Лабораторная работа №2
Разработка приложений с несколькими Activity.

Передача данных между Activity

Цель: формирование у студентов знаний о жизненном цикле Activity, навыков создания и вызова нового Activity, передачи данных между Activity, хранения данных с помощью класса SharedPreferences, создания всплывающих сообщений и логирования.

План занятия

1. Изучить теоретические сведения.

2. Выполнить практическое задание по лабораторной работе.

3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Работа мобильного приложения может сопровождаться возникновением всплывающих сообщений, диалоговых окон, работой с меню и т.д. Рассмотрим реализацию простейшего вспомогательного элемента – всплывающего сообщения.

Всплывающие сообщения

Приложение может показывать всплывающие сообщения с помощью класса Toast:

Toast.makeText(context, text , duration).show();

Статический метод makeText создает View-элемент Toast.

Далее рассмотрим параметры метода.

context – объект, который предоставляет доступ к базовым функциям приложения, т.к.

Activity является подклассом Context, то в качестве объекта от Context используем текущее Activity, т.е. this.

text – текст, который надо вывести.

duration – продолжительность показа (Toast.LENGTH_LONG – длинная (3,5 сек),

Toast.LENGTH_SHORT – короткая (2 сек)).

Чтобы Toast отобразился на экране, вызывается метод show(). Пример реализации (рисунок 2.1):

Toast.makeText (this, "Show Text Views!", Toast.LENGTH_SHORT).show();

а б

Рисунок 2.1 – Пример отображения всплывающих сообщений: а – вывод сообщения "Show Text Views!" внизу окна после нажатия на кнопку Show и его размещаемого по умолчанию; б – вывод сообщения "Hide Text Views!" в центре окна после нажатия на кнопку Hide

По умолчанию всплывающее сообщение отображается в нижней части рабочего окна по центру. Для изменения места расположения всплывающего сообщения используется метод setGravity:

```
Toast toast = Toast.makeText(this, "Hide Text Views!", Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();
```

Логирование.

При тестировании работы приложения можно отслеживать логи в окне logcat (рисунок 2.2). Логи имеют разные уровни важности: error, warn, info, debug, verbose (по убыванию). Установка Log level в окне logcat приводит к фильтрации логов указанного уровня, а также уровней более высокой важности. Имеется возможность создавать, редактировать и удалять свои фильтры. Логирование является одним из инструментов разработчика на этапе тестирования приложения. Для этого программно создаются логи, по их выводу в окне logcat отслеживается логика работы приложения, а перед выпуском финальной версии созданные логи из программы удаляются.

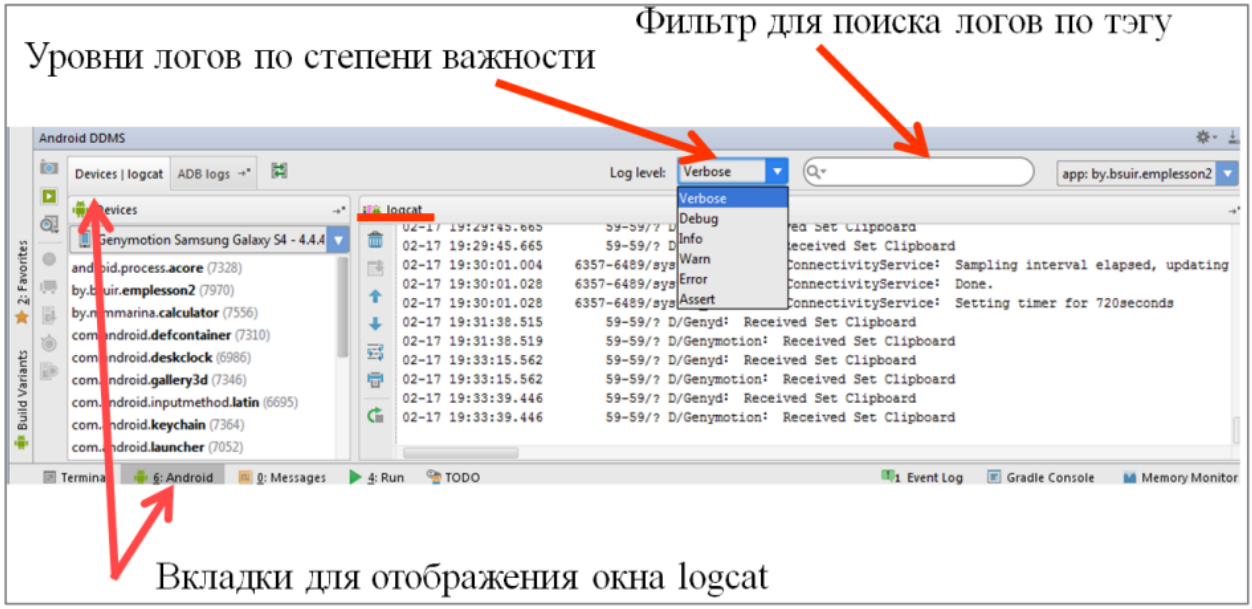

Рисунок 2.2 – Работа с окном logcat

Программно логирование выполняется с помощью класса Log и его методов Log.v() Log.d() Log.i() Log.w() and Log.e(). Названия методов соответствуют уровню логов.

Пример синтаксиса методов:

Log.d(tag, text);

Методы требуют на вход тэг и текст сообщения. Тэг – это метка для реализации возможности поиска конкретного лога путем создания собственного фильтра.

Пример реализации логирования в коде:

```
private static final String TAG = "myLogs";
public void onClick(View v) {
  switch (v.getId()) {
        case R.id.btnShow: Log.d(TAG, "Нажата кнопка Show");
        break;
        case R.id.btnHide: Log.d(TAG, "Нажата кнопка Hide");
        break;
  }
}
```

Жизненный цикл Activity.

Созданное при работе приложения Activity может быть в одном из трех состояний:

1. Resumed: Activity видно на экране, оно находится в фокусе, пользователь может с ним взаимодействовать. Это состояние также иногда называют Running.

2. Paused: Activity не в фокусе, пользователь не может с ним взаимодействовать, но его видно (оно перекрыто другим Activity, которое занимает не весь экран или полупрозрачно).

3. Stopped: Activity не видно (полностью перекрывается другим Activity), соответственно оно не в фокусе и пользователь не может с ним взаимодействовать. Когда Activity переходит из одного состояния в другое, система автоматически вызывает соответствующие методы, в которые можно добавлять свой код. Методы Activity, которые вызывает система (рисунок 2.3):

– onCreate() – вызывается при первом создании Activity.

– onStart() – вызывается перед тем, как Activity будет видно пользователю.

– onResume() – вызывается перед тем как будет доступно для активности пользователя (взаимодействие).

– onPause() – вызывается перед тем, как будет показано другое Activity.

– onStop() – вызывается когда Activity становится не видно пользователю.

– onDestroy() – вызывается перед тем, как Activity будет уничтожено.

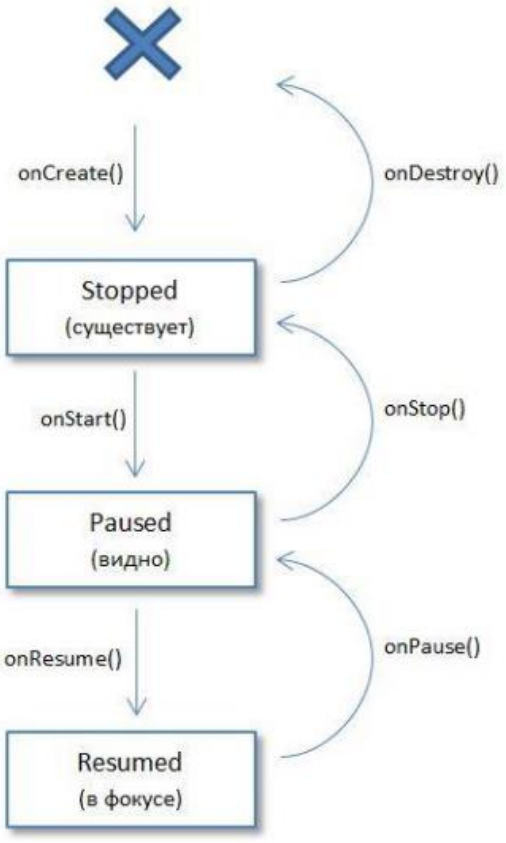

Рисунок 2.3 – Жизненный цикл Activity

Обратите внимание на то, что сами методы НЕ вызывают смену состояния. Наоборот, смена состояния Activity является триггером, который вызывает эти методы. Тем самым мы можем реагировать на произошедшее событие необходимым образом.

Для того, чтобы в ответ на происходящие изменения состояния Activity отработал требуемый код, необходимо переопределить вышеописанные методы жизненного цикла Activity (вызов и вставка в код переопределяемого метода производится с помощью комбинации клавиш Ctrl + O).

Далее приведен фрагмент кода, переопределяющего методы onStart() и onResume() путем добавления логирования.

```
@Override
protected void onStart() {
    super.onStart(); // метод родительского класса (его нельзя удалять!)
    Log.d(TAG, "MainActivity: onStart"); // добавляем логи
}
@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "MainActivity: onResume"); // добавляем логи
}
```

Аналогичным образом можно переопределить все 6 методов, включая onCreate(). Следует отметить, что добавочный код следует располагать только после вызова родительского метода (например, super.onStart() для onStart()).

Создание нового Activity.

Мобильное приложение, как правило, содержит несколько Activity и реализует переход между ними с передачей данных при необходимости. Создать новое Activity можно вручную или воспользоваться встроенной функцией добавления нового компонента – Activity – в существующий проект.

Далее рассмотрим оба способа. Алгоритм создания нового Activity вручную включает следующие шаги:

1.Создаем новый xml файл разметки (например, activity_second.xml).

2.Создаем новый java-класс второго Activity (например, SecondActivity).

3.Прописываем логику SecondActivity.java (наследуемся от суперкласса, подключаем графическую разметку):

```
public class SecondActivity extends ActionBarActivity{
    @Override
     protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_second);
     }
}
```

4.Регистрируем новое Activity в манифест-файле (это обязательно):

```
<activity
        android:name=".MainActivity"
        android:label="@string/app_name">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>

<activity
        android:name=".SecondActivity"
        android:label="@string/app_name">
</activity>
```

Программный способ добавления нового компонента в Androidприложение реализуется с помощью меню File → New или путем нажатия комбинации клавиш Alt + Insert. Далее необходимо заполнить форму кастомизации нового Activity с указанием имен java-файла, xml-файла по аналогии с созданием Activity в новом проекте (см. рисунок 1.4). Запись в файле манифеста будет создана автоматически.

Явный вызов нового Activity.

Следующий шаг после создания нового Activity – научиться вызывать это Activity (например, в ответ на нажатие на одноименную кнопку в исходном Activity) (рисунок 2.4).

Для того чтобы из одного Activity вызвать другое, существует два способа: явный и неявный вызов. Далее рассмотрим оба способа.

Реализуем явный вызов нового Activity из основного:

```
Intent intent = new Intent(this, SecondActivity.class);
startActivity(intent);
```

Намерение (Intent) – это механизм для описания одной операции (выбрать фотографию, отправить письмо, сделать звонок, запустить браузер и перейти по

указанному адресу). В Android-приложениях многие операции работают через намерения. Наиболее распространенный сценарий использования намерения - запуск другой активности в своём приложении.

С помощью класса Intent явно указывают, какое Activity необходимо отобразить (это обычно используется внутри одного приложения):

Intent (Context packageContext, Class cls)

Context – это объект, который предоставляет доступ к базовым функциям приложения, таким как: доступ к ресурсам, к файловой системе, вызов Activiy и т.д. Activity является подклассом Context, поэтому можно использовать ее – this.

Class – имя класса, указанное в манифест-файле. Так как при создании записи Activity в манифест-файле было указано имя класса, то это же имя следует указать во втором параметре Intent. Далее просмотрев манифест-файл система обнаружит соответствие и покажет соответствующее Activity.

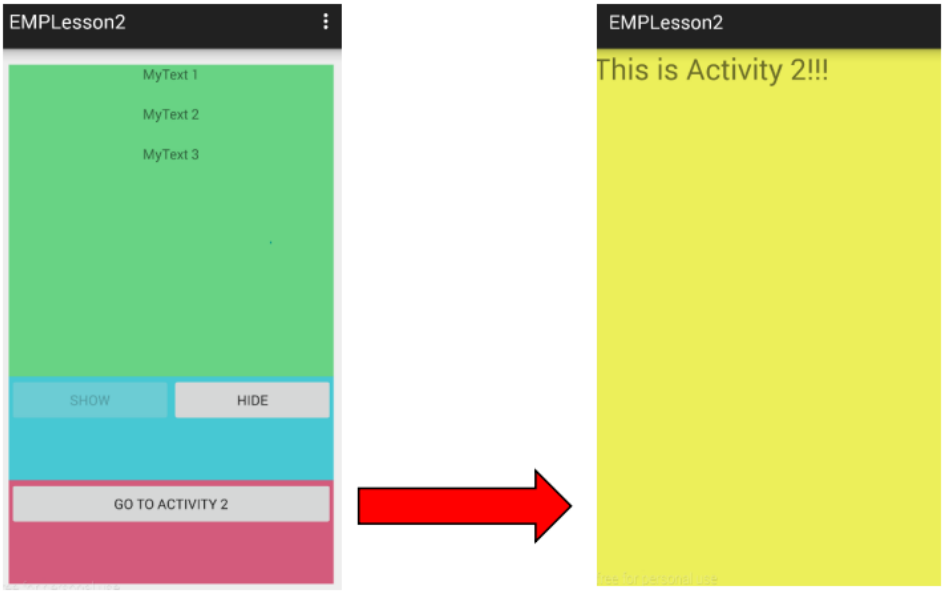

Рисунок 2.4 – Пример вызова нового Activity в ответ на нажатие на одноименную кнопку в исходном Activity

Неявный вызов нового Activity. В отличие от явного вызова, основанного на поиске нового Activity по имени класса в манифест-файле, неявный вызов работает с настройками Intent Filter искомого Activity в манифест-файле.

Intent Filter включает ряд параметров (action, data, category), комбинация которых определяет желаемую цель (отправка письма, открытие гиперссылки, редактирование текста, просмотр картинки, звонок по определенному номеру и т.д.).

Для настройки этих параметров в манифест-файле для нового Activity добавляем Intent Filter:

<activity

android:name=".SecondActivity"

android:label="@string/app_name">

```
android:name=".SecondActivity"
android:label="@string/app_name">

<intent-filter>
  <action android:name="android.intent.action.SecondActivity" />
  <category android:name="android.intent.category.DEFAULT" />
</intent-filter>

</activity>
```

Тогда неявно вызвать SecondActivity из MainActivity можно следующим образом:

intent = new Intent("android.intent.action.SecondActivity");

startActivity(intent);

Параметры объекта Intent из вышеприведенного кода совпадают с условиями фильтра в манифест-файле, и SecondActivity будет вызвано.

Следует отметить, что в случае с неявным вызовом поиск идет уже по всем Activity всех приложений в системе. Если таковых находится несколько, то система предоставляет выбор, какой именно программой необходимо воспользоваться.

Передача данных между Activity.

Передача данных между Activity осуществляется с помощью класса Intent. Сохранение данных в Intent реализуется с помощью метода putExtra(), а извлечение – getExtra().

Метод putExtra: добавляет к объекту пару: первый параметр – это ключ(имя), второй - значение.

Пример кода из MainActivity.java:

Intent intent = new Intent(this, SecondActivity.class);

intent.putExtra("Name", et_Name.getText().toString()); in

tent.putExtra("Email", et_Email.getText().toString());

startActivity(intent);

Метод getExtra: извлекает значение по ключу. Пример кода из SecondActivity.java:

rotected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

```
        setContentView(R.layout.activity_second);
        initViews();
        Intent intent = getIntent();
        tv_Name.setText(intent.getStringExtra("Name"));
        tv_Email.setText(intent.getStringExtra("Email"));
    }
```

Поместить в Intent можно данные не только типа String. В списке методов Intent:

http://developer.android.com/reference/android/content/Intent.html#pubmethods

можно посмотреть все многообразие типов, которые умеет принимать на вход метод putExtra.

Чтобы сохранить пользовательский тип данных используется метод putSerializable(), а извлечение – getSerializable() соответственно.

Метод startActivityForResult.

Запуск другой Activity не обязательно должен быть односторонним. Можно запустить другое Activity и получить обратно результат. Для этого используется метод startActivityForResult() вместо startActivity() (рисунок 2.5).

Алгоритм действий следующий:

1. Вызвать из исходного Activity новое Activity не с помощью метода startActivity, а с помощью startActivityForResult.

   Синтаксис метода startActivityForResult:

   startActivityForResult(Intent intent, int requestCode);

   requestCode – идентификатор для определения, с какого Activity пришел результат.

2. Реализовать в новом Activity метод setResult для указания, какие данные необходимо вернуть в «родительское» Activity и вызвать finish().
   Синтаксис метода setResult:
   setResult(int resultCode, Intent intent);
   resultCode – код возврата. Определяет успешно прошел вызов или нет. intent адресует данные в «родительское» Activity.
3. В исходном Activity в методе onActivityResult прописать логику обработки результата, который придет из нового Activity.
   Синтаксис метода onActivityResult:

protected void onActivityResult(int requestCode, int resultCode, Intent data);
requestCode – тот же идентификатор, что и в startActivityForResult. По нему определяем, с какого Activity пришел результат.
resultCode – код возврата.
data – Intent, в котором возвращаются данные.

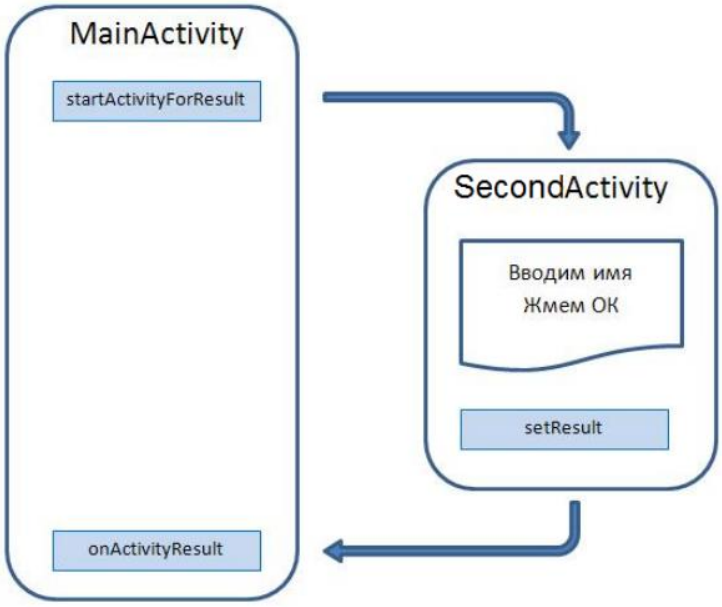

Рисунок 2.5 – Механизм вызова нового Activity и возврат из него с результатом

Например, необходимо из MainActivity вызвать SecondActivity, в SecondActivity ввести имя пользователя и вернуться в MainActivity, чтобы отобразить это имя в текстовом поле.

MainActivity.java:

```
@Override
public void onClick(View v) {
        switch (v.getId()) {
                case R.id.btnGotoActivity2:
                Intent intent = new Intent(this, SecondActivity.class);
                startActivityForResult(intent, 1);
        break;
        }
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
```

```
        if (data == null) {
            return;
        }
        String name = data.getStringExtra("Name"); tv_Name.setText(name);
    }
```

SecondActivity.java:

```
    @Override
    public void onClick(View view) {
        Intent intent = new Intent();
        intent.putExtra("Name", et_ Name.getText().toString());
         setResult(RESULT_OK, intent); finish();
    }
```

Хранение данных в виде пары ключ, значение с помощью класса SharedPreferences.

Если имеется относительно небольшая коллекция пар ключ-значение, которую требуется сохранить, целесообразно использовать SharedPreferences API. Объект SharedPreferences указывает на файл, содержащий пары ключзначение и обеспечивает простые методы для их чтения и записи. Каждый SharedPreferences файл управляется библиотекой и может быть личным или общим. Примером использования SharedPreferences может служить задача сохранения персональных данных пользователя и их восстановления при последующих запусках приложения.

Для реализации задачи хранения данных создадим два пользовательских метода saveText() – сохранение данных и loadText() – загрузка данных. Загрузку будем выполнять при запуске приложения (т.е. в методе onCreate), а сохранение – при закрытии приложения (т.е. в методе onDestroy).

Для сохранения данных сначала с помощью метода getPreferences получаем объект sPref класса SharedPreferences, который позволяет работать с данными (читать и писать):

```
sPref = getPreferences(MODE_PRIVATE);
```

Константа MODE_PRIVATE используется для настройки доступа и означает, что после сохранения, данные будут доступны только этому приложению. Далее, чтобы редактировать данные, необходим объект Editor – получаем его из sPref:

```
 SharedPreferences.Editor editor = sPref.edit();
```

В метод putString указываем наименование переменной – это константа SAVED_TEXT, и значение – содержимое поля myEditText. Чтобы данные сохранились, необходимо выполнить commit. И для наглядности выводим сообщение, что данные сохранены:

```
editor.putString(SAVED_TEXT, myEditText.getText().toString());
editor.commit();
Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
```

Для загрузки данных получаем объект sPref класса SharedPreferences. Чтение осуществляется с помощью метода getString: в параметрах указываем константу - это имя, и значение по умолчанию (пустая строка). Далее пишем значение в поле ввода myEditText и выводим сообщение, что данные считаны:

```
sPref = getPreferences(MODE_PRIVATE);
String savedText = sPref.getString(SAVED_TEXT, "");
myEditText.setText(savedText);
Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
```

Работа приложения продемонстрирована на рисунке 2.6. Полный код приведен ниже:

```
public class MainActivity extends ActionBarActivity {
        private EditText myEditText;
        SharedPreferences sPref;
        final String SAVED_TEXT = "my_saved_text";
  @Override protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView(); loadText();
  }
  private void initView() {
        myEditText = (EditText) findViewById(R.id.myEditText);
  }
  @Override
  protected void onDestroy() {
        super.onDestroy();
```

```
            saveText();
        }
        void saveText() {
            sPref = getPreferences(MODE_PRIVATE);
            SharedPreferences.Editor editor = sPref.edit();
            editor.putString(SAVED_TEXT, myEditText.getText().toString());
            editor.commit();
            Toast.makeText(this, "Text saved", Toast.LENGTH_SHORT).show();
        }
        private void loadText() {
            sPref = getPreferences(MODE_PRIVATE);
            String savedText = sPref.getString(SAVED_TEXT, "");
            myEditText.setText(savedText);
            Toast.makeText(this, "Text loaded", Toast.LENGTH_SHORT).show();
        }
}
```

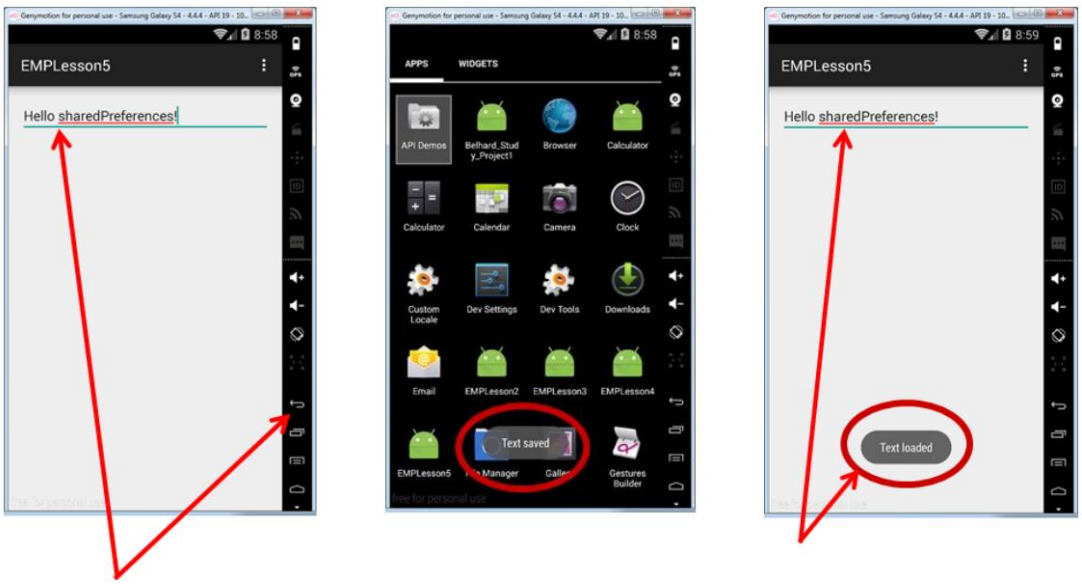

Рисунок 2.6 – Хранение данных в виде пары: имя, значение с использованием класса SharedPreferences:

а – сохраняем данные и выходим из приложения; б – выводится всплывающее сообщение; в – открываем приложение и видим восстановленную информацию

Практическое задание

1. Разработать приложение Taxi, состоящее из трех Activity (рисунок 2.7).

2. В первом Activity создать три редактируемых текстовых поля (EditText) для ввода пользователем регистрационных данных (телефона, имени и фамилии), кнопку Registration для запуска второго Activity.

3. При нажатии на кнопку Registration выполнить явный вызов второго Activity с передачей данных о пользователе (телефон, имя и фамилия).

4. Во втором Activity создать два текстовых поля (TextView) для вывода переданной информации о пользователе (имя+фамилия, телефон), пустое по умолчанию текстовое поле (TextView) для вывода маршрута движения, кнопку Set path для ввода этого маршрута, кнопку вызова такси Call Taxi (недоступна, пока не введен маршрут движения).

5. При нажатии на кнопку Set path выполнить неявный вызов третьего Activity с помощью метода startActivityForResult.

6. В третьем Activity создать шесть редактируемых текстовых полей (EditText) для ввода параметров маршрута движения, кнопку ОК для возврата во второе Activity.

7. При нажатии на кнопку ОК реализовать возврат во второе Activity с передачей в качестве результата параметров маршрута движения.

8. После возврата во второе Activity в текстовое поле вывести информация о маршруте движения и предложение вызвать такси, кнопку вызова такси Call taxi сделать доступной.

9. При нажатии на кнопку Call Taxi вывести всплывающее сообщение об успешной отправке такси.

10. Реализовать сохранение регистрационных данных пользователя в исходном Activity с помощью класса SharedPreferences и восстанавливать эту информацию при повторных запусках приложения. При этом название кнопки Registration должно программно меняеться на Log in.

11. Вывести в лог очередность вызовов методов жизненного цикла перого, второго и третьего Activity.

12. Сделать вывод на основании логов о жизненном цикле Activity.

13. Продемонстрировать работу приложения Taxi на эмуляторе или реальном устройстве.

14. Дополнительное задание, предполагающее самостоятельное углубленное освоение материала: реализовать функционал geo-локации для автоматического определения исходной точки маршрута движения и/или Google карт для обозначения конечной точки маршрута движения пользователя.

Содержание отчета

1. Скриншоты графических представлений первого, второго и третьего Activity в Android Studio, демонстрирующие логику работы приложения Taxi.

2. Код xml-файлов графических представлений первого, второго и третьего Activity приложения Taxi.

3. Код java-файлов первого, второго и третьего Activity приложения Taxi.

4. Результаты вывода в лог очередности вызовов методов жизненного цикла первого, второго и третьего Activity приложения Taxi.

Контрольные вопросы

1. Как с помощью класса Toast создать всплывающее сообщение?

2. В каких случаях необходимо логирование?

3. Что представляет собой окно LogCat? Какие существуют уровни логирования?

4. Как программно реализовать логирование?

5. Как создать новое Activity (опишите работу с java-классом, layoutфайлом, файлом конфигурации AndroidManifest.xml)?

6. Для чего используется контекст приложения Context?

7. Для чего используются объекты класса Intent?

8. Как выполнить явный вызов Activity?

9. Как выполнить неявный вызов Activity?

10. Какие состояния предусмотрены жизненным циклом Activity?

11. Какие методы автоматически срабатывают при смене состояния Activity? Как можно использовать эти методы?

12. Как выполняется передача данных с помощью Intent?

13. В каком виде хранятся данных с помощью класса SharedPreferences?

14. В каких случаях целесообразно хранить данные с помощью класса SharedPreferences?