ЛАБОРАТОРНАЯ РАБОТА №7

Фрагменты. ViewPager. Хранение информации в базе данных SQLite

Цель: формирование у студентов знаний и навыков работы с фрагментами, использования View Pager для перелистывания фрагментов, хранения информации в базе данных SQLite.

План занятия

1. Изучить теоретические сведения.

2. Выполнить практическое задание по лабораторной работе.

3. Оформить отчет и ответить на контрольные вопросы.

Теоретические сведения

Базовые сведения о фрагментах. Устройства с мобильной платформой Android характеризуются многообразием размеров и разрешений экрана, т.е. высокой степенью фрагментации. Если для смартфонов с небольшими экранами реализация Activity выглядит приемлемо, то на больших экранах (например, планшетах) остается незадействованное пространство. В связи с этим начиная с API 11 в Android были добавлены фрагменты, чтобы разработчики могли создавать более гибкие пользовательские интерфейсы на больших экранах (рисунок 1).

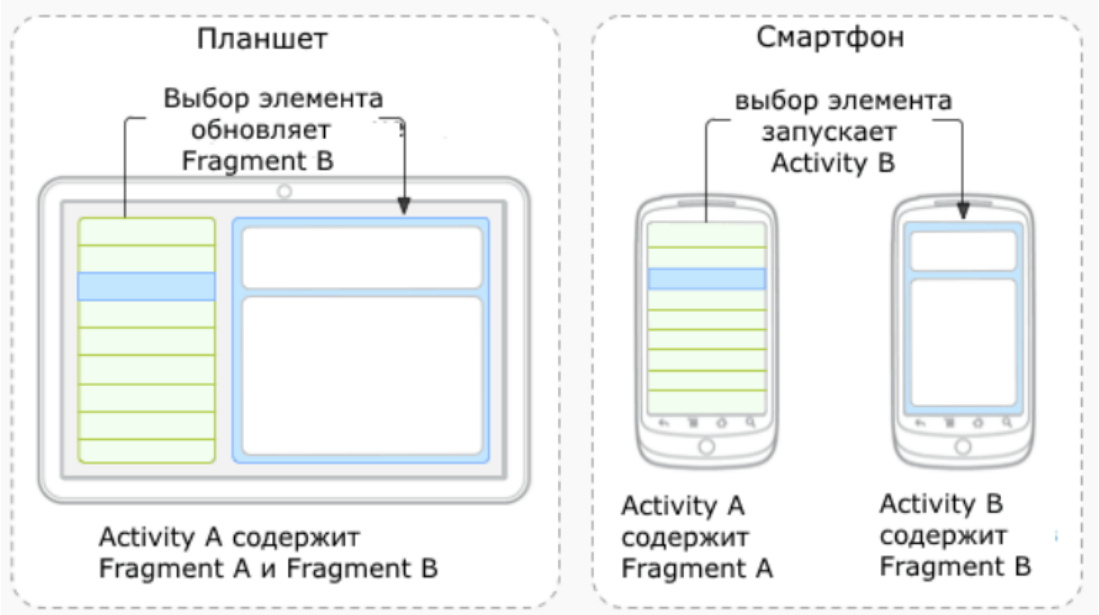


Рисунок 1 – Пример реализации концепции фрагментов на Android-устройствах различных размеров

Фрагмент существует в контексте Activity и имеет свой жизненный цикл, вне Activity он существовать не может. Activity может иметь несколько фрагментов.

Алгоритм создания и подключения фрагмента к Activity следующий:

1. В отдельном xml-файле создаем графическое представление фрагмента (например, fragment1.xml).

2. Создаем в папке java класс Fragment1.java. Класс фрагмента должен наследоваться от класса Fragment:

public class Fragment1 extends **Fragment**

3. Переопределяем метод onCreateView для подключения разметки фрагмента:

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

View view = inflater.inflate(R.layout.**fragment1**, container, false);

return view;

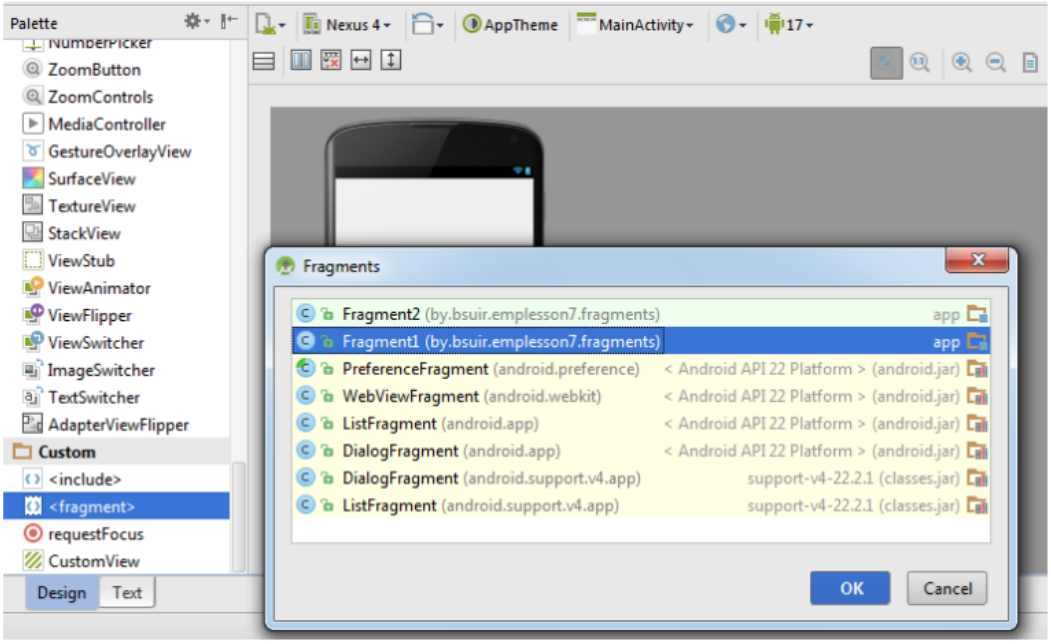
}

При необходимости в методе onCreateView можно находить Viewэлементы фрагмента, устанавливать обработчики для этих элементов и реализовывать логику обработки событий подобно Activity.

4. Добавляем фрагмент в Activity одним из двух способов: статическим или динамическим.

Статическое добавление фрагмента. В activity_main.xml фрагмент добавляется как элемент (рисунок 2). Для каждого фрагмента должно быть установлены высота, ширина, id, а также имя. В качестве имени устанавливается полное имя класса с учетом пакета:

```
<fragment
    android:layout_width=" match_parent "
    android:layout_height="match_parent"
    android:name="by.bsuir.emplesson7.fragments.Fragment1"
    android:id="@+id/fragment1"/>
```



Кроме фрагмента можно добавить в разметку activity_main.xml другие элементы или фрагменты.

Код класса MainActivity остается тем же, что и при обычном создании проекта.

Динамическое добавление фрагмента.

К activity_main.xml добавляем FrameLayout, который впоследствии будет служить контейнером для размещения фрагмента из кода приложения:

```
<FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

Добавляем фрагмент в сформированный контейнер непосредственно в коде приложения:

MainActivity.java:

```
private Fragment1 fragment1;
private FragmentManager fragmentManager;
private FragmentTransaction fragmentTransaction;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main); initView();
```

```
        initFragments();
    }
    private void initFragments() {
        fragment1 = new Fragment1();
        fragmentManager = getFragmentManager();
        fragmentTransaction = fragmentManager.beginTransaction();
        fragmentTransaction.add(R.id.fragment_container, fragment1);
        fragmentTransaction.commit();
    }
```

Класс FragmentManager служит для управления фрагментами. Чтобы получить его, следует вызвать метод getFragmentManager() из кода операции.

Класс FragmentTransaction позволяет выполнять транзакции с фрагментами: add() – добавление фрагмента; remove()– удаление фрагмента; replace()– замена фрагмента; hide()– делает фрагмент невидимым; show() – отображает фрагмент. Пример реализации данных методов для двух фрагментов приведен ниже:

```
frag1 = new Fragment1();
frag2 = new Fragment2();
fTrans = getFragmentManager().beginTransaction();
 fTrans.add(R.id.fragmentContainer, frag1); // добавление фрагмента1
 fTrans.remove(frag1); // удаление фрагмента1
 fTrans.replace(R.id.fragmentContainer, frag2); // размещение фрагмента2
```

Взаимодействие фрагментов и Activity.

Доступ к фрагменту из Activity осуществляется в MainActivity.java посредством метода findFragmentById:

```
// для статических фрагментов: указываем id фрагмента
Fragment frag1 = getFragmentManager().findFragmentById(R.id.fragment1);
TextView tv_frag1 = (TextView) frag1.getView().findViewById(R.id.tv_frag1);
tv_frag1.setText("Hello to Fragment 1 from Activity");
// для динамических фрагментов: указываем id контейнера
```

Fragment frag2 = getFragmentManager().findFragmentById(R.id. fragment_container);

TextView tv_frag2 = (TextView) frag2.getView().findViewById(R.id.tv_frag2);

tv_frag2.setText("Hello to Fragment 2 from Activity");

Доступ к Activity из фрагмента осуществляется в коде класса фрагмента с помощью метода

getActivity: TextView tv_activity = getActivity().findViewById(R.id.tv_activity); tv_activity.setText("Hello from Fragment1");

Обработка в Activity (или в другом фрагменте) события из фрагмента выполняется посредством интерфейса в качестве прослойки для отслеживания событий. Нельзя напрямую связываться из одного фрагмента с другим!

Фрагменты: ViewPager.

Для реализации слайдера и обеспечения эффекта перелистывания страниц в Android-приложениях используется ViewPager. Каждая страница ViewPager представляет собой фрагмент. Имеется возможность формирования верхнего меню вкладок с заголовками страниц. Алгоритм создания ViewPager следующий:

1. В activity_main.xml добавляем ViewPager, при необходимости для формирования верхнего меню вкладок с заголовками страниц дополнительно прописываем PagerTabStrip:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
<android.support.v4.view.ViewPager
  android:id="@+id/pager"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

  <android.support.v4.view.PagerTabStrip
    android:id="@+id/pagerTabStrip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top">
  </android.support.v4.view.PagerTabStrip>

</android.support.v4.view.ViewPager>

</RelativeLayout>
```

```
MainActivity extends ActionBarActivity {
    ViewPager pager;
    PagerAdapter pagerAdapter;
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        pager = (ViewPager) findViewById(R.id.pager);
        pagerAdapter = new MyFragmentPagerAdapter(getSupportFragmentManager());
        pager.setAdapter(pagerAdapter);
    }
}
```

3. Реализуем класс MyFragmentPagerAdapter:

```
public class MyFragmentPagerAdapter extends FragmentPagerAdapter {
    static final int PAGE_COUNT = 3;
    public MyFragmentPagerAdapter(FragmentManager fm) {
        super(fm);
        }
```

```
        @Override
        public Fragment getItem(int i) {
                switch (i){
                case 0: return new Fragment1();
                case 1: return new Fragment2();
                case 2: return new Fragment3();
                default: return null;
        }
}
@Override
public int getCount() {
        return PAGE_COUNT;
} // при необходимости добавляем верхнее меню вкладок с заголовками
@Override
public CharSequence getPageTitle(int i) {
switch (i){
        case 0: return "Frag1";
        case 1: return "Frag2";
```

```
            case 2: return "Frag3";
            default: return null;
            }
        }
}
```

4. Создаем графические разметки и java-код фрагментов. Пример структуры проекта с ViewPager приведен на рисунке 4.

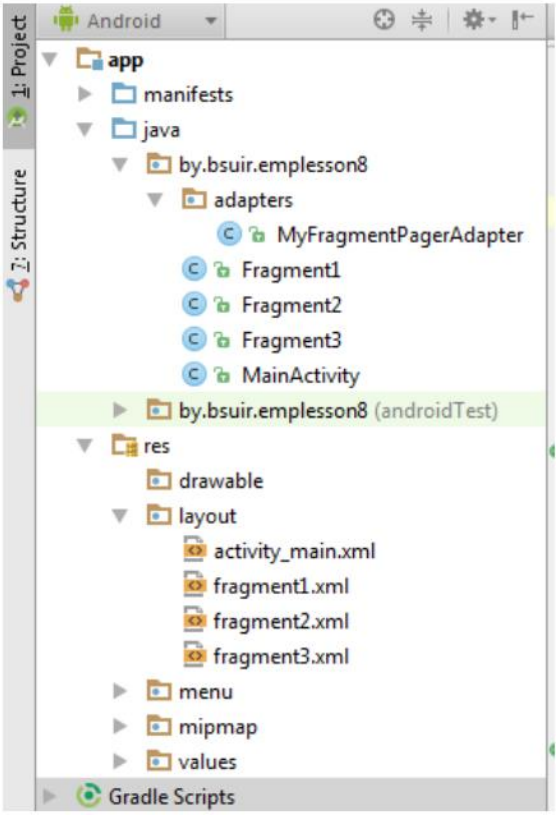


Рисунок 4.4 – Структура проекта с ViewPager Работа с базой данных SQLite.

SQLite доступен на любом Android-устройстве, его не нужно устанавливать отдельно. SQLite поддерживает типы TEXT (аналог String в Java), INTEGER (аналог long в Java) и REAL (аналог double в Java). Остальные типы следует конвертировать, прежде чем сохранять в базе данных. Для хранения изображений в базе данных указывают путь к изображениям, а сами изображения хранят в файловой системе.

Для понимания механизма работы с SQLite в Android-приложениях рассмотрим простейший пример (рисунок 5): создадим таблицу товаров с полями id (номер), name (наименование товара), price (цена), count (количество единиц товара в наличии). Для работы с таблицей товаров реализуем функции добавления (кнопка Add), чтения всех данных таблицы и вывода информации в лог (кнопка Read), очистки всей таблицы (кнопка Clear), ввода name, price, count в качестве новых

данных уже существующего товара с номером id (кнопка Update), удаления товара по id (кнопка Delete).

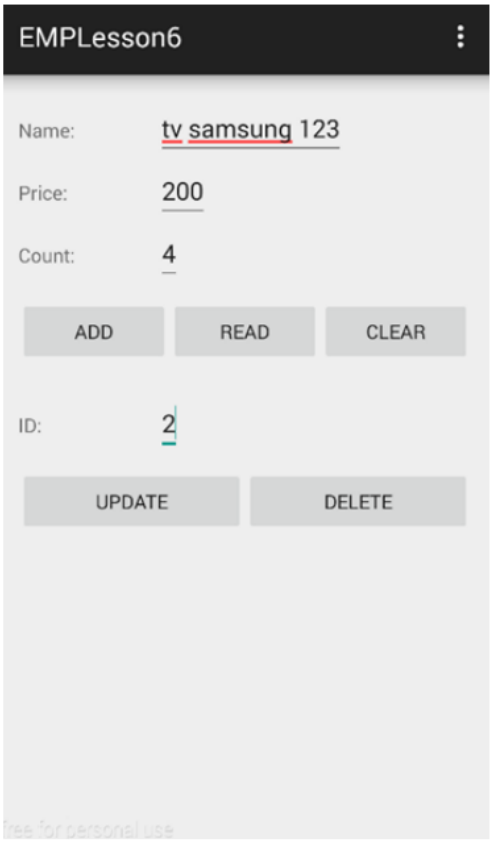


Рисунок 4.5 – Пример работы с SQLite в Android-приложениях: добавление новой позиции в таблицу (Add), чтение всех данных (Read), очистка всех данных (Clear), обновление позиции по номеру (Update), удаление позиции по номеру (Delete)

Работа с базой данных сводится к следующим задачам:

1. Создание и открытие базы данных, создание таблицы. Данные задачи реализуется с помощью класса SQLiteOpenHelper.

2. Создание интерфейса для вставки данных. Класс ContentValues используется для добавления новых строк в таблицу. Каждый объект этого класса представляет собой одну строку таблицы и выглядит как ассоциативный массив с именами столбцов и значениями, которые им соответствуют.

3. Создание интерфейса для выполнения запросов (выборки данных). Запросы к базе данных возвращают объекты класса Cursor. Вместо того чтобы извлекать данные и возвращать копию значений, курсоры ссылаются на результирующий набор исходных данных. Курсоры позволяют управлять текущей позицией (строкой) в результирующем наборе данных, возвращаемом при запросе.

4. Закрытие базы данных. Как правило выполняется в методе onDestroy() java-класса Activity.

С помощью абстрактного класса SQLiteOpenHelper можно создавать, открывать и обновлять базы данных. Это основной класс, с которым необходимо работать в Android-проектах. Класс SQLiteOpenHelper содержит два обязательных абстрактных метода:

1. onCreate(): вызывается при первом создании базы данных.

2. onUpgrade(): вызывается при модификации базы данных (в частности, при попытке подключения к БД более новой версии, чем существующая).

В приложении необходимо создать собственный класс, наследуемый от SQLiteOpenHelper. В этом классе необходимо реализовать указанные обязательные методы, описав в них логику создания и модификации используемой в приложении базы. В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных. Для решения ранее поставленной задачи (см. рисунок 5) создадим класс, являющийся наследником SQLiteOpenHelper и назовем его DBHelper:

```
class DBHelper extends SQLiteOpenHelper {
    private static final String DB_TABLE = "goods";
    public DBHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + DB_TABLE + "(" + "id integer primary key autoincrement," + "name text," + "price integer," + "count integer" + ");");
}
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
// Удаляем старую таблицу и создаём новую
    db.execSQL("DROP TABLE IF IT EXISTS " + DB_TABLE); onCreate(db);
     }
}}
```

Реализуем вспомогательный класс DB для работы с таблицей товаров.

```
public class DB {
        private static final String DB_NAME = "mydb";
        private static final int DB_VERSION = 1;
        private static final String DB_TABLE = "goods";
        private final Context mCtx;
        private DBHelper mDBHelper;
        private SQLiteDatabase mDB;
        public DB(Context ctx) { mCtx = ctx;
} // открыть подключение
public void open() {
        mDBHelper = new DBHelper(mCtx, DB_NAME, null, DB_VERSION);
        mDB = mDBHelper.getWritableDatabase();
} // закрыть подключение
public void close() {
        if (mDBHelper!=null) mDBHelper.close();
} // заполнить таблицу исходными данными при необходимости
public void write() {
        int i=0;
        while (i< 25) {
        i++;
        addRec("My good №" + i, i, i);
} } // получить все данные из таблицы DB_TABLE
public Cursor getAllData() {
return mDB.query(DB_TABLE, null, null, null, null, null, null);
} // добавить запись в DB_TABLE
public void addRec(String name, int price, int count) {
        ContentValues cv = new ContentValues();
        cv.put("name", name);
        cv.put("price", price);
```

```
        cv.put("count", count);
        mDB.insert(DB_TABLE, null, cv);
    } // обновить запись в DB_TABLE
    public void update(int id, String name, int price, int count) {
        ContentValues cv = new ContentValues();
        cv.put("name", name); cv.put("price", price);
        cv.put("count", count);
        mDB.update(DB_TABLE, cv, "id = ?", new String[]{String.valueOf(id)});
    } // удалить запись из DB_TABLE
    public void delRec(long id) {
        mDB.delete(DB_TABLE, "id = " + id, null);
    } // удалить все записи из DB_TABLE
    public void delAll() {
        mDB.delete(DB_TABLE, null, null);
    }
}
```

Следует отметить, что набор методов класса DB в общем случае не является исчерпывающим и может быть изменен/дополнен при необходимости, например, методами выгрузки не всех данных, а удовлетворяющих некоторому набору условий. Для чтения данных используется метод query():

Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder)

В метод query() передают семь параметров. Если какой-то параметр для запроса не имеет значения – указывают null:

table — имя таблицы, к которой передается запрос;

String[] columnNames — список имен возвращаемых полей (массив). При передаче null возвращаются все столбцы.

String whereClause — параметр, формирующий выражение WHERE (исключая сам оператор WHERE). Значение null возвращает все строки. Например: id = 19 and summary = ?

String[] selectionArgs — значения аргументов фильтра. Вы можете включить ? в "whereClause"". Подставляется в запрос из заданного массива;

String[] groupBy - фильтр для группировки, формирующий выражение GROUP BY (исключая сам оператор GROUP BY).

String[] having — фильтр для группировки, формирующий выражение HAVING (исключая сам оператор HAVING). Если не нужен, передается null;

String[] orderBy — параметр, формирующий выражение ORDER BY (исключая сам оператор ORDER BY). При сортировке по умолчанию передается null.

Объект Cursor, возвращаемый методом query(), обеспечивает доступ к набору записей результирующей выборки. Для обработки возвращаемых данных объект Cursor имеет набор методов для чтения каждого типа данных — getString(), getInt() и getFloat().

Примеры поиска по имени, по стоимости:

```
mDB.query.query(DB_TABLE, null, " name = ?", new String[] {
"Samsung Galaxy"}, null, null, null);
mDB.query(DB_TABLE, null, "count = ?", new String[] {
Integer.toString(100)}, null, null, null);
```

Работа класса MainActivity, осуществляющего взаимодействие с базой данных, приведена ниже.

```
public class MainActivity extends ActionBarActivity implements View.OnClickListener{
final String LOG_TAG = "myLogs";
private Context context;
private DB db;
private Cursor cursor;
private int idColIndex;
private int nameColIndex;
private int priceColIndex;
private int countColIndex;
private EditText etName, etPrice, etCount, etId;
```

```
    private Button btnAdd, btnRead, btnClear, btnUpdate, btnDelete; private String name_temp;
    private int id_temp, price_temp, count_temp;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initView(); initDB();
    }
    private void initView() {
        etName = (EditText) findViewById(R.id.etName);
        etPrice = (EditText) findViewById(R.id.etPrice);
        etCount = (EditText) findViewById(R.id.etCount);
        etId = (EditText) findViewById(R.id.etId);
        btnAdd = (Button) findViewById(R.id.btnAdd);
         btnAdd.setOnClickListener(this); btnRead = (Button) findViewById(R.id.btnRead);
        btnRead.setOnClickListener(this); btnClear = (Button) findViewById(R.id.btnClear);
        btnClear.setOnClickListener(this);
        btnUpdate = (Button) findViewById(R.id.btnUpdate);
        btnUpdate.setOnClickListener(this); btnDelete = (Button) findViewById(R.id.btnDelete);
        btnDelete.setOnClickListener(this);
    }
    private void initDB() { // открываем подключение к БД
        db = new DB(this);
        db.open(); //db.delAll(); // если нужно все вернуть к исходному состоянию
        db.write(); // при каждом запуске дописываем данные из write()
    }
```

```
@Override
public void onClick(View view) {
        switch (view.getId()){
         case R.id.btnAdd: name_temp = etName.getText().toString();
                price_temp = Integer.parseInt(etPrice.getText().toString());
                count_temp = Integer.parseInt(etCount.getText().toString());
                db.addRec(name_temp, price_temp, count_temp);
                break;
        case R.id.btnRead: // получаем курсор cursor = db.getAllData();
                // ставим позицию курсора на первую строку выборки
                // если в выборке нет строк, вернется false
                if (cursor.moveToFirst()) {
                // определяем номера столбцов по имени в выборке
                 int idColIndex = cursor.getColumnIndex("id");
                int nameColIndex = cursor.getColumnIndex("name");
                int priceColIndex = cursor.getColumnIndex("price");
                int countColIndex = cursor.getColumnIndex("count");
                do {
                        // получаем значения по номерам столбцов и пишем в лог
                        Log.d(LOG_TAG,
                        "ID = " + cursor.getInt(idColIndex) + ", name = " +
                cursor.getString(nameColIndex) + ", price = " +
                cursor.getInt(priceColIndex) + ", count = " +
                cursor.getInt(countColIndex)); // переход на следующую строку, а
                если следующей нет (текущая - последняя), то выходим из цикла
                } while (cursor.moveToNext());
        } else {
                Log.d(LOG_TAG, "0 rows");
```

```
} Log.d(LOG_TAG, "cursor.getCount()=" + String.valueOf(cursor.getCount())); cursor.close(); break; case R.id.btnClear: db.delAll();

break;

case R.id.btnUpdate: id_temp = Integer.parseInt(etId.getText().toString());

    name_temp = etName.getText().toString();

    price_temp = Integer.parseInt(etPrice.getText().toString());

    count_temp = Integer.parseInt(etCount.getText().toString());

    db.update(id_temp, name_temp, price_temp, count_temp); break;

    case R.id.btnDelete: id_temp = Integer.parseInt(etId.getText().toString());

    db.delRec(id_temp); break;

}

} }
```

Пример предоставления списком ListView данных из SQLite.

Далее приведен пример реализации класса GoodsAdapter для решения задачи приложения MiniShop, ранее реализуемой посредством предоставления адаптеру набора объектов класса Good (см. рисунок 3).

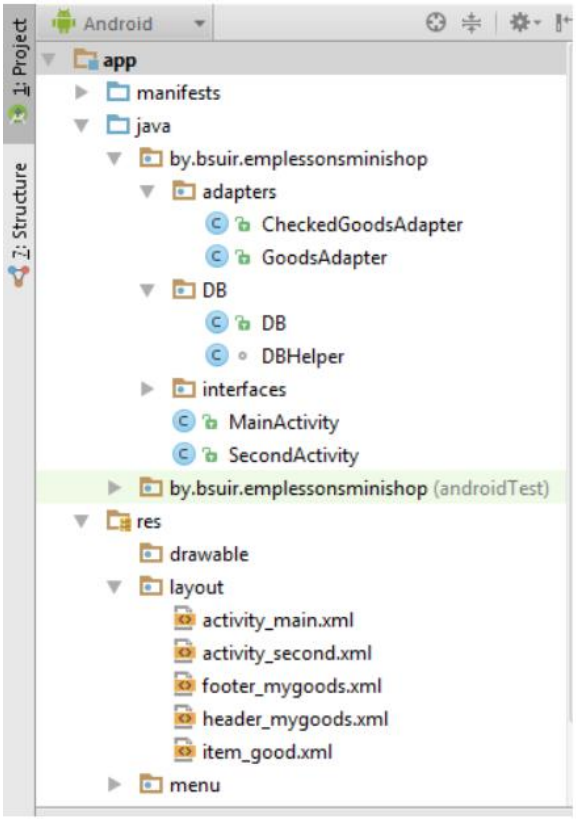


Рисунок 4.6 – Структура проекта MiniShop с использованием базы данных

```
public class GoodsAdapter extends BaseAdapter implements CompoundButton.OnCheckedChangeListener {
    private Context context; private LayoutInflater inflater;
    private DB db; private Cursor cursor;
    private int idColIndex; private int nameColIndex;
    private int priceColIndex;
    private ArrayList arr_is_checked_goods_adapter = new ArrayList();
    private OnChangeListener onChangeListener;
    public GoodsAdapter(Context context, DB db, OnChangeListener onChangeListener) { this.context = context;
    this.inflater = LayoutInflater.from(context);
    this.db = db; this.onChangeListener = onChangeListener;
    cursor = db.getAllData();
    idColIndex = cursor.getColumnIndex("id");
    nameColIndex = cursor.getColumnIndex("name");
    priceColIndex = cursor.getColumnIndex("price");
    for (int i=0; i<cursor.getCount(); i++) {
    arr_is_checked_goods_adapter.add(false);
} }
@Override
public int getCount() {
    return cursor.getCount();
}
@Override
public Cursor getItem(int i) {
    cursor.moveToPosition(i);
    return cursor;
}
```

```
        @Override
        public long getItemId(int i) {
                return i;
        } @Override
        public View getView(int position, View view, ViewGroup parent) {
                if (view == null) {
                        view = inflater.inflate(R.layout.item_good, null);
                } cursor.moveToPosition(position);
        ViewHolder vh = new ViewHolder(); vh.initViewHolder(view);
        vh.tv_goodPrice.setText(cursor.getInt(priceColIndex)+"");
        vh.tv_goodName.setText(cursor.getString(nameColIndex)+"");
        vh.cb_good.setChecked(arr_is_checked_goods_adapter.get(position));
        vh.cb_good.setOnCheckedChangeListener(this);
        vh.cb_good.setTag(position); return view;
}
@Override
public void onCheckedChanged(CompoundButton compoundButton, boolean isChecked)
{
        if (compoundButton.isShown()) {
                int i = (int) compoundButton.getTag(); if (isChecked){
                        arr_is_checked_goods_adapter.set(i, true);
                }else {
                         arr_is_checked_goods_adapter.set(i, false);
                } cursor.moveToPosition(i);
                int id = cursor.getInt(idColIndex);
                String name = cursor.getString(nameColIndex);
                int price = cursor.getInt(priceColIndex);
                int check = 0; if (isChecked){ check = 1; }
                db.update(id, name, price, check);
```

```
                notifyDataSetChanged();
                onChangeListener.onDataChanged();
        }
}
public ArrayList getIsCheckedGoods() {
        return arr_is_checked_goods_adapter;
}
public class ViewHolder {
        private TextView tv_goodPrice;
        private TextView tv_goodName; private CheckBox cb_good;
        public ViewHolder() { }
         public void initViewHolder(View view) {
        tv_goodPrice = (TextView) view.findViewById(R.id.tv_goodPrice);
        tv_goodName = (TextView) view.findViewById(R.id.tv_goodName);
        cb_good = (CheckBox) view.findViewById(R.id.cb_good); }
        }
}
```

Практическое задание

1. Разработать приложение MyNotes представляющее собой View Pager.
2. Поместить в View Pager четыре фрагмента: FragmentShow, FragmentAdd, FragmentDel, FragmentUpdate.
3. В View Pager добавить верхнее меню вкладок (PagerTabStrip) с заголовками Show, Add, Del, Update.
4. Во фрагменте FragmentShow реализовать кастомизированный список заметок ListView с помощью собственного адаптера.
5. В каждом пункте списка отобразить следующую информацию о заметке пользователя: номер, описание заметки.
6. Хранение, а также предоставление информации о заметках адаптеру реализовать с помощью базы данных SQLite.
7. Во фрагменте FragmentAdd реализовать функционал добавления новой заметки посредством ввода описания заметки в поле EditText и добавления информации в базу данных SQLite по нажатию на кнопку Add.

8. Во фрагменте FragmentDel реализовать функционал удаления новой заметки посредством ввода ее номера в поле EditText и удаления информации из базы данных SQLite по нажатию на кнопку Del.
9. Во фрагменте FragmentUpdate реализовать функционал обновления существующей заметки посредством ввода ее номера в поле EditText, ввода нового описания в поле EditText и обновления информации в базе данных SQLite по нажатию на кнопку Update.
10. Предусмотреть обработку исключительной ситуации отсутствия заметки по указанному номеру посредством вывода пользователю всплывающего сообщения соответствующего содержания.
11. Продемонстрировать работу приложения MyNotes на эмуляторе или реальном устройстве.
12. Дополнительное задание, предполагающее самостоятельное углубленное освоение материала: реализовать приложение MiniShop с помощью фрагментов и базы данных SQLite. Предусмотреть различное расположение фрагментов в портретной и альбомной ориентациях (см. рисунок).

Содержание отчета

1. Скриншоты графических представлений фрагментов FragmentShow, FragmentAdd, FragmentDel, FragmentUpdate в Android Studio, демонстрирующие логику работы приложения MyNotes.
2. Код xml-файлов графических представлений, используемых в приложении MyNotes.
3. Код java-файлов приложения MyNotes, включая классы для работы с базой данных, Activity, фрагменты, адаптеры.

Контрольные вопросы

1. Что представляет собой такое фрагмент? Для чего нужны фрагменты?
2. Как статически добавить фрагмент в Activity?
3. Как динамически добавить фрагмент в Activity?
4. Какие динамические операции возможны с фрагментами?
5. Что такое ViewPager?
6. Как создать ViewPager?
7. Для чего используются PagerTitleStrip и PagerTabStrip?
8. Как называется базовый класс для работы с базой данных SQLite в Android?
9. Какие методы обязательны для переопределения при работе с базой данных SQLite?
10. Для чего используется класс ContentValues?
11. Для чего используется класс Cursor?
12. Как реализуются методы insert, query, delete, update для вставки, чтения, удаления и добавления записи в SQLite?