволяющий использовать 16-разрядные инструкции, пользуясь при этом всеми преимуществами 32-разрядной системы – например, доступом к памяти с 32-разрядным адресным пространством.
- Технология SIMD (несколько данных в одной инструкции), позволяющая ускорить обработку медиаданных. Например, быстродействие аудио/видеокодеков может быть увеличено с ее помощью в 4 раза.
- Набор инструкций DSP, интегрирующий в процессор инструкции DSP, выполняющие 16- и 32-разрядные арифметические операции для обработки сигналов в реальном времени.
- Технология Jazelle, позволяющая процессору выполнять Java-код на аппаратном уровне.
- Технология Intelligent Energy Manager (IEM), объединяющая аппаратные и программные компоненты, которые совместно выполняют динамическое распределение питания.

Рис. 1.4. Процессор ARM7TDMI от Atmel

- Технология TrustZone, предназначенная для защиты памяти устройства от критических сбоев, вызываемых вирусами и неправильной работой программного обеспечения.

Аппаратная платформа

Как уже можно было понять из описания платформы ARM, процессор в портативных устройствах отличается от процессоров настольных персональных компьютеров (ПК) тем, что состоит из большого количества различных модулей, отвечающих не только за вычисление. Для примера можно рассмотреть «чипсет» MSM7200 компании Qualcomm, под управлением которого работает большинство современных коммуникаторов под управлением ОС Windows Mobile. В этом чипе два ARM процессора: 400 МГц ARM11 для обработки приложений и 279 МГц ARM9 для работы модема, каждый из которых имеет свой цифровой сигнальный процессор (QDSP5000 и QDSP4000). Встроенный графический процессор ATi Imageon 2300 позволяет работать с 2D и 3D графикой, проигрывать видео встроенным MPEG4 декодером и захватывать фото и видео с помощью JPEG кодека. Контроллер памяти позволяет работать с NAND и SDRAM.

1.2.3. Оперативная память мобильных устройств

Используемую в мобильных устройствах память можно разделить на два типа: Random access memory (RAM) и Read-only memory (ROM).

Память с произвольной выборкой (Random access memory, RAM) представляет собой один из видов памяти, позволяющий в любой момент времени получить доступ к любой ячейке по еѐ адресу на чтение или запись. RAM подразделяется на статическую и динамическую. В статических ОЗУ запоминающий элемент представляет собой триггер, что позволяет считывание информации без еѐ потери. В динамических ОЗУ элементом памяти является ѐмкость (например, входная ѐмкость полевого транзистора), что требует восстановления записанной информации в процессе еѐ хранения и использования. Это усложняет применение ОЗУ динамического типа, но позволяет реализовать больший объѐм памяти. В современных динамических ОЗУ имеются встроенные системы синхронизации и регенерации, поэтому по внешним сигналам управления они не отличаются от статических.

Можно выделить следующие типы RAM:

- Полупроводниковая статическая (SRAM) память – ячейки представляют собой полупроводниковые триггеры. Достоинства – небольшое энергопотребление, высокое быстродействие, отсутствие необхо-

димости производить «регенерацию». Недостатки – малый объѐм, высокая стоимость. В настоящее время широко используется в качестве кэш-памяти процессоров.

Полупроводниковая динамическая (DRAM) память – каждая• ячейка представляет собой конденсатор на основе перехода КМОП-транзистора. Достоинства – низкая стоимость, большой объѐм. Недостатки – необходимость периодического считывания и перезаписи каждой ячейки, т.н. «регенерация», и, как следствие, понижение быстродействия, большое энергопотребление. Процесс регенерации реализуется специальным контроллером, установленным на материнской плате или в центральном процессоре. DRAM обычно используется в качестве оперативной памяти, в т.ч. и мобильных устройств.

Постоянное запоминающее устройство (Read-only memory, ROM) – это второй существующий вид памяти. Особенностью его является то, что данные, хранящиеся в ROM, не могут быть изменены (по крайней мере, это невозможно выполнить быстро).

В строгом смысле этого слова, термин ROM может относиться только к самому старому типу твердотельных ПЗУ, известному как mask (масочный) ROM, которые производятся с заранее введенными данными и в принципе не может быть изменены. Однако в более современных типах ПЗУ, данные могут быть стерты и перепрограммированы заново множество раз. Термин ROM (т.е. память, предназначенная только для чтения) применяется к ним потому, что процесс перепрограммирования предполагается нерегулярным, относительно медленным и, зачастую, не позволяющим произвольные выборки на запись для конкретных ячеек памяти. Несмотря на простоту масочного ПЗУ, возможности по масштабированию и перепрограммированию на лету делает современные типы ROM более гибкими и недорогими в использовании. Это привело к тому, что в настоящее время масочное ПЗУ в устройствах практически не используется.

Когда речь идет о мобильных устройствах, то под термином ROM подразумевается флеш-память. Флеш-память (flash-memory) – разновидность твердотельной полупроводниковой энергонезависимой перезаписываемой памяти. Ее особенностью является то, что она может быть прочитана сколько угодно раз, но количество перезаписей такой памяти ограничено. В настоящее время выделяют два основных типа флеш-памяти: NOR и NAND. Эти архитектуры в настоящее время существуют параллельно и не конкурируют друг с другом, поскольку находят применение в разных областях хранения данных.

Применение RAM и ROM в мобильных устройствах

Многие современные мобильные устройства применяют оба типа памяти и это обусловлено преимуществами и недостатками каждого из

них. Так, RAM, являясь очень быстрым типом памяти, потребляющим достаточно много энергии, в основном используется в качестве оперативной памяти. ROM же намного медленнее, но гораздо более экономично расходует энергию, т.к. не нуждается в постоянном питании для хранения данных. Соответственно, в качестве носителя информации в настоящее время применяется ROM. Впрочем, этот принцип не является догматическим. Так, в устройствах Windows Mobile, вплоть до пятой версии, вся используемая память была энергозависимой, что позволяло операционной системе динамически выделять необходимое ей количество оперативной памяти и мгновенно запускать процессы, находящиеся в RAM. Обратной стороной такого подхода было то, что при разрядке батарей, вся информация, в т.ч. контакты, установленные программы и т.д., стиралась.

Применение двух основных типов флеш-памяти: NAND и NOR тоже обуславливается особенностями их работы. Адресное пространство NOR-памяти позволяет работать с отдельными байтами или словами (2 байта). В NAND ячейки группируются в небольшие блоки (по аналогии с кластером жесткого диска). Из этого следует, что при последовательном чтении и записи преимущество по скорости будет у NAND. Однако с другой стороны NAND значительно проигрывает в операциях с произвольным доступом и не позволяет напрямую работать с байтами информации.

1.3. Коммуникационные технологии

Возникновение Интернета превратило персональный компьютер из автономного устройства, которое может работать исключительно с программами и данными в его памяти, в участника локальных и глобальной компьютерной сети, которому доступны совершенно новые объемы данных и возможности.

В конце 90-х годов прошлого века на рынке появились первые гибриды карманных персональных компьютеров и телефонов – Handspring Visorphone или Ericsson R380 (рис. 1.5). Чтобы дать представление о том, какой путь за это время прошли мобильные устройства необходимо отметить, что первое из указанных устройств было не коммуникатором, а отдельной приставкой, подключавшейся к КПК на основе Pal-

mOS; а второе, хоть и работало на основе ОС EPOC, не позволяло установку стороннего программного обеспечения.

Рис. 1.5. Ericsson R380

Современные технологии по передаче данных рассмотрены ниже.

1.3.1. Стандарт GSM

Самым популярным стандартом сотовой связи в мире является стандарт GSM (Global System for Mobile Communications). Согласно данным Ассоциации GSM, ему принадлежат 82 % рынка глобальной связи, более 3 миллиардов людей в 212 странах используют именно его. Необходимо отметить, что существуют мобильные устройства, которые поддерживают другие стандарты сотовой связи, такие как американский CDMA или японский FOMA. Стандарт GSM является самым популярным и наиболее распространенным стандартом сотовой связи в России и мире.

Поколения мобильных сетевых технологий

Всю историю развития мобильных сетевых технологий, которая берѐт свое начало во второй половине прошлого века, можно разделить на 3 поколения (табл. 1.1).

Таблица 1.1

Поколения мобильных сетевых технологий

Поколение	1G	2G	2.5G	3G
Начало разработок	1970 г.	1980 г.	1985 г.	1990 г.
Реализация	1984 г.	1991 г.	1999 г.	2002 г.
Сервисы	аналоговый стандарт, синхронная передача данных со скоростью до 9,6 кбит/с.	цифровой стандарт, поддержка коротких сообщений (SMS)	большая ѐмкость, пакетная передача данных	ещѐ большая ѐмкость, скорости до 2 Мбит/с

Поколение	1G	2G	2.5G	3G
Стандарты	AMPS, TACS, NMT и др.	TDMA, CDMA, GSM, PDC	GPRS, EDGE,	WCDMA, CDMA2000,

	1xRTT	UMTS		
Ширина канала	1,9 кбит/с	14,4 кбит/с	384 кбит/с	2 Мбит/с

Первое поколение (1G)

Европейским стандартом сотовой связи первого поколения стал NMT (Nordic Mobile Telephone system). Его окончательные спецификации были приняты в 1978 году пятью скандинавскими странами (Данией, Финляндией, Исландией, Норвегией и Швецией). Стандарт NMT является стандартом аналоговой сотовой связи и работает в диапазоне частот 453,0–457,5 МГц, используя до 180 каналов связи по 25 кГц каждый. Радиус действия одной базовой станции достигает 5–25 км в зависимости от нагрузки на каждую из них. В 1983 году была разработана модернизированная версия NMT-900 (первая условно называлась NMT-450), работавшая на частоте 900 МГц. Выход обновлѐнного стандарта позволил уменьшить размеры телефонных аппаратов, а также добавить несколько новых сервисов. Тем не менее, спустя некоторое время NMT был заменен на более прогрессивные цифровые стандарты. Вполне естественно, что первое поколение сотовой связи не смогло с ними конкурировать, т.к. несмотря на то, что качество аналоговой беспроводной связи в целом было удовлетворительным, разговор можно было легко перехватить и расшифровать.

Второе поколение (2G)

Принципиально новым подходом к передаче информации (в частности, голоса) отличалось второе поколение мобильных коммуникаций – на этот раз в его основу легли цифровые стандарты. В Европе на смену NMT пришел стандарт GSM, который применяется по настоящее время. Цифровой стандарт предполагает, что голос человека теперь проходит оцифровку (кодирование). То есть по каналу связи, как и в 1G-стандарте, передается модулированная несущая частота, но уже не аналоговым сигналом, а цифровым кодом.

Внедрение цифрового стандарта привело к возникновению новых сервисов, самым известным из которых, пожалуй, можно назвать сервис обмена короткими сообщениями (SMS). Одним из существенных недостатков сетей сотовой связи стандарта GSM – низкая скорость передачи данных (максимум 9,6 кбит/с). Организация этого процесса тоже была далека от совершенства – для передачи данных абоненту выделялся один голосовой канал, а биллинг осуществлялся, исходя из времени соединения.

Второе с половиной поколение (2,5G)

Для расширения возможностей передачи данных посредством существующих GSM-сетей была разработана услуга пакетной передачи данных по радиоканалу GPRS (General Packet Radio Service). Для передачи данных в GPRS начали использоваться одновременно многие каналы связи в паузах между передачей речи, что позволило увеличить скорость и производить учет и оплату услуг (биллинг) пропорционально объему переданной информации. В связи с минимальной приоритетностью GPRS пакетов при звонке соединение по GPRS временно приостанавливается или обрывается.

Именно появление GPRS позволило реализовать мобильный Интернет, предоставив возможность получать доступ к глобальной сети с сотовых телефонов, смартфонов и коммуникаторов. Тем не менее, скорость передачи данных при использовании GPRS оставляет желать лучшего. Официально максимальный его предел равен 115 кбит/с, а в реальности обмен информацией производится не быстрее, чем на скорости 40–50 кбит/с, что в два раза меньше теоретического максимума. По сегодняшним меркам такой пропускной способности не хватит для нормальной работы в сети Интернет.

В качестве решения этой проблемы был разработана технология EDGE (Enhanced Data rates for GSM Evolution), которую иногда относят к отдельному поколению – 2,75G. EDGE является расширением GPRS и не может существовать отдельно от GPRS. Главное различие между GPRS и EDGE – в использовании иной модуляционной схемы на физическом уровне, что позволило достичь пропускной способности примерно втрое больше, чем в технологии GPRS.

Третье поколение (3G)

Термин 3G используется для описания сервисов мобильной связи стандарта следующего поколения, которые обеспечивают более высокое качество звука, а также высокоскоростной доступ в Интернет и мультимедийные сервисы. Мобильные сети третьего поколения отличаются от сетей второго поколения гораздо большей скоростью передачи данных, а также более широким набором и высоким качеством предоставляемых услуг.

Хотя существует много различных интерпретаций того, что представляет собой 3G, единственным определением, принимаемым универсально, является определение, опубликованное Международным институтом электросвязи (ITU). ITU, работающий с промышленными организациями по всему миру, определяет и утверждает технические требова-

ния и стандарты, а также правила использования спектра для систем 3G в рамках программы IMT-2000 (International Mobile Telecommunications-2000). IMT-2000 – это рекомендации, разработанные Международным институтом электросвязи, касающиеся вопросов использования частотного спектра и технических особенностей для всего семейства стандартов 3-го поколения. Рекомендации описывают пути эволюции существующих в мире стандартов 2-го поколения в стандарты 3-го поколения. ITU требует, чтобы сети IMT-2000 (3G), помимо прочих свойств, обеспечивали улучшенную ѐмкость системы и эффективность использования спектра для систем 2G и поддерживали сервисы передачи данных со скоростями – минимум 144 кбит/с, при использовании в мобильном режиме (не в помещениях), и максимум 2 Мбита/с, в не мобильных условиях (в помещениях).

Основываясь на этих требованиях, в 1999 году ITU одобрил пять радиоинтерфейсов для стандартов IMT-2000, как часть рекомендаций ITU-R M.1457. Ключевыми стали два стандарта 3G: UMTS (Universal Mobile Telecommunications Systems – универсальная мобильная телекоммуникационная система), поддерживаемая европейскими странами, и CDMA 2000 (Code Division Multiple Access – множественный доступ с кодовым разделением каналов), сторонниками которой традиционно являются азиатские страны и США. В принципе, эти две технологии предполагают два различных подхода к организации сетей 3G: революционный (UMTS) и эволюционный (разновидности CDMA – CDMA2000, CDMA2000 IX, CDMA2000 IX EvDo). Эволюционный путь подразумевает сохранение частот и постепенный переход к новым технологиям путѐм наращивания технических мощностей оператора. UMTS – совершенно новый стандарт, в то время как разновидности CDMA, предложенные для 3G, являются развитием уже эксплуатирующейся в мире технологии второго поколения cdmaOne (IS-95).

Всего существует три основных стандарта 3G: UMTS (Universal Mobile Telecommunications Service), CDMA2000 и WCDMA (Wide CDMA). Все они настроены на пакетную передачу данных и, соответственно, на работу с цифровыми компьютерными сетями, включая Интернет. Скорость передачи данных в новом поколении стандартов может достигать 2,4 Мбит/с. Это позволит поднять качество звука, а также добавить такой сервис, как видеозвонок, о котором, вероятно, слышали уже многие. Мобильный Интернет теперь станет доступнее и значительно быстрее.

В настоящее время сети 3G уже работают в Азии, США, в то время как в Европе существует пока только в тестовых вариантах. Наиболее

впечатляющих успехов в области 3G на мировом фоне добилась Япония.

1.3.2. Технология Wi-Fi

В основе WLAN-технологий лежит принцип высокочастотной радиосвязи между узлами сети. В качестве узла сети может выступать как отдельный компьютер, ноутбук или КПК, так и специальное устройство – «точка доступа» («Access Point»), обеспечивающее доступ к кабельному сегменту сети Ethernet, Интернету или другому компьютеру.

Специальные стандарты для WLAN-сетей разрабатываются Институтом инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers), более известного под аббревиатурой IEEE. Первый стандарт IEEE 802.11 для беспроводных локальных сетей был принят в 1997 году. Он подразумевал работу оборудования на частоте 2,4 ГГц со скоростями 1 и 2 Мбит/с. Стандарт разрабатывался в течение 7 лет и поэтому ко времени принятия уже не мог соответствовать выросшим потребностям.

Новый расширенный вариант стандарта, названный 802.11b (802.11 High Rate), был принят в 1999 году. С его принятием стала возможной работа беспроводных сетей на скоростях до 11 Мбит/с, что было сопоставимо по скорости с обычными сетями Ethernet. Такая скорость позволила существенно расширить область применения беспроводных сетей и поднять уровень задач, для которых стало возможным использование WLAN.

В том же 1999 году была создана независимая международная организация Wi-Fi Alliance (Wi-Fi – сокращение от Wireless Fidelity), занимающаяся сертификацией на совместимость WLAN-устройств от различных производителей. Эта организация объединяет практически всех ведущих производителей Intel, IBM, Cisco, HP, Dell и др. В настоящий момент в нее входит более 200 компаний, и уже более 1500 устройств получили сертификат Wi-Fi с момента начала сертификации в марте 2000 года.

Торговая марка Wi-Fi гарантирует совместимость оборудования от разных производителей. Первоначально в ноутбуках использовались адаптеры стандарта 802.11b, поэтому логотип WiFi часто ассоциировался именно с этим стандартом. В настоящее время под Wi-Fi понимается любой из стандартов 802.11: a, b или g.

1.3.3. Стандарты передачи данных IEEE 802.11

Существует несколько разновидностей WLAN-сетей, которые различаются схемой организации сигнала, скоростями передачи данных, радиусом охвата сети, а также характеристиками радиопередатчиков и приемных устройств. Наибольшее распространение получили беспроводные сети стандарта IEEE 802.11b, IEEE 802.11g и IEEE 802.11a. Их сравнение представлено в таблице 1.2.

Первыми в 1999 году были утверждены спецификации 802.11a и 802.11b, однако наибольшее распространение получили устройства, выполненные по стандарту 802.11b.

В стандарте 802.11b применяется метод широкополосной модуляции с прямым расширением спектра – DSSS (Direct Sequence Spread Spectrum). Весь рабочий диапазон делится на 14 каналов, разнесенных на 25 МГц для исключения взаимных помех. Данные передаются по одному из этих каналов без переключения на другие. Возможно одновременное использование всего 3 каналов. Скорость передачи данных может автоматически меняться, в зависимости от уровня помех и расстояния между передатчиком и приемником.

Таблица 1.2

Стандарты IEEE 802.11

Характеристики	Спецификации		
	IEEE 802.11a	IEEE 802.11b	IEEE 802.11g
Скорость передачи данных	до 54 Мбит/с	11 Мбит/с	до 54 Мбит/с
Расстояние и скорость передачи данных	В закрытых помещениях: 12 м (54 Мбит/с), 91 м (6 Мбит/с). В открытых помещениях в пределах прямой видимости: 30 м (54 Мбит/с), 305 м (6 Мбит/с)	В закрытых помещениях: 30 м (11 Мбит/с), 91 м (1 Мбит/с). В открытых помещениях в пределах прямой видимости: 120 м (11 Мбит/с), 460 м (1 Мбит/с)	В закрытых помещениях: 30 м (54 Мбит/с), 91 м (1 Мбит/с). В открытых помещениях в пределах прямой видимости: 120 м (54 Мбит/с), 460 м (1 Мбит/с)
Рабочая частота	5 ГГц (5,15–5,350 ГГц и 5,725–5,825 ГГц)	2,4 ГГц (2,4–2,4835 ГГц)	2,4 ГГц (2,4–2,4835 ГГц)

Стандарт IEEE 802.11b обеспечивает максимальную теоретическую скорость передачи 11 Мбит/с, что сравнимо с обычной кабельной сетью

10 BaseT Ethernet. Однако такая скорость возможна лишь при условии, что в данный момент только одно WLAN-устройство осуществляет передачу. При увеличении числа пользователей полоса пропускания делится на всех и скорость работы падает. Несмотря на ратификацию стандарта 802.11a в 1999 году, он реально начал применяться только с 2001 года. Данный стандарт используется в основном в США и Японии. В России и в Европе он не получил широкого распространения.

В стандарте 802.11a используется OFDM-схема модуляции сигнала – мультиплексирование с разделением по ортогональным частотам (Orthogonal Frequency Division Multiplexing). Основной поток данных разделяется на ряд параллельных подпотоков с относительно низкой скоростью передачи, и далее для их модуляции используется соответствующее число несущих. В стандарте определены три обязательные скорости передачи данных (6, 12 и 24 Мбит/с) и пять дополнительных (9, 18, 24, 48 и 54 Мбит/с). Также имеется возможность одновременного использования двух каналов, что обеспечивает увеличение скорости вдвое.

Стандарт 802.11g окончательно был ратифицирован в июне 2003 года. Он является дальнейшей разработкой спецификации IEEE 802.11b и осуществляет передачу данных в том же частотном диапазоне. Основным преимуществом этого стандарта является увеличенная пропускная способность – скорость передачи данных составляет до 54 Мбит/с по сравнению с 11 Мбит/с у 802.11b. Как и IEEE 802.11b, новая спецификация предусматривает использование диапазона 2,4 ГГц, но для увеличения скорости применена та же схема модуляции сигнала – что и в 802.11a – ортогональное частотное мультиплексирование (OFDM).

Особенностью данного стандарта является совместимость с 802.11b. Например, адаптеры 802.11b могут работать в сетях 802.11g (но при этом не быстрее 11 Мбит/с), а адаптеры 802.11g могут снижать скорость передачи данных до 11 Мбит/с для работы в старых сетях 802.11b.

Сейчас ведутся разработки нового стандарта WLAN – IEEE 802.11n. Он должен работать вдвое быстрее, чем 802.11a и 802.11g, на скорости от 100 Мбит/с до максимального значения 540 Мбит/с.

1.3.4. Протокол Bluetooth

Bluetooth – технология беспроводной связи, созданная в 1998 году группой компаний: Ericsson, IBM, Intel, Nokia, Toshiba. В настоящее время разработки в области Bluetooth ведутся Bluetooth SIG (Special Interest Group), в которую входят также Lucent, Microsoft и многие другие.

Основное назначение Bluetooth – обеспечение экономичной (с точки зрения потребляемого тока) и дешевой радиосвязи между различными типами электронных устройств, причем немалое значение придается компактности электронных компонентов, что дает возможность применять Bluetooth в малогабаритных устройствах размером с наручные часы.

Интерфейс Bluetooth позволяет передавать голос и данные. Для передачи данных могут быть использованы асимметричный и симметричный методы. Работающий на частоте 2,4 ГГц приемопередатчик, коим является Bluetooth-чип, позволяет в зависимости от степени мощности устанавливать связь в пределах 10 или 100 метров. Разница в расстоянии, безусловно, большая, однако соединение в пределах 10 м позволяет сохранить низкое энергопотребление, компактный размер и достаточно невысокую стоимость компонентов.

Одним из выгодных моментов в реализации этой технологии является рабочий диапазон частот – 2,45 ГГц; это так называемый нижний диапазон, обозначаемый аббревиатурой ISM (Industrial, Scientific, Medical) и использующийся для работы промышленного, научного и медицинского оборудования. Он разрешен к свободному использованию по всему миру, и для применения соответствующих устройств не требуется дополнительных лицензий и разрешений.

Всего существует три коммерческие версии протокола: 1.1, 1.2 и 2.0. Все они обратно совместимы друг с другом. В 2004 году к протоколу Bluetooth 2.0 была добавлена функция «Enhanced Data Rate» (EDR), позволившая увеличить скорость передачи данных втрое.

Классов в Bluetooth три и отвечают они за радиус действия адаптера/модуля: class 1 (до 100 метров), class 2 (до 10 метров) и class 3 (до 1 метра). Однако необходимо учитывать, что дальность действия в случае технологии Bluetooth – понятие весьма абстрактное: стабильная работа спаренных устройств на расстоянии 10 или 100 метров (в зависимости от класса) может быть обеспечена только в идеальных условиях, которые в реальной жизни практически недостижимы. Кроме того, если одно устройство поддерживает Bluetooth class 2, а другое – class 1, то функционировать они смогут лишь на расстоянии до 10 метров.

Функциональность устройств по данному протоколу реализуется с использованием профилей Bluetooth. Профили Bluetooth – это своеобразный механизм, обеспечивающий функционирование связки двух и более Bluetooth-устройств: если каждое из них поддерживает один и тот же профиль, определенный в спецификации Bluetooth, то они смогут взаимодействовать соответствующим образом. Теоретически единственным обязательным профилем, который поддерживается любым Bluetooth-модулем, является GAP (Generic Access Profile) – профиль

общего доступа. Однако де-факто к нему в подавляющем большинстве случаев добавляются еще три профиля, необходимые для организации передачи данных: профиль последовательного порта SPP (Serial Port Profile), протокол приложения определения предлагаемых сервисов SDAP (Service Discovery Application Profile) и протокол операции клиент-сервер при обмене объектами GOEP (Generic Object Exchange Profile). Помимо вышеперечисленных профилей, Bluetooth-устройство может (но отнюдь не обязано) поддерживать какой-либо из девяти основных или двенадцати дополнительных профилей.

1.3.5. Организация беспроводных сетей

Выделяют два вида организации беспроводных сетей: Ad-Hoc и Infrastructure Mode. Режим Ad-Hoc является простейшей структурой локальной сети, при которой узлы сети (ноутбуки или компьютеры) связываются напрямую друг с другом. Такая структура удобна для быстрого развертывания сетей. Для ее организации требуется минимум оборудования – каждый узел должен быть оборудован адаптером WLAN.

В режиме Infrastructure Mode узлы сети связаны друг с другом не напрямую, а через точку доступа (Access Point). Различают два режима взаимодействия с точками доступа – BSS (Basic Service Set) и ESS (Extended Service Set). В режиме BSS все узлы связаны между собой через одну точку доступа, которая может играть роль моста для соединения с внешней кабельной сетью. Режим ESS представляет собой объединение нескольких точек доступа, т.е. объединяет несколько сетей BSS. В этом случае точки доступа могут взаимодействовать и друг с другом. Расширенный режим удобно использовать тогда, когда необходимо объединить в одну сеть несколько пользователей или подключить нескольких проводных сетей.

Важным вопросом при организации WLAN-сетей является дальность покрытия. На этот параметр влияет сразу несколько факторов:

1) используемая частота (чем она больше, тем меньшая дальность действия радиоволн);

2) наличие преград между узлами сети (различные материалы поразному поглощают и отражают сигналы);

3) режим функционирования (Infrastructure Mode или Ad Hoc);

4) мощность оборудования.

Если рассматривать идеальные условия, то зона покрытия с одной точкой доступа будет иметь следующий средний радиус покрытия:

- сеть стандарта IEEE 802.11a – 50 м;

сети 802.11b и 802.11g – порядка 100 м.•

За счет увеличения количества точек доступа (в режиме Infrastructure ESS) можно расширять зоны покрытия сети на всю необходимую область охвата.

1.3.6. Безопасность беспроводных сетей

Для WLAN-сетей очень актуальны вопросы безопасности и защиты передаваемых данных, т.к. для перехвата данных в общем случае достаточно просто оказаться в зоне действия сети.

Первоначально созданные в этой сфере технологии обладали невысокой степенью защиты, данная проблема остается актуальной и на сегодняшний день.

Для защиты передаваемых данных предусмотрены следующие методы:

использование MAC-адресов (Media Access Control ID): у каждо-• го адаптера есть свой, абсолютно уникальный код, установленный производителем. Эти адреса необходимо занести в списки адресов доступа у используемых для организации сети точек доступа. Все остальные WLAN-адаптеры с неправильными адресами будут исключены из сети автоматически.

использование ключей SSID (Service Set Identifier): каждый ле-• гальный пользователь сети должен получить от администратора сети свой уникальный идентификатор сети.

шифрование данных.•

Первые два способа не обеспечивают защиты от прослушивания и перехвата пакетов данных, поэтому защитить сеть в случае перехвата данных можно только с помощью шифрования.

Изначально стандарт 802.11 предусматривал аппаратный протокол шифрования данных WEP (Wired Equivalent Privacy – защищенность, эквивалентная беспроводным сетям), основанный на алгоритме шифрования RC4. Однако в скором времени было обнаружено, что защищенную с его помощью сеть довольно легко взломать. Ранние версии предусматривали шифрование с использованием 40-битного ключа, более поздние – 64-, 128- или 256-битное. Но даже такая длина ключа в WEP не может обеспечить высокий уровень защиты сети, т.к. основная слабость данной технологии заключается в статичности ключа шифрования. Хотя при использовании данного ключа увеличивается время взлома и количество пакетов данных, которые нужно перехватить, чтобы вычис-

лить ключ, сама возможность взлома остается. Это абсолютно неприемлемо для определенного круга серьѐзных компаний и организаций.

На смену WEP была создана новая технология WPA (Wi-Fi Protected Access), разрабатывающаяся IEEE совместно с Wi-Fi Alliance. Главной особенностью новой системы безопасности является шифрование данных с динамическими изменяемыми ключами и проверка аутентификации пользователей [2].

В отличие от WEP здесь используется протокол целостности временных ключей TKIP (Temporal Key Integrity Protocol), который подразумевает обновление ключей перед началом каждой сессии шифрования и проверкой пакетов на принадлежность к данной сессии.

Для аутентификации пользователей используются сертификаты RADIUS (Remote Authentication Dial-In User Service – сервер RADIUS должен подтвердить право доступа). Такой метод подразумевает, главным образом, корпоративное использование.

Второй упрощенный вариант аутентификации требует предварительной установки разделяемых паролей на сетевые устройства (режим аутентификации PSK (Pre-Shared Keys)). Этот метод лучше всего применять в домашних условиях или там, где не происходит обмен важной информацией.

Изначально WPA разрабатывалась как временная технология, которая со временем она должна быть заменена новым стандартом 802.11i. Данный стандарт, с учетом всех уже существующих наработок, призван обеспечить надежное шифрование передаваемых данных, а также аутентификацию пользователей сети. В большинство уже выпущенных WiFi-устройств (точки доступа, сетевые карты) можно установить протокол WPA посредством обновления программного обеспечения.

1.4. Программные платформы

Разработка и выполнения программ на мобильных устройствах выполняется с помощью комплекса программ (операционная система, промежуточное программное обеспечение и ключевые приложения), которые совместно называются программной платформой. В настоящее время для мобильных устройств разработаны различные программные платформы. На разных типах мобильных устройств преобладающими являются разные платформы.

Исследование компании Gartner также показало, что в 2010 году на рынке КПК преобладали устройства под управлением операционной системы Android – 36,1 %.

1.4.1. Платформа Android

Программная платформа Android разрабатывается консорциумом, в который входит 34 компании – Open Handset Alliance. Целью SDK является предоставление инструментов и API, необходимых для начала разработки приложений на языке Java для платформы Android.

Средства разработки программного обеспечения

- каркас приложений, позволяющий использовать много раз и заменять существующие компоненты;
- виртуальная машина Dalvik, оптимизированная для мобильных устройств;
- встроенный браузер на основе WebKit;
- оптимизированная работка графики, реализованная специальной 2D-библиотекой и 3D-графикой согласно спецификациям OpenGL ES 1.0;
- SQLite для структурированного хранения данных;
- поддержка основных фото-, аудио- и видеоформатов (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF);
- GSM-телефония;
- Bluetooth, EDGE, 3G и Wi-Fi;
- камера, GPS, компас и акселерометр.

Архитектура платформы Android

На рис. 1.6 показана диаграмма с описанием основных компонентов операционной системы Android. Каждый из элементов подробнее описывается ниже.

Рис. 1.6. Архитектура платформы Android

Приложения

Платформа Android будет поставляться с набором основных приложений, таких как почтовый клиент, программа по работе с SMS, календарь, карты, браузер, контакты и др. Все эти программы написаны на языке Java.

Разработчики программ на платформе Android имеют полный доступ ко всем API, доступным ключевым приложениям. Архитектура приложений создается для упрощения повторного использования компонентов, любое приложение может опубликовать свои возможности и любое другое приложение может ими воспользоваться (учитывая ограничения безопасности, накладываемые каркасом приложений). Этот

же механизм позволяет пользователю заменять компоненты.

Работоспособность всех приложений обеспечивает набор сервисов и систем, включающий:

- набор графических элементов, которые используется для построения приложений, включая списки, сетки, текстовые окна, кнопки и даже встраиваемый web-браузер;

- контент-провайдер, который позволяет приложениям обращаться к данным из других приложений (таких как кКонтакты) и делиться собственными данными;
- менеджер Ресурсов, обеспечивающий доступ к ресурсам, не относящимся к коду, таким как файлы локализации, графические файлы и т.д.;
- менеджер Напоминаний, позволяющий выводить на строку статуса предупреждения;
- менеджер Процессов, управляющий жизненным циклом приложений и обеспечивающий возможность переключения между ними.

Библиотеки

Платформа Android включает в себя набор C/C++ библиотек, используемых различными компонентами системы Android. Все эти возможности доступны разработчикам через каркас приложений Android. Некоторые из них приводятся ниже:

- Системная библиотека на языке C – производная от BSD имплементация стандартной системной библиотеки языка C (libc), настроенной для работы с мобильными устройствами на основе Linux.
- Медиабиблиотеки – основанные на OpenCORE от PacketVideo. Это библиотеки обеспечивают возможность проигрывания и записи большинства наиболее популярных фото-, аудио- и видеоформатов, таких как MPEG4, H.264, MP3, AAC, AMR, JPG и PNG.
- Экранный менеджер – управляет доступом к подсистеме дисплея, позволяя одновременно работать 2D- и 3D-графике различных приложений.
- LibWebCore – современное базовое ПО для поддержки работы браузеров, обеспечивающее работу не только самого браузера Android, но и встраиваемого в приложения web-обозревателя.
- SGL – встроенная 2D-система графической визуализации.
- 3D-библиотеки – основанные на API OpenGL ES 1.0, библиотеки могут использовать и аппаратное ускорение 3D, если оно предусмотрено, и оптимизированную программную 3D-растеризацию.
- FreeType – работа с векторными и растровыми шрифтами.

- SQLite – производительная и компактная реляционная база данных, доступная всем приложениям.

Android Runtime

Android включает в себя набор библиотек, которые представляют бóльшую часть функциональности ключевых библиотек языка программирования Java. Каждое приложение для Android работает в своем отдельном процессе с собственным экземпляром виртуальной машины Dalvik. Dalvik был написан с расчетом, что несколько виртуальных ма-

шин могут одновременно эффективно работать. Dalvik выполняет файлы формата .dex, которые оптимизированы для работы с минимальным объемом оперативной памяти. Виртуальная машина основана на регистрах, запускает классы, скомпилированные компилятором языка Java, которые были превращены в формат .dex специальной утилитой dx из Android SDK.

Для реализации функциональности низкого уровня, такой как организация поточной обработки и низкоуровневое управление памятью, виртуальная машина Dalvik работает с ядром Linux.

Ядро Linux

Основные системные сервисы Android, такие как безопасность, управление памятью, управление процессами, сетевой стек и модель драйверов, основываются на Linux версии 2.6. Ядро также работает как уровень абстракции между аппаратным обеспечением и стеком программ.

1.4.2. Java 2 Micro Edition (J2ME)

Язык программирования Java был создан в 1995 году компанией Sun Microsystems. Разработан он для того, чтобы программы написанные программистами один раз могли бы работать на разных типах мобильных устройств. В 1998 году произошло разделение языка Java на Standard Edition (J2SE), который предназначается для обычных компьютеров, Enterprise Edition (J2EE), используемый на серверах, и Micro Edition (J2ME), который и устанавливается в мобильные устройства.

Как уже говорилось в первой главе, основным отличием мобильных телефонов от смартфонов и коммуникаторов является то, что все они работают не под управлением полноценной мобильной ОС, а под управлением прошивки и установка каких-либо новых программ невозможна. Однако решение было найдено именно с помощью Java ME.

Идея Java состоит в том, что команды отдаются не напрямую процессору, а виртуальной Java-машине (JVM – Java Virtual Machine). Вместо команд процессора программа на Java представляет собой байт-код – команды, которые должна выполнять Java-машина. Соответственно, для

работы программы достаточно, чтобы в системе была установлена Java-машина. Решением поддержки приложений в сотовых телефонах стала установка в прошивки виртуальной Java ME машины под названием KVM – Kilobyte Virtual Machine.

Для программ, которые рассчитаны на Java ME, есть особое название – мидлет. Мидлеты распространяются не в виде разрозненных файлов, а в виде специальных архивов JAR – Java Archive. В нем хранятся все файлы программы: .class (они содержат байт-код), файлы ресурсов

(например, картинки или звуки) и файл-манифест, описывающий программу: название, производитель, версия и другие данные.

Функционал телефонов может серьезно отличаться, поэтому для Java ME существует целый ряд различных API. Базовый API, на котором строятся все остальные, – это либо CDC (Connected Device Configuration), либо CLDC (Connected Limited Device Configuration). Для смартфонов, коммуникаторов и КПК предназначен CDC, как более функциональный; для мобильников – CLDC.

Поскольку сотовые телефоны отличаются по устройству от компьютеров, потребовался API, обеспечивающий работу GUI и хранение настройки приложений и поддержку других специфических для мобильников возможностей. Эту задачу берет на себя API под названием Mobile Information Device Profile (MIDP). На данный момент существует несколько версий: MIDP 1.0 был создан еще в 2000 году и накладывал ряд серьезных ограничений, поэтому в 2002 была выпущена новая версия – MIDP 2.0, использующаяся до сих пор.

Необходимо отметить, что список встраиваемых в телефоны API не ограничивается CLDC и MIDP. Существует целый ряд спецификаций мобильных стандартов Java, разработанных по запросам на спецификацию (JSR – Java Specification Request), которые также реализованы в качестве API и обычно носят номер соответствующего им запроса.

Возможности Java ME

Возможности каждого отдельного телефона можно описать несколькими параметрами, самым главным из которых будут поддерживаемые API. Основные API – это уже описанные CLDC и MIDP. Но кроме них есть и множество других. Некоторые используются широко, некоторые – не очень.

Например, заявленная поддержка Java 3D для телефона означает поддержку JSR 184 – Mobile 3D Graphics API (сокращенно M3G), а за возможности работы java-мидлета с видео и аудио отвечает JSR 135 – Mobile Media API.

Преимущества и недостатки Java ME

Java обладает следующими достоинствами:

Один из главных плюсов Java ME – полное отсутствие вирусов• и червей и очень небольшие возможности для троянских программ. Если исключить случаи некорректной реализации Java-машин, существование вирусов и червей на Java ME принципиально невозможно. Дело в том, что для всех Java-машин на современных телефонах такие действия, как посылка sms или передача данных через Bluetooth должны подтверждаться пользователем, что практически блокирует любые возможности по работе на java-машине вредоносного кода.

Другой плюс Java ME – гибкость реализации. Каждый произво-• дитель решает, какой именно функционал поддерживается тем или иным телефоном. С другой стороны, это создает определенные сложности для разработки мидлетов из-за большого разнообразия аппаратов.

Недостатки Java:

Работа с устройством через API. В обычной программе, которая• выполняется через API операционной системы, команды выполняются процессором, и тут контроль над действиями программы никто не осуществляет. Возможности Java сильно ограничены заложенными API и если аппарат не обладает нужной функциональностью, то сделать ничего нельзя.

Другой минус кроется в CLDC, являющейся сокращенной верси-• ей API Java SE. Этот API был разработан как набор базовых функций для телефонов, поэтому функциональность API была серьезно урезана. В настоящее время практически все телефоны поддерживают MIDP 2.0, заметно более требовательный к аппаратному обеспечению, однако это не привело к появлению новых версии CLDC.

С этим связан и еще один недостаток: в отличие от Java SE, где по-• стоянно совершенствуются сами возможности языка, добавляются новые средства, мобильная Java не изменяется со времени создания версии 3.

1.4.3. Архитектура мобильных приложений

Архитектура клиент-сервер

Архитектура приложения часто создается для иллюстрации общих свойств программного обеспечения (например, код приложения и платформа) и оборудования (например, клиент, сервер и сетевые устройства). В литературе выделяют ряд базовых шаблонов.

Архитектура приложений, как правило, проектируется в терминах архитектуры «клиент-сервер», при которой один или несколько клиентских устройств запрашивают информацию с сервера. Сервер обычно отвечает необходимой информацией (рис. 1.7).

Рис. 1.7. Клиент-серверная архитектура

Рассмотрим следующие понятия клиент-серверной архитектуры: слои, уровни и связи между слоями и уровнями.

Слои

Функциональность кода по всему приложению необязательно равномерная. Некоторые разделы кода приложения лучше подходят для обработки пользовательского интерфейса, в то время как другие разделы разработаны для управления бизнес-логикой или соединения с базой данных или серверных систем.

Расслоение описывает разделение работ внутри кода приложения на одной машине. Зачастую слои это не более чем программные модули, которые размещены в разных папках или каталогах на стороне клиента или сервера.

Со стороны клиента обычно имеются от нуля до трех слоев в коде приложения. Со стороны сервера – от одного до трѐх слоев кода приложения. Частично это важно для хорошего проектирования программного обеспечения, которое обеспечивает повторное использование кода, частично это важно для безопасности, а отчасти из соображений удобства.

Клиент с нулевым количеством слоѐв кода, по существу, не имеет специального кода приложения. Этот тип клиента обычно упоминается как тонкий клиент и возможен в архитектуре клиент-сервер, при которой сервер содержит весь пользовательский код приложения. Клиент с одним до трех слоев код приложения обычно называется толстым клиентом.

Сервер также может содержать от одного до трех слоев специального кода приложения. Тем не менее, по определению не может быть нулевого количества слоев кода на сервере.

Слой, код которого наиболее тесно взаимодействует с пользователем, часто упоминается в литературе как слой представления [4]. Второй слой обычно обрабатывает бизнес-логику кода. Третий слой назы-

вают уровнем доступа к данным (связь с базой данных или с источником данных).

Вполне возможно наличие более чем с трех уровней на стороне клиента или сервера, но слишком много слоев может привести к трудностям при управлении.

Уровни

Разбиение функциональности кода приложения на слои помогает повторному использованию, но это не делает архитектуру автоматически масштабируемой. Для того чтобы это сделать, важно распределять код на нескольких машинах.

Уровень описывает разделение работ прикладного кода на нескольких машинах. Многоуровневость, как правило, предполагает размещение программных модулей на разных машинах в распределенной серверной среде. Если код приложения уже распределѐн по слоям, это делает многоуровневость гораздо более простым процессом.

Код, который наиболее тесно взаимодействует с пользователем, часто расположен на уровне представления. Второй уровень, который содержит логику бизнес-приложений и логику доступа к данным, часто упоминается как уровень приложений. Третий ярус обычно содержит базу данных или источник данных и упоминается как уровень базы данных.

Серверы, входящие в состав каждого уровня, могут отличаться как по возможностям, так и по номерам. Например, в крупномасштабной распределенной среде веб-приложения может быть большое количество веб-серверов на уровне представления, меньшее число серверов приложений на прикладном уровне и два кластеризованных сервера баз данных на уровне баз данных. Возможность добавления дополнительные серверы часто называют горизонтальным масштабированием. Возможность добавления более мощных серверов часто называют вертикальным масштабированием. Многоуровневость прикладного кода в таких случаях значительно облегчает возможности масштабирования приложений.

В крупномасштабных распределенных веб-приложениях уровни часто ограничены брандмауэрами. Например, брандмауэр может быть помещен перед уровнем представления, в то время как второй брандмауэр может быть помещен перед уровнем приложений. Таким образом,

уровень представления находится между брандмауэрами в так называемой демилитаризованной зоне (ДЗ), в то время как уровни приложения и базы данных сервера защищены вторым брандмауэром и расположе-

ны в так называемой зоне интрасети. Следовательно, масштабируемость способствует безопасности и позволяет крупным предприятиям защитить важные внутренние системы от трафика, исходящего из ненадежных зон, таких как Интернет и ДЗ. Без масштабируемости очень трудно защитить внутренние системы.

Клиент

Как правило, мобильные устройства могут работать в качестве тонких или толстых клиентов, или могут быть разработаны таким образом, что могут быть хостами веб-страниц (рис. 1.8).

Тонкий клиент

Тонкие клиенты не имеют пользовательского прикладного кода и их функциональность полностью обеспечивается сервером (рис. 1.8). Таким образом, они не зависят от операционной системы мобильного устройства или мобильного устройства типа толстого клиента.

Рис. 1.8. Тонкий клиент

Тонкие клиенты обычно используют веб-браузеры и протокол беспроводного доступа (WAP), чтобы отображать следующие типы страниц прикладного содержания:

- Web (например, HTML, XML);
- WAP (например, WML).

Например, для отображения веб-страниц КПК может отображать их через Microsoft Pocket Internet Explorer, в то время как планшетные и портативные ПК также могут отображать их через Microsoft Internet Explorer или Google Chrome. Аналогичным образом WAP-браузер на сотовом телефоне может отображать WML-страницы.

Тонкие клиенты имеют ряд преимуществ перед толстыми клиентами. Например, они гораздо проще в обслуживании и поддержке, поскольку не имеют прикладного кода и данных. В результате, нет необходимости рассматривать версий прикладного кода и механизмы их распределения клиенту.

Однако трудностью при работе с тонкими клиентами, является то, что они должны быть в постоянной связи с сервером, т.к. это их источ-

ник для обновления и получения данных. Если связь не является надежной, возможно вместо этого типа клиента потребуется рассмотреть автономные толстые клиентские приложения.

Толстый клиент

Толстые клиенты обычно имеют от одного до трех слоев прикладного кода и могут работать независимо от сервера в течение некоторого периода времени.

Как правило, толстые клиенты наиболее полезны в ситуациях, когда связь между клиентом и сервером не может быть гарантированной. Например, толстое клиентское приложение может быть в состоянии принимать ввод пользователя и хранение данных в локальной базе данных пока восстанавливается связь с сервером и данные могут быть перемещены на сервер. Это позволяет пользователю продолжать работать, даже если он/она находится вне контакта с сервером.

Тем не менее толстые клиенты в значительной степени зависят от операционной системы и типа мобильного устройства, поэтому могут быть сложности при повторном использовании и распространении кода. Вы также можете иметь несколько версий кода для поддержки на нескольких устройствах.

Толстые клиенты могут быть реализованы с помощью одного, двух или трех слоев прикладного кода. Тем не менее если вы используете только один слой, крайне трудно выделить отдельные сферы/области

функциональности и повторно использовать и распространять код на несколько типов устройств. Таким образом, в целом лучше использовать два или, предпочтительно, три слоя, чтобы использовать как можно большую часть прикладного кода (рис. 1.9).

Рис. 1.9. Толстый клиент – три слоя

Хостинг веб-страниц

Кроме того, на мобильном устройстве можно отображать и обрабатывать веб-страницы, даже если мобильный клиент лишь периодически подключен к сети и серверным системам. Для того чтобы это сделать, на мобильном устройстве небходим эквивалент «мини» веб-сервера.

Например, компания Microsoft выпустила HTTP-сервер, который работает на КПК как раз для таких целей. Данные, введенные пользователем на веб-странице, обрабатываются HTTP-сервером и хранятся

в локальной базе данных, если не могут быть загружены на сервер, пока с ним не восстановится соединение.

Клиенты, которые используют хостинг веб-страницы, также могут иметь от одного до трех слоев (рис. 1.10). Главное отличие между веб-страницами толстого клиента и окнами Windows заключается в том, что

уровень представления отображает и использует веб-страницы, а не формы Windows.

Рис. 1.10. Хостинг веб-страницы

Сервер

Архитектура сервера, как правило, состоит из одного-трех слоев кода, реализованного на одном-трех уровнях. Зачастую строят трехуровневую архитектуру, но у неѐ есть плюсы и минусы. Например, крупная трехуровневая архитектура может быть достаточно дорогостоящей для реализации. Если приложение ориентировано на ограниченное число пользователей, трехуровневая архитектура может быть избыточной.

В следующих разделах мы обсудим одноуровневую, двухуровневую и трехуровневую архитектуры и некоторые плюсы и минусы их реализации.

Одноуровневая архитектура

Одноуровневая архитектура предполагает размещение всех трѐх слоѐв кода на одном сервере (рис. 1.11). Есть несколько преимуществ и недостатков такого типа архитектуры.

Преимущества: очень удобно, быстрая разработка и развертывание.

Недостатки: небольшая масштабируемость и безопасность.

С одной стороны, очень удобно разрабатывать код на одной машине, с другой – очень трудно масштабировать приложение. Для веб-приложения также трудно защитить сервер с помощью брандмауэров и зон безопасности, поскольку сервер почти наверняка будет в зоне ДЗ, что может привести к высокому риску безопасности доступа к базе данных.

Рис. 1.11. Одноуровневая архитектура

Двухуровневая архитектура

При реализации двухуровневой архитектуры сервер базы данных отделяется от сервера приложений (рис. 1.12). Имеется несколько плюсов и минусов такого подхода.

Рис. 1.12. Двухуровневая архитектура

Плюсы: удобство и возможность специализации сервера баз данных.

Минусы: меньшая масштабируемость, сложности при обеспечении безопасности и дороговизна.

Отделение сервера баз данных позволяет ему стать более специализированным, но по-прежнему крайне сложно применять масштабирование. При такой архитектуре сложно оградить серверы с брандмауэрами и зоны безопасности, хотя это разделение происходит лучше, чем в одноуровневой архитектуре. Тем не менее, безопасность приложения все равно небольшая.

Трехуровневая архитектура

Трехуровневая архитектура обеспечивает разделение базы данных, сервера приложений и сервера представлений друг от друга (рис. 1.13). Такой подход имеет свои преимущества и недостатки.

Рис. 1.13. Трехуровневая архитектура

Преимущества: масштабируемость, защита с помощью брандмауэров и зон, возможность специализации сервера баз данных.

Недостатки: избыточность, более сложно развиваться, труднее управлять, большая дороговизна.

Отделение базы данных позволяет серверу базы данных стать более специализированным сервером. Отделение презентационного и прикладного серверов также позволяет провести специализацию этих серверов, что в совокупности дает огромный потенциал масштабируемости. Также можно лучше защитить приложение с использованием брандмауэров и зон ДЗ.

Типы соединений

Мобильные устройства обычно работают в одном из трех режимов: всегда на связи, частично связаны или не подключены.

Постоянное подключение

Мобильные устройства, такие как сотовые телефоны или КПК, как правило, работают в режиме постоянной связи. Мобильные устройства, такие как КПК, планшетные и портативные ПК, по существу стали расширением существующих приложений и инфраструктуры, позволяющих пользователям всегда быть соединенными с приложениями во время работы.

Частичное подключение

Иногда мобильные устройства могут быть отключены в течение длительного периода времени. Например, сотрудники мобильных офи-

сов могут периодически подключаться к серверу в офисе для получения электронной почты, контактной информации или задач, которые предстоит сделать. Работник отключает мобильное устройство и выполняет работу вне офиса, во время которой он может ссылаться на скачанную информацию. Пользователь также может локально обновлять информацию на своем мобильном устройстве перед повторной синхронизацией мобильного устройства с сервером в более позднее время.

Без подключения

Существуют также мобильные устройства, которые никогда не подключаются к серверам, например игровые устройства.

Синхронизация

Тип соединения влияет на то, каким образом можно синхронизировать данные между мобильным устройством и сервером. Синхронизация реализуется двумя основными способами: непрерывно или методом с промежуточным хранением.

Непрерывное соединение

Когда между клиентом и сервером непрерывное соединение, синхронизация данных между клиентом и сервером также является непрерывной и может быть достигнута синхронным или асинхронным спосо-

бами (рис. 1.14).

46

Рис. 1.14. Синхронное и асинхронное взаимодействие

Синхронное соединение возникает, когда запрос для хранения данных отправляется на сервер, на котором хранятся данные. Затем данные помещаются в хранилище, такое как базы данных на сервере.
При синхронном соединении все данные сохраняются полностью перед тем как сервер подтверждает получение данных и освобождает пользовательский интерфейс клиента.

Асинхронное соединение происходит, когда сначала на сервер отправляется запрос для хранения данных, а только потом данные будут сохранены. Данные помещаются в хранилище, такое как база данных на сервере. Однако при асинхронной связи данные не могут быть полностью сохранены до распознавания сервером клиента. Обычно сервер сначала распознает запрос клиента и только затем сохраняет данные. После выполнения запроса на хранение сервер сообщит об этом клиенту.

Метод с промежуточным хранением

47

Когда не может быть гарантировано соединение между клиентом и сервером, есть возможность безопасно хранить и передавать информацию, используя метод с промежуточным хранением. Предположим, что мобильный пользователь желает ввести данные в то время, когда его мобильное устройство не подключено к серверу. Приложение мобильного клиента может сначала сохранить данные в локальной базе данных. Позже, когда установится соединение, мобильное приложение передаст данные из локальной базы данных базе данных на сервере (рис. 1.15).

Рис. 1.15. Метод соединения с промежуточным хранением

Важно отметить, что если вы позволите мобильным пользователям

таким образом хранить данные в локальной базе данных, необходимо также обеспечить целостность данных, когда данные синхронизируются с сервером базы данных, т.к. другие пользователи могут добавить или изменить противоречивые данные на своих мобильных устройствах.

Архитектурные шаблоны

В следующих разделах представлены некоторые шаблоны для малых, средних и крупномасштабных архитектур мобильных приложений.

Если предположить, что существует четыре возможных слоя клиента, три уровня сервера и три типа подключения, то, в общей сложности, получается 36 возможных комбинаций. Однако не все из этих комбинаций являются полезными. В настоящее время самая распространенная модель – это «частичное подключение», поскольку связь не всегда может быть гарантирована.

Нулевая, трехуровневая архитектура непрерывного подключения

Если мобильный клиент не имеет слоев прикладного кода, это означает, что он является тонким клиентом. Сервер хранит весь код приложения и имеет трехуровневую архитектуру. На рис. 1.16 представлена простая мобильная архитектура.

Рис. 1.16. Пример архитектуры с непрерывным подключением

Уровень представления имеет прикладной код, который может отображать страницы мобильного устройства, такого как КПК. Обычные веб-страницы (например, ASP.NET, JSP, HTML) доступны для просмотра с помощью веб-браузера, например Microsoft Pocket Internet Explorer.

Уровень представления также взаимодействует с данными и имеет доступ к объектам на уровнях приложения и базы данных. Как правило, данные могут быть прочитаны из базы данных и обратно записаны во время обновления.

Эта архитектура очень простая, потому что предполагается, что мобильный клиент всегда будет подключен к серверу. Таким образом, нет необходимости в хранении данных приложения на мобильном уст-

ройстве. Если мобильное устройство будет отключено, то не будет возможности получения всей актуальной информации, пока соединение не будет восстановлено.

Три слоя, три уровня, архитектура частичного подключения

Мобильный клиент имеет три слоя кода прикладной программы, т.е. он является толстым клиентом. Сервер также содержит код приложения и организован в виде трехуровневой архитектуры. На рис. 1.17 представлена более сложная мобильная архитектура.

Рис. 1.17. Сложная архитектура мобильного приложения

Мобильный клиент имеет полностью автономное приложение, которое может читать и записывать вводимые пользователем данные в локальную базу данных, когда он не подключен к серверу. Когда связь восстанавливается, данные могут быть получены из локальной базы данных и загружены на сервер с помощью механизма промежуточного хранения данных.

Принципы разработки архитектуры мобильного приложения

В следующих разделах мы опишем некоторые из принципов создания хорошей архитектуры. На практике может оказаться невозможным достижение всех этих принципов. Тем не менее, вы обнаружите, что большинство архитектур мобильных приложений удовлетворяют этим принципам. Этап проектирования архитектуры должен решать вопросы бизнеса и функциональные потребности пользователей.

Метод независимости

функциональности и повторно использовать и распространять код на несколько типов устройств. Таким образом, в целом лучше использовать два или, предпочтительно, три слоя, чтобы использовать как можно большую часть прикладного кода (рис. 1.9).

Рис. 1.9. Толстый клиент – три слоя

Хостинг веб-страниц

Кроме того, на мобильном устройстве можно отображать и обрабатывать веб-страницы, даже если мобильный клиент лишь периодически подключен к сети и серверным системам. Для того чтобы это сделать, на мобильном устройстве небходим эквивалент «мини» веб-сервера.

Например, компания Microsoft выпустила HTTP-сервер, который работает на КПК как раз для таких целей. Данные, введенные пользователем на веб-странице, обрабатываются HTTP-сервером и хранятся в локальной базе данных, если не могут быть загружены на сервер, пока с ним не восстановится соединение.

Клиенты, которые используют хостинг веб-страницы, также могут иметь от одного до трех слоев (рис. 1.10). Главное отличие между веб-страницами толстого клиента и окнами Windows заключается в том, что

уровень представления отображает и использует веб-страницы, а не формы Windows.

Рис. 1.10. Хостинг веб-страницы

Сервер

Архитектура сервера, как правило, состоит из одного-трех слоев кода, реализованного на одном-трех уровнях. Зачастую строят трехуровневую архитектуру, но у неѐ есть плюсы и минусы. Например, крупная трехуровневая архитектура может быть достаточно дорогостоящей для реализации. Если приложение ориентировано на ограниченное число пользователей, трехуровневая архитектура может быть избыточной.

В следующих разделах мы обсудим одноуровневую, двухуровневую и трехуровневую архитектуры и некоторые плюсы и минусы их реализации.

Одноуровневая архитектура

Одноуровневая архитектура предполагает размещение всех трѐх слоѐв кода на одном сервере (рис. 1.11). Есть несколько преимуществ и недостатков такого типа архитектуры.

Преимущества: очень удобно, быстрая разработка и развертывание.

Недостатки: небольшая масштабируемость и безопасность.

С одной стороны, очень удобно разрабатывать код на одной машине, с другой – очень трудно масштабировать приложение. Для веб-приложения также трудно защитить сервер с помощью брандмауэров и зон безопасности, поскольку сервер почти наверняка будет в зоне ДЗ, что может привести к высокому риску безопасности доступа к базе данных.

Рис. 1.11. Одноуровневая архитектура

Двухуровневая архитектура

При реализации двухуровневой архитектуры сервер базы данных отделяется от сервера приложений (рис. 1.12). Имеется несколько плюсов и минусов такого подхода.

Рис. 1.12. Двухуровневая архитектура

Плюсы: удобство и возможность специализации сервера баз данных.

Минусы: меньшая масштабируемость, сложности при обеспечении безопасности и дороговизна.

Отделение сервера баз данных позволяет ему стать более специализированным, но по-прежнему крайне сложно применять масштабирование. При такой архитектуре сложно оградить серверы с брандмауэрами и зоны безопасности, хотя это разделение происходит лучше, чем в одноуровневой архитектуре. Тем не менее, безопасность приложения все равно небольшая.

Трехуровневая архитектура

Трехуровневая архитектура обеспечивает разделение базы данных, сервера приложений и сервера представлений друг от друга (рис. 1.13). Такой подход имеет свои преимущества и недостатки.

Рис. 1.13. Трехуровневая архитектура

Преимущества: масштабируемость, защита с помощью брандмауэров и зон, возможность специализации сервера баз данных.

Недостатки: избыточность, более сложно развиваться, труднее управлять, большая дороговизна.

Отделение базы данных позволяет серверу базы данных стать более специализированным сервером. Отделение презентационного и прикладного серверов также позволяет провести специализацию этих серверов, что в совокупности дает огромный потенциал масштабируемости. Также можно лучше защитить приложение с использованием брандмауэров и зон ДЗ.

Типы соединений

Мобильные устройства обычно работают в одном из трех режимов: всегда на связи, частично связаны или не подключены.

Постоянное подключение

Мобильные устройства, такие как сотовые телефоны или КПК, как правило, работают в режиме постоянной связи. Мобильные устройства, такие как КПК, планшетные и портативные ПК, по существу стали расширением существующих приложений и инфраструктуры, позволяющих пользователям всегда быть соединенными с приложениями во время работы.

Частичное подключение

Иногда мобильные устройства могут быть отключены в течение длительного периода времени. Например, сотрудники мобильных офи-

сов могут периодически подключаться к серверу в офисе для получения электронной почты, контактной информации или задач, которые предстоит сделать. Работник отключает мобильное устройство и выполняет работу вне офиса, во время которой он может ссылаться на скачанную информацию. Пользователь также может локально обновлять информацию на своем мобильном устройстве перед повторной синхронизацией мобильного устройства с сервером в более позднее время.

Без подключения

Существуют также мобильные устройства, которые никогда не подключаются к серверам, например игровые устройства.

Синхронизация

Тип соединения влияет на то, каким образом можно синхронизиро-

вать данные между мобильным устройством и сервером. Синхронизация реализуется двумя основными способами: непрерывно или методом с промежуточным хранением.

Непрерывное соединение

Когда между клиентом и сервером непрерывное соединение, синхронизация данных между клиентом и сервером также является непрерывной и может быть достигнута синхронным или асинхронным способами (рис. 1.14).

Рис. 1.14. Синхронное и асинхронное взаимодействие

Синхронное соединение возникает, когда запрос для хранения данных отправляется на сервер, на котором хранятся данные. Затем данные помещаются в хранилище, такое как базы данных на сервере.
При синхронном соединении все данные сохраняются полностью перед тем как сервер подтверждает получение данных и освобождает пользовательский интерфейс клиента.

Асинхронное соединение происходит, когда сначала на сервер отправляется запрос для хранения данных, а только потом данные будут сохранены. Данные помещаются в хранилище, такое как база данных на сервере. Однако при асинхронной связи данные не могут быть полностью сохранены до распознавания сервером клиента. Обычно сервер сначала распознает запрос клиента и только затем сохраняет данные. После выполнения запроса на хранение сервер сообщит об этом клиенту.

Метод с промежуточным хранением

Когда не может быть гарантировано соединение между клиентом и сервером, есть возможность безопасно хранить и передавать информацию, используя метод с промежуточным хранением. Предположим, что мобильный пользователь желает ввести данные в то время, когда его мобильное устройство не подключено к серверу. Приложение мобильного клиента может сначала сохранить данные в локальной базе данных. Позже, когда установится соединение, мобильное приложение передаст данные из локальной базы данных базе данных на сервере (рис. 1.15).

Рис. 1.15. Метод соединения с промежуточным хранением

Важно отметить, что если вы позволите мобильным пользователям таким образом хранить данные в локальной базе данных, необходимо также обеспечить целостность данных, когда данные синхронизируются с сервером базы данных, т.к. другие пользователи могут добавить или изменить противоречивые данные на своих мобильных устройствах.

Архитектурные шаблоны

В следующих разделах представлены некоторые шаблоны для малых, средних и крупномасштабных архитектур мобильных приложений.

Если предположить, что существует четыре возможных слоя клиента, три уровня сервера и три типа подключения, то, в общей сложности, получается 36 возможных комбинаций. Однако не все из этих комбинаций являются полезными. В настоящее время самая распространенная модель – это «частичное подключение», поскольку связь не всегда может быть гарантирована.

Нулевая, трехуровневая архитектура непрерывного подключения

Если мобильный клиент не имеет слоев прикладного кода, это означает, что он является тонким клиентом. Сервер хранит весь код приложения и имеет трехуровневую архитектуру. На рис. 1.16 представлена простая мобильная архитектура.

Рис. 1.16. Пример архитектуры с непрерывным подключением

Уровень представления имеет прикладной код, который может отображать страницы мобильного устройства, такого как КПК. Обычные веб-страницы (например, ASP.NET, JSP, HTML) доступны для просмотра с помощью веб-браузера, например Microsoft Pocket Internet Explorer.

Уровень представления также взаимодействует с данными и имеет доступ к объектам на уровнях приложения и базы данных. Как правило, данные могут быть прочитаны из базы данных и обратно записаны во

время обновления.

Эта архитектура очень простая, потому что предполагается, что мобильный клиент всегда будет подключен к серверу. Таким образом, нет необходимости в хранении данных приложения на мобильном уст-

ройстве. Если мобильное устройство будет отключено, то не будет возможности получения всей актуальной информации, пока соединение не будет восстановлено.

Три слоя, три уровня, архитектура частичного подключения

Мобильный клиент имеет три слоя кода прикладной программы, т.е. он является толстым клиентом. Сервер также содержит код приложения и организован в виде трехуровневой архитектуры. На рис. 1.17 представлена более сложная мобильная архитектура.

Рис. 1.17. Сложная архитектура мобильного приложения

Мобильный клиент имеет полностью автономное приложение, которое может читать и записывать вводимые пользователем данные в локальную базу данных, когда он не подключен к серверу. Когда связь восстанавливается, данные могут быть получены из локальной базы данных и загружены на сервер с помощью механизма промежуточного хранения данных.

Принципы разработки архитектуры мобильного приложения

В следующих разделах мы опишем некоторые из принципов создания хорошей архитектуры. На практике может оказаться невозможным достижение всех этих принципов. Тем не менее, вы обнаружите, что большинство архитектур мобильных приложений удовлетворяют этим принципам. Этап проектирования архитектуры должен решать вопросы бизнеса и функциональные потребности пользователей.

Метод независимости

В идеальном варианте необходимо разработать мобильные приложения, которые насколько возможно независимы от типа устройства и от платформы. Это не всегда легко или возможно, но хорошие приложения, как правило, написаны так, что они могут работать на многих устройствах и платформах.

На практике, однако, большинство приложений скорее всего не соответствуют этим парадигмам. По всей вероятности, вам будет необходимо выбрать наиболее подходящее устройство и платформу и, соответственно, писать приложения. Таким образом, вам почти наверняка придется выбирать мобильные устройства, такие как смартфоны или КПК. Каждое устройство и платформа имеет различные характеристики, что необходимо принимать во внимание при разработке приложений.

Высокая производительность и доступность

Архитектура должна, как правило, иметь отличную производительность при нормальном и пиковом периодах потребности в ресурсах. Например, для сайта брокерской площадки электронного бизнеса в день пиковой торговли акциями. Если люди могут использовать сайт в любой момент времени, архитектура должна обладать также высокой доступностью.

Масштабируемость

Архитектура должна быть масштабируемой, чтобы быстро адаптироваться при возможном значительном увеличении числа пользователей, приложений и функциональных возможностей. Архитектура должна быть разработана так, чтобы легко позволить горизонтальное (дополнительные сервера) и вертикальное (добавление более быстрых серверов) масштабирование без ущерба для любых существующих приложений.

Системные требования пользователя

Архитектура, как правило, должна позволять обрабатывать как можно бóльшие типы и количество пользователей. Например, веб-приложения с большим объемом графики для отображения на КПК могут быть красивыми, но если пользователи имеют только низкоскоростные линии соединения, производительность не будет удовлетворительной. Таким образом, следует иметь в виду полный спектр пользователей для высоко- и низкопроизводительных систем.

Вопросы для самопроверки

1. Дайте определение мобильных вычислительных устройств.
2. В чем преимущества использования мобильных устройств?

3. Какие типы мобильных устройств существуют?
4. Какие операционные системы используются в мобильных устройствах?
5. Какие типы памяти применяются в мобильных устройствах? В чем заключаются их достоинства и недостатки?

2. РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ JAVA 2 MICRO EDITION

Java 2 Micro Edition (J2ME) – это платформа, которая была специально разработана для портативных устройств и представляет собой виртуальную машину с необходимым минимумом библиотек и более легкими версиями стандартных Java-классов.

Выделяют следующие основные категории портативных устройств:

- Устройства с небольшой мощностью и ограниченными функциональными возможностями, которые способны к нестационарным сетевым коммуникациям – мобильные телефоны, двусторонние пейджеры, карманные персональные компьютеры и органайзеры.
- Устройства, обладающие большей мощностью и средствами пользовательского интерфейса (UI), подсоединенные к фиксированному беспрерывному сетевому соединению – компьютерные приставки к телевизору, телевизоры с выходом в Интернет, навигационные автомобильные системы, смартфоны, коммуникаторы высокого класса.

Использование принципа модульного построения программ на J2ME, предоставляет возможность разработки приложений для разных типов устройств в зависимости от их функциональных возможностей, объема памяти и типа связи. С этой целью в J2ME используются конфигурации и профили.

2.1. Конфигурации и профили J2ME

Конфигурация J2ME определяет минимальную Java-платформу для категории портативных устройств. Другими словами, конфигурация – это спецификация, которая определяет доступные ресурсы системного уровня.

Конфигурация включает три базовых элемента:

- набор свойств языка программирования Java,
- набор свойств виртуальной машины Java,
- набор поддерживаемых библиотек Java и программных интерфейсов приложения (API).

Выделяют две базовые конфигурации J2ME для соответствующих

категорий портативных устройств:

1. Connected, Limited Device Configuration (CLDC) – конфигурация для не стационарно подключаемых мобильных устройств с ограниченными возможностями.
2. Connected Device Configuration (CDC) – конфигурация для постоянно подключенных устройств с объемом памяти 2 или более мегабайт и с большими функциональными возможностями.

Профиль J2ME определяет программный интерфейс для определенного класса устройств. Реализация профиля состоит из набора библиотек классов Java, которые обеспечивают интерфейс программного уровня, т.е. определяют все виды функциональных возможностей и служб.

Профили используются для обеспечения реальной платформы для семейства устройств, которые имеют возможности, заданные конфигурацией. Таким образом, гарантируется взаимодействие – которая необязательно предполагает совместимость конечных продуктов различных производителей – между всеми устройствами одной категории для определения стандартной платформы разработки приложений на Java.

Например, профиль может поддерживать возможность сетевой коммуникации для стандарта Short Message Service (SMS). Поскольку стандарт SMS является повсеместно распространенным свойством в сотовой телефонии, имеет смысл задать эту службу в профиле, который предназначен для мобильных телефонов, вместо того чтобы встраивать еѐ в конфигурацию.

Профиль внедряется поверх конфигурации и включает библиотеки, которые соответствуют более специфичным характеристикам заданной категории устройств, чем библиотеки, которые содержат конфигурации. Приложения затем встраиваются поверх конфигурации и профиля и могут использовать только библиотеки классов, предоставляемые этими двумя низкоуровневыми спецификациями. Профили могут быть встроены поверх друг друга, но приложение J2ME может содержать только одну конфигурацию. На рисунке 2.1 показаны уровни, из которых состоит платформа J2ME.

Рис. 2.1. Платформа J2ME

2.1.1. Конфигурация Connected, Limited Device Configuration

Цель использования CLDC заключается в том, чтобы установить стандартную платформу Java для этих устройств. Из-за широкого выбора системного программного обеспечения на различных персональных устройствах CLDC исходит из минимальных предположений о среде, в которой она существует. Например, одна ОС может поддерживать множественные параллельные процессы, другая может или не может поддерживать файловую систему и тому подобное.

CLDC отличается от CDC и представляет из себя ее подгруппу. Однако эти конфигурации независимы друг от друга, так что они не должны использоваться вместе при описании платформы. На рисунке 2.2 показана связь между двумя конфигурациями и платформой J2SE.

Рис. 2.2. CLDC является подгруппой CDC

Как и CDC, CLDC определяет требуемый уровень поддержки языка программирования Java, требуемую функциональную поддержку соответствующей требованиям виртуальной машины Java и требуемый набор библиотек классов.

Поддержка языка Java. Спецификация CLDC не включает поддержку следующих свойств языка Java:

- вычисления с плава•ющей точкой;
- финализация объекта;•
- иерархия класса Java.lang.Error во всей его полноте.•

Отсутствие поддержки плавающей точки является основным отличием на языковом уровне виртуальной машины Java, которая поддерживает CLDC, от стандартной VM J2SE, что очевидно для программистов. Это означает, что программы, предназначенные для запуска на CLDC, не могут использовать константы, типы и величины с плавающей точкой. Вы не можете использовать встроенный тип float и класс Java.lang.Float был удален из библиотек CLDC. Это свойство не присутствует из-за отсутствия аппаратного или программного обеспе-

чения с плавающей точкой на большинстве мобильных устройств.

Финализация объекта также отсутствует. Это означает, что метод Object.finalized был удален из библиотек CLDC.

Иерархия исключений Java.lang.Error также была удалена из библиотек CLDC и поэтому недоступна для приложений. Основная причина того, что обработка ошибок отсутствует, заключается в ограниченной памяти мобильных устройств. Это обычно не создает никаких неудобств при разработке приложений, как-никак, приложения не рассчитаны на восстановление из ошибочных состояний. И ресурсная цена реализации обработки ошибок высока и лежит за пределами возможностей сегодняшних мобильных устройств. Кроме того, нейтрализация ошибок на портативных устройствах, таких, как мобильные телефоны, зависит от конкретного устройства. И, наконец, не имеет смысла оговаривать механизм восстановления, который устройства должны использовать. Этот механизм легко может находиться за пределами встроенной виртуальной машины.

Поддержка виртуальной машины Java и библиотек. В CLDC определены требования для виртуальной машины Java. Они зависят от VM, которая высоко-портативна и создана для ресурсно ограниченных небольших устройств. Поддержка нескольких свойств, которые существуют в стандартной J2SE VM, была исключена из спецификации CLDC. В следующем списке перечислены свойства, которые не поддерживаются в CLDC-совместимой виртуальной машине. Свойства, перечислен-

ные в этом списке, были исключены как из-за изменения библиотек, так и из-за соображений безопасности:

- Java Native Interface (JNI, собственный интерфейс Java);
- загрузчики определяемых пользователем классов;
- отражение (reflection);
- группы нитей и демоны нитей (thread daemons);
- финализация (отсутствие метода Object.finalize() в библиотеках CLDC);
- слабые ссылки (weak references);
- ошибки (поддерживается небольшая подгруппа ошибок J2SE);
- проверка класса файла.

Среди этих неподдерживаемых свойств проверка класса файла заслуживает дополнительного пояснения. Виртуальная машина в спецификации CLDC все еще выполняет этот процесс, но она использует двухшаговый процесс и отличный алгоритм, который требует меньшей затраты вычислительных ресурсов, чем стандартный J2SE верификатор.

Виртуальная машина, которая устанавливается вместе с внедрени-

ем CLDC, называется Kilobyte Virtual Machine (KVM), названа она таким образом потому, что использует всего лишь несколько килобайт рабочей памяти [6]. KVM не является полнофункциональной J2SE VM.

Спецификация свойств, которые поддерживает виртуальная машина, включает спецификацию библиотек, которые она поддерживает. Спецификация CLDC подробно описывает библиотеки, внедрение которых должно поддерживаться.

CLDC – это конфигурация, поверх которой встраиваются один или более профилей, поэтому CLDC предназначена для разработчиков отдельных приложений. В таблице 2.1 перечислены пакеты, которые включает в себя CLDC.

Таблица 2.1.

Пакеты CLDC

Название пакета CLDC	Описание
java.io	Стандартные классы и пакеты ввода/вывода Java, подмножество пакета J2SE
java.lang	Классы и интерфейсы VM, подмножество пакета J2SE
java.util	Классы и интерфейсы стандартных утилит, подмножество пакета J2SE
javax.microedition.io	Классы и интерфейсы структуры общих соединений CLDC

Первые три пакета используют префикс Java, в своем имени, потому что каждый из них содержит подгруппу стандартных классов платформы J2SE. Последний, однако, должен использовать префикс javax., поскольку он описывает новое «стандартное расширение», которое не является частью основной платформы Java.

2.1.2. Конфигурация Connected Device Configuration

Используемая в CDC виртуальная машина Java, называется компактной виртуальной машиной (Compact Virtual Machine) и имеет следующие основные свойства:

- улучшенная запоминающая система;
- небольшие временные интервалы сборки мусора в среднем;
- полное отделение виртуальной машины от системы памяти;

модульные сборщики мусора;•
сборка мусора по поколениям.•

В частности, CVM была спроектирована с учетом предоставления следующих свойств:

портативность;•
быстрая синхронизация;•
выполнение классов Java отдельно от постоянной памяти (ROM);•
поддержка естественных потоков;•
зоны обслуживания малых классов;•
предоставление интерфейсов и поддержка служб операционной• системы реального времени (RTOS);
преобразование потоков Java непосредственно в естественные• потоки;
поддержка всех свойств и библиотек виртуальной машины Java• версии 1.3, таких как интерфейс вызова удаленных методов, интерфейс отладки виртуальной машины Java и пр.

CDC устанавливает минимальный набор библиотек классов и API. Она поддерживает следующие стандартные пакеты Java:

java.lang – системные классы виртуальной машины Java;•
java.util – базовые утилиты Java;•
java.net – дейтаграмма Universal Datagram Protocol (UDP) и• ввод/вывод (I/O);
java.io – файловый ввод/вывод Java;•

java.text – самая минимальная поддержка интернационализа-• ции;
java.security – минимальная защита на низком уровне и• шифрование сериализации объекта.

Данные API не включают полный набор пакетов набора инструментальных средств разработки программного обеспечения (Software Development Kit (SDK)) Java 2. В некоторых случаях эти пакеты и классы являются подгруппами пакетов и классов Java 2 SDK. В таблице 2.2 перечислен полный набор пакетов, поддерживаемых CDC.

Таблица 2.2.

Пакеты CDC

Название пакета CDC	Описание
java.io	Стандартные классы и интерфейсы ввода/вывода

java.lang	Классы виртуальной машины
java.lang.ref	Классы для работы с ссылками на объекты
java.lang.reflect	Классы и интерфейсы, поддерживающие отражение (динамическую информацию о классах)
java.math	Математический пакет
java.net	Сетевые классы и интерфейсы
java.security	Классы и интерфейсы безопасности
java.security.cert	Классы сертификации безопасности
java.text	Текстовой пакет
java.util	Классы стандартных утилит
java.util.jar	Классы утилиты архиватора Java (JAR)
java.util.zip	Классы утилиты ZIP
javax.microedition.io	Классы и интерфейсы структуры общих соединений CDC

2.1.3. Профиль Foundation Profile

Конфигурация вместе с профилем формирует исполняемую среду J2ME. Свойства и службы системного уровня, поддерживаемые конфигурацией, более или менее спрятаны от разработчика приложений. В действительности разработчику приложения запрещен прямой доступ к ним. Если это не было соблюдено, приложение не будет считаться соответствующим требованиям J2ME.

С точки зрения программиста профиль необходим для «полезной» работы. Профиль определяет уровень, который содержит API. Создатели J2ME вначале задали один профиль CDC, профиль Foundation, который основан на выпуске J2SE версии 1.3. Он был разработан стандартным комитетом Java Community Process, экспертной группой компаний, работающих в сфере потребительских электронных товаров. Профиль Foundation содержит в себе пакеты J2SE, перечисленные в таблице 2.3.

Таблица 2.3.

Пакеты профиля Foundation

Название пакета профиля Foundation	Описание
java.lang	Поддержка языка Java.

	Добавляет полную поддержку zip и других утилит
java.util	
	J2SE.
Java.net	Добавляет TCP/IP сокеты и HTTP-соединения.
java.io	Поддержка механизмов ввода/вывода.
	Обеспечивает поддержку интернационализации
java.text	
	пакета.
java.security	Добавляет подпись и сертификацию кодов.

Согласно приведенной таблице, вся иерархия java.awt (Abstract Window Toolkit, AWT) и java.swing (пакет Swing), которая определяет API графического пользовательского интерфейса (GUI), отсутствует в поддерживаемых пакетах. Если приложению необходим GUI, потребуется дополнительный профиль. Профили могут быть внедрены поверх друг друга. Продукт платформы J2ME, однако, может содержать только одну конфигурацию.

Отсутствие поддержки GUI в профиле Foundation имеет меньшее воздействие на семейство постоянно подключенных сетевых устройств с общим доступом, таких, как компьютерные приставки к телевизору, чем оно влияет на персональные мобильные устройства, с которыми работают при помощи второй конфигурации J2ME, CLDC.

В общем, решение включать или не включать свойства и библиотеки в конфигурацию или профиль основано на их зонах обслуживания, требованиях к статическим и динамическим ресурсам и к безопасности.

2.1.4. Профиль Personal Profile

Спецификация профиля Personal была разработана в Java Community, конечным результатом которой стал JSR-62. Профиль Personal обеспечивает среду с полной поддержкой AWT. Замысел его создателей заключался в том, чтобы обеспечить платформу, подходящую для Web-апплетов. Он также предоставляет способ перемещения J2ME для приложений Personal Java.

Профиль Personal версии 1.0 требует внедрения профиля Foundation версии 1.0. Это расширенный набор профиля Personal Basis Profile версии 1.0. Однако профиль Personal является подгруппой платформы J2SE версии 1.3.1, которая дает приложениям, созданным в профиле Personal, большую совместимость снизу вверх с J2SE версии 1.3.1.

В таблице 2.4 перечислены пакеты, которые включены в профиль

Personal версии 1.0.

Таблица 2.4.

Пакеты профиля Personal

Название пакета профиля	Описание
Personal	
java.applet	Классы, необходимые для создания апплетов, и используемые апплетами.
java.awt	Классы AWT для создания пользовательского интерфейса программ.
java.awt.data transfer	Классы и интерфейсы для пересылки данных внутри и между приложениями.
]ava.awt.event	Классы и интерфейсы для обработки событий AWT.
java.awt.font	Классы и интерфейсы для работы со шрифтами.
java.awt.im	Классы и интерфейсы для описания редакторов методов ввода.
java.awt.im.spi	Интерфейсы, которые помогают в разработке редакторов методов ввода для любой среды исполнения Java.
java.awt.image	Классы для создания и изменения изображений
java.beans	Классы, которые поддерживают разработку компонентов JavaBean.
javax.microedition.xlet	Интерфейсы, используемые приложениями и диспетчерами приложений профиля J2ME Personal для коммуникации.

2.1.5. Профиль RMI

Профиль Remote Method Invocation (RMI) является профилем, созданным для платформ, которые поддерживают конфигурацию CDC. Он был задан JSR-66 и определен различными компаниями, принимавшими участие в Java Community Process.

Профиль RMI требует внедрения профиля Foundation и внедряется поверх него. Продукты профиля RMI должны поддерживать следующие свойства:

- полную семантику RMI вызовов;
- поддержку объектов маршалинга;
- RMI проводного протокола;
- экспорт удаленных объектов через API UnicastRemoteObject;
- распределенную сборку мусора и интерфейсы еборщика мусора как для клиента, так и для сервера;
- интерфейс активатора и протокол активации для клиента;
- интерфейсы реестра RMI и экспорт реестра удаленных объектов.

Профиль RMI поддерживает подгруппу RMI API J2SE в версии 1.3. Следующие интерфейсы и свойства являются частью спецификации RMI J2SE в. 1.3 и публичных API, но поддержка этих интерфейсов и функциональных возможностей исключена из технических требований профиля RMI из-за ограниченности вычислительных мощностей устройств, сетевой производительности и пропускной способности:

- RMI через брандмауэры и прокси;
- RMI мультиплексный протокол;
- модель реализации «активизируемого» («activatable») удаленного объекта;
- нерекомендуемые методы, классы и интерфейсы;
- поддержка протокола скелетона/заглушки для RMI в. 1.1;
- компилятор скелетона и заглушки.

Поддержка следующих свойств J2SE RMI в 1.3 не включена:

- java.rmi.server.disableHttp;
- java.rmi.activation.port;
- java.rmi.loader.packagePrefix;
- java.rmi.registry.packagePrefix;
- java.rmi.server.packagePrefix.

2.2. Профиль Mobile Information Device Profile

Поскольку категория, обслуживаемая CLDC, включает в себя такое множество различных типов персональных устройств, потенциально для их поддержки необходимо множество различных профилей. Наиболее популярным и хорошо известным из них является профиль Mobile Information Device (MIDP), иногда называемый MID Profile. MIDP «лежит» поверх CLDC и задает набор API пользовательского интерфейса (UI), созданного для современных беспроводных устройств.

Следуя традициям языка Java, MIDP-приложения называются мидлеты [8]. С практической точки зрения, MIDP является единственным профилем, доступным на сегодняшний день.

Другой профиль, профиль PDA, в настоящее время находится на стадии описания. Профили PDA также принадлежат к общей категории мобильных информационных устройств. Однако профиль PDA, возможно, никогда не будет внедрен, поскольку сомнительно, предлагает ли он достаточно отличий и улучшений к спецификации MIDP, чтобы оправдать его разработку [10].

Спецификация MIDP, как и профиль Foundation конфигурации CDC, была создана экспертной группой, в этом случае экспертной группой профиля Mobile Information Device Profile, которая является международным форумом, включающим представителей нескольких компаний со сферой деятельности в области мобильных устройств. MIDP предназначен для мобильных информационных устройств, таких, как мобильные телефоны, двусторонние пейджеры и тому подобного, которые приблизительно соответствуют следующим характеристикам:

- размер экрана примерно (как минимум) 96x54 пикселей;
- глубина экрана 1 бит;
- клавиатура для работы одной или двумя руками, устройство ввода с сенсорного экрана;
- 128 Кб энергонезависимой памяти для MIDP-компонентов;
- 8 Кб энергонезависимой памяти для данных постоянного хранения;
- 32 Кб энергозависимой оперативной памяти для области динамической памяти Jra:
- двусторонняя беспроводная связь.

Поскольку диапазон возможностей MID столь широк, MIDP устанавливает рабочую величину минимального общего знаменателя возможностей устройств. Поэтому MIDP определяет следующие API:

- приложения (семантика и управление приложениями MIDP);
- пользовательский интерфейс;

- постоянное хранение;
- организация сетей;
- таймеры.

В таблице 2.5. перечислены пакеты, которые содержит MIDP.

Таблица 2.5.

Пакеты MIDP

Название пакета MIDP	Описание
javax.microedition.icdui	Классы и интерфейсы интерфейса пользователя
javax.microedition.rms	Система организации ведения записей (Record management system, RMS), поддерживающая постоянное хранение устройства
javax.microedition.midlet	Типы классов поддержки определения приложений MIDP
javax.microedition.io	Классы и интерфейсы структуры общих соединений MIDP
java.io	Классы и интерфейсы стандартного ввода/ вывода Java
java.lang	Классы и интерфейсы виртуальной Java машины
java.util	Классы и интерфейсы стандартных утилит

Реализация MIDP должна состоять из пакетов и классов, указанных в спецификации MIDP. Кроме того, она может иметь зависимые от реализации классы для доступа программного и аппаратного обеспечения родной системы.

В таблице 2.6 сопоставляются структуры данных платформ CDC и CLDC. Как в CDC, так и в CLDC нет ничего такого, что препятствует производителю подключать любую из платформ к данному семейству устройств. Тем не менее, структуры платформ – особенно свойства конфигураций и профилей – были определены для работы с практическими ограничениями различных семейств аппаратных устройств.

Таблица 2.6.

Пакеты CDC

Название пакета MIDP	Описание
javax.microedition.midlet	Типы классов поддержки определения приложений MIDP.

Название пакета MIDP	Описание
javax.microedition.io	Классы и интерфейсы структуры общих соединений MIDP.

java.io	Классы и интерфейсы стандартного ввода/ вывода Java.
java.lang	Классы и интерфейсы виртуальной Java машины.
java.util	Классы и интерфейсы стандартных утилит.

2.2.1. Модель состояний мидлета

Жизненный цикл мидлета включает несколько состояний. В таблице 2.7 перечислены возможные состояния MID-лета и их соответствующие описания.

Таблица 2.7.

Состояния MID-лета

Состояние	Описание
Active (Активен)	MID-лет либо готов к запуску, либо запущен. Процесс, который управляет MID-летом, может не быть в запущенном состоянии, но MID-лет все равно активен.
Destroyed (Прерван)	MID-лет не запущен и уже не может переходить в другие состояния.
Paused (Приостановлен)	MID-лет не выполняется. Он не может начать работу до тех пор, пока не перейдет в активное состояние.

На рисунке 2.3 показана диаграмма перехода из состояния в состояние, которая представляет из себя режимы мидлета и события, которые служат причиной перехода из одного состояния в другое. Методы startApp(), pauseApp() и destroyApp() позволяют MID-лету изменять свое состояние.

Рис. 2.3. Состояния MID-лета

В таблице 2.8 перечислены методы класса javax.microedition.midlet.MIDlet, которые управляют состоянием MID-лета.

Таблица 2.8.

Методы классов MID-летов, управляющие состояниями MID-лета

Название метода класса MID-лета	Описание
protected abstract void destroyApp()	AMS сигнализирует MID-лету о прекращении работы. MID-лет входит в прерванное состояние.
void notifyDestroyed()	MID-лет запрашивает о входе в прерванное состояние.
void notifyPaused()	MID-лет запрашивает о дезактивации и входе в приостановленное состояние.
protected abstract void pauseApp()	AMS сигнализирует MID-лету остановиться, MID-лет входит в приостановленное состояние.
void resumeRequest()	MID-лет запрашивает о повторном входе в активное состояние.
protected abstract void startApp()	AMS сигнализирует MID-лету, что он активен.

Приложения MID отличаются от приложений J2SE тем, каким образом они заканчивают работу. Для завершения работы MID-лета необходимо вызвать метод notifyDestroyed(), который сигнализирует AMS, что MID-лет завершил выполнение. AMS закрывает MID-лета и все его объекты. Однако виртуальная машина продолжает работу.

Структура мидлета может быть представлена следующим образом:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class MIDPSkeleton extends MIDlet implements Comman-
dListener
{
```

```
private Command quit; // The Quit command
private Display display; // Declaring the display
// Initialize the Display and place system controls in the
Constructor
public MIDPSkeleton()
{
  display = Display.getDisplay(this);
  quit = new Command("Quit", Command.SCREEN, 2);
}
/**
* Initialize all the classes to be used in the program here
(startApp())
*/
public void startApp()
{
  ...
  ...
}
// If the Application needs to be paused temporarily.
public void pauseApp()
{
  ...
  ...
}
// Clean up when the application is destroyed.
public void destroyApp(boolean unconditional)
{
  ...
  ...
}
// Event handling routine.
public void commandAction(Command c, Displayable s)
{
  if (c == quit) // If Quit button is pressed.
  {
    notifyDestroyed(); // Call the destroyApp method.
  }
}
}
```

2.2.2. Процесс разработки приложений MIDP

Процесс разработки приложений на J2ME состоит из следующих этапов:

1. Проектирование и кодирование – написание программы.
2. Компилирование – компилирование программы с помощью стандартного компилятора J2SE.
3. Предварительная проверка – выполнение предварительной проверки обработки классов Java до упаковки: проверка использования операций с плавающей точкой и методов завершения в классах Java.
4. Упаковка – создание архивного файла JAR, содержащего ресурсы приложения, создание файла описания приложения, содержащего метаинформацию о приложении.
5. Раскрытие – распаковка и размещение ресурсов приложения под контролем эмулятора.
6. Выполнение – запуск приложения с использованием эмулятора.
7. Отладка – нахождение и выделение ошибок в программе и внесение исправлений в исходный код.

2.2.2.1. Проектирование и кодирование

Первоначально необходимо создать структуру каталогов, которая будет поддерживать разработку набора MID-летов, т.е. комплекта MID-летов, которые используют общие ресурсы приложений.

Сначала создадим каталог под названием Hello, что будет названием примера первого приложения, под директорией apps/, предназначенной для установки инструментария для работы с беспроводными устройствами. Эта директория является корневой для нового проекта. Проект – это организованное объединение ресурсов (исходного кода, файлов ресурсов, откомпилированных файлов), специфических для одного или более связанных приложений.

Для демонстрации примера будем использовать платформу для разработки мобильных приложений – Java ME SDK 3.0. Еѐ можно загрузить на сайта Sun Microsystems, http://java.sun.com.

На рисунке 2.4 показано главное окно приложения.

Рис. 2.4. Главное окно создания проекта

Создание проекта состоит из следующих этапов:

1) Выберите пункт меню Файл – Новый проект (File – New Project) или щелкните на значке проекта.
2) В новом окне выберите категорию Java ME SDK и тип проекта – MIDP приложение (MIDP Application). Нажмите кнопку Далее (Next) (рис. 2.5).

69

Рис. 2.5. Окно создания нового проекта

3) В поле название проекта (Project Name) укажите новое название (рис. 2.6). При необходимости, с помощью кнопки Обзор (Browse) измените расположение проекта (Project Location). Снимите флажок Создать Мидлет Привет (Create Hello MIDlet), и нажмите на кнопку Завершить (Finish).

70

Рис. 2.6. Окно создания названия и расположения нового проекта

В результате выполнения этих действий будет создан корневой каталог и подкаталоги, содержание которых представлено в таблице 2.9.

Таблица 2.9.

Каталоги проекта

Название каталога	Содержание директории
Bin	Файлы приложения: файл.jar, файл.jad, MANIFEST.MF.
classes	Откомпилированные и предварительно проверенные файлы .class.
Res	Файлы ресурсов приложения, такие, как файлы изображений .png в формате PNG.
Src	Файлы исходного приложения.
tmpclasses	Откомпилированные, непроверенные файлы .class.

2.2.2.2. Компиляция

Следующим этапом в цикле разработки является компиляция исходной программы. Прежде чем приступить к компиляции, следует проверить, что список командных путей среды оболочки включает маршрут к директории, в которой содержатся утилиты J2ME на компьютере.

Общая форма строки компиляции представляет из себя следующее:

```
S javac -d <tmpclasses dir> -bootclasspath <midpapi.zip lo-
cation> \
<location of Jva sourcce fie(s)>
```

Указание -d сообщает компилятору директорию, в которую нужно записывать непроверенные откомпилированные классы. Указание -bootclasspath указывает местоположение файла midpapi.zip, который поставляется вместе с инструментарием J2ME Wireless Toolkit, разработанным «Java Software», и содержит все классы MIDP, которые необходимы для написания приложений на J2ME. Среды разработки коммерческих производителей также включают этот файл. Указание -bootclasspath также сообщает компилятору о превосходстве над любой спецификацией CLASSPATH, которая, возможно, была установлена в среде своей оболочки. Заметьте, что это должен быть относительный маршрут доступа к файлу – относительный к корневой директории проекта. Наконец, следует указать имена путей исходных файлов Java, которые компилируете.

Чтобы откомпилировать набор MID-летов HelloWorld из директории apps/HelloWorld/, используйте следующую команду:

```
$ javac -d tmpclasses \
-bootclasspach ../../lib/midpapi.zip src/HelloWorld.Java
$
```

Указание -d сообщает компилятору записать непроверенные компилированные классы в директорию tmpclasses, которая является поддиректорией каталога HelloWorld/. Указание -bootclasspath определяет имя пути относительно данного каталога. Последний параметр указывает относительное имя пути исходного файла HelloWorld.Java.

После завершения компиляции файлов директория tmpclasses будет содержать непроверенные файлы .class:


```
$ Is -I tmpclasses/
total 0
-rw-r--r-- 1 vartan None 922 HelloWorld.class
$
```

2.2.2.3. Предварительная проверка

Следующим этапом после компиляции является предварительная проверка файлов .class, которые только что откомпилировали. Чтобы выполнить проверку, запустите следующую команду:

```
$ preverify -classpath "../../lib/midpapi.zip;tmpclasses" -d
classes \
tmpclasses
S
```

Параметр -d указывает директорию, в которую должны быть записаны предварительно проверенные выходные классы, генерируемые с помощью этой команды. Наконец, имя замыкающей директории, tmpclasses, показывает местонахождение, из которого можно получить непроверенные файлы классов, которые были созданы на предыдущем этапе компиляции.

Запуск команды preverify создает предварительно проверенные файлы .class в директории классов в соответствии с указаниями:

```
S Is -I classes/
total 0
-rw-r--r-- 1 vartan None 922 HelloWorld.class
$
```

Команда preverify является инструментом предварительной проверки файлов классов, который используется в процессе проверки файлов классов. Проверка файлов классов в CLDC, как и в J2SE, является процессом проверки истинности файлов классов Java и отклоняет неправильные файлы. Однако в отличие от процесса проверки в J2SE проверка файлов классов в CLDC включает два этапа:

1. предварительная проверка вне устройства,
2. проверка на устройстве.

Использование команды preverify, представляет собой фазу предварительной проверки вне устройства. В реальной среде эта первая фаза обычно осуществляется на сервере, с которого MIDP-приложения загружаются на мобильные устройства. Обычно сервер выполняет это до того, как делает приложение доступным для загрузки.

Причина появления этого нового процесса проверки заключается в том, что обычный верификатор файлов классов J2SE требует больше памяти и возможностей по обработке данных, чем стандартные мобильные устройства могут реально предоставлять. Новый верификатор CLDC требует намного меньше RAM и является при этом намного более эффективным.

2.2.2.4. Упаковка

Следующим этапом после предварительной проверки является упаковка приложения. Упаковка набора мидлетов включает 2 объекта:

- архивный файл Java файлов MID-лета,
- необязательный файл дескриптора приложения.

Согласно спецификации MIDP необходимо, чтобы упакова набора MID-летов осуществлялась с помощью утилиты архивации Java (JAR).

Архив JAR набора MID-летов может содержать несколько типов файлов, как показано в следующем списке:

- файл манифеста (manifest file) – файл, который описывает содержимое JAR-файла,
- файлы классов Java, которые содержат MID-леты из набора MID-летов архива,
- файлы ресурсов приложения, используемые MID-летами из набора MID-летов.

JAR файл манифеста (manifest file) содержит атрибуты, которые описывают содержимое самого JAR-файла. Его наличие в JAR-файле необязательно.

Другой необязательный описательный файл, называемый файлом дескриптора приложения, содержит информацию о наборе MID-летов. Этот файл иногда называется дескриптором приложения Java (JAD). Каждый набор MID-летов может иметь связанный с ним файл описания.

Файл дескриптора приложения используется по двум причинам. Программное обеспечение управления приложениями устройства (AMS) использует информацию из этого файла для первоначальной проверки того, что все MID-леты в файле JAR соответствуют требованиям устройства, перед тем как оно загрузит полный файл JAR. AMS также использует эту информацию для управления MID-летом. AMS устройства отвечает за установку и удаление наборов MID-летов. Оно

также обеспечивает MID-леты средой исполнения, требуемой спецификацией MIDP. Наконец, AMS управляет выполнением MID-летов, а именно запуском, приостановкой и закрытием всех MID-летов.

Наконец, сами MID-леты могут извлекать из конфигурации JAD-файла специфические атрибуты, которые представляют собой параметры MID-лета. Файл ресурсов приложения является основным механизмом для распаковки конфигураций MIDP-приложений.

Спецификация MIDP требует, чтобы в файле Manifest присутствовали определенные поля. Требуемые поля показаны в таблице 2.10.

Таблица 2.10.

Обязательные атрибуты файла Manifest.MF

Имя атрибута	Описание
MIDlet-Name	Название набора MID-летов.
MIDlet-Versiorv	Номер версии набора MID-летов в форме <major>.<minor>.<micro>, определяемой схемой спецификации управления версиями продукта JDK.
MIDlet-Vendor	Разработчик приложения (компания или частное лицо).
MIDlet-<n>	По одному на MID-лет в данном наборе, содержит разделяемый запятой список из текстового имени MID-лета, значка и имени класса n-ного MID-лета в наборе.
MicroEdition-Profile	Профиль J2ME, необходимый для исполнения MID-лета.
MicroEdition-Configuration	Конфигурация J2ME, необходимая для исполнения MID-лета.

Файл манифеста содержит строки атрибутов, один атрибут на строку. Каждый атрибут состоит из ключа и значения. После ключа ставится двоеточие, которое отделяет его от связанного с ним значения.

Файл MANIFEST.MF программы Hello находится в папке Hello/bin/. Он выглядит следующим образом:

```
MIDlet-l: Hello, Hello.png, Hello
MIDlet-Narae: Hello
MIDlet-Vendor: Vartan Piroumian
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```


Файл манифеста различает различные MID-леты, нумеруя их от MIDlet-l до MIDlet-/n. Число 1 должно идентифицировать первый MID-лет.

У атрибута MIDlet-1 существует три значения. Первое – название набора MID-летов, который содержит данный MID-лет, это значение может быть именем, воспринимающимся человеком. Второе значение является именем файла изображения PNG, который AMS использует как значок, представляющий этот MID-лет. Последнее значение является именем файла класса MID-лета, который определяет входную точку исполнения MID-лета.

AMS использует атрибуты MicroEdition-Configuration и MicroEdition-Profile для определения того, подходит ли MID-лет для данного устройства.

Спецификация MIDP позволяет также создавать необязательные поля в файле манифеста. В таблице 2.11 показаны необязательные поля файла манифеста.

Таблица 2.11.

Необязательные атрибуты файла MANIFEST.MF

Имя атрибута	Описание
MIDlet-Description	Описание набора MID-летов
MIDlet-Icon	Имя файла PNG, содержащегося в JAR
MIDlet-Info-URL	URL, который содержит дополнительную информацию об этом наборе MID-летов
	Минимальное количество байт данных постоянного

MIDlet-Data-Size	хранения, требуемое набором

Для создания файла JAR приложения, используйте следующую команду:

```
$ jar craf bin/MANIFEST.MF bin/HelloWorld.jar -C classes/ .
-C res .
$
```

Программное обеспечение управления приложениями на мобильном устройстве использует файл JAD для получения информации, необходимой для управления ресурсами во время выполнения мидлета. Файл дескриптора приложения является необязательным, однако ин-

формативным. Атрибуты файла дескриптора приложения перечислены в таблицах 2.12 и 2.13.

Таблица 2.12.

Обязательные атрибуты файла дескриптора приложения

Имя атрибута	Описание
MIDlet-Jar-URL	URL файла JAR набора MID-летов.
MIDlet-Jar-Size	Размер (в байтах) файла JAR.
MIDlet-Name	Имя набора MID-летов.
MIDlet-Vendor	Разработчик приложения (например, название компании или имя частного лица).
MIDlet-Version	Номер версии набора MID-летов в форме <major>.<minor>.<micro>, определяемой схемой спецификации управления версиями продукта JDK.
MicroEdition-Configuration	Конфигурация J2ME, необходимая для исполнения MID-лета.
MicroEdition-Profile	Профиль J2ME, необходимый для исполнения MID-лета.

Таблица 2.13.

Необязательные атрибуты файла дескриптора приложения

Атрибут	Описание
MIDlet-Data-Size	Минимальное количество байт данных постоянного хранения, требуемое набором.
MIDlet-Delete-Confirm	Указывает, должна ли AMS запрашивать подтверждение пользователя перед удалением MID-лета.
MIDlet-Description	Описание набора MID-летов.
MIDlet-Icon	Имя файла PNG, содержащегося в JAR.
MIDlet-Info-URL	URL, который содержит дополнительную информацию об этом наборе MID-летов.
MIDlet-Install-Notify	Указывает, должна ли AMS уведомлять пользователя перед установкой нового MID-лета.

В дополнение к необязательным полям, перечисленным в таблице 2.13, файл JAD может содержать отдельные поля атрибутов для каждого MID-лета, описанные и названные разработчиком приложения. В

названиях дополнительных атрибутов нельзя использовать зарезервированное слово «MIDlet-».

2.2.2.5. Раскрытие и выполнение

Системы инициализации предлагают распаковку приложения вслед за его загрузкой. Пользователи загружают файл JAR набора MID-летов на свои устройства и запускают его с помощью программного обеспечения системы управления приложениями устройства.

Для создания MID-лета необходимо выполнить следующие действия:

1) Правой кнопкой щелкните на Проект (Project) и выберите пункт Новый MID-лет (New – MIDlet).
2) В появившемся окне (рис. 2.7) укажите название (MIDlet Name) и класс (MIDlet Class Name) MID-лета. Затем нажмите на кнопку Завершить (Finish).

Рис. 2.7. Окно создания MID-лета

3) В результате будет создан новый файл, который будет добавлен в проект, и показан исходный код MID-лета в редакторе. Полученный код необходимо будет заменить на следующий:

```
import javax.microedition.midlet.*;
```



```
import javax.microedition.lcdui.*;
public class Hello extends MIDlet implements CommandListener
{
   public Hello() {
      exitCommand = new Command ("Exit", Command.EXIT, 1);
      tbox = new TextBox ("Hello world MIDlet", "Hello
World!", 25, 0);
      tbox.addCommand (exitCommand);
      tbox.setCommandListener (this);
   }
   protected void startApp() {
      Display.getDisplay (this).setCurrent (tbox);
   }
   protected void pauseApp() {}
   protected void destroyApp (boolean bool) {}
   public void commandAction (Command cmd, Displayable
disp) {
      if (cmd == exitCommand) {
         destroyApp (false);
         notifyDestroyed();
      }
   }
   private Command exitCommand;
   private TextBox tbox;
}
```

4) Нажмите кнопку Пуск (Run), в результате чего в эмуляторе появится сообщение «Hello World!» (рис. 2.8).

Рис. 2.8. Выполнение MID-лета

2.2.3. Модель компонентов пользовательского интерфейса MIDP

Классы для построения пользовательсого интерфейса в MIDP содержатся в пакете javax.microedition.lcdui. Эти классы инкапсулируют логику для работы с дисплеем мобильного устройства и создания основных высокоуровневых компонентов интерфейса.

Класс Display является менеджером дисплея и средств ввода мобильного устройства. Каждый MID-лет имеет ссылку только на один экземпляр этого класса. Эта ссылка может быть получена с помощью статического метода getDisplay():

```
public static Display getDisplay(MIDlet midlet);
```


Обычно приложение вызывает getDisplay() при первом вызове метода startApp() для получения ссылки на объект Display.

```
public void startApp()
{
    Display display = Display.getDisplay();
    …
}
```

Класс Display также позволяет получить информацию о мобильном устройсте, необходимую для адаптации приложения к различным типам дисплеев (табл. 2.14).

Таблица 2.14.

Методы класса Display

Метод	Описание
public int numColors()	Возвращает количество цветов, если значение истинно, или, в обратном случае, градаций серого, поддерживаемых данным дисплеем.
public boolean isColor()	Возвращает значение истинно, если данный дисплей является цветным.
	Возвращает количество уровней прозрачности, поддерживаемых данным устройством.

	Минимальное значение равно двум, что сви-
public int numAlphaLe-	детельствует о поддержке полной прозрач-
vels()	ности и полной непрозрачности без смеши-
	вания. Возвращение значения больше двух
	означает, что поддерживается смешивание.
	Возвращает один цвет из цветовой схемы
public int getColor(int	высокоуровневого пользовательского ин-
colorSpecifier)	терфейса на основании спецификации цвета.
	Возвращает стиль, используемый для рисо-
public int getBorderS-	вания границы в зависимости от состояния
tyle(boolean highlighted)	компонента (выделен/ не выделен).
public boolean flashBack-	Вызывает эффект мигания подсветки уст-
light(int duration)	ройства.
public boolean vibrate(int	Включает сигнал вибровызова.
duration)	

Компоненты пользовательского интерфейса, которые отображаются на экране, содержатся внутри контейнеров, наследуемых от абстрактного класса Displayable. Объект Displayable не виден на эк-

ране, до тех пор, пока он не ассоциирован с объектом Display с помощью метода setCurrent():

public void setCurrent(Displayable displayable),

public void setCurrent(Alert alert, Displayable displayable).

Здесь alert – это окно с сообщением, которое демонстрируется пользователю перед активацией Displayable.

Также, для получения текущего объекта Displayable можно использовать метод: public Displayable getCurrent().

Методы класса Displayable перечислены в таблице 2.15.

Таблица 2.15.

Методы класса Displayable

Метод	Описание
public void isShown()	Возвращает значение истинно, если есть отображение на дисплее.
public void addCommand(Command cmd)	Добавляет команду.
public void removeCommand(Command cmd)	Удаляет команду.
public void setCommandListener(CommandListener l)	Устанавливает слушателя для команд, заменяя предыдущего слушателя команд.
public void setTitle(String title)	Задает заголовок.
public String getTitle()	Получает заголовок.
public void setTicker(Ticker ticker)	Устанавливает таймер прерывателя, заменяя предыдущие значения.
public Ticker getTicker()	Получает значение таймера прерывателя.

MIDP содержит два абстрактных подкласса Displayable – Screen и Canvas.

Canvas – это базовый класс для написания приложений, имеющих дело с низкоуровневым GUI API. Этот класс предоставляет разработчику методы to handle game actions, key events и pointer events. Чтобы создать пользовательский интерфейс с помощью низкоуровнего API, необходимо создать подкласс от Canvas и реализовать метод paint(). После этого использовать различные методы для рисования напрямую по холсту.

Screen – это базовый класс, от которого наследуются все окна высокоуровневого API. В отличие от предыдущего класса, разработчи-

кам не нужно создавать подкласс от Screen. Существует четыре типа данного класса: TextBox, List, Alert и Form. Если Screen является видимым, то содержимое дисплея обновляется автоматически. Низкоуровневые события, генерируемые дисплеем или клавиатурой, обрабатываются внутри API, и если необходимо, конвертируются в высокоуровневые события, доступные разработчику.

2.2.4. Высокоуровневое API пользовательского интерфейса MIDP

Мидлет, написанный с помощью высокоуровневого API, обычно

состоит из одного или более экранов и набора команд, которые позволяют пользователю сообщать мидлету, какие действия выполнять и как переходить от одного экрана к другому.

Обработка событий с помощью Command

Класс Displayable поддерживает обработку событий пользовательского интерфейса с помощью команд.

Класс Command – это конструкция, которая инкапсулирует семантическую информацию о действии. Сама логика действия содержится не в этом объекте, а в CommandListener, ассоциированном с Displayable. Команды, как правило, имеют внешнее представление в пользовательском интерфейсе, которое зависит от семантической информации команды.

Команды могут быть добавлены и удалены с помощью следующих методов:

public void addCommand(Command cmd);
public void removeCommand(Command cmd).

Для создания команды можно воспользоваться одним из двух конструкторов:

public Command(String label, int commandType, int priority);
public Command(String shortLabel, String longLabel, int commandType, int priority).

Атрибуты label или shortLabel обычно отображаются на экране, а атрибут commandType используется для указания типа команды. Значением этого параметра должна быть одна из констант, определенных в классе Command (табл. 2.16).

Таблица 2.16.

Типы команд

Константа типа команды	Описание
public static int OK	Стандартный положительный ответ.
public static int CANCEL	Стандартный отрицательный ответ.
public static int BACK	Возвращает пользователя к предыдущему экрану.
public static int STOP	Останавливает текущий процесс.
public static int HELP	Запрос на онлайновую помощь.
public static int SCREEN	Команда приложения, которая выполняется в текущем экране.
public static int EXIT	Команда для выхода из приложения.
public static int ITEM	Команда, специфичная для компонентов экрана

или пунктов списка.

Атрибут priority определяет, какая команда должна отображаться на экране, если всем не хватает места. Чем ниже значение, тем выше приоритет.

Для того, чтобы команда вызывала конкретные действия при ее активации, необходимо зарегистрировать CommandListener для Displayable. Это делается с помощью следующего метода:

```
public void setCommandListener(CommandListener l);
```

CommandListener – это интерфейс с одним методом:

```
public void commandAction(Command c, Displayable d);
```

Этот метод вызывается каждый раз, когда какая-нибудь команда активируется на Displayable. Аргументами этого метода являются активируемая команда и Displayable, на котором эта команда была активирована.

TextBox

TextBox – это подкласс от экрана (Screen), который позволяет пользователю вводить и редактировать текст. Методы класса представлены в таблице 2.17.

Для создания компонета используется следующий конструктор:

```
public TextBox(String title, String text, int maxSize, int
constrants);
```

Здесь title – это заголовок, который отображается вверху компонента, text – текст в компоненте при инициализации, maxSize – максимальный размер компонента в символах и constrants – константа, управляющая ограничениями на пользовательский ввод.

Таблица 2.17.

Методы класса TextBox

Метод	Описание
public int getConstraints()	Получает ограничения для класса TextBox.
public void setConstraints(int constraints)	Устанавливает ограничения.
public int getMaxSize()	Возвращает максимальный размер (в символах), которые могут храниться в TextBox.

public int setMaxSize(int maxSize)	Устанавливает максимальный размер (в символах), которые могут содержаться в TextBox.
public String getString()	Получает содержимое TextBox в виде строки.
public void setString(String text)	Устанавливает содержимое TextBox в виде строки, заменяя предыдущее содержимое.
public int size()	Получает количество символов, которое в настоящее время хранится в TextBox.
public void delete(int offset, int length)	Удаляет символы из TextBox.
public void insert(String src, int position)	Добавляет строки в TextBox.
public void insert(char[] data, int offset, int length, int position)	Вставляет диапазон массива символов в TextBox.
public void setChars(char[] data, int offset, int length)	Устанавливает содержимое TextBox из набора символов, заменяя предыдущее содержимое.
public void getCaretPosition()	Получает текущую позицию ввода.

Для демонстрации создадим класс TextBox с одной командой выхода из приложения. Для TextBox создадим заголовок, таймер прерывателя, первоначальный текст и ограничим его размер 120 символами.

Листинг 2.1: Пример создания элемента TextBox.

```
/*
 * TextBoxExample.java demonstrates a typical use of the
TextBox class.
 */
package lab2;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
```

```
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
```



```
import javax.microedition.lcdui.Ticker;
import javax.microedition.midlet.*;
public class TextBoxExample extends MIDlet {
//    Maximum text size in the TextBox
   private static final int MAX_TEXT_SIZE = 120;
   public void startApp() {
//       Create the TextBox
       TextBox textBox = new TextBox("TextBox Example",
"Initial TextBox text",
             MAX_TEXT_SIZE, TextField.ANY);
//       Create a ticker
       Ticker ticker = new Ticker("Ticker for the Text-
Box");
       textBox.setTicker(ticker);
//       Create Exit Command
       Command exitCommand = new Command("Exit", Com-
mand.EXIT, 0);
       textBox.addCommand(exitCommand);
       textBox.setCommandListener(new CommandListener(){
          public void commandAction(Command c, Displayable
d) {
             notifyDestroyed();
          }
       });
//     Make the TextBox to be visible on the display
       Display.getDisplay(this).setCurrent(textBox);
   }
   public void pauseApp() {
   }
   public void destroyApp(boolean unconditional) {
   }
}
```

Alert

Элемент alert представляет собой экран, содержащий уведомление для пользователя.

В MIDP существует два типа уведомлений:

1) timed alert – уведомление находится на экране определенное время и затем исчезает;
2) modal alert – уведомление демонстрируется пользователю до тех пор, пока он его не закроет.

Для создания алертов используются следующие конструкторы:

```
public Alert(String title);
public Alert(String title, String alertText, Image alertImage, AlertType alertType);
```


Здесь title – заголовок окна, alertText – текст уведомления, alertImage – изображение, ассоциируемое с уведомлением, alertType – тип уведомления, определяемый одной из констант класса (табл. 2.18).

Таблица 2.18.

Методы класса Alert

Константа типа класса Alert	Описание
public static final int ALARM	Предупреждает пользователя о событии, для которого пользователь предварительно сделал уведомление.
public static final int CONFIRMATION	Уведомляет пользователя о подтверждении действий.
public static final int ERROR	Предупреждает пользователя об ошибочной операции.
public static final int INFO	Показывает пользователю не угрожающую информацию.
public static final int WARNING	Предупреждает пользователя о потенциально опасной операции.

Методы класса Alert перечислены в таблице 2.19.

Таблица 2.19.

Методы класса Alert

Метод	Описание

public int getDefaultTimeout()	Получает время по умолчанию для отображения оповещений.
public int getTimeout()	Получает время, на которое будет отображаться предупреждение.
public void setTimeout(int time)	Устанавливает время, за которое должно быть показано оповещение.
public Image getImage()	Получает изображения, используемые в уведомлении.
public void setImage(Image img)	Устанавливает набор изображений, используемых в оповещении.
public AlertType getType()	Возвращает тип уведомления.
public void setType(AlertType type)	Устанавливает тип оповещения.

По умолчанию создаются уведомления, рассчитанные по времени (timed alert), значение времени для которого можно получить с помощью метода getDefaultTimeout().

Если необходимо получить модальное уведомление, то необходимо вызвать метод setTimeout(Alert.FOREVER).

При создании оповещения можно использовать специальную команду DISMISS_COMMAND, которя устанавливает следующий экран, если истекает время или пользователь закрыл текущее сообщение. При необходимости, можно задать свою логику для выполнения этой команды. Для этого следует использовать статическую переменную public static final Command DISMISS_COMMAND, которая позволяет задать свой собственный сигнал управления.

В примере рассмотрим два типа оповещений: модальное и рассчитанное по времени. MID-лет сначала запускает оповещение с таймаутом 3000 мс. После окончания таймаута текущим экраном становится TextBox. Это достигается с помощью метода setCurrent(Alert alert, Displayable displayable). Этот метод делает текущим оповещение, а после того как оно закрывается – displayable. Из класса TextBox можно вызвать модальное оповещение (modalAlert). Оно ждет своего закрытия пользователем, после чего активным снова становится TextBox, который имеет только команду Завершить (Exit).

Листинг 2.2: Пример создания уведомлений.

```
/*
```

```
 * AlertExample.java
 */
package lab2;
import java.io.IOException;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.*;
public class AlertExample extends MIDlet {
// Get the instance of Display
    private Display display = Display.getDisplay(this);
// The Alerts
```



```
    private Alert timedAlert, modalAlert;

//   The TextBox
     private TextBox textBox;
//   Command for exit the application
     private Command exit = new Command("Exit", Command.EXIT,
0);
// Command for transition to the modal alert
    private Command nextAlert = new Command("Next alert",
"Show next alert",
            Command.OK, 0);
// Path to the image file
    private static final String pathToImage = "images/alert-
image.png";
    public void startApp() {
//         Load the image
        Image image = loadImage();
//       Create the TextBox
        textBox = new TextBox("", "", 120, TextField.ANY);
        textBox.setCommandListener(new CommandListener() {
            public void commandAction(Command c, Displayable
d) {
```

```
            if (c == exit) {
                notifyDestroyed();
            }
            else if (c == nextAlert) {
                textBox.setTitle("TextBox after modal
alert");
                textBox.removeCommand(nextAlert);
                display.setCurrent(modalAlert, textBox);
            }
        }
    });
//    Create the timed Alert
    timedAlert = new Alert("Timed alert", "The alert is
going to be gone",
            image, AlertType.INFO);
    timedAlert.setTimeout(3000);
//    Create the modal Alert
    modalAlert = new Alert("Modal alert", "The alert is
waiting for action",
            image, AlertType.WARNING);
    modalAlert.setTimeout(Alert.FOREVER);
    textBox.setTitle("TextBox after timed alert");
    textBox.addCommand(exit);
    textBox.addCommand(nextAlert);

//    Make the timed Alert visible
```



```
    display.setCurrent(timedAlert, textBox);

  }
  public void pauseApp() {
  }
  public void destroyApp(boolean unconditional) {
  }
  private Image loadImage() {
    try {
//      Try to load the image by path
      return Image.createImage(pathToImage);
    }
    catch (IOException e) {
//      If an exception occurred we return null
```

```
            System.out.println(e.getMessage());
            return null;
        }
    }

}
```

List

Компонент List – это подкласс класса экран (Screen), который позволяет пользователю выбирать элементы в списке. Элементами могут быть текстовые строки и изображения.

Текущая версия MIDP поддерживает три типа компонента List:

1. EXCLUSIVE позволяет выбирать только один элемент из списка.
2. MULTIPLE позволяет выбирать несколько элементов из списка.
3. IMPLICIT выполняет команду каждый раз при выборе элемента списка.

Создание объекта List возможно с помощью следующих конструкторов:

public List(String title, int listType);
public List (String title, int listType, String[] stringElements, Image[] imageElements).

Здесь title – заголовок окна, stringElements – массив, не содержащий null, задает текстовые элементы списка, imageElements – графические элементы списка. Аргумент ListType определяет тип списка и задается тремя константами: Choice.EXCLUSIVE, Choice.MULTIPLE, Choice.IMPLICIT.

Кроме того, класс List также реализует интерфейс Choice. Методы данного класса представлены в таблице 2.20.

Методы класса List

Метод	Описание
public void set(int elementNum, String stringPart, Image imagePart)	Задает строку и изображение для указанного элемента, заменяя при этом предыдущее содержимое элемента.
public void insert(int elementNum, String stringPart, Image imagePart)	Вставляет элемент в список до указанного элемента.
public int append(String stringPart, Image imagePart)	Добавляет элемент в список.
public String get-	Получает строку указанного элемен-

String(int elementNum)	та.
public Image getI-mage(int elementNum)	Получает изображение указанного элемента.
public void de-lete(int elementNum)	Удаляет указанный элемент.
public void deleteAll()	Удаляет все элементы из списка.
public int size()	Получает количество элементов в списке.
public void setFitPoli-cy(int fitPolicy)	Устанавливает предпочтительную политику приложения, которая настраивает элемент Choice в зависимости от доступного места экрана.
public void set-Font(int elementNum, Font font)	Устанавливает шрифт для указанного элемента.
public Font get-Font(int elementNum)	Возвращает шрифт для выбранного элемента.
public boolean isSe-lected(int elementNum)	Получает логическое значение, указывающее, является ли данный элемент выбранным.
public int getSelectedIndex()	Возвращает порядковый номер выбранного элемента в списке.
public void setSelectedIn-dex(int elementNum, boo-lean selected)	Устанавливает выбранное состояние элемента.
public int getSelected-Flags(boolean[] selectedArray_r eturn)	Запрашивает состояние списка и возвращает состояние всех элементов в логическом массиве.
public void setSelected-Flags(boolean[] selectedArray)	Устанавливает выбранное состояние для всех элементов списка.

Когда пользователь выбирает элемент в списке IMPLICIT, вызывается специальная команда, ассоциированная с данным списком. Дей-

ствие, выполняемой этой командой, зависит от типа устройства. Для создания приложений рекомендуется самим выбирать команды с помощью метода setSelectCommand().

Для демонстрации работы со списками рассмотрим пример приложения, состоящего из магазина музыкальных композиций – Store и персонального списка предпочтений – Favorites. Магазин (Store) реализован как список List.MULTIPLE, а список предпочтений (Favorites) как List.IMPLICIT. Также приложение содержит меню в

виде List.IMPLICIT. Пользователь может выбирать композиции из магазина и добавлять их в список предпочтений. В списке предпочтений (Favorites) композиции могут быть «прослушаны» (показывается сообщение (alert) на 5 сек) и перемещены обратно в магазин (Store).

Листинг 2.3: Пример работы со списками.

```
package lab2;
/*
 * ListExample.java
 */
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.*;
public class ListExample extends MIDlet {
    Display display = Display.getDisplay(this);
// The Store list
    private List storeList;
// The Favorite list
    private List favoriteList;
// The Menu list
    private List menuList;
// Commands
    private Command exitCommand = new Command("Exit", Com-
mand.EXIT, 0);
    private Command gotoFavoriteCommand = new Com-
mand("Favorites", Command.OK, 0);
    private Command playCommand = new Command("Play", Com-
mand.ITEM, 0);
    private Command deleteCommand = new Command("Delete",
Command.ITEM, 1);
    private Command gotoMenuCommand = new Command("Menu",
Command.OK, 0);
```



```
    private Command takeSongsCommand = new Command("Take",
Command.SCREEN, 0);
    public void startApp() {
```

```
// Create all lists
        createStore();
        createFavoriteList();
        createMenu();
// Start the app with the main menu
        display.setCurrent(menuList);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    private void createStore() {
// Form the list of songs
        String[] items = {
            "Beethoven. Symphony No. 5: I",
            "Tchaikovsky. 1812 Overture",
            "Mozart. Eine Kleine Nachtmusik: Allegro",
            "Bach. Toccata And Fugue In D Minor",
            "Rossini. William Tell Overture",
            "Pachelbel. Canon In D",
            "Strauss. Blue Danube"
        };
// Create the list
        storeList = new List("Store", List.MULTIPLE, items,
null);
// Add the commands
        storeList.addCommand(takeSongsCommand);
        storeList.addCommand(gotoFavoriteCommand);
// Define the logic for commands
        storeList.setCommandListener(new CommandListener() {
            public void commandAction(Command c, Displayable
d) {
                if (c == takeSongsCommand) {
// Find selected items, remove them from the Store, and
// add them to the Favorites
                    boolean[] selected = new boo-
lean[storeList.size()];
                    storeList.getSelectedFlags(selected);
                    for (int i = 0; i < storeList.size();
i++) {
                        if (selected[i]) {
                            String item = store-
List.getString(i);
                            favoriteList.append(item, null);
```

```
                    storeList.delete(i);
```



```
                }
            }
        }
        else {
// Go to the Favotites
            display.setCurrent(favoriteList);
        }
    }
  });
 }
 private void createFavoriteList() {
// Create the list
    favoriteList = new List("Favorites", List.IMPLICIT);
// Set the select command
    favoriteList.setSelectCommand(playCommand);
// Set other commands
    favoriteList.addCommand(deleteCommand);
    favoriteList.addCommand(gotoMenuCommand);
    favoriteList.setCommandListener(new CommandListen-
er() {
        public void commandAction(Command c, Displayable
d) {
            if (c == playCommand) {
// Show an alert for 5 seconds
                int index = favoriteL-
ist.getSelectedIndex();
                String item = favoriteL-
ist.getString(index);
                Alert alert = new Alert("Playing...",
item, null, AlertType.INFO);
                alert.setTimeout(5000);
                display.setCurrent(alert, favoriteList);
            }
            else if (c == deleteCommand) {
// Remove the item from this list and add it to Store
                int index = favoriteL-
ist.getSelectedIndex();
                String item = favoriteL-
ist.getString(index);
```

```
                favoriteList.delete(index);
                storeList.append(item, null);
            }
            else {
                display.setCurrent(menuList);
            }
        }
    });
}
```



```
private void createMenu() {
    String[] items = {"Store", "Favorites", "Exit"};
    menuList = new List("Menu", List.IMPLICIT, items,
null);
    menuList.addCommand(exitCommand);
    menuList.setCommandListener(new CommandListener() {
        public void commandAction(Command command, Dis-
playable d) {
            if (command == List.SELECT_COMMAND) {
// Transfer to a screen depending on the selected item
                int index = menuList.getSelectedIndex();
                switch (index) {
                    case 0:
                        display.setCurrent(storeList);
                        break;
                    case 1:
                        dis-
play.setCurrent(favoriteList);
                        break;
                    default:
                        notifyDestroyed();
                }
            }
            else if (command == exitCommand) {
// Exit the application
                notifyDestroyed();
            }
        }
    });
}
```

}

Form

Класс Form – это подкласс класса экран (Screen), который может содержать произвольный набор компонентов пользовательского интерфейса, называемых items. Элементы, которые можно добавлять на форму, должны расширять класс Item.

Класс Form можно создать с помощью следующих конструкторов:

```
public Form(String title);
public Form(String title, Item[] items);
```

Первый вариант позволяет создать пустую форму с заголовком title, второй вариант с элементами пользовательского интерфейса items.

Управление items, находящимися на форме, можно осуществлять с помощью методов, перечисленных в таблице 2.21.

Таблица 2.21.

Методы класса Form

Метод	Описание
public int append(Item item)	Добавляет элемент в форму.
public int append(String str)	Добавляет элемент, состоящий из одной строки в форму.
public int append(Image img)	Добавляет элемент, состоящий из одного изображения в форму.
public void set(int itemNum, Item item)	Устанавливает указанный элемент, заменяя предыдущий.
public void insert(int itemNum, Item item)	Вставляет элемент в форму перед указанным элементом.
public void delete(int itemNum)	Удаляет указанный пункт.
public void deleteAll()	Удаляет все элементы из формы.
public int size()	Получает количество элементов в форме.
public Item get(int itemNum)	Получает элемент для данной позиции.

Item

Item – это суперкласс для компонетов, которые могут быть раз-

мещены на форме. Все объеты Item имеют поле подписи (label field). Подпись обычно размещается рядом с компонентом на одной с ним горизонтальной линии, или над компонентом. Items имеют минимальный и предпочтительный размеры, которые используются для компоновки Items на форме. Минимальный размер вычисляется реализацией устройства и может быть получен с помощью методов getMinimumWidth() и getMinimumHeight(). Предпочтительный же размер может устанавливаться как реализацией устройства, так и приложением с помощью метода setPreferredSize().

Класс Item содержит директиву размещения компонента на форме. Эта директива задается целым числом и управляется с помощью методов getLayout() и setLayout(). Значением директивы, как правило, является комбинация статических констант класса Item, которые перечислены в таблице 2.22.

Таблица 2.22.

Константы класса Item

Константы класса	Описание
LAYOUT_2	Отображает свойства MIDP 2.0.
LAYOUT_LEFT LAYOUT_RIGHT	Указывают на выравнивание по горизонтали и взаимно исключают друг друга.
LAYOUT_CENTER	
LAYOUT_TOP LAYOUT_BOTTOM	Указывают вертикальное выравнивание и взаимно исключают друг друга.
LAYOUT_VCENTER	
LAYOUT_SHRINK LAYOUT_EXPAND LAYOUT_VSHRINK LAYOUT_VEXPAND	Указывают на сокращение или расширение.
LAYOUT_NEWLINE_BEFORE LAYOUT_NEWLINE_AFTER	Указывают на запрос новой линии до или после элемента.

Также класс Item имеет три константы для отображения:

1. PLAIN – отображает Item в обычном виде.
2. HYPERLINK – показывает Item как гиперссылку.
3. BUTTON – отображает Item в виде кнопки.

StringItem

StringItem – это простой подкласс Item, который представляет

собой текстовую надпись. Создать StringItem можно с помощью следующих конструкторов:
public StringItem(String label, String text);
public StringIten(String label, String text, int appearance-
Mode);

Здесь, параметр appearanceMode может принимать одно из трех значений: Item.PLAIN, Item.HYPERLINK и Item.BUTTON.

Методы класса StringItem представлены в таблице 2.23.

Таблица 2.23.

Методы класса StringItem

Метод	Описание
public String getText()	Получает текстовое содержимое или null, если надписи нет.
public void setText(String text)	Устанавливает текстовое содержимое в надпись.
public Font getFont()	Получает шрифт для надписи.
public void setFont(Font font)	Устанавливает предпочтительный шрифт для надписи.
public int getAppearanceMode()	Возвращает тип обображения напдписи.

Spacer

Класс Spacer – представляет собой пустое пространство на форме и используется для размещения компонентов. Создать объект Spacer можно с помощью следующего конструктора:
public Spacer(int minWidth, int minHeight);

TextField

Класс TextField – это компонет для редактирования текста, который может быть размещен на форме. Для создания объекта используется следующий конструктор:
public TextField(String label, String text, int maxSize,
int constraints);

Здесь label – надпись компонета, text – первоначальный текст компонента, maxSize – максимальный размер компонента в символах, constraints – ограничения ввода текста пользователем. Задать input constraints для компонента можно с помощью следующих констант, являющихся статическими членами класса TextField:

- ANY – позволяет вводить любые данные,
- NUMERIC – позволяет вводить только числа,

- DECIMAL – позволяет вводить числа с десятичной частью,
- PHONENUMBER – позволяет вводить данные только в формате телефонных номеров,
- EMAILADDR – позволяет вводить числа в виде e-mail,
- URL – ввод должен соответствовать формату URL.

Также сюда входят модифицирующие флаги, позволяющие задавать внешний вид вводимых данных: PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_WORD, INITIAL_CAPS_SENTENCE.

Методы класса TextField в основной совпадают с методами класса TextBox.

ImageItem

Класс ImageItem – это компонент, который может содержать изображение. Существуют две формы конструктора для ImageItem:

```
public   ImageItem(String label,   Image image,        int layout,
String altText);
public   ImageItem(String label,   Image image,        int layout,
String altText, int appearanceMode);
```


Здесь label – строковая метка для изображения, image – графические данные, представленные классом Image, layout – директива свойств, altText – альтернативный текст, appearanceMode – тип отображения. Параметры Label и altText могут быть пропущены.

DateField

Класс DateField – это редактируемый компонет для представления даты и времени. DateField имеет два конструктора:

```
public DateField(String label, int mode);
public DateField(String label, int mode, TimeZone timeZone);
```

Здесь label – текстовая метка компонента, mode – режим, который может принимать значения DateField.DATE, DateField.DATE_TIME или DateField.TIME, timeZone – объект, описывающий часовой пояс.

Методы класса DateField перечислены в таблице 2.24.

Таблица 2.24.

Методы класса DateField

Метод	Описание
public Date getDate()	Возвращает значение даты.
public int	Получает режим входа для поля даты.

getInputMode()	
public void setDate(Date date)	Устанавливает дату.
public void setInputMode(int mode)	Устанавливает режим ввода для поля даты.

Gauge

Класс Gauge представляет целое число в графической форме. Он содержит текущее значение, которое лежит между нулем и максимальным значением.

Объект класса Gauge можно создать с помощью следующего конструктора:

public Gauge(String label, boolean interactive, int maxValue, int initialValue);

Здесь label – строковое значение для компонента, параметр interactive – определяет возможность изменения значения компонента пользователем, maxValue – максимальное значение, initialValue – первоначальное значение в интервале от 0 до maxValue, или, если maxValue равно значению Gauge.INDEFINITE, то одна из следующих констант: Gauge.CONTINUOUS_IDLE,

Gauge.CONTINUOUS_RUNNING, Gauge.INCREMENTAL_IDLE или Gauge.INCREMENTAL_UPDATING.

Класс Gauge имеет следующие методы, перечисленные в таблице 2.25.

Таблица 2.25.

Методы класса Gauge

Метод	Описание
public boolean isInteractive()	Сообщает, может ли пользователь изменить значение.
public int getMaxValue()	Получает значение максимального диапазона работы процесса.
public void setValue(int value)	Устанавливает текущее значение процесса.
public int getValue()	Получает текущее значение в процессе работы.
public void setMaxValue(int maxValue)	Устанавливает максимальное значение течения процесса.

ChoiceGroup

Класс ChoiceGroup также как и List, реализует интерфейс

Choice и имеет такие же методы. Конструкторы для создания объектов класса ChoiceGroup:
public ChoiceGroup(String label, int choiceType);
public ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements);

В целом, ChoiceType может принимать такие же значения, как и List, за исключением того, что вместо IMPLICIT объявлена константа POPUP, которая придает компоненту вид поля со списком или выпадающего меню.

CustomItem

Класс CustomItem – это абстрактный класс, предназначенный для создания новых компонентов, размещаемых на форме. Внешний вид и логика работы этих компонентов задаются с помощью методов, имеющих доступ к низкоуровневому API.

Пример, представленный ниже, содержит три формы. Первая форма демонстрирует возможности размещения элементов на форме. Трем элементам StringItem назначены разные режимы отображения: BUTTON, PLAIN и HYPERLINK. Первому StringItem в виде BUTTON назначена команда по умолчанию, которая меняет свойства компонентов. На второй форме размещены DateField и StringItem в виде кнопок (BUTTON). Активация кнопки циклически меняет вид Date-

Field. На третьей форме расположены компоненты Gauge и TextField, при изменении TextField меняется Gauge. Для этого для формы с помощью метода setItemStateListener регистрируем слушатель событий компонентов формы. В примере анонимный класс, реализующий интерфейс ItemStateListener, читает текущее значение TextField и устанавливает значение Gauge.

Листинг 2.4: Пример обработки форм.

```
/*
 * FormExample.java
 */
package lab2;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Gauge;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemCommandListener;
```

```
import javax.microedition.lcdui.ItemStateListener;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.*;
public class FormExample extends MIDlet
    implements CommandListener {
    private Display dispaly = Display.getDisplay(this);
// The global Next and Exit commands
    private Command exitCommand = new Command("Exit", Com-
mand.EXIT, 0);
    private Command nextCommand = new Command("Next", Com-
mand.OK, 0);
// The array of forms and the index of the current form
    private int currentForm = 0;
    private Form[] forms = new Form[3];
// The form items
    private StringItem item1, item2, item3;
    private DateField dateField;
    private Gauge gauge;
    public void startApp() {
        makeForms();
        dispaly.setCurrent(forms[nextFormIndex()]);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
```



```
// Process the global commands
    public void commandAction(Command command, Displayable
d) {
        if (command == exitCommand) {
            notifyDestroyed();
        }
        else if (command == nextCommand) {
            dispaly.setCurrent(forms[nextFormIndex()]);
        }
    }
    private int nextFormIndex() {
        if (currentForm == forms.length) {
            currentForm = 0;
        }
```

```
        return currentForm++;
    }
    private void makeForms() {
// Create the first form
        Form form1 = new Form("Form1. Layout of StringI-
tems");
        item1 = new StringItem(null, "one", Item.BUTTON);
        item2 = new StringItem(null, "two");
        item3 = new StringItem(null, "three",
Item.HYPERLINK);
// Define command and command listener for item1
        Command changeLayoutCommand = new Command("one",
Command.ITEM, 0);
        item1.setDefaultCommand(changeLayoutCommand);
        item1.setItemCommandListener(new ItemCommandListen-
er() {
            public void commandAction(Command c, Item item)
{
// Change the layout mode for all StringItems
                item1.setLayout(Item.LAYOUT_2 |
Item.LAYOUT_LEFT | Item.LAYOUT_NEWLINE_AFTER);
                item2.setLayout(Item.LAYOUT_2 |
Item.LAYOUT_CENTER | Item.LAYOUT_NEWLINE_AFTER);
                item3.setLayout(Item.LAYOUT_2 |
Item.LAYOUT_RIGHT | Item.LAYOUT_NEWLINE_AFTER);
            }
        });
        form1.append(item1);
        form1.append(item2);
        form1.append(item3);
// Create the second form
        Form form2 = new Form("Form2. DateField");
// Create the DateField item and place it on the form2
```



```
        dateField = new DateField("DateField", Date-
Field.DATE);
        dateField.setLayout(Item.LAYOUT_CENTER |
Item.LAYOUT_VCENTER);
// Create StringItem, place it and define Command and Com-
mandListener
```

```
        StringItem dateControl = new StringItem(null,
"Change", Item.BUTTON);
        dateControl.setLayout(Item.LAYOUT_CENTER |
Item.LAYOUT_BOTTOM);
        dateControl.setDefaultCommand(new Command("Change",
Command.ITEM, 0));
        dateControl.setItemCommandListener(new ItemComman-
dListener() {
            public void commandAction(Command c, Item item)
{
// Change DateField's input mode
                date-
Field.setInputMode((dateField.getInputMode() + 1) % 3 + 1);
            }
        });
        form2.append(dateField);
        form2.append(dateControl);
// Create the third form
        Form form3 = new Form("Form3. Gauge");
        gauge = new Gauge(null, false, 10, 5);
        TextField textField = new TextField(null, "5", 2,
TextField.NUMERIC);
        form3.setItemStateListener(new ItemStateListener() {
// Transfer changes from the textField to the gauge
            public void itemStateChanged(Item item) {
                TextField tField = (TextField) item;
                int value = Integ-
er.parseInt(tField.getString());
                if (value < 0)
                    value = 0;
                else if (value > 10)
                    value = 10;
                gauge.setValue(value);
            }
        });
        form3.append(gauge);
        form3.append(textField);
        forms[0] = form1;
        forms[1] = form2;
        forms[2] = form3;
// Add global Commands and define the CommandListener for
them
```

```
        for (int i = 0; i < forms.length; i++) {
            forms[i].addCommand(exitCommand);
            forms[i].addCommand(nextCommand);
            forms[i].setCommandListener(this);
        }
    }
}
```

2.3. Система управления записями

Одним из важных компонентов MIDP является система управления записями (Record Management System), которая позволяет сохранить данные в памяти устройства. Record Management System (RMS) – это программный интерфейс, используемый для хранения и манипулирования данными на мобильных устройствах, большинство из которых не поддерживают доступ к обычной файловой системе.

Запись (record) – это элемент данных, который может содержать число, строку, массив, изображение, т.е. всѐ то, что можно представить в виде последовательности байтов. На уровне API записи – это массивы байтов.

Функция интерпретации содержимого записи целиком возлагается на приложение. RMS предоставляет хранилище записей и уникальный идентификатор.

Хранилище (RecordStore) – это упорядоченная коллекция записей. Когда создается запись, хранилище присваивает ей уникальный целочисленный идентификатор (record ID). Например, первая запись получает record ID равный 1, вторая – 2 и т. д. Однако это не индекс, поскольку при удалении записи, оставшиеся элементы не перенумеровываются.

Название (name) используется для идентификации хранилища внутри MID-лета. Оно может содержать от 1 до 32 символов и должно быть уникально внутри MID-лета, создавшего хранилище. В MIDP 1.0 хранилища не могут использоваться более чем одним приложением. В версии MIDP 2.0 хранилища могут использоваться разными приложениями. В этом случае, хранилище идентифицируется не только названием, но и производителем приложения, создавшего это хранилище.

Кроме того, хранилище содержит информацию о дате последнего изменения и версии. Приложения также могут использовать обработчик события для учета изменения данных в хранилище.

В приложении Record Store представлена экземпляром javax.microedition.rms.RecordStore. Если необходимо получить

доступ к Record Store из другого MID-лета, то имя Record Store составляется из имѐн MID-лета и Record Store.

Для открытия Record Store используются следующие методы:

1. public static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary);
2. public static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable);
3. public static RecordStore openRecordStore(String recordStoreName, String vendorName, String suiteName);

Здесь, recordStoreName – название хранилища; createIfNecessary определяет, нужно ли создать новое хранилище, если такого не существует с данным именем; authmode – применяется только тогда, когда создается хранилище.

Константа authmode может принимать два значения: AUTHMODE_PRIVATE – позволяет осуществить доступ только из данной программы и AUTHMODE_ANY – позволяет осуществить доступ для любого комплекта мидлетов. Истинное значение аргумент writable принимает, если хранилище доступно для записи другим мидлетам; vendorName и suiteName – название вендора и мидлета соответственно.

Закрыть хранилище можно с помощью метода public void closeRecordStore(). Рекомендуется всегда закрывать хранилище записей, если в нем больше нет необходимости.

Для нахождения всех хранилищ, доступных для мидлета, используется метод: public static String[] listRecordStores().

Для удаления записи по имени необходимо применить метод public static void deleteRecordStore(String recordStoreName).

Режим доступа к хранилищу может быть изменен с помощью метода public void setMode(int authmode, boolean writable).

Чтобы узнать количество записей в хранилище, используется метод public int getSize().

Количество доступной памяти для записи можно узнать следующим образом: public int getSizeAvailable().

Исключения, входящие в состав пакета javax.microedition.rms, которые могут генерироваться при работе с хранилищем, перечислены в таблице 2.26.

Таблица 2.26.

Исключения при работе с хранилищем записей

Исключение	Описание
InvalidRecordIDException	Указывает, что операция не может быть выполнена, потому что передан неправильный идентификатор записи.
RecordStoreException	Суперкласс предыдущих исключений, который появляется при общих ошибках.
RecordStoreFullException	Сообщает, что закончился доступный объем памяти в хранилище.
RecordStoreNotFoundException	Указывает, что данное хранилище не существует.
RecordStoreNotOpenException	Сообщает, что приложение пытается использовать хранилище, которое было закрыто.

Чтобы добавить новую запись в хранилище, можно использовать следующий метод:
public int addRecord(byte[] data, int offset, int numBytes).

Здесь data – данные, которые необходимо сохранить в записи; offset – индекс массива data, начиная с которого данные будут сохранены в записи; numBytes – количество байтов, которое необходимо сохранить в записи. Метод возвращает идентификатор созданной записи.

Чтобы извлечь запись из хранилища, можно воспользоваться следующими методами:
1. public byte[] getRecord(int recordId).
2. public int getRecord(int recordId, byte[] buffer, int offset).

Здесь, recordId – идентификатор записи; buffer – байтовый массив, в который будут копироваться данные; offset – индекс в массиве, с которого нужно копировать. Первый метод возвращает копию записи, второй – данные записи.

Для удаления записи по идентификатору предназначен следующий метод: public void deleteRecord(int recordId).

Обновить запись можно с помощью метода:
public void setRecord(int recordId, byte[] newData,
int offset, int numBytes). Этот метод заменяет данные записи с указанным идентификатором на новые данные из массива newData.

Метод int getNextRecordID() возвращает идентификатор сле-

дующей записи, которая будет добавлена в Record Store.

Метод getNumRecords() позволяет получить количество доступных записей в хранилище.

Для демонстрации работы хранилища (RecordStore) создадим пример, в котором элемент TextBox будет сохранять свой текст после выхода из приложения. Данные из TextBox хранятся в записи с фиксированным идентификатором (id), задаваемым константой RECORD_ID. При закрытии приложения, в методе destroyApp(), происходит запись данных TextBox в хранилище. В свою очередь, в конструкторе, если хранилище содержит какую-либо информацию, происходит инициализация TextBox данными из хранилища.

Листинг 2.5: Пример работы хранилища записей.

```
/*
 * TextMiniEditor.java
 */
package lab3;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.*;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
public class TextMiniEditor extends MIDlet
    implements CommandListener
{
    private static final String STORE_NAME = "TextMiniEdi-
torStore";
    private static final int RECORD_ID = 1;
    private TextBox textBox;
    private RecordStore store;
    private Command exitCommand = new Command("Exit", Com-
mand.EXIT, 0);
    public TextMiniEditor()
    {
        textBox = new TextBox("Text Mini Editor", null,
2000, TextField.ANY);
```

```
        textBox.addCommand(exitCommand);
        textBox.setCommandListener(this);
        try {
            store = openStore();
            if (store.getNumRecords() > 0) {
                byte[] data = store.getRecord(RECORD_ID);
```



```
            textBox.setString(new String(data));
        }
    }
    catch (RecordStoreException ex) {
        showError(ex.getMessage(), false);
    }
    finally {
        try {
            store.closeRecordStore();
        }
        catch (RecordStoreException ex) {
            ex.printStackTrace();
        }
    }
}
protected void destroyApp(boolean unconditional)
{
    try {
        store = openStore();
        byte[] data = textBox.getString().getBytes();
        store.addRecord(data, 0, data.length);
    }
    catch (RecordStoreException ex) {
        showError(ex.getMessage(), true);
    }
    finally {
        try {
            store.closeRecordStore();
        }
        catch (RecordStoreException ex) {
            ex.printStackTrace();
        }
    }
}
protected void pauseApp() {}
```

```
protected void startApp()
   throws MIDletStateChangeException
{
   Display.getDisplay(this).setCurrent(textBox);
}
public void commandAction(Command c, Displayable d)
{
   if (c == exitCommand) {
      destroyApp(true);
      notifyDestroyed();
   }
}
```



```
   private RecordStore openStore() throws RecordStoreExcep-
tion
   {
      return RecordStore.openRecordStore(STORE_NAME,
true);
   }
   private void showError(String message, boolean exit)
   {
      Alert alert = new Alert("Error",
            "Record Store Error: " + message,
            null, AlertType.ERROR);
      alert.setTimeout(5000);
      if (exit) {
         alert.setCommandListener(new CommandListener() {
            public void commandAction(Command c, Dis-
playable d) {
               notifyDestroyed();
            }
         });
      }
      Display.getDisplay(this).setCurrent(alert);
   }
}
```

Мидлет содержит интерфейс RecordListener, который позволяет создавать объекты, получающие уведомления об изменениях в хранилище. Интерфейс содержит три метода:

1. void recordAdded(RecordStore recordStore, int

recordId) – вызывается после добавления записи;
2. void recordChanged(RecordStore recordStore, int recordId) – появляется после изменения записи;
3. void recordDeleted(RecordStore recordStore, int recordId) – может быть вызван после удаления записи.

Чтобы добавить/удалить обработчик событий к хранилищу, можно воспользоваться методами:
1. void addRecordListener(RecordListener listener).
2. void removeRecordListener(RecordListener listener).

Метод enumerateRecords класса RecordStore возвращает экземпляр класса, реализующего интерфейс RecordEnumeration, представляющего собой отсортированное подмножество записей из хранилища:
public RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated).

Здесь, filter – если не равен null, определяет, какие записи будут включены в результат; comparator используется для сортировки полученных записей; keepUpdated – если значение истинно, то интерфейс RecordEnumeration будет автоматически обновляться при изменениях в хранилище.

Интерфейс RecordFilter содержит единственный метод, который определяет совпадение записей в хранилище: boolean matches(byte[] candidate). Метод должен возвращать значение true, если запись с данными (candidate) соответствует заданному критерию.

Интерфейс RecordComparator осуществляет сортировку записей и также содержит один метод: int compare(byte[] rec1, byte[] rec2). Метод возвращает одну из трех констант (табл. 2.27).

Таблица 2.27.

Значение константы интерфейса RecordComparator

Значение константы	Описание
static int RecordComparator.EQUIVALENT	Записи равнозначны.
static int RecordComparator.FOLLOWS	Первая запись больше второй.
static int RecordComparator.PRECEDES	Вторая запись больше, чем первая.

Экземпляр интерфейса RecordEnumeration предназначен для перебора записей, извлеченных из хранилища. Для этих целей сущест-

вуют методы, перечисленные в таблице 2.28.

Таблица 2.28.

Интерфейс RecordEnumeration

Метод	Описание
byte[] nextRecord()	Возвращает копию следующей записи.
boolean hasNextElement()	Возвращает истинное значение, если есть следующий элемент.
int nextRecordId()	Возвращает идентификатор следующей записи.
byte[] previousRecord()	Возвращает копию предыдущей записи.
int previousRecordId()	Возвращает идентификатор предыдущей записи.
boolean hasPreviousElement()	Возвращает значение TRUE, если есть предыдущий элемент.
int numRecords()	Возвращает количество доступных записей.
void reset()	Сбрасывает индекс записи к первоначальному значению.
void destroy()	Освобождает используемые ресурсы.

2.4. Взаимодействие с сетью

Для работы с сетью в J2ME используется специальный MIDP API – Generic Connection Framework (GCF). Поддержка GCF осуществляется на уровне конфигурации. Этот набор интерфейсов расположен в пакете javax.microedition.io.

Основу GCF составляет класс Connector и набор интерфейсов. Класс Connector является основой для образцов подключения. Интерфейсы определяют типы поддерживаемых соединений, которые включают следующие основные этапы:

1. Мобильное приложение запрашивает класс Connector для открытия соединения с сетевым ресурсом.
2. Метод Connector.open() анализирует URI и возвращает объект Connection, который содержит ссылки на входной и выходной потоки к сетевому ресурсу.
3. Приложение получает объект InputStream или OutputStream из объекта Connection.
4. Мобильное приложение считывает данные из InputStream или записывает их в OutputStream в процессе своей обработки.
5. Приложение закрывает Connection при завершении работы

После устновки соединения анализируются сообщения протокола и полезная нагрузка сообщения. Например, если клиент устанавливает HTTP-соединение, он должен проанализировать синтаксис и семантику ответного сообщения HTTP-протокола, возвращенного сервером.

Интерфейс Connection – это наиболее общий тип соединения,

который может быть только открыт и закрыт. Метод open не объявлен как public, поскольку всегда вызывается только через статический метод open() класса Connector. Метод void close () – закрывает соединение с сетью. На момент вызова метода соответствующие потоки могут оставаться открытыми. В этом случае, закрытие соединения будет отложено до закрытия потоков, но доступ к соединению будет запрещен.

Интерфейс InputConnection используется для создания входной связи с сетью. Методами данного интерфейса являются следующие:

1. DataInputStream openDataInputStream() – открывает и возвращает поток ввода данных для конкретного соединения;
2. InputStream openInputStream() – открывает и возвращает входной поток для конкретного соединения.

Интерфейс OutputConnection необходим для создания выходной связи с сетью. Его методы:

1. DataOutputStream openDataOutputStream() – открывает и возвращает поток вывода данных для конкретного соединения;
2. OutputStream openOutputStream() – открывает и возвращает выходной поток для конкретного соединения.

Интерфейс StreamConnection определяет связь с потоком и не имеет методов и констант. Это просто интерфейс, сочетающий интерфейсы InputConnection и OutputConnection.

Интерфейс ContentConnection используется для определения связи с потоком. Данный интерфейс включает 3 метода:

1. string getEncoding() – определяет кодировку потока;
2. long getLength() – возвращает продолжительность соединения;
3. string getType() – возвращает тип соединения.

Интерфейс StreamConnectionNotifier необходим для ожидания установки соединения. Он использует метод acceptAndOpen(), который возвращает разъем сервера, с которым произошло соединение. Как и для всех других соединений, возвращенный поток следует закрыть по окончании использования.

Интерфейс javax.microedition.io.DatagramConnection дополняет интерфейс Connection и описывает соединения, которые посылают и получают дейтаграммы через протокол дейтаграмм.

Термин протокол дейтаграмм обозначает протокол, который находится на более низком уровне в модели OSI, чем протоколы уровня приложений. Протоколы дейтаграмм переносят дейтаграммы, которые иногда называются пакетами. Эти протоколы обычно переносят сооб-

щения дейтаграмм с одного устройства на другое, основываясь исключительно на информации, содержащейся в этой дейтаграмме. Несколько пакетов, отправленных с одного устройства на другое, могут быть переданы по различным маршрутам и приходить в любом порядке, при этом доставка пакетов не гарантирована.

Универсальный интернет-протокол передачи дейтаграмм – Internet Universal Datagram Protocol (UDP) – встроен непосредственно поверх интернет-протокола (IP) сетевого уровня и поддерживаеся некоторыми реализациями MIDP. В UDP отсутствуют многие свойства, которые имеются в транспортных протоколах, такие, как согласование вариантов соединений, повторная сборка пакетов, сквозной контроль потока, управление окнами, устранение ошибок, разбиение на части и гарантированная доставка. Эти свойства не поддерживаются в пользу очень бы-

строй скорости пересылки. Мидлеты могут использовать дейтаграммные соединения, когда им нужны быстрые соединения без перехода из состояния в состояние и когда не требуется гарантированная пересылка.

В таблице 2.29 перечислены методы интерфейса DatagramConnection.

Таблица 2.29.

Методы DatagramConnection

Метод	Описание
int getMaximumLength()	Выдает максимально возможную длину дейтаграммы, которая определена базовым протоколом реализации.
int getNominalLength()	Показывает номинальную длину дейтаграммы.
Datagram newDatagram(byte[] buf, int size)	Создает новый объект дейтаграммы, получая данные из указанного массива.
Datagram newDatagram(byte[] buf, int size, String addr)	Создает новый обьект дейтаграммы с указанными массивом данных и с указанным адресом назначения.
Datagram newDatagram(int size)	Создает новый обьект дейтаграммы.
Datagram newDatagram(int size, String addr)	Создает новый обьект дейтаграммы с указанным адресом.
void receive(Datagram dgram)	Получает дейтаграмму.
void send(Datagram dgram)	Отправляет дейтаграмму.

Чтобы использовать дейтаграммное соединение, приложение-клиент выполняет следующие шаги:

- Создает объект DatagramConnection.
- Получает объект Datagram из объекта DatagramConnection.
- Заполняет объект Datagram данными, составляющими полезную нагрузку, которая будет послана принимающему объекту.
- Запрашивает соединение о посылке дейтаграммы.
- Запрашивает соединение о получении ответной дейтаграммы.

Чтобы создать дейтаграммное соединение, необходимо использовать класс Connector. Следует указать, что требуется получить дейтаграммное соединение, поставляя строковую дейтаграмму в поле схемы URI, который передается формам метода Connector.open().

Полный синтаксис дейтаграммных адресов следующий:

address:= <протокол>://<адресат>
protocol:= "datagram"

target:= [<хост>]:<порт>
host:= Значимое DNS-имя хоста или его номер
port:= Значимый системный номер порта

Указание полей хоста необязательно. Если поле хоста пропущено, то соединение представляется со стороны сервера. Соединение сервера ожидает клиента для посылки ему дейтаграммы. Сервер извлекает адрес посылающего из дейтаграммы, полученной им, и использует его для ответа. Если поле хоста указано, соединение открывается как соединение со стороны клиента, т.е. запрашивающий соединение, является клиентом, чтобы отправить дейтаграмму адресованному узлу. Когда соединение установлено, мобильное приложение может использовать его для отправки и получения дейтаграмм.

Интерфейс javax.microedition.io.Datagram определяет дейтаграммы, которые являются частями сообщения, посланными и полученными протоколами передачи дейтаграмм. Объект DatagramConnection посылает и получает объекты Datagram.

В таблице 2.30 перечислены методы интерфейса Datagram

Таблица 2.30.

Методы интерфейса Datagram

Метод	Описание
String getAddress ()	Показывает адрес в дейтаграмме.
byte [] getData()	Выдает буфер, содержащий полезную нагрузку дейтаграмм.
int getLength()	Отображает длину полезной нагрузки дейтаграммы.

int getOffsetO	Выдает смещение указателя для чтения/записи в буфере полезной нагрузки.
void reset()	Восстанавливает позицию указателя для чтения/записи в буфере полезной нагрузки.
void setAddress (Datagram reference)	Устанавливает, что адрес данной дейтаграммы является адресом указанной дейтаграммы.
void setAddress (String addr)	Устанавливает адрес, указываемый строкой.
void setData (byte[] buffer, int offset, int len)	Устанавливает полезную нагрузку данной дейтаграммы.
void setLength (int len)	Устанавливает длину полезной нагрузки дейтаграммы.

Полезная нагрузка – это данные дейтаграммы без интерпретации их формы, структуры или типа. В интерфейсе Datagram отсутствует информация о синтаксисе или семантике полезной нагрузки. Дейтаграммы просто рассматривают полезную нагрузку как последовательность байтов.

Интерфейс SecureConnection определяет безопасную связь с сетью. Данный интерфейс имеет только один метод: SecurityInfo getSecurityInfo() – получает информацию о безопасности связи.

Профиль MIDP расширяет поддержку сетевых соединений, предоставляемую конфигурацией CLDC (Connection, ContentConnection, InputConnection, OutputConnection, StreamConnection). MIDP поддерживает подмножество протокола HTTP, который может быть реализован как поверх протоколов стека TCP/IP, так и поверх других протоколов, например WAP, с использованием шлюзов для доступа к HTTPсерверам в Internet. Кроме того, Generic Connection Framework используется для организации соединений клиент-сервер и датаграмных соединений. Однако стандартом гарантируется только реализация в рамках MIDP протокола HTTP 1.1.

Интерфейс javax.microedition.io.HttpConnection расширяет интерфейсы CLDC и обеспечивает дополнительную функциональность необходимую для создания заголовков HTTP-запросов, раз-

бора заголовков ответов и выполнения других функций протокола (табл. 2.31).

HTTP является протоколом запрос-ответ и соединение может находиться в одном из трех состояний:

1. установка (setup) – соединение с сервером не установлено и идет формирование параметров запроса;
2. соединено (connected) – установлено соединение, запрос передан серверу и ожидается ответ;
3. закрыто (closed) – получен ответ сервера и соединение завершено.

Таблица 2.31.

Методы интерфейса HttpConnection

Название метода HttpConnection	Описание
long getDate()	Возвращает данные.
long getExpiration()	Выдает значение поля заголовка Expires.
String getFile()	Отображает значение поля URL данного соединения.

Название метода HttpConnection	Описание
String getHeaderField (int n)	Возвращает заголовок файла по индексу.
String getHeaderField (String name)	Выдает заголовок файла по названию.
long getHeaderFieldDate (String name, long def)	Выдает значение поля лоя даты.
int getHeaderFieldInt (String name, int def)	Возвращает значение заданного поля для номера.
String getHeaderFieldKey (int n)	Получает файл заголовка по ключу.
String getHost()	Выдает информацию о соединении.
long getLastModified()	Возвращает значение модифицированного заголовка.
int getPort()	Отображает значение порта соединения.
String getProtocol()	Выдает имя протокола URL.
String getQueryO	Выдает запрос на соединение.
String getRef()	Выдает часть ссылки URL.

String getRequestMethod ()	Выдает метод текущего запроса.
String getRequestProperty (String key)	Выдает свойства запроса для соединения.
int getResponseCode()	Выдает код состояния протокола HTTP.
String getResponseMessage()	Возвращает сообщение о коде состояния протокола HTTP.
String getURL()	Выдает адрес соединения.
void setRequestMethod (String method)	Устанавливает метод запроса адреса (GET, POST или HEAD).
void setRequestProperty (String key, String value)	Устанавливает свойства общего запроса.

В дополнение к этим методам интерфейс HttpConnection также определяет полную совокупность констант, представляющих коды статуса и ошибок HTTP.

Соединения сокета являются последним типом соединений, явно представленных сетевой инфраструктурой MIDP. Реализации MIDP, поддерживающие сокеты, реализуют традиционные сокеты стиля UNIX.

Интерфейс StreamConnectionNotifier представляет известный сокет сервера. Интерфейс StreamConnection, который описан ранее, представляет сокет клиента.

Сокет – это сетевой механизм транспортного уровня, который обычно реализует пару протоколов TCP/IP в фиксированных межсетевых средах. Сокет сервера является программой, предоставляющей сетевую службу для соединений через сокеты.

Сокеты не требуют создания абсолютно никакой структуры полезной нагрузки, которую они транспортируют. Как и дейтаграммы, они просто транспортируют последовательность байтов. Служба определяет формат, синтаксис и семантику транспортируемых данных, составляющих сообщения. Клиенты должны соблюдать эти правила, для того чтобы использовать службу.

Соединения сокета находятся на транспортном уровне. Их поддержка осуществляется автоматически, если сокеты реализованы с помощью соединений TCP. TCP является ориентированным на соединения протоколом транспортного уровня, предназначенным для хранения данных в течение нескольких пересылок между клиентом и сервером.

Протоколы уровня приложений могут быть определены поверх протоколов транспортного уровня, если это необходимо. Реализация протокола уровня приложений использует любой доступный механизм транспортировки. Например, HTTP является протоколом уровня приложений. Создатели приложения MIDP могут выбирать, не создать ли протокол уровня приложений непосредственно поверх механизма сокета, если таковой поддерживается. Если сокеты не поддерживаются, сообщения протокола уровня приложений могут быть туннелированы с помощью HTTP. Протокол уровня приложений ответственен за определение своего собственного состояния, которое отличается от состояния протокола транспортного уровня.

Соединения сокета устанавливаются также как и другие типы соединений, клиенты используют метод Connector.open() и указывают URI базирующейся на сокетах службы, с которой они хотят соединиться. Однако со стороны сервера модель соединения немного отличается из-за определяемой соединениями природы сокетов. Эта модель необходима для серверов, чтобы иметь возможность обеспечивать многочисленные одновременные соединения клиентов.

Этапы соединения сокета:

- Сервер устанавливает соединение, которое связано с известным сокетом – сокетом сервера, чей порт и служба были предварительно установлены и объявлены.
- Сервер прослушивает запросы соединения клиента.
- Клиент создает запрос соединения для сервера и ожидает отклика.

- Сервер принимает запрос соединения и создает новое соединение с клиентом, сервер связывает соединение с новым сокетом. Сервер создает новый объект приложения, который будет взаимодействовать с клиентом через новое соединение и порождает нить для контролирования этого объекта.
- Возвращается запрос соединения клиента. Приложение клиента теперь имеет объект соединения, чьей конечной точкой является новый сокет, созданный сервером.
- Клиент и сервер взаимодействуют через новое соединение.
- Сервер продолжает прослушивать последующие запросы соединения на известном сокете.

Когда сервер принимает соединение на известном сокете, он не может взаимодействовать с другими клиентами, пока это соединение открыто. Поэтому сервер открывает второе соединение через новый сокет. Реализация на сервере уведомляет клиента и пересылает ему ин-

формацию о соединении с этим новым сокетом. Реализация клиента создает объект соединения, который общается с сервером через новый сокет. Сервер теперь свободен для прослушивания запросов соединения других клиентов на своем известном сокете.

Процесс открытия сокетов похож на процесс открытия дейтаграмм. Приложения пересылают URI в метод создания Connector.open() для получения соединения:

address:= <протокол>://<адресат>
protocol:= "socket"
target:= [<хост>]:<порт>
host:= <Значимое DNS-имя хоста или его номер>
port:= <Значимый системный номер порта>

Присутствие или отсутствие имени компьютера в URI говорит о том, является ли соединение серверным или клиентским.

Интерфейс ServerSocketConnection реализует связь с сервером. С этой целью используются следующие методы:

1. String getLocalAddress() – получает локальный адрес связи с разъемом (socket);
2. int getLocal Port() – получает локальный адрес связи с портом.

Интерфейс SocketConnection находит разъем (socket) для потока связи. Методы данного интерфейса следующие:

1. String getAddress() – получает адрес связи;
2. String getLocalAddress() – получает локальный адрес связи;

3. int getLocalPort() – получает локальный порт соединения;
4. int getPort() – получает порт соединения;
5. int getSocketOption (byte option) – получает необходимую опцию разъема для создания соединения;
6. void setSocketOption (byte option, int value) – устанавливает необходимую опцию разъема для создания соединения.

Важно заметить, что не все реализации поддерживают серверные сокеты.

Приведем пример соединения с сетью мидлета для заполнения HTML-формы. Веб-сайт имеет следующую HTML-форму (листинг 2.6).

Листинг 2.6: Пример соединения с сетью.

```
<html>
<body>
<form action="FormHandler.jsp" method="post">
```

```
Name: <input type="text" name="firstName">
  <input type="text" name="lastName"><br>
Sex:
  <input type="radio" checked name="sex" value="male">Male
  <input type="radio" name="sex" value="female">Female
<br>
<input type="submit">
</form>
</body>
</html>
```

Данный мидлет MidletFormHandler возвращает содержание HTML-страницы с помощью метода getHtml(), преобразуя HTML-код в объекты с использованием метода convertFormData, и вызывает метод composeInterface класса FormData.

```
package learn.j2me.lab4;
import java.io.DataInputStream;
import java.io.IOException;
import java.util.Vector;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.*;
import learn.j2me.lab4.forms.FormData;
import learn.j2me.lab4.forms.RadioInput;
import learn.j2me.lab4.forms.TextInput;
```



```
public class MidletFormHandler extends MIDlet {
   public static final String URL =
"http://localhost:8080/sample/";

   public void startApp() {
      String html = null;
      try {
         html = getHtml(URL);
         FormData form = convertFormData(html);
         form.composeInterface(Display.getDisplay(this));
      }
```

```
        catch (IOException ex) {
            Alert alert = new Alert(
                    "I/O Error", ex.getMessage(), null,
AlertType.ERROR);
            Display.getDisplay(this).setCurrent(alert);
        }
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    private String getHtml(String url) throws IOException {
        HttpConnection connection = null;
        DataInputStream dis = null;
        byte[] data = null;
        try {
            connection = (HttpConnection) Connec-
tor.open(url);
            int length = (int) connection.getLength();
            data = new byte[length];
            dis = new DataInput-
Stream(connection.openInputStream());
            dis.readFully(data);
        }
        finally {
            if (dis != null)
                dis.close();
            if (connection != null)
                connection.close();
        }
        return new String(data);
    }
    private FormData convertFormData(String html) {
//        We miss the process of html parsing here. Just
create the FormData
//        object by hand
        Vector inputs = new Vector();
```



```
  inputs.addElement(new TextInput("firstName"));
        inputs.addElement(new TextInput("lastName"));
        Vector radioValues = new Vector();
        radioValues.addElement("male");
        radioValues.addElement("female");
```

```
        inputs.addElement(new RadioInput("sex", radioValues,
0));
        FormData formData = new FormData("FormHandler.jsp",
inputs);
        return formData;
    }
}
```

Метод getHtml() создает Http-соединение и считывает данные с помощью метода readFully() класса DataInputStream. Метод convertFormData() необходим, чтобы конвертировать HTML-код в объекты HTML-формы. Объект FormData содержит структуру и методы, необходимые для отображения форм объекта в пользовательском интерфейсе MIDP.

```
package learn.j2me.lab4.forms;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Vector;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.ItemCommandListener;
import javax.microedition.lcdui.StringItem;
import javax.microedition.lcdui.TextField;
import learn.j2me.lab4.MidletFormHandler;
public class FormData {
    private String action;
    private Vector inputs;
    private Display display;
    public FormData(String action, Vector inputs) {
        this.action = action;
        this.inputs = inputs;
    }
    public void composeInterface(Display display) {
        this.display = display;
```

```
        Form form = new Form("Generated form");
        for (int i = 0; i < inputs.size(); i++) {
            InputData input = (InputData) in-
puts.elementAt(i);
            form.append(input.getComponent());
        }
        InputData submit = new Submit("Submit");
        form.append(submit.getComponent());
        display.setCurrent(form);
    };
    public void submit() {
        String params = makeParams();
        HttpConnection connection = null;
        OutputStream out = null;
        InputStream in = null;
        try {
            connection = (HttpConnection) Connector.open(
                    MidletFormHandler.URL + "/" + action);
            connec-
tion.setRequestMethod(HttpConnection.POST);
            connection.setRequestProperty(
                    "Content-Type", "application/x-www-form-
urlencoded");
            connection.setRequestProperty(
                    "Content-Length", Integ-
er.toString(params.length()));
            out = connection.openDataOutputStream();
            out.write(params.getBytes());
            in = connection.openInputStream();
            int length = (int) connection.getLength();
            byte[] data = new byte[length];
            in.read(data);
            String response = new String(data);
            Alert alert = new Alert("Response", response,
null, AlertType.INFO);
            display.setCurrent(alert);
        }
        catch (IOException e) {
            Alert alert = new Alert("I/O Error",
                    e.getMessage(), null, AlertType.ERROR);
            display.setCurrent(alert);
        }
        finally {
```

```
            try {
                if (out != null)
                    out.close();
                if (connection != null)
                    connection.close();
```



```
            }
            catch (IOException e) {}
        }
    }
    private String makeParams() {
        StringBuffer sb = new StringBuffer();
        Form form = (Form) display.getCurrent();
        for (int i = 0; i < inputs.size(); i++) {
            InputData htmlItem = (InputData) in-
puts.elementAt(i);
            Item formItem = form.get(i);
            String value;
            String className = formI-
tem.getClass().getName();
            if (className.endsWith("TextField")) {
                value = ((TextField) formItem).getString();
            }
            else if (className.equals("ChoiceGroup")) {
                ChoiceGroup cg = (ChoiceGroup) formItem;
                int index = cg.getSelectedIndex();
                value = cg.getString(index);
            }
            else {
                continue;
            }
            sb.append(htmlItem.name);
            sb.append("=");
            sb.append(value);
            sb.append("&");
        }
        return sb.toString();
    }
    public class Submit extends InputData {
        public Submit(String name) {
            super(name);
```

```
        }
        protected Item getComponent() {
            StringItem item = new StringItem(null, name,
Item.BUTTON);

            item.setDefaultCommand(new Command(name, Com-
mand.ITEM, 0));
            item.setItemCommandListener(new ItemCommandLis-
tener() {
                public void commandAction(Command c, Item
item) {
                    submit();
                }
```



```
            });
            return item;
        }
    }
}
```

Метод ComposeInterface() создает форму, добавляет в форму все элементы и отображает объекты на экране приложения. Метод Submit() является обработчиком события отправки для кнопки OnClick. Целью данного метода является формирование параметров для запроса POST.

Внутренний класс Submit расширяет абстрактный класс ввода данных (InputData). Он имеет один абстрактный метод getComponent(), который создает объект javax.microedition.lcdui.Item в зависимости от типа элемента.

```
package learn.j2me.lab4.forms;
import javax.microedition.lcdui.Item;
public abstract class InputData
{
    protected String name;
    public InputData(String name)
     {
        this.name = name;
    }
    protected abstract Item getComponent();
}
```

В этом примере реализованы три подкласса ввода данных (Input-

Data): Submit, TextInput и RadioInput.

Листинг 2.7: Класс TextInput.java.

```
package learn.j2me.lab4.forms;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.TextField;
public class TextInput extends InputData {
   public TextInput(String name) {
      super(name);
   }
   public Item getComponent() {
      TextField item = new TextField(name, "", 20, Text-
Field.ANY);
      return item;
   }

}
```


Листинг 2.8: Класс RadioInput.java.

```
package learn.j2me.lab4.forms;
import java.util.Vector;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Item;
public class RadioInput extends InputData {
   protected Vector values;
   protected int checked;
   public RadioInput(String name, Vector values, int
checked) {
      super(name);
      this.values = values;
      this.checked = checked;
   }
   protected Item getComponent() {
      String[] array = new String[values.size()];
      values.copyInto(array);
      ChoiceGroup item = new ChoiceGroup(name, Choice-
Group.EXCLUSIVE,
            array, null);
      return item;
   }
}
```

Результат работы мидлета представлен на рисунке 2.9.

Рисунок 2.9. Пример мидлета с формой ввода данных

Вопросы для самопроверки

1. Что такое конфигурация J2ME? Какая конфигурация была разработана для мобильных устройств с небольшой памятью?
2. Что такое профиль J2ME?
3. Как называется основной профиль для мобильных устройств? Каковы его свойства?
4. С помощью какого механизма реализуется работа с данными?
5. Какова архитектура классов, обеспечивающих соединения мобильных устройств с сетью?

3. СОЗДАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ОС ANDROID

В настоящее время существует два основных варианта разработки программных систем для мобильных устройств. Во-первых, создание клиентских приложений, обладающих графическим интерфейсом и сложной функциональностью, которые устанавливаются и выполняются на компьютере пользователя.

Такие программы могут работать двумя разными способами:

- локально (например, записные книжки, калькуляторы);
- совместно с программами, работающими на сервере, для организации взаимодействия, для получения доступа к базам данных, для выполнения сложных расчетов.

Взаимодействие с серверами может выполняться с использованием различных коммуникационных технологий. Это могут быть протоколы низкого уровня (например, сокеты) или протоколы более высокого

уровня (например, веб-сервисы).

Другим вариантом создания приложений, работающих с сервером, является создание веб-приложений. Веб-приложения – это прикладные программы, которые выполняются на стороне сервера, а для взаимодействия с пользователем необходима только одна программа – веб-браузер.

3.1. Основные инструменты разработки

Для разработки и отладки приложений под Android необходимы следующие инструменты:

1. Java Development Kit (JDK) 5 или выше (Если на вашем компьютере не установлен JDK, то его можно взять на странице http://www.oracle.com/technetwork/java/javase/downloads/index.html).
2. Android SDK – содержит отладчик, библиотеки, эмулятор, документацию и др. Он может быть получен по адресу http://developer.android.com/sdk/index.html.
3. Eclipse – данная интегрированная среда разработки (IDE) не является инструментом, без которого невозможно создание приложений, однако ее использование очень сильно упрощает и ускоряет процесс. Рекомендуемой IDE для создания приложений под Android является Eclipse. Eclipse можно скопировать по адресу http://www.eclipse.org/downloads.

4. Android development tools (ADT) – плагин Android development tools (ADT) является расширением для Eclipse, которое поддерживает создание и отладку приложений для Android.

3.1.1. Создание виртуальных устройств для Android (AVD)

Для тестирования Android-приложений необходим AVD. AVD – это сконфигурированный эмулятор, который позволяет моделировать устройства с заданными аппаратными и программными характеристиками. Рекомендуется создавать несколько AVD, чтобы тестировать приложения на нескольких конфигурациях.

Создать AVD можно с помощью программы AVD Manager, которая находится в корне каталога с установленной Android SDK.

3.1.2. Компоненты Android-приложения

Есть четыре типа компонентов приложения:

- Activity – представляет собой отдельный экран с пользовательским интерфейсом.

- Service – это фоновый компонент для выполнения длительных по времени операций или удаленных процессов. Сервис не имеет пользовательского интерфейса.
- Content provider – управляет разделяемыми данными приложений. Данные могут быть сохранены в файловой системе, в базе данных SQLite или в любом другом хранилище. Через content provider другие приложения могут получать доступ к этим данным.
- Broadcast receiver – это компонент, который отвечает на общесистемные широковещательные сообщения.

Уникальной особенностью системы Android является то, что любое приложение может запускать компонент другого приложения.

3.1.3. Первое Android-приложение

Создадим и запустим Hello World приложение, которое состоит одной текстовой метки с надписью Hello World.

1. Откройте ПО Eclipse. Создайте новый проект, используя пункты меню Файл → New → Проект (File → Новый → Project). В открывшемся окне Новый проект (New Project) выберите элемент Андроид (Android → Android Project) и нажмите кнопку Далее (Next) (рис. 3.1).

Рис. 3.1. Окно создания нового проекта

2. В окне Новый Проект Андроид (New Android Project) заполните поля, как показано на рисунке 3.2, и нажмите кнопку Завершить (Finish).

Рис. 3.2. Окно нового проекта

3. В Package Explorer откройте проект HelloWorld, затем откройте папку res/values и дважды щелкните по файлу strings.xml (рис. 3.3).

130

Рис. 3.3. «Package Explorer»

Для того чтобы модифицировать файл в текстовом режиме, выберите вкладку strings.xml, расположенную внизу. Измените файл strings.xml, как показано ниже:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World!</string>
   <string name="app_name">HelloWorld</string>
</resources>
```

4. Сохраните изменения с помощью комбинации клавиш Ctrl+S.

5. Выполните команду Пуск (Run). Откроется окно Запустить Как (Run As), в котором нужно выбрать пункт Приложение Андроид (Android Application) (рис. 3.4).

131

Рис. 3.4. Окно «Сохранить как»

Запустится эмулятор Android (если эмулятор заблокирован, нужно с помощью кнопки разблокировки разблокировать его). На рисунке 3.5 изображен работающий эмулятор с запущенным приложением Hello-World.

Рис. 3.5. Окно отображения результатов

6. Нажмите на кнопку Домой (Home), после чего будет виден «Домашний» экран (Home screen) (рис. 3.6).

132

Рис. 3.6. Окно «Home screen»

7. Нажмите кнопку Запуск приложений (Application Launcher) для просмотра списка установленных приложений. Приложение HelloWorld теперь установлено в Application Launcher (рис. 3.7).

Рис. 3.7. Окно просмотра установленных приложений

Таким образом, мы установили первое приложение на эмулятор.

3.1.4. Структура Android-приложения

Всю структуру Android-приложения в Eclipse можно увидеть с помощью Package Explorer (рис. 3.8).

Рис. 3.8. Окно «Package Explorer»

Проект Android-приложения содержит следующие папки и файлы:

- src – содержит исходные файлы .java и .aidl.

- Android 2.1-update1 – содержит только один файл, android.jar, необходимый для запуска Android-приложений.
- gen – содержит R.java, автоматически генерируемый файл, который содержит ссылки на все ресурсы проекта.
- assets – содержит другие файлы, необходимые для приложения, такие как HTML, текстовые файлы, базы данных.
- res – содержит ресурсы, используемые в приложении. Состоит из следующих подпапок
  - anim – содержит XML-файлы, компилируемые в анимационные объекты.
  - color – содержит XML-файлы, которые описывают цвета.
  - drawable – содержит графические ресурсы.
  - layout – содержит XML-файлы, которые описывают размещение компонентов на экране.
  - menu – содержит XML-файлы, которые описывают меню

приложения.

- raw – содержит разные файлы.
- values – содержит описания ресурсов. Ссылки на эти ресурсы определяются не именем файла, а данными из класса R.
- xml – содержит различные XML-файлы для конфигурирования компонентов.

• AndroidManifest.xml – управляющий файл, который описывает приложение и каждый из его компонентов.

Файл main.xml описывает пользовательский интерфейс и представляет собой файл компоновки:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
</LinearLayout>
```

Каждый элемент файла компоновки представляет собой имя файла, наследованного от View. В нашем примере файл компоновки состоит из двух элементов, LinearLayout и TextView. Элемент Linear-

Layout предназначен для управления размещением компонентов и более подробно будет рассмотрен далее. Здесь только укажем, что как корневой элемент XML-документа, он имеет обязательный атрибут xmlns:android. Это XML-декларация пространства имен, и каждый самый первый элемент должен иметь такой атрибут.

Элемент TextView предназначен для отображения текста пользователю и имеет в нашем примере три атрибута:

• android:layout_width – задает ширину компонента на экране. Значение fill_parent предписывает компоненту занимать всю область родительского компонента в указанном направлении.

• android:layout_height – задает высоту компонента. Значение wrap_content предписывает увеличение размера компонента только для отображения содержимого компонента.

• android:text – текст, который TextView должен показывать. В

данном примере вместо текстовой константы используется строковый ресурс. Строка hello определена в файле res/values/strings.xml. Это рекомендуемая практика использования строк, так как это упрощает локализацию приложений.

Файл strings.xml содержит текстовые строки, используемые в пользовательском интерфейсе:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World!</string>
   <string name="app_name">HelloWorld</string>
</resources>
```

Рассмотрим подробнее файл AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
     package="learn.android.lab1"
     android:versionCode="1"
     android:versionName="1.0">
   <uses-sdk android:minSdkVersion="7" />

   <application android:icon="@drawable/icon" andro-
id:label="@string/app_name">
       <activity android:name=".HelloWorldActivity"
              android:label="@string/app_name">
          <intent-filter>
             <action andro-
id:name="android.intent.action.MAIN" />
```



```
             <category andro-
id:name="android.intent.category.LAUNCHER" />
          </intent-filter>
       </activity>

   </application>
</manifest>
```

Файл AndroidManifest.xml содержит информацию о приложении:

- имя пакета приложения learn.android.lab1.
- версия кода приложения 1.

имя версии приложения 1.0.•

приложение может быть запущено только с установленным API• версии не ниже 7.

приложение использует графический файл icon.png из каталога• drawable.

имя приложения определено строкой app_name в файле• strings.xml.

в приложении одно activity, представленное классом Hello-• WorldActivity.java. Метка для этого activity совпадает с именем приложения.

action• с атрибутом android:name="android.intent.action.MAIN" означает, что данная activity служит входной точкой для приложения.

category• с атрибутом android:name="android.intent.category.LAUNCHER" означает, что приложение может быть запущено из Application Launcher.

Eclipse автоматически генерирует и обновляет файл R.java:

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */
package learn.android.lab1;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
      public static final int icon=0x7f020000;
   }
```



```
   public static final class layout {
      public static final int main=0x7f030000;
   }
   public static final class string {
      public static final int app_name=0x7f040001;
      public static final int hello=0x7f040000;
   }
}
```

Не рекомендуется вручную модифицировать этот файл, так как

его содержимое автоматически обновляется при изменениях в проекте.

Класс HelloWorldActivity.java связывает Activity с main.xml с помощью метода setContentView:

```
package learn.android.lab1;
import android.app.Activity;
import android.os.Bundle;
public class HelloWorldActivity extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```

Здесь R.layout.main является ссылкой на файл main.xml. При добавлении новых файлов компоновки в фолдер res/layout R.java автоматически изменяется. Метод onCreate() – это один из методов, активируемых, когда activity загружена.

3.1.5. Архитерктура Android GUI

Activity – это компонент приложения, предназначенный для создания пользовательского интерфейса. Каждый activity расширяет класс Activity и представляет собой экран, который приложение может демонстрировать пользователю.

Приложение обычно состоит из нескольких activities, которые слабо связаны друг с другом. Обычно одна activity указана как «главная», и она показывается пользователю, когда тот запускает приложение первый раз. Каждая activity затем может запускать другую activity для выполнения различных задач. Каждый раз, когда запускается новая activity, предыдущая останвливается, и система сохраняет ее в стеке, называемом бэкстэк (back stack).

Класс Activity опрелделяет несколько событий, которые управляют жизненным циклом activity:

- onCreate() – вызывается при создании activity,
- onStart() – вызывается, когда activity становится видимой,
- onResume() – вызывается, когда activity начинает взаимодействовать с пользователем,
- onPause() – вызывается, когда activity переходит в состояние паузы,

onStop() – вызывается, когда activity больше не видна пользова-• телю,

onDestroy() – вызывается перед уничтожением activity систе-• мой,

onRestart() – вызывается, когда activity была остановлена, а за-• тем снова запущена.

3.2. Создание пользовательского интерфейса

3.2.1. Объект View

В Android-приложениях пользовательский интерфейс строится с помощью объектов View и ViewGroup. Каждый экран в Android содержит иерархическое дерево элементов View. Класс View служит базой для подклассов, называемых «виджетами», которые представляют собой полностью реализованные объекты пользовательского интерфейса. Класс ViewGroup служит базой для подклассов, называемых «компоновка», которые предлагают различные типы архитектуры компоновки.

Объект View – это структура данных, свойства которой хранят данные и параметры компоновки для определенной области экрана. Объект View сам отвечает за свои размеры, компоновку, прорисовку, изменение фокуса, скроллинг и горячие клавиши.

На платформк Android интерфейс с пользователем строится в виде иерархического дерева, состоящего из узлов, относящихся к классам Veiw и ViewGroup. Для того чтобы добавить иерархию элементов пользовательского интерфейса к activity, необходимо вызвать метод setContentView() класса Activity и передать ему ссылку на корневой узел иерархии. Метод может принимать как ID ресурса, так и ссылку на View. Например, при инициализации activity является типичным следующий код:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```



```
setContentView(R.layout.main);
}
```

3.2.2. Компоновка

Наиболее распространенным способом размещения компонентов пользовательского интерфейса является использование XML-файлов

компоновки. Каждый элемент в файле является объектом View или ViewGroup. Объекты View – это листья, ViewGroup – ветви в иерархии.

Имя XML-элемента соответствует классу Java. Например, элемент TextView создает объект класса TextView. Когда загружается ресурс компоновки, Android инициализирует объекты, соответствующие элементам XML-файла.

Android поддерживает следующие объекты ViewGroup:

- LinearLayout.
- TableLayout.
- RelativeLayout.
- FrameLayout.
- ScrollView.

LinearLayout упорядочивает компоненты в один столбец или в одну строку.

Рассмотрим пример:

```
<?xml version=”1.0” encoding=”utf-8”?>
<LinearLayout
xmlns:android=”http://schemas.android.com/apk/res/android”
android:orientation=”vertical”
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
>
<TextView
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”@string/hello”
/>
<Button
android:layout_width=”160dp”
android:layout_height=”wrap_content”
android:text=”Button”
android:layout_gravity=”right”
android:layout_weight=”0.2”
/>
<EditText
android:layout_width=”fill_parent”
```



```
android:layout_height=”wrap_content”
android:textSize=”18sp”
android:layout_weight=”0.8”
/>
</LinearLayout>
```

В этом примере корневым элеметом является LinearLayout, и он имеет три дочерних элемента, TextView, Button и EditText. Элемент LinearLayout управляет порядком, в котором размещаются компоненты, указанные внутри этого элемента. Атрибут orientation может иметь имеет два значения: vertical и horizontal.

Все объекты View и ViewGroup имееют общий набор атрибутов, основные из которых перечислены в таблице 3.1.

Таблица 3.1.

Атрибуты объекта View

Атрибут	Описание
layout_width	Ширина компонента.
layout_height	Высота компонента.
layout_marginTop	Отступ сверху.
layout_marginBottom	Отступ снизу.
layout_marginLeft	Отступ слева.
layout_marginRight	Отступ справа.
layout_gravity	Управляет позиционированием дочерних элементов.
layout_weight	Влияет на количество места, выделяемого компоненту.
layout_x	Задает x-координату.
layout_y	Задает y-координату.

В примере ширина элемента TextView задана равной ширине родительского компонента с помощью значения атрибута fill_parent. Высота задана константой wrap_content, что значит, что его высота равна высоте его содержимого.

Ширина элемента Button задана с помощью абсолютного значения. В нашем примере она равна 160 dp. Размеры элементов пользовательского интерфейса Android можно задавать с помощью следующих единиц измерения:

- dp – это пиксель, не зависящий от плотности. Рекомендуемая единица измерения для задания размеров элементов экрана.
- sp – пиксель, не зависящий от масштаба. Рекомендуется для задания размеров шрифтов.

На рисунке 3.9 изображен результат выполнения примера.

Рис. 3.9. Пример задания свойств

AbsoluteLayout позволяет указывать точное значение координат дочерних элементов. Координаты задаются с помощью атрибутов

android:layout_x и android:layout_y. С версии 1.5 AbsoluteLayout не рекомендуется использовать, так как не гарантируется его поддержка в следующих версиях.

TableLayout группирует элементы в строки и столбцы. Строка в таблице создается с помощью элемента TableRow. Каждый View внутри строки создает ячейку. Ширина столбца определяется самой широкой ячейкой в столбце.

Листинг 3.1: Пример компоновки экрана с помощью TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    >
    <TableRow>
        <TextView
            android:text="UserName:"
            android:width="120px"
            />
```



```
        <EditText
            android:id="@+id/txtUserName"
            android:width="200px"
            />
    </TableRow>
    <TableRow>
        <TextView
            android:text="Password:"
            />
        <EditText
            android:id="@+id/txtPassword"
            android:password="true"
            />
    </TableRow>
    <TableRow>
        <TextView/>
        <CheckBox
            android:id="@+id/chkRememberPassword"
            android:layout_width="fill_parent"
```

```
            android:layout_height="wrap_content"
            android:text="RememberPassword"
            />
    </TableRow>
    <TableRow>
        <Button
            android:id="@+id/buttonSignIn"
            android:text="LogIn"
            />
    </TableRow>
</TableLayout>
```

Результат выполнения данного примера представлен на рисунке 3.10.

Рис. 3.10. результат использования свойства «TableLayout»

В примере два столбца и четыре строки. Ячейка под TextView с надписью Password заполнена пустым элементом TextView. Если этого не сделать, то checkbox RememberPassword будет расположен под TextView, а не под TextEdit.

Атрибут ID

Каждый объект View может иметь целочисленный идентификатор (ID), который уникальным образом идентифицирует объект в дереве компоновки. В компилируемом приложении этот ID имеет целочисленный тип, но в XML-файле этот атрибут задается в виде строки и имеет следующий синтаксис: android:id="@+id/txtUserName".

Символ @ в начале строки указывает, что XML-парсер должен проанализировать остальную часть строки и идентифицировать ее как ID. Символ + означает, что это новое имя ресурса, которое должно быть создано и добавлено в файл ресурсов.

В коде ссылка на экземпляр объекта может быть получена следующим образом:

```
EditText editText=(EditText) findViewById(R.id.txtUserName);
```

RelativeLayout позволяет размещать компоненты, задавая их взаимное положение. Рассмотрим пример.

Листинг 3.2: Пример использования RelativeLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/andro
id"
    android:id="@+id/RLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/lblComments"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Comments"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
    />
    <EditText
        android:id="@+id/txtComments"
        android:layout_width="fill_parent"
        android:layout_height="170px"
        android:textSize="18sp"
        android:layout_alignLeft="@+id/lblComments"
        android:layout_below="@+id/lblComments"
        android:layout_centerHorizontal="true"
    />
    <Button
        android:id="@+id/btnSave"
        android:layout_width="125px"
        android:layout_height="wrap_content"
        android:text="Save"
        android:layout_below="@+id/txtComments"
        android:layout_alignRight="@+id/txtComments"
    />
    <Button
        android:id="@+id/btnCancel"
        android:layout_width="124px"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_below="@+id/txtComments"
        android:layout_alignLeft="@+id/txtComments"
```

```
    />
</RelativeLayout>
```

Каждый объект View внутри RelativeLayout имеет атрибуты, которые позволяют выравнивать их относительно других элементов. Эти следующие атрибуты:

- layout_alignParentTop,
- layout_alignParentLeft,
- layout_alignLeft,
- layout_alignRight,
- layout_below,
- layout_centerHorizontal.

Значением каждого из этих атрибутов является ID объекта View, относительно которого производится выравнивание. Экран, получившийся в результате выполнения этого кода, выглядит следующим образом (рис. 3.11).

Рис. 3.11. Результат использования «RelativeLayout»

FrameLayout используется для показа одного View. Все объекты, добавляемые в FrameLayout, прикрепляются к верхнему левому углу.

ScrollView – это специальный тип FrameLayout, который позволяет прокручивать список объектов View, не вмещающихся в экран. ScrollView может содержать только один дочерний View или ViewGroup.

3.2.3. Обзор основных виджетов

Android содержит набор стандартных объектов View для создания простых пользовательских интерфейсов. Используя эти элементы управления, модифицируя и расширяя их, можно упростить разработку и улучшить совместимость приложений.

Следующий список содержит наиболее часто используемые элементы управления:

- TextView – отображает текст и опционально позволяет его редактировать.
- EditText – подкласс TextView, который предназначен для редактирования текста.
- Button – стандартная кнопка.
- CheckBox – кнопка с двумя состояниями.
- RadioButton – также кнопка с двумя состояниями, но, в отличие от CheckBox, RadioButton не может быть переведена пользователем в неотмеченное состояние.
- ImageButton – кнопка с картинкой вместо текста.
- ToggleButton – кнопка с двумя состояниями, использующая индикатор с подсветкой.

3.2.4. Обработка событий пользовательского интерфейса

Для обработки событий пользовательского интерфейса можно выполнить одно из двух действий:

- Создать слушатель событий и зарегистрировать его в объекте View.
- Перекрыть существующий метод обратного вызова для объекта View.

Слушатели событий

Слушатель событий – это интерфейс в классе View, который содержит один метод. Этот метод вызывается фрэймворком Android, когда View, с которым слушатель зарегистрирован, активируется пользователем.

Следующие методы включены в интерфейсы слушателей событий:

- onClick() – вызывается, когда пользователь или касается элемента, или фокусируется на элементе с помощью навигационных клавиш или трэкбола и нажимает кнопку ввод (enter) или на трэкбол.

- onLongClick() – вызывается, когда пользователь касается и удерживает элемент или фокусируется на нем с помощью навигационых клавиш или трэкбола и нажимает и удерживает клавишу ввод (enter) или нажимает и удерживает трэкбол.
- onFocusChange() – вызывается, когда пользователь наводит или убирает фокус с элемента, используя навигационные клавиши или трэкбол.

- onKey() – вызывается, когда пользователь фокусируется на элементе и нажимает или отпускает клавишу.
- onTouch() – вызывается, когда пользователь выполняет действие, квалифицируемое как событие касания, включая нажатие, отпускание, или любое движение по экрану.
- onCreateContextMenu() – вызывается при создании контекстного меню.

Каждый метод содержится в своем собственном интерфейсе в единственном числе. Чтобы задать один из этих методов, необходимо реализовать соответсвующий интерфейс и передать его методу View.set<eventname>Listener().

Например, setOnClickListener().

Всплывающие уведомления

Всплывающее уведомление – это сообщение, которое появляется на поверхности окна приложения. Оно занимает только область на экране, необходимую для сообщения, а основное приложение остается видимым и интерактивным. Уведомление автоматически появляется и затухает и не принимает событий взаимодействия с пользователем.

Для того чтобы создать всплывающее уведомление, необходимо создать экземпляр Toast с помощью статического метода makeText(). Этот метод принимает три параметра: контекст приложения Context, текстовое сообщение и длительность уведомления по времени. Метод возвращает инициализированный объект Toast. Затем уведомление можно отобразить с помощью метода show(), как показано в следующем примере.

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```


Стандартное уведомление появляется внизу экрана, по центру. Можно изменить это положение с помощью метода setGravity(int gravity, int xOffset, int yOffset). Метод принимает три параметра: константу Gravity, отступ по x и отступ по y.

Например, если нужно показать уведомление в верхнем левом углу, то можно использовать:

toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);.

Для демонстрации создадим приложение, включающее элементы интерфейса – RadioGroup, EditText и двух кнопок (Button). Первая кнопка выводит уведомление, содержащее текст из поля EditText. Вторая кнопка очищает содержимое EditText. Выбор RadioButton в RadioGroup помещает текстовую метку соответствующей RadioButton в EditText.

Листинг 3.3: Файл компоновки.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    android:id="@+id/relativeLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/andro
id"
    >
   <RadioGroup
      android:id="@+id/rg"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      >
      <RadioButton
        android:text="RadioButton 1"
        android:layout_width="wrap_content"
        android:id="@+id/radioButton1"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:checked="true"
        />
      <RadioButton
        android:text="RadioButton 2"
        android:id="@+id/radioButton2"
        android:layout_height="wrap_content"
        android:layout_below="@+id/radioButton1"
        android:layout_alignParentLeft="true"
        android:layout_width="144dp"
        />
```



```
      <RadioButton
        android:text="RadioButton 3"
```

```
        android:layout_width="wrap_content"
        android:id="@+id/radioButton3"
        android:layout_height="wrap_content"
        android:layout_below="@+id/radioButton2"
        android:layout_alignParentLeft="true"
        />
    </RadioGroup>
    <EditText
     android:layout_width="wrap_content"
     android:id="@+id/editText1"
     android:layout_height="wrap_content"
     android:layout_alignParentTop="true"
     android:layout_alignBottom="@+id/rg"
     android:layout_alignParentRight="true"
     android:layout_toRightOf="@+id/rg"
     />
    <Button
     android:text="Show Toast"
     android:layout_width="wrap_content"
     android:id="@+id/button1"
     android:layout_height="wrap_content"
     android:layout_below="@+id/rg"
     android:layout_alignParentLeft="true"
     android:layout_marginLeft="20dp"
     android:layout_marginTop="18dp"
     />
    <Button
     android:text="Clear TextEdit"
     android:layout_width="wrap_content"
     android:id="@+id/button2"
     android:layout_height="wrap_content"
     android:layout_alignBaseline="@+id/button1"
     android:layout_alignBottom="@+id/button1"
     android:layout_alignParentRight="true"
     android:layout_marginRight="34dp"
     />
</RelativeLayout>
```

Листинг 3.4: Файл MainActivity.java.

```
package learn.android.lab2;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
import android.widget.EditText;
```



```
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.RadioGroup.OnCheckedChangeListener;
import android.widget.Toast;
public class MainActivity extends Activity
    implements OnCheckedChangeListener
{
    private RadioGroup mRadioGroup;
    private Button mButton1, mButton2;
    private EditText mEditText;

    private OnClickListener mButtonListener = new OnClick-
Listener() {
        @Override
        public void onClick(View v) {
            int id = v.getId();

            switch (id) {
            case R.id.button1:
                String text = mEdit-
Text.getText().toString();
                Toast.makeText(getApplicationContext(),
text, Toast.LENGTH_SHORT).show();
                break;
            case R.id.button2:
                mEditText.setText("");
                break;
            }
        }
    };

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      mRadioGroup = (RadioGroup) findViewById(R.id.rg);
      mRadioGroup.setOnCheckedChangeListener(this);

      mButton1 = (Button) findViewById(R.id.button1);
```

```
        mButton1.setOnClickListener(mButtonListener);

        mButton2 = (Button) findViewById(R.id.button2);
        mButton2.setOnClickListener(mButtonListener);

        mEditText = (EditText) findViewById(R.id.editText1);
    }
    @Override
```



```
    public void onCheckedChanged(RadioGroup group, int
checkedId) {
        RadioButton rb = (RadioButton) findViewBy-
Id(checkedId);
        mEditText.setText(rb.getText().toString());
    }
}
```

В примере обработчик событий mButtonListener для Button реализован как внутренний класс, в то время как обработчиком для RadioGroup зарегистрирован сам класс MainActivity. Для этого он реализует интерфейс OnCheckedChangeListener и определяет метод onCheckedChanged.

Результат выполнения данного кода выглядит следующим образом (рис. 3.12).

Рис. 3.12. Результат использования обработчика событий

Объект ListView отображает элементы в вертикальном списке с прокруткой. Следующий пример демонстрирует, как отобразить список текстовых элементов с помощью ListView.

Листинг 3.5: Пример использования элемента ListView.

```
package learn.android.lab2;
import android.app.ListActivity;
import android.os.Bundle;
```

```
import   android.view.View;
import   android.widget.ArrayAdapter;
import   android.widget.ListView;
import   android.widget.Toast;
public   class ListViewExampleActivity extends ListActivity {

String[] strings;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
//         setContentView(R.layout.main);

        ListView listView = getListView();
        list-
View.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);

        strings                =                getRe-
sources().getStringArray(R.array.strings);

         setListAdapter(new ArrayAdapter<String>(
           this,    android.R.layout.simple_list_item_checked,
strings
         ));
    }

   public void onListItemClick(ListView parent, View v, int
position, long id) {
    parent.setItemChecked(position,                par-
ent.isItemChecked(position));
    Toast.makeText(this,
         "The string " + strings[position] + " has been se-
lected",
         Toast.LENGTH_SHORT).show();
   }
}
```

В данном примере класс ListViewExampleActivity расширяет ListActivity. Класс ListViewExampleActivity, в свою очередь, расширяет Activity и отображает список элементов путем связывания с источником данных. Также здесь нет необходимости модифицировать main.xml, класс ListActivity сам содержит ListView. Поэтому в методе onCreate() закомментирован вызов setContentView(R.layout.main).

Метод getListView() используется для получения ссылки на ListView. Это нужно, потому что необходимо программно изменить

поведение ListView. Затем использовался метод setChoiceMode()

153

для указания ListView, как он должен обрабатывать действия пользователя.

Строковые значения для списка хранятся в strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">Hello World, ListViewExampleActivity!</string>
   <string name="app_name">ListViewExample</string>
   <string-array name="strings">
    <item>string1</item>
    <item>string2</item>
    <item>string3</item>
    <item>string4</item>
   </string-array>
</resources>
```

Эти значения извлекаются программно с помощью выражения: getResources().getStringArray(R.array.strings).

Метод setListAdapter() используется для программного заполнения всей видимой области activity данными. Объект ArrayAdapter управляет массивом строк, который отображается в ListView. В нашем примере ListView настраивается для отображения в режиме simple_list_item_checked.

Метод onListItemClick() вызывается, когда активируется элемент в ListView. Внутри этого метода вызывается setItemChecked для показа иконки рядом с каждым элементом.

После запуска экран мобильного приложения выглядит следующим образом (рис. 3.13).

154

Рис. 3.13. Результат использования списков

3.2.5. Создание меню

Меню – это один из важных элементов пользовательского интер-

фейса, который позволяет пользователю выбирать действия.

В Android существует три типа меню:

- Меню опций – главное меню для activity, которое появляется, когда пользователь нажимает кнопку MENU.
- Контекстное меню – всплывающее меню, которое появляется, когда пользователь нажимает и удерживает компонент экрана, для которого зарегистрировано контекстное меню.
- Подменю – всплывающее меню, которое появляется, когда нажимает элемент меню, содержащее вложенной меню.

Создание ресурса меню

Не рекомендуется создавать меню программно. Вместо этого нужно описывать меню и его компоненты в XML-файле ресурса меню, затем загружать его в программном коде.

Для создания ресурса меню нужно создать XML-файл в каталоге проекта res/menu/ и построить меню с помощью следующих элементов:

- menu – задает объект Menu, который является контейнером для элементов меню. Элемент menu должен быть корневым узлом файла и может содержать один или более элементов item и group.
- item – создает объект MenuItem, который представляет собой отдельные пункт меню. Этот элемент может содержать вложенный элемент menu для создания подменю.
- group – опциональный невидимый контейнер для элементов item. Позволяет группировать пункты меню, так что они имеют одинаковые свойства, например активное состояние и видимость.

Пример меню, заданного файлом example_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
   <item          android:id="@+id/item1"          andro-
id:title="@string/menu_item1"/>
   <item          android:id="@+id/item2"          andro-
id:title="@string/menu_item2"/>
   <item          android:id="@+id/item3"          andro-
id:title="@string/menu_item3">
    <menu>
      <item          android:id="@+id/item4"          andro-
id:title="@string/sub_menu"/>
    </menu>
```

```
    </item>
</menu>
```

Этот пример описывает меню с тремя пунктами и одним подменю с одним пунктом. Пункты меню имеют атрибут android:title, который является ссылкой на строку пункта меню.

Загрузка ресурса меню

Из кода приложения ресурс меню конвертирован в программный объект с помощью MenuInflater.inflate(). Например, следующий код загружает example_menu.xml в методе onCreateOptionsMenu():

```
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.example_menu, menu);
    return true;
}
```


Метод getMenuInflater() возвращает MenuInflater текущей activity. В этом объекте вызывается метод inflate(), который загружает ресурс меню в объект Menu.

Создание меню опций

Меню опций – это место, куда обычно включаются основные действия и необходимые навигационные элементы.

Когда на платформе Android создается меню опций в первый раз, в Activity вызывается метод onCreateOptionsMenu(). Для создания пользовательского меню опций необходимо переопределить этот метод, как показано в примере.

Когда пользователь выбирает пункт в меню опций, система вызывает метод onOptionsItemSelected() в Activity. Этому методу передается MenuItem, который пользователь выбрал. Выбранный пункт меню может быть найден с помощью метода getItemId(), который возвращает уникальный ID пункта меню.

Например:

```
public boolean onOptionsItemSelected(MenuItem item)
{
    switch (item.getItemId())
```

```
    {
        case R.id.item1:
        Toast.makeText(this, R.string.menu_item1,
    Toast.LENGTH_SHORT).show();
        return true;
        case R.id.item2:
        Toast.makeText(this, R.string.menu_item2,
    Toast.LENGTH_SHORT).show();
        return true;
    case R.id.item3:
        Toast.makeText(this, R.string.menu_item3,
    Toast.LENGTH_SHORT).show();
        return true;
        default:
        return super.onOptionsItemSelected(item);
    }
}
```

В этом примере метод getItemId() запрашивает идентификатор ID выбранного пункта меню и использует оператор switch, которые заданы в файле example_menu.xml. Если цикл switch case успешно обрабатывает пункт меню, он возвращает значение TRUE для индикации того, что пункт меню был обработан. Иначе значение по умолчанию (default) передает пункт меню суперклассу.

Создание контекстного меню

Контекстное меню подобно меню, возникающего, когда пользователь выполняет правый щелчок мышью. Контекстное меню должно использоваться для обеспечения доступа к действиям, которые принадлежат конкретному компоненту пользовательского интерфейса. В Android контекстное меню вызывается, когда пользователь выполняет «длинное нажатие».

Для того чтобы назначить контекстное меню для View, необходимо зарегистрировать View для контекстного меню с помощью метода registerForContextMenu(). Для задания внешнего вида и поведения контекстного меню нужно переопределить методы onCreateContextMenu() и onContextItemSelected().

Пример метода onCreateContextMenu(), который использует ресурс context_menu.xml:

```
public void onCreateContextMenu(ContextMenu menu,        View   v,
ContextMenuInfo menuInfo)
{
```

```
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.context_menu, menu);
}
```

Параметры метода включают выбранный пользователем View и объект ContextMenu.ContextMenuInfo, который предоставляет дополнительную информацию о выбранном View. Когда пользователь выбирает пункт контекстного меню, система вызывает метод onContextItemSelected().

Пример, как обрабатывать выбранные пункты меню:

```
public boolean onContextItemSelected(MenuItem item)
{
  AdapterContextMenuInfo   info   =   (AdapterContextMenuInfo)
item.getMenuInfo();
  switch (item.getItemId())
  {
  case R.id.edit:
    editNote(info.id);
    return true;
  case R.id.delete:
    deleteNote(info.id);
    return true;
  default:
```



```
      return super.onContextItemSelected(item);
    }
}
```

Структура этого кода подобна примеру для меню опций, в котором getItemId() запрашивает ID выбранного пункта меню,а выражение switch этот ID с идентификаторами ресурса меню. И также, как в примере меню опций, выражение default вызывает суперкласс. Также в этом примере для выполнения действий с выбранным пунктом меню приложение должно знать ID пункта меню в списке (его позицию в ListView). Чтобы получить этот ID, приложение вызывает метод getMenuInfo(), который возвращает объект AdapterView.AdapterContextMenuInfo, включающий ID выбранного пункта меню в поле id.

3.3. Связывание Activities с помощью Intent

Android-приложение может содержать ноль и более activities. Когда приложение имеет более чем одно activity, может возникнуть необходимость перехода от одной activity к другой. В Android навигация между activities осуществляется с помощью элемента Intent.

Объект Intent – это пассивная структура данных, хранящая абстрактное описание выполняемой операции. Intent может содержать:

- Component name – имя компонента, который должен обрабатывать Intent. Это поле является объектом ComponentName – комбинация полного имени класса и имени пакета, указанного в файле манифеста приложения.
- Action – строка, содержащая выполняемое действие. Класс Intent содержит несколько констант, например Intent.ACTION_MAIN.
- Data – URI данных и тип MIME этих данных. Различным действиям соответсвуют различные типы спецификаций данных.
- Category – строка, содержащая дополнительную информацию о типе компонента, который должен обрабатывать Intent.
- Extras – пара ключ-значение для дополнительной информации, которая должна быть представлена компоненту, обрабатывающему Intent.
- Flags – флаги разных сортов.

Intents могут быть разделены на две группы:

1. Явные Intents указывают целевой компонент по его имени.
2. Неявные Intents не содержат цель.

Android доставляет явный Intent экземпляру указанного целевого класса. Чтобы определить, какой компонент должен получить Intent, нужно только имя компонента.

В отсутствие указанной цели, т.е. в неявном Intent, система Android должна найти лучший компонент (или компоненты) для обработки Intent. Она делает это сравнением содержимого объекта Intent с фильтрами Intent, структур, ассоциированных с компонентами, которые потенциально могут получать Intents.

3.3.1. Фильтры Intent

Для информирования системы, какие явные Intents они могут обработать, activities, сервисы и широковещательные опросы могут иметь один или более фильтров Intent. Каждый фильтр описывает набор интентов, которые он хочет получать. Он пропускает интенты нужного типа. Явный интент всегда доставляется к своей цели независимо от того, что он содержит. Но неявный интент доставляется компоненту,

только если он может пройти через один из фильтров компонента.

Фильтр интентов – это экземпляр класса IntentFilter. Однако, так как система Android должна знать о возможностях компонента до того, как она может запустить компонент, фильтры интентов обычно настраиваются не в Java-коде, а в файле манифеста приложения (AndroidManifest.xml) как элементы intent-filter.

Фильтр имеет поля, которые соответствуют полям action, data и category объекта Intent. Неявный интент, чтобы попасть к компоненту, должен пройти все три теста. Но так как компонент может иметь несколько фильтров интентов, то интент, не прошедший один фильтр, может пройти через другой.

Рассмотрим более подробно каждый из трех тестов:

Фильтр Action. Элемент intent-filter в файле манифеста• содержит действия в виде элементов action.

Например:

```
<intent-filter . . . >
   <action   android:name="com.example.project.SHOW_CURRENT"
/>
   <action    android:name="com.example.project.SHOW_RECENT"
/>
   <action   android:name="com.example.project.SHOW_PENDING"
/>
   . . .
</intent-filter>
```


Чтобы пройти этот тест, действие в объекте Intent должно соответствовать одному из действий в фильтре.

Фильтр Category. Элемент• <intent-filter> также содержит категории:

```
<intent-filter . . . >
   <category android:name="android.intent.category.DEFAULT"
/>
   <category
android:name="android.intent.category.BROWSABLE" />
   . . .
</intent-filter>
```

Константы, описаные ранее для действий и категорий, не используются в файле манифеста. Вместо этого используются полные строковые значения. Например, строка "android.intent.category.BROWSABLE" соответствует константе CATEGORY_BROWSABLE.

Чтобы интент мог пройти тест категорий, каждой категории в объекте Intent должна соответствовать категория в фильтре.

Фильтр Data. Подобно действиям и категориям, данные в•

фильтре интентов специфицируются как элементы.

Например:

```
<intent-filter . . . >
   <data android:mimeType="video/mpeg"
android:scheme="http" . . . />
   <data android:mimeType="audio/mpeg"
android:scheme="http" . . . />
   . . .
</intent-filter>
```

Каждый элемент data может указывать URI и тип данных. Существует несколько атрибутов – scheme, host, port и path – для каждой части URI: scheme://host:port/path

Когда URI в объекте Intent сравнивается с URI в фильтре, сравнение происходит только по частям URI, действительно упоминаемым в фильтре. Например, если фильтр указывает только схему, все URI с такой схемой проходят фильтр.

Атрибут type элемента data задает тип MIME данных. Этот атрибут используется чаще, чем URI. И объект Intent, и фильтр могут использовать символ «*» для подтипов. Например, «text/*» или «audio/*».

3.3.2. Запуск и завершение Activity

Другая activity может быть запущена вызовом startActivity() с аргументом Intent, который описывает запускаемую activity.

Обычно в приложениях существует необходимость запуска известной activity. Этого можно достичь созданием интента, который явно описывает activity, используя имя класса. Например, есть activity с именем SignInActivity:

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

Если в приложении нужно выполнить действие, такое как отправка почты, сообщения или обновление статуса, используя свои данные, то, вместо того, чтобы создавать свои activity, можно использовать activity, предоставляемые другими приложениями.

Например, если нужно дать возможность пользователю отправить почтовое сообщение, то можно создать следующий интент:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
```

startActivity(intent);

Иногда нужно получить результат из activity. В этом случае нужно использовать метод startActivityForResult(), а затем получить результат, реализовав onActivityResult(). Когда созданная activity заканчивает выполняться, она возвращает завернутый в Intent результат методу onActivityResult().

Закрыть activity можно методом finish(). Также уже запущенную activity можно закрыть с помощью finishActivity().

3.3.3. Пример использования Intent

Пусть имеется список, состоящий из интернет адресов. Каждый элемент списка содержит две строки: подпись и адрес. Элемент ListView отображает подписи. Обычный клик открывает браузер с соответствующим адресом. Контекстное меню для каждого пункта меню содержит две команды: open и edit. Команда open аналогична простому клику, а edit открывает новую activity, в которой можно отредактировать подпись и адрес выбранного элемента списка.

Листинг 3.6: Пример использования Intent.

Файл манифеста AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="learn.android.lab2"
```



```
    android:versionCode="1"
    android:versionName="1.0">
  <uses-sdk android:minSdkVersion="7" />

  <application android:icon="@drawable/icon" andro-
id:label="@string/app_name">
      <activity android:name=".AddressBookActivity"
            android:label="@string/app_name">
        <intent-filter>
            <action andro-
id:name="android.intent.action.MAIN" />
            <category andro-
id:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
      <activity android:name="EditItemActivity" andro-
```

```
id:label="Edit">
          <intent-filter>
              <action andro-
id:name="learn.android.lab2.EDIT_ITEM"></action>
              <category andro-
id:name="android.intent.category.DEFAULT"></category>
          </intent-filter>
       </activity>
   </application>
</manifest>
```

Здесь добавлен activity с именем EditItemActivity для редактирования элемента списка.

Файл AddressBookActivity.java расширяет ListActivity и является списком адресов.

```
package learn.android.lab2;
import java.util.HashMap;
import java.util.Map;
import android.app.ListActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.ContextMenu.ContextMenuInfo;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView.AdapterContextMenuInfo;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class AddressBookActivity extends ListActivity {
```



```
        public final static int SUCCESS_CODE = 100;
        public final static String LABEL_KEY = "label";
    public final static String URL_KEY = "key";
        Map<String, String> mListData;
    String[] mListItems;
        TextView mSelectedListItem = null;
       public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
        mListData = new HashMap<String, String>();
```

```
        mListData.put("Google search",
"http://google.com");
        mListData.put("Google maps",
"http://maps.google.com");
        mListData.put("Google news",
"http://news.google.com");
        mListData.put("Google translate",
"http://translate.google.com");
       mListItems = mListData.keySet().toArray(new
String[0]);
       setListAdapter(new ArrayAdapter<String>(
         this, android.R.layout.simple_list_item_1, mListI-
tems)
    );
       registerForContextMenu(getListView());
   }
    public void onCreateContextMenu(ContextMenu menu, View
v,
                                    ContextMenuInfo me-
nuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
   }
   public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    mSelectedListItem = (TextView) info.targetView;
        String menuItemLabel = mSelectedListI-
tem.getText().toString();

    switch (item.getItemId()) {
    case R.id.open:
        openURL(menuItemLabel);
        return true;
    case R.id.edit:
        Intent i = new In-
tent("learn.android.lab2.EDIT_ITEM");
        Bundle extras = new Bundle();
```



```
        extras.putString(LABEL_KEY, menuItemLabel);
```

```
        extras.putString(URL_KEY, mListDa-
ta.get(menuItemLabel));
        i.putExtras(extras);
        startActivityForResult(i, SUCCESS_CODE);
        return true;
        default:
            return super.onContextItemSelected(item);
    }
   }
   public void onListItemClick(ListView parent, View v, int
position, long id) {
    openURL(mListItems[position]);
   }
   protected void onActivityResult(int requestCode, int re-
sultCode, Intent data) {
    if (requestCode == SUCCESS_CODE && resultCode ==
RESULT_OK) {
        Bundle bundle = data.getExtras();
        if (bundle != null) {
            String label = bundle.getString(LABEL_KEY);
            String url = bundle.getString(URL_KEY);

            TextView textView = (TextView) mSelectedLis-
tItem;
            textView.setText(label);
            mListData.remove(label);
            mListData.put(label, url);
        }
    }
   }
   private void openURL(String menuLabel) {
    String url = mListData.get(menuLabel);
    Intent i = new In-
tent(android.content.Intent.ACTION_VIEW, Uri.parse(url));
    startActivity(i);
   }
}
```

После выполнения AddressBookActivity выглядит следующим об-
разом (рис. 3.14).

Рис. 3.14. Результат использования «AddressBookActivity»

В методе onCreate() создаются данные для списка mListData и mListItems. Затем эти данные с помощью ArrayAdapter делаются источником данных для списка. В заключение список регистрируется для контекстного меню.

Приватный метод openURL() создает Intent для просмотра web-страницы по ее URL и запускает новую activity с этим интентом. Обработчик событий onListItemClick() только лишь вызывает этот метод.

Метод onCreateContextMenu() вызывается при создании контекстного меню. В нем создается меню с помощью файла menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu

xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/open" android:title="Open"></item>
  <item android:id="@+id/edit" android:title="Edit"></item>
</menu>
```

Контекстное меню выглядит следующим образом (рис. 3.15).

Рис. 3.15. Пример использования контекстного меню

Метод onContextItemSelected вызывается при активации пункта контекстного меню. Метод сначала вычисляет переменную mSelectedListItem, которая является ссылкой на активируемый TextView. Затем, если выполнялась команда open, вызывается метод openURL(). Если выполняется команда edit, то создается Intent, с помощью класса Bundle в нем сохраняются подпись и адрес текущего пункта меню. Затем интент запускается с помощью метода startActivityForResult().

Действию с именем learn.android.lab2.EDIT_ITEM соответствует activity EditItemActivity.

Файл компоновки для этого activity выглядит следующим обра-

зом:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
    <TableRow>
     <TextView
         android:text="Label:"
         android:layout_weight="0.3"
```



```
         android:gravity="right"
         android:layout_marginRight="10px"
         />
     <EditText
         android:id="@+id/label"
         android:layout_weight="0.7"
         />
    </TableRow>
    <TableRow>
     <TextView
         android:text="URL:"
         android:layout_weight="0.3"
         android:gravity="right"
         android:layout_marginRight="10px"
         />
     <EditText
         android:id="@+id/url"
         android:layout_weight="0.7"
         />
    </TableRow>
    <TableRow>
     <Button
         android:id="@+id/btn_ok"
         android:text="Ok"
         android:layout_weight="0.5"
         android:layout_margin="20px"
         />
     <Button
         android:id="@+id/btn_cancel"
         android:text="Cancel"
```

```
        android:layout_weight="0.5"
        android:layout_margin="20px"
        />
   </TableRow>
</TableLayout>
```

Файл EditItemActivity.java выглядит следующим образом:

```
package learn.android.lab2;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class EditItemActivity extends Activity
{
    private EditText labelField;
```



```
   private EditText urlField;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.edit_item);

      labelField = (EditText) findViewById(R.id.label);
      urlField = (EditText) findViewById(R.id.url);

      Bundle extras = getIntent().getExtras();
      String label = ex-
tras.getString(AddressBookActivity.LABEL_KEY);
      String url = ex-
tras.getString(AddressBookActivity.URL_KEY);

      labelField.setText(label);
      urlField.setText(url);

      Button okBtn = (Button) findViewById(R.id.btn_ok);
      okBtn.setOnClickListener(new View.OnClickListener()
{
       public void onClick(View v) {
          Intent data = new Intent();
```

```
            Bundle bundle = new Bundle();

            bundle.putString(
                AddressBookActivity.LABEL_KEY,
                labelField.getText().toString()
            );
            bundle.putString(
                AddressBookActivity.URL_KEY,
                urlField.getText().toString()
            );

            data.putExtras(bundle);
            setResult(RESULT_OK, data);
            finish();
        }
        });

        Button cancelBtn = (Button) findViewBy-
Id(R.id.btn_cancel);
        cancelBtn.setOnClickListener(new
View.OnClickListener() {
         public void onClick(View v) {
            setResult(RESULT_CANCELED);
            finish();
         }

                          169


         });
    }

}
```

Внешний вид EditItemActivity представлен на рисунке 3.16.

Рис. 3.16. Пример использования «EditItemActivity»

Вся логика этого activity расположена в методе onCreate(). Сначала из интента извлекаются подпись и адрес пункта меню. Затем эти значения устанавливаются методом setText() для элементов EditView. Обработчик событий onClick для кнопки Cancel просто

возвращает результат RESULT_CANCELED и заканчивает текущую activity. В обработчике кнопки Ok создается новый интент, в нем сохраняются данные из элементов EditView, результат передается вызвавшему процессу, и activity завершается.

В классе AddressBookActivity, в методе onActivityResult() обрабатывается результат из EditItemActivity. Здесь из интента снова извлекаются данные, обновляется список и структуры данных.

Вопросы для самопроверки

1. Какие инструменты необходимо установить, чтобы проектировать мобильные приложения для платформы Андроид?
2. Что такое эмулятор мобильного приложения, почему и зачем он используется?
3. Из каких компонентов состоит Андроид-приложение?
4. Что такое управляющий файл в приложении Андроид?
5. Какие объекты могут использоваться для разработки пользовательского интерфейса?

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Mikkonen T. Programming mobile devices an introduction for practitioners. – London: John Wiley & Sons Ltd., 2007. – 245 p.
2. Paavilainen J. Mobile business strategies – understanding the technologies and opportunities. – London: IT Press, 2002. – 257 p.
3. Lee V., Schneider H., Schell R. Mobile Applications: architecture, design, and development. – Prentice Hall, 2004. – 368 p.
4. Fling B. Mobile design and development: practical concepts and techniques for creating mobile sites and web apps. – O'Reilly Media, 2009. – 336 p.
5. Verbraeck A. Designing mobile service systems. – Amsterdam: IOS Press, 2007. – 249 p.
6. Zheng P., Lionel N. Smart Phone and next-generation mobile computing. – Morgan Kaufmann, 2005. – 350 p.

7. Горнаков С.Г. Программирование мобильных телефонов на Java 2 Micro Edition. – М.: ДМК Пресс, 2004. – 336 с.
8. Пирумян В. Платформа программирования J2ME для портативных устройств. – М.: КУДИЦ-Образ, 2002. – 352 с.
9. Topley K. J2ME in a nutshell. – O'Reilly, 2002. – 478 p.
10. Yuan M. J. Enterprise J2ME: Developing Mobile Java Applications. – Prentice Hall PTR, 2003. – 480 p.
11. Голощапов А.Л. Google Android: программирование для мобильных устройств. – СПб.: БХВ-Петербург, 2011. – 448 с.
12. Friesen J. Learn Java for Android development. – Apress, 2010. – 656 p.
13. Jackson W. Android apps for absolute beginners. – Apress, 2011. – 344 p.
14. Burnette E. Hello, Android: introducing Google's mobile development platform. – Pragmatic Bookshelf, 2010. – 300 p.
15. Ableson W. F., Sen R., King C. Android in action. – Manning Publications, 2011. – 592 p.
16. Rogers R., Lombardo J., Mednieks Z., G. Blake Meike. Android application development: programming with the Google SDK. – O'Reilly Media, 2009. – 336 p.
17. Murphy M. L. Android programming tutorials. – CommonsWare, 2011. – 334 p.
18. Meier R. Professional Android 2 application development. – Wrox, 2010. – 576 p.
19. Sayed Y. Hashimi. Pro Android 2. – Apress, 2010. – 500 p.

20. Conder S., Darcey L. Android wireless application development. – Addison-Wesley Professional, 2009. – 600 p.
21. To N., Steele J. The Android developer's cookbook: building applications with the Android SDK (Developer's Library). – Addison-Wesley Professional, 2010. – 400 p.
22. DiMarzio J. F. Android: a programmer's guide. – McGraw-Hill Osborne Media, 2008. – 400 p.
23. Komatineni S., MacLean D., Hashimi S. Pro Android 3. – Apress, 2011. – 1200 p.

Учебное издание

СОКОЛОВА Вероника Валерьевна

РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Учебное пособие

Научный редактор, д.т.н. В.А. Силич

Редактор Л.А Егорова

Верстка В.В. Соколова

Подписано к печати Формат 60×84/16.
Бумага «Снегурочка». Печать Xerox.
Усл. печ. л. 10,17. Уч.-изд. л. 9,21.
Заказ . Тираж экз.
Национальный исследовательский
Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту BS EN ISO 9001:2008
. 634050, г. Томск, пр. Ленина, 30.
Тел./факс: 8(3822)56-35-35, www.tpu.ru