Курс лекций по предмету «Программирование для мобильных приложений»

Кафедра ПМиК

ст. преп. Нечта И.В.

Содержание

Введение

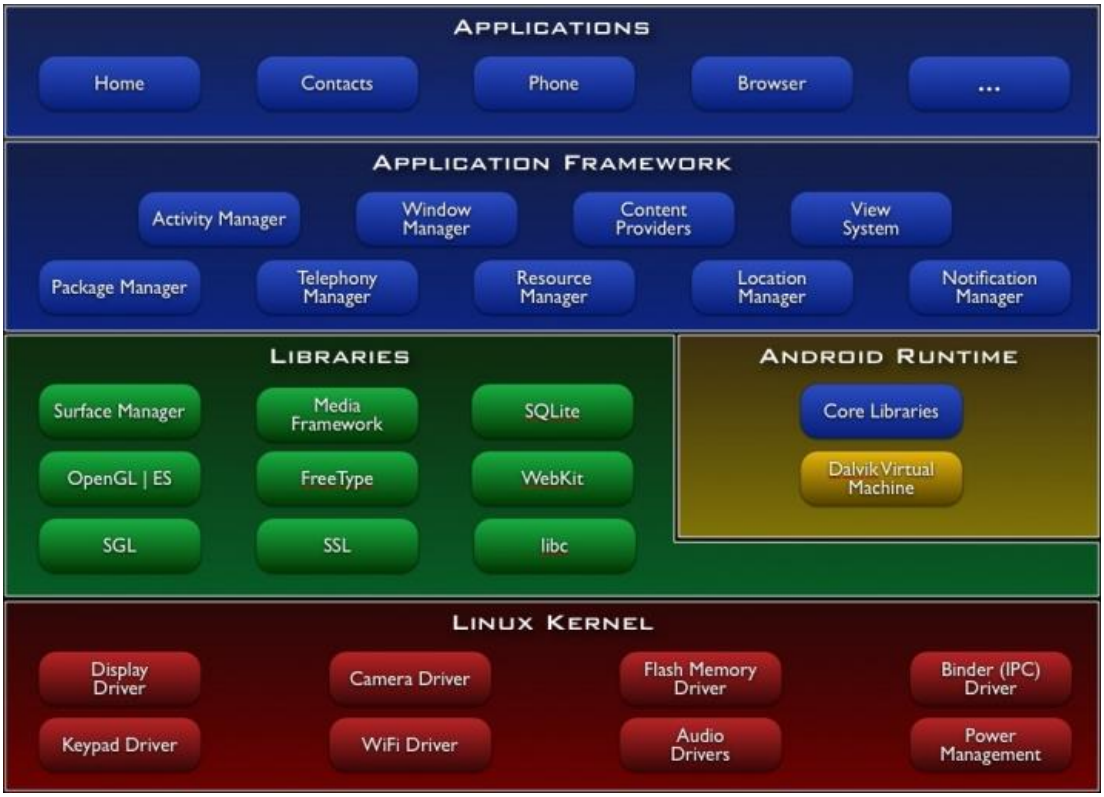
В данном курсе лекций будут рассмотрены основные приемы программирования простейших мобильных приложений, работающих под управлением операционной системы *Google Android*. В настоящий момент рынок мобильных устройств бурно развивается в ряде стран, и соизмерим по доходности с рынком продажи персональных компьютеров. На основании исследований, проведенных компанией *Google*, мобильные устройства, в основном, используются как средство для выхода в интернет, средство аудио и видео записи. Базовая функция устройств – телефонные звонки, стала использоваться значительно реже. Поэтому большинство мобильных устройств пользуются как миниатюрные карманные компьютеры, пришедшие взамен ноутбукам.

В настоящий момент на рынке имеются большой набор мобильных устройств (смартфоны, планшеты, и т.д.) различных производителей. Указанные устройства используют различные платформы: *Google Android, Apple iOS, Windows Mobile, Symbian, iPhone*. С точки зрения специалистов одной из самых перспективных платформ является *Android*.

Android [i] – одна из ведущих мобильных программных платформ на мировом рынке. Она включает в себя не только операционную систему, но также связующее ПО (middleware) и ключевые приложения. ОС *Android* основана на модифицированной версии ядра Linux. Новые версии этой операционной системы выпускаются Google и Open Handset Alliance. Техническое обслуживание и дальнейшее развитие *Android* осуществляет Android Open Source Project (AOSP). В рамках инновационной компании Promwad работает специальное подразделение Promwad Mobile, которое специализируется на разработке мобильных Android-приложений для мобильных телефонов, планшетных компьютеров, ридеров, систем автомобильной навигации и других мультимедийных устройств. Разработка пользовательских приложений для *Android* осуществляется при помощи специального SDK на языке программирования Java. Java байт-код

преобразуется в собственный формат байт-кода dex, исполняемый виртуальной машиной Dalvik. Отличительные характеристики платформы *Android*:

- виртуальная машина Dalvik
- встроенный браузер на движке Webkit
- развитые графические библиотеки для 2D графики и 3D графики на базе спецификации OpenGL ES 1.0 с поддержкой аппаратного ускорения
- использование SQLite для хранения структурированных данных
- поддержка мультимедийных аудио и видео форматов (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- коммуникационный стек для работы с GSM
- коммуникационный стек для работы с Bluetooth, EDGE, 3G и WiFi
- поддержка дополнительных устройств: видеокамера, GPS, компас, акселерометр
- развитые средства разработки: отладчик, эмулятор, средства профилирования, интеграция со средой разработки Eclipse



Данный курс лекций предназначен для изучения основ программирования мобильных приложений под ОС *Android*.

Основы объектно-ориентированного программирования в Java

Среда разработки Eclipse предоставляет комплекс программных средств для создания приложений на языке Java. Язык программирования Java является объектно-ориентированным. Исходный код компилируется в специальный байт-код, который может быть выполнен на виртуальной машине, что обеспечивает Java кроссплатформенность. Основные возможности языка[ii]:

- автоматическое управление памятью;
- расширенные возможности обработки исключительных ситуаций;
- богатый набор средств фильтрации ввода/вывода;
- набор стандартных коллекций, таких как массив, список, стек и т. п.;
- наличие простых средств создания сетевых приложений;
- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы;
- встроенные в язык средства создания многопоточных приложений;
- унифицированный доступ к базам данных;
- поддержка шаблонов (начиная с версии 1.5);
- параллельное выполнение программ.

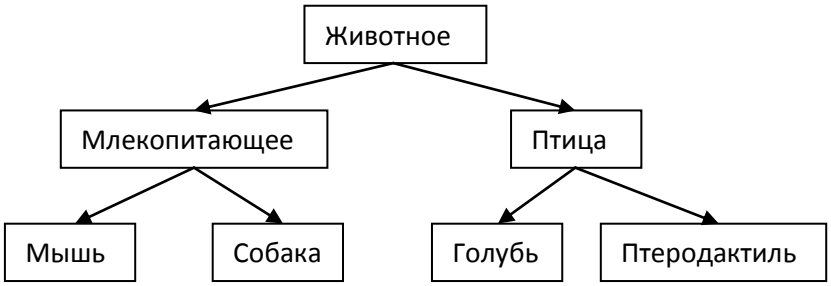
Проводя аналогию с языком C++ можно отметить наличие пространства имен, реализуемое в Java через пакеты (Java-package). Аналогично имеется понятие класса, как механизма создания объектов, и присущих ему механизмов. Инкапсуляция – механизм ограничения доступа к определенным компонентам объекта. Инкапсуляция применяется, когда требуется защитить поля класса от доступа извне. Существуют специальные модификаторы доступа определяющие область видимости полей класса.

Модификатор доступа	Область видимости элементов раздела		
	Из внешних функций	Из методов наследуемых классов	Из методов самого класса
public	Да	Да	Да
protected	Нет	Да	Да
private	Нет	Нет	Да

```
public class A{
private String name;
public void SetName(String n){
        name=n;
        }
public String GetName(){
        return name;
        }
}
```

Здесь мы видим ограничение прямого доступа к полю name извне. Работа с полем осуществляется посредством двух методов GetName и SetName.

Основным достоинством объектно-ориентированного программирования является возможность повторного использования ранее созданных классов через наследование. **Наследование** – создание иерархии классов для того, чтобы поля данных и методы предков автоматически становились полями данных и методами потомков. Наследование реализуется возможностью объединять класс с другим, во время объявления второго класса. Рассмотрим следующую иерархию объектов.



Класс, лежащий на вершине иерархии, содержит поля характерные для всех классов иерархии, например, вес. По мере углубления вниз иерархии производится добавление отдельных полей характерных только для данного вида существ (например, поле Размах крыла для класса Птица), причем остальные поля автоматически наследуются из родительского класса. Такой механизм очень удобен в использовании и позволяет быстро разрабатывать большое количество объектов разных типов. Пример:

```
pubic class Animal{
        public int weight;
};
```

Создадим новый класс на основе предыдущего:

```
pubic class Bird extends Animal{
        public:
        int wingspan;
};
```

Таким образом, класс Bird имеет не только свое поле wingspan, но и унаследованное weight.

Существует еще один базовый механизм ООП. Полиморфизм – возможность объектов иметь разную реализацию при одинаковых интерфейсах. Когда наследуются классы, то часть идентификаторов (например, имен методов) оказывается уже использованными, что заставляет программиста придумывать другие имена для схожих по «смыслу» методов. Например:

```
pubic class technic{
        public int move(){
                printf("Передвигаемся");
                }
};
pubic class tank extends technic{
@Override
        public int move(){
                printf("Едем в танке");
                }
};
pubic class plane extends technic{
@Override
        public int move(){
                printf("Летим на самолете");
                }
```

```
};

public void proc(){
        technic a;
        tank    b;
        plane   c;

        a.move();     // Передвигаемся
        b.move();     // Едем в танке
        c.move();     // Летим на самолете
}
```

Как видно из примера полиморфизм дает возможность использовать один и тот же идентификатор move для различных реализаций. Спецификатор @Override указывает на то, что наследуемы метод move заменяется на новый.

Также существует понятие виртуальных методов. Виртуальный метод – функция класса, которая может быть переопределена в наследниках так, что конкретная реализация метода для вызова будет определяться во время выполнения программы. Рассмотрим следующий пример.

```
void proc(){
        technic a[2];

        a[0]= new technic();
        a[1]= new tank();
        a[2]= new plane();

        for (int i=0;i<=2;i++)
        a[i].move();
```

Результат:

```
Передвигаемся
Едем в танке
Летим на самолете
```

При наследовании членов класса существует возможность запрещать их переопределение потомками. Например,

```
pubic class tank extends technic{
        public final int move(){
                printf("Передвигаемся");
                }
```

В данном случае спецификатор final запрещает переопределять метод move в производных от technic классах. Если final стоит в описании класса, то данный класс запрещается наследовать.

Существуют поля класса, называемые статическими. Доступ к таким полям может производиться еще до создания объекта и является общим для всех экземпляров класса. Причем, изменение значение поля в одном объекте повлечет за собой изменения во всех объектах.

```
public class technic{
        public static int x;
        …
        };
```

Таким образом, статическая переменная может использоваться как разделяемый ресурс между всеми объектами данного класса.

Еще одна особенность ООП состоит в наличии абстрактных методов. Такие методы не имеют реализации в рамках данного класса, но обязательно должны быть переопределены в производных классах. Класс, содержащий абстрактный метод, также называется абстрактным и должен иметь в описании спецификатор abstract. Пример,

```
public abstract class technic{
    public abstract int move();
    }
```

Вопросы для самопроверки

1. В чем заключается принцип инкапсуляции?
2. Что такое полиморфизм?
3. Когда удобнее применять наследование?

Компоненты приложений. Файлы проекта.

Исходный код простейшего приложения состоит из набора модулей, каждый из которых содержит описание одного класса. Рассмотрим пример простейшей программы.

```
package com.example.save;
import android.os.Bundle;
public class myclass extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_save_xml);
}
    @Override
    protected void onStop() {
        super.onStop();
}
```

Первой строка обозначает имя проекта (оно должно быть уникальным). Далее идет набор подключаемых модулей (аналог #include в языке c++). В нашем приложении мы будем создавать класс myclass на базе существующего класса Activity (что указано как: myclass **extends** Activity). При запуске приложения будет вызываться функция onCreate(), а при остановке onStop(). Строка super.onCreate(savedInstanceState) означает вызов функции onCreate

родительского класса. setContentView(R.layout.activity_save_xml) определяет отображаемый интерфейс приложения (в данном примере интерфейс находится в ресурсе R.layout.activity_save_xml). Строго говоря, мы будем использовать вышеприведенную программу в качестве шаблона при разработке более сложных приложений.

Приложение Android содержит различные ресурсы, находящиеся в папке res. Все ресурсы имеют уникальные идентификаторы. Синтаксис для выделения идентификаторов получается следующим образом @[package:]type/name. Type соответствует одному из пространств имен с типами ресурсов:

- R.drawable – ресурсы изображений
- R.id – идентификаторы ресурсов
- R.layout – макет приложения
- R.string – строковые константы

Таким образом, идентификатор может выглядеть следующим так: R.id.TextView1. Различные папки используются для хранения ресурсов:

- anim – скомпилированные файлы анимации и фидео;
- drawable – растровые изображения (например, bmp);
- layout – определения пользовательского интерфейса;
- values – массивы, цвета, строки … ;
- xml – компилированные произвольные xml файлы;
- raw – не компилированные исходные файлы.

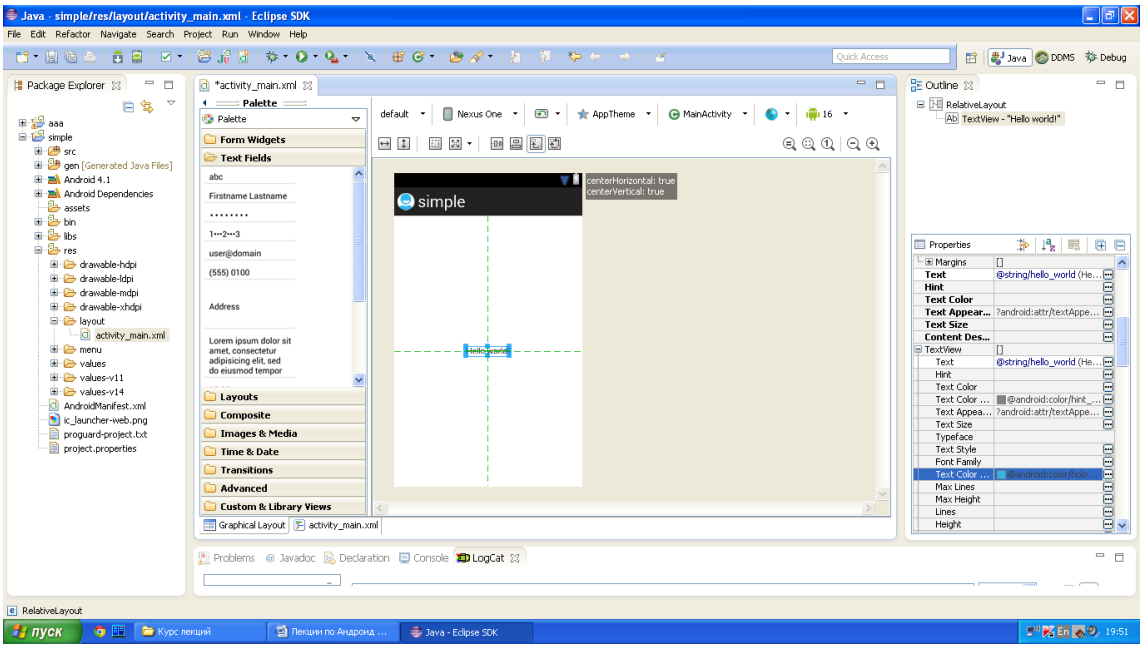
Eclipse накладывает ограничения на имена файлов, являющихся ресурсами. Так имена файлов должны состоять только из строчных букв латинского алфавита, цифр и символа подчеркивания (a – z, 0 – 9, _).

Для создания работоспособного приложения нам потребуется разработать интерфейс. В Eclipse существует наборы готовых компонентов, которые можно с легкостью использовать для построения интерфейса.

Для отображения формы (интерфейса) проекта необходимо отобразить файл res/layout/activity_main.xml в режиме Graphical Layout.

Рассмотрим набор различных компонентов, которые будут активно использоваться в рамках данного курса.

TextView – обычное текстовое поле, предназначенное для вывода строковых сообщений. Свойство Text задает текст выводимого сообщения. TextColor – цвет текста (например, «?android:color/holo_green_light»). TextSize – размер текста (например, «20px»). Visibility – видимость на форме.



При наведении курсора мыши на свойства класса в панели свойств (правое нижнее окно Properties) можно получить всплывающую подсказку, описывающую данное свойство. Доступ к элементу из программы осуществляется следующим образом:

```
TextView txt=(TextView)findViewById(R.id.textView1);
txt.setText("Hello Android");
```

TextView – тип переменной txt. Все элементы приложения андройд имеют уникальные идентификаторы, которые определены в файле gen/R.java (Файл модифицировать категорически запрещено). findViewById – это функция поиска элемента проекта по идентификатору. В качестве параметра здесь указан путь к данному элементу. В нашем примере *textView1* – значение поля Id класса TextView.

Компонент editText – предназначен для ввода текста пользователем. Имеющиеся свойства такие же как у TextView.

```
EditText e= (EditText) findViewById(R.id.editText1);
txt.setText(e.getText());
```

В работающем приложении, при постановке курсора в текстовое, поле на экране устройства автоматически отображается клавиатура.

Компонент Button – обычная кнопка. Отображаемый на кнопке текст задается свойством Text. В рабочей среде Eclipse обработчик нажатия клавиши необходимо создавать самостоятельно. Например,

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView txt=(TextView)findViewById(R.id.textView1);
    Button b=(Button)findViewById(R.id.button1);
    b.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            txt.setText("Hello");
        }
    });
```

Функция setOnClickListener – устанавливает обработчик нажатия кнопки. Функция onClick(View v) может быть описана в другой части кода программы, но в таком случае локальные переменные функции onCreate станут недоступны.

SeekBar – слайдер, когда пользователь может передвигать ползунок пальцем на экране. Также можно двигать ползунок при помощи клавиш-стрелок. Следующие методы часто используются:

- onProgressChanged() - уведомляет об изменении положения ползунка;
- onStartTrackingTouch() - уведомляет о том, что пользователь начал перемещать ползунок;
- onStopTrackingTouch() - уведомляет о том, что пользователь закончил перемещать ползунок. Свойство Progress указывает положение ползунка. Максимальное значение определяется свойством Max.

Проект приложение хранит исходные коды java в папке src. В корневой папке имеется файл AndroidManifest.xml[iii] - файл, описывающий глобальные параметры приложения. Файл должен содержать требуемые приложением разрешения. Например, если приложению требуется доступ к сети, то это должно быть определено здесь. *AndroidManifest.xml* можно рассматривать, как описание для развертывания Android-приложения.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.save"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Convert"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Основные атрибуты

- package - базовый пакет для следующих элементов Java. Он также обязан быть уникальным, т.к. Android Marketplace только единожды принимает заявку на каждый *package*. Следовательно, необходимо обеспечить уникальность имени *package*, во избежание конфликтов с другими разработчиками.
- android:versionName и android:versionCode - определяют версию Вашего приложения. versionName - то, что видит пользователь и может быть любой строкой. versionCode должен быть целым, и Android Market

использует это для определения, предоставили ли Вы новую версию, чтобы запустить обновления на устройствах, на которых установлено Ваше приложение. Как правило, начинается с **1** и увеличивается на единицу, если Вы выпускаете новую версию приложения.

- Activity – определяет активити (главный класс программы, аналог функции main()), в этом примере указывает на класс «com.example.save». Определение категории (категория android:name= «android.intent.category.LAUNCHER») определяет, что это приложение добавлено в директорию приложений на Android-устройстве. Значения со знаком **@** ссылаются на файлы ресурсов, которые содержат актуальные значения. Это упрощает работу с разными ресурсами, такими как строки, цвета, иконки, для разных устройств и упрощает перевод приложений. Каждый активти в приложение должен иметь собственный блок <activity> в файле манифеста.
- uses-permission описывает права необходимые для того что бы приложение могло получить доступ к выполнению определенных операций. Например, в приложение использующем доступ к данным GPS, необходимо явно указать разрешение на доступ, например так:

```
<uses-permission android:name= «android.permission.ACCESS_GPS» />
```

- permission секция описывает права, которые должны запрашивать другие приложения для доступа к данному.
- uses-sdk определяет минимальную версию SDK, на котором можно запускать Ваше приложение. Это предотвращает установку Вашего приложения на устройства с более старой версией SDK.

Вопросы для самопроверки

1. Зачем нужен файл AndroidManifest.xml ?
2. Что выполняет функция findViewById() ?

4. Поддержка 2-D графики

Для создания графического интерфейса приложения необходимо использовать набор графических примитивов (окружность, линия, точка …).

Имеющаяся библиотека для работы с графическими объектами предоставляет интерфейс программирования двумерной графики на базе класса Canvas. Рассмотрим простейший пример создания приложения, выводящего на экран графические примитивы.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new Panel(this));
}
class Panel extends View{
        public Panel (Context context){
                super(context);
                …
        }
        @Override
        public void onDraw(Canvas c){
                Paint p=new Paint();
                p.setColor(Color.White);
                        //Color.rgb(70, 130, 180)
                p.setStyle(Style.FILL);
                c.drawRect(10,10,50,50,p);
        }
}
```

В методе onCreate основного класса происходит переключение отображаемого интерфейса приложения на класс Panel. Класс Panel способен выводить на экран различные графические примитивы. Метод onDraw вызывается всякий раз, когда требуется прорисовка панели. В данном примере на панель выводится квадрат цветом, задаваемым с помощью кисти p.

Доступны следующие графические примитивы:

Линия	drawLine
Квадрат	drawRect
Овал	drawOval
Точка	drawPoint
Текст	drawText

Для вывода на панель картинки сперва ее следует загрузить из ресурсов приложения.

```
@Override
    public void onDraw(Canvas c){
        Paint p=new Paint();
        p.setColor(Color.White);
        p.setStyle(Style.FILL);
        Bitmap b=BitmapFactory.decodeResource(getResources(),
        R.drawable.mypicture);
        c.drawBitmap(b,100,100,p); //отображаем картинку на экране
    }
```

В данном примере картинка называется mypicture.

Для постоянной перерисовки панели обычно используют таймер.

```
class Panel extends View{
    public Panel (Context context){…}
    @Override
    public void onDraw(Canvas c){…}
    new CountDownTimer(1000,100){
        public void onTick(long millisUntilFinished){
            invalidate(); //перерисовать экран
        }
        public void onFinish(){
            this.start(); //по окончанию вызвать повторно
        }
    }.start();
}
```

Функция CountDownTimer имеет два параметра: первый указывает общее время (в милисекундах) работы таймера, после которого будет вызван метод onFinish. В данном примере после завершения работы таймер будет запускаться вновь. Второй параметр указывает интервал, с которым будет вызываться метод onTick. Функция invalidate() указывает приложению, что на экране произошли изменения и его необходимо перерисовать, что заставляет приложение вызывать метод onDraw (делать смену кадров).

Для эффективного взаимодействия с приложением необходимо задействовать сенсорную панель экрана. Обработка событий, происходящих на сенсорном дисплее, осуществляется в программе.

```
class Panel extends View{
    public Panel (Context context){
        super(context);
        this.setOnTouchListener(new OnTouchListener(){
            @OverRide
            public Boolean onTouch(View v, MotionEvent event){
                int x,y;
                x=Math.round(event.getX());
                y=Math.round(event.getY());
    // совершено нажатие на панель в точке (x,y)
                return true;
            }
        });
    }
}
```

Здесь происходит установка обработчика событий панели OnTouchListener. Событие event имеет следующие методы:

- getX() – возвращает координату X события;
- getY() – возвращает координату Y события;
- getAction() – возвращает тип события:
    - MotionEvent.ACTION_DOWN – нажатие;
    - MotionEvent.ACTION_UP – отжатие;

- MotionEvent.ACTION_MOVE – перемещение.

Вопросы для самопроверки

1. Какой метод используется для перерисовки объекта Panel ?
2. Какие графические примитивы доступны для рисования?
3. Каким образом обрабатываются нажатия на панели экрана?

Намерения

Для взаимодействия приложений друг с другом используется специальный механизм – Намерения, реализованный в классе Intent. Намерения используются для передачи запросов к другим приложениям, в том числе и к компонентам операционной системы Android. Существует несколько видов намерений, вот некоторые из них:

- Явные. Применяются, когда исходное приложение отправитель намерения запрашивает обработку данных у другого заранее заданного приложения (т.е. известно название пакета package обработчика).
- Не явные. Применяется, когда исходное приложение отправитель намерения запрашивает обработку данных у приложения и заранее неизвестным именем package.

Обработка намерений осуществляется по следующей схеме. Исходное приложение отправитель формирует намерение, используя класс Intent, и отправляет его на обработку. Операционная система принимает намерение и если оно явное, т.е. указано какому приложение оно направлялось, то его перенаправляют конечному приложению. Далее намерение обрабатывается специальными функциями приложения-получателя и жизненный цикл намеренья завершается. При неявном намерении, когда неизвестна приложение-получатель, операционная система собирает все файлы AndroidManifest.xml со всех установленных приложений и начинает их анализировать. Файл манифеста содержит в себе тег <*intent-filter*> в которых указано, какие намерения может обрабатывать приложение. Таким образом, будет найдено приложение, которое может обработать намеренье. Если в системе есть два или более приложений, способных обработать намерение, то

система предложит пользователю самостоятельно выбрать приложение-получатель.

Явные намерения используются для переходов между активностями приложения. Например, необходимо создать дополнительную активность приложения, то создается новый класс и xml разметка. Обратите внимание на то, какая разметка привязывается к новому классу.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(/* Здесь должна быть новая разметка*/);
}
```

Также необходимо добавить в файл AndroidManifest.xml разрешение на запуск данного класса. Если это не будет делано, то при переключении приложение экстренно остановится с ошибкой.

```
<activity android:name=".NewActivity"
        android:label="@string/app_name2">
    </activity>
```

Далее сам процесс переключения производится с помощью следующих команд в исходном классе.

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

Для передачи данных вместе с намерением используется метод putExtra()

```
intent.putExtra("Ключ", "Значение");
```

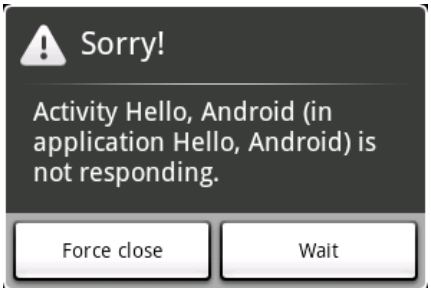
Принимающее приложение использует серию функций getExtra()

```
Intent intent = getIntent();
Key = intent.getStringExtra("Ключ");
```

Потоки

Удобный пользовательский интерфейс предполагает минимального время отклика приложения на осуществляемые пользователем действия. Обычно задержки (подвисание) в работе программы вызваны не трудоемким

алгоритмом работы, а какими-либо внешними задержками, например, ожидание загрузки файла из сети Интернет. Во время такого ожидания приложение временно перестает обрабатывать события, и операционная система считает такое приложение зависшим, предлагая его принудительное закрытие.



Операционная система предоставляет возможность создания полноценного (аналогичного с ОС Linux, Windows) многопоточного приложения. Рассмотрим жизненный цикл одного потока на следующем примере.

1. Создание
2. Запуск

2.1. Постановка в очередь планировщика задач

2.2. Работа

2.3. Блокировка

3. Уничтожение

```
//Опишем класс потока
class Task extends Thread{
        @Override
        protected void run(){
                // действия выполняющиеся в потоке
        }
}
//Запустим поток
public void onCreate(Bundle savedInstanceState) {
        …
        Thread p=new Task(); //создание потока
```

```
        p.start(); // Запуск
    }
```

Рассмотрим пример работы двух одновременно работающих потоков.

```
    //Опишем класс потока1
    class Task1 extends Thread{
        @Override
        protected void run(){
            ProgressBar p=(ProgressBar)findViewById(R.id.progressBar1);
            for(int i=0;i<100;i++){
                p.setProgress(i);
                SystemClock.sleep(100);
            }
        }
    }
    //Опишем класс потока2
    class Task2 extends Thread{
        @Override
        protected void run(){
            ProgressBar p=(ProgressBar)findViewById(R.id.progressBar2);
            for(int i=0;i<100;i++){
                p.setProgress(i);
                SystemClock.sleep(100);
            }
        }

    }
    //Запустим потоки
    public void onCreate(Bundle savedInstanceState) {
        …
        Thread p1=new Task1(); //создание потока1
        Thread p2=new Task2(); //создание потока2
        p1.start(); // Запуск
        p2.start(); // Запуск
    }
```

Работающее приложение будет показывать две заполняющиеся полосы одновременно. SystemClock.sleep(100) приостанавливает выполнение потока на 100 миллисекунд.

Обычно потоки рекомендуется применять для загрузки файлов из сети интернет или для проигрывания звуковых файлов.

```
class Task2 extends Thread{
    @Override
    protected void run(){
        MediaPlayer mp;
        mp=MediaPlayer.create(getApplicationContext(),R.raw.sound);
        mp.start();
    }
}
```

Дополнительные возможности

Сохранение состояния

При запуске приложения иногда требуется загрузка настроек, сделанных во время предыдущей работы программы. Самым простым способом хранения настроек является хранение данных в виде xml файлов. Файл xml может располагаться в папке res/values/. Рассмотрим пример чтения настроек-ресурсов из файла.

Содержимое файла *res/values/settings.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="abc">Hello world</string>
    <item format="integer" name="n" type="integer">12345</item>
</resources>
```

Ниже приводится исходный код части программы, запускающийся при вначале работы приложения.

```
public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_save_xml);
        EditText loginEdit = (EditText)findViewById(R.id.editText1);
        int i;
        i=getResources().getInteger(R.integer.n);
        loginEdit.setText(String.valueOf(i));
    }
```

Текстовое поле loginEdit загружает из файла xml значение элемента ***n*** при помощи функции getResources(). Таким образом, настройки задаются на стадии компиляции программы и могут быть считаны приложением во время его работы.

Настройки могут также создаваться во время работы приложения и быть доступными для изменений. В таком случае доступ к настройкам должен быть только из программы (а не на стадии компиляции через редактор). Рассмотрим следующий пример.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_save_xml);
        EditText loginEdit = (EditText)findViewById(R.id.editText1);
    // загружаем настройки
        SharedPreferences settings=getPreferences(0);
        String login=settings.getString("Login","");
    // вставляем в текстовое поле загруженные настройки
        loginEdit.setText(login);
    }
    @Override
    protected void onStop() {
        super.onStop();
        EditText loginEdit = (EditText)findViewById(R.id.editText1);
    //получаем содержимое текстового поля
```

```
        String login=loginEdit.getText();
    // загружаем редактор настроек
        SharedPreferences settings=getPreferences(0);
        SharedPreferences.Editor editor=settings.edit();
    //записывем в редактор настройки
        editor.putString("Login",login);
    // сохраняем данные из редактора
    editor.commit();
}
```

В данном примере в качестве настройки используется строка Login. Она загружается при запуске программы и сохраняется при завершении работы приложения. Доступ к таким настройкам не может быть осуществлен на стадии компиляции через редактор.

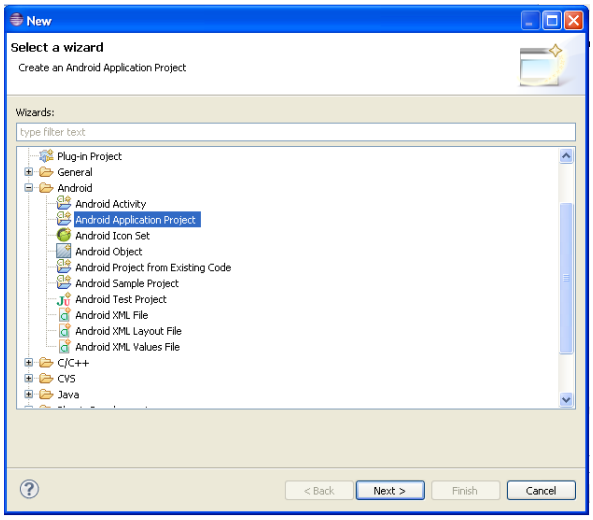
Вопросы для самопроверки

1. Где приложение хранит свои настройки?
2. Как можно ли задать настройки приложения на стадии компиляции?

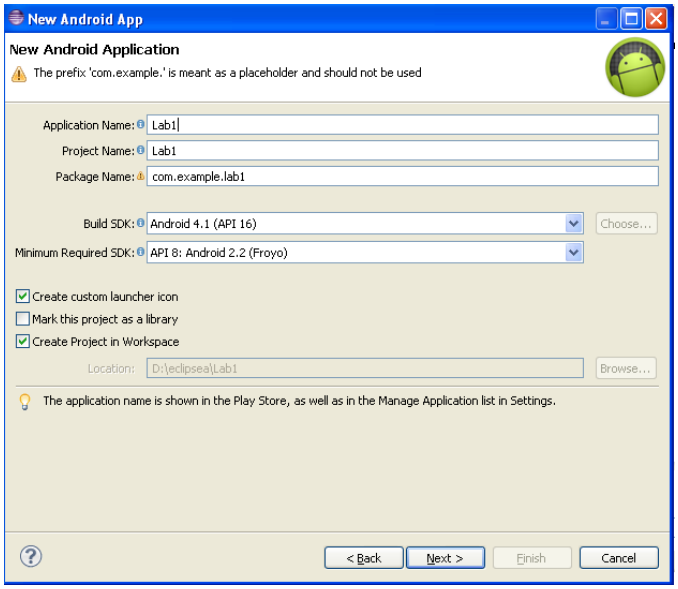
Курсовая работа

Задание Пример: Создайте простейшее приложение Hello World. Порядок выполнения: Запустите Eclipce.

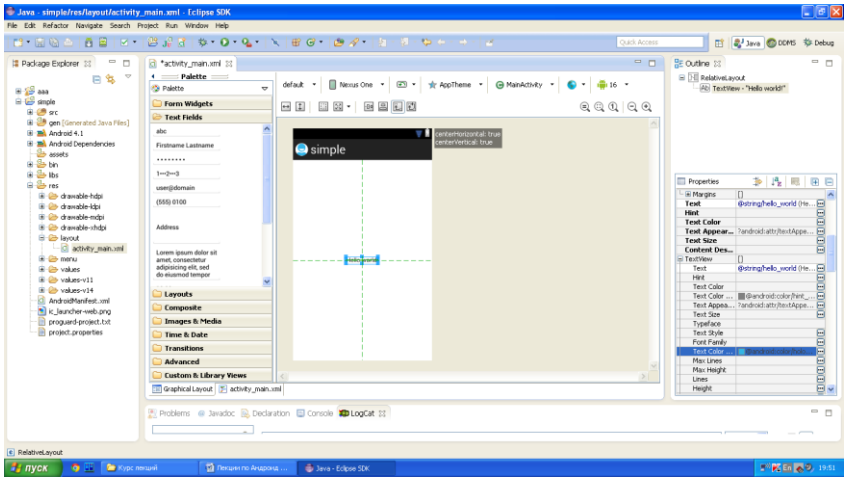
Создайте новый проект.



Задайте имя проекта.



Остальные параметры проекта оставьте без изменений. Далее перед Вами откроется следующий вид проекта.

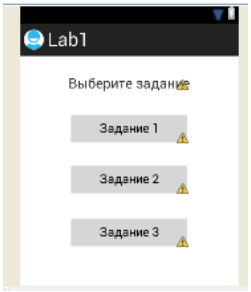


Слева расположено дерево файлов проекта. В центре находится редактор интерфейса приложения. В данном случае можно изменить положение надписи *Hello World* простым перетаскиванием мышки. Для запуска проекта необходимо нажать Ctrl+F11. Затем появится окно, в котором указывается способ запуска приложения. Выберите *Android Application*.

Через несколько секунд начнется загрузка Эмулятора. Эмулятор загружается примерно (3-4 мин). В дальнейшем, при внесении изменений в проект,

закрывать или перезапускать эмулятор не нужно. Для редактирования исходного кода приложения выберите файл *src/com.example.lab1/MainActivity.java*

В рамках курсового проекта необходимо выполнение четырех следующих заданий в соответствии с вариантом. Все задания курсовой работы объединены в одну программу (один проект). Таким образом, проект будет состоять из набора классов, каждый из которых реализует какое-то одно задание. При запуске программы на экране должен появиться набор кнопок, каждая из которых запускает одно из следующих заданий. (Для переключения между интерфейсами заданий можно использовать функцию setContentView()).



Вариант задания вычисляется так: остаток от деления на 3 последней цифры Вашего пароля (идентификатора) плюс единица. Например, для цифры 7: (7%3)+1 = (1)+1 = 2. Итог: вариант №2.

Задание 1:

Реализуйте простейший Калькулятор. Имеется набор кнопок, циферблат. Калькулятор позволяет вычислять (сумму, разность, произведение и частное). Предусмотреть обработку ситуации деления на ноль. Кроме этого, добавьте функции вычисления (вариант 1: вычисление синуса, вар. 2: возведение в степень, вар. 3 деление с остатком).

Задание 2:

Нарисуйте <Объект>. На нем разместите несколько шаров, которые будут медленно менять цвет с разной скоростью. Обработку изменения цвета реализовать с помощью потоков. (Вариант 1 <объект>- Елка, Вариант 2 <объект>-Светофор, Вариант 3 <объект>- Снеговик).

Задание 3:

Прыгающий <объект>. Начальная позиция появления объекта определяется нажатием пальца по экрану. Объект падает с ускорением вниз,

отражаясь от нижней границы экрана, и с незначительным смешением вправо. С каждым разом высота отскока объекта уменьшается, и когда объект практически остановится, он пропадает с экрана. Объектов может быть несколько, поэтому каждый должен обрабатываться в отдельном потоке приложения. (Вариант 1 <объект>- Цифра восемь, Вариант 2 <объект>- Октаэдр, Вариант 3 <объект>- Треугольник).

Задание 4:

Дополните Задание 3 возможностью изменения настроек (цвета объекта, ускорения, цвет/рисунок фона). Добавьте кнопку настройки в основное меню программы. Отскок объекта от границы экрана должен сопровождаться звуком (любым на ваше усмотрение). Настройки должны сохраняться после закрытия приложения через класс (Preferences).

---

[i] http://www.promwad.com/technologies/mobile-platforms-ru.html
[ii] http://ru.wikipedia.org/wiki/Java
[iii] http://droid.ameego.ru/wiki/AndroidManifest.xml