МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ИНСТИТУТ ТЕХНОЛОГИЙ (ФИЛИАЛ) ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» В Г. ВОЛГОДОНСКЕ РОСТОВСКОЙ ОБЛАСТИ

(Институт технологий (филиал) ДГТУ в г. Волгодонске)

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для проведения лабораторного практикума
по дисциплине
«РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ»
для обучающихся по направлению подготовки
09.03.02 Информационные системы и технологии
программа бакалавриата «Информационные системы»

2021г.

Практическая работа № 1 Портирование приложений с использованием Intel XDK

Задачи практической работы:

- Изучить основы работы в Intel XDK.
- Исследовать переносимое приложение и переработать интерфейс.
- Перенести интерфейс в среду Intel XDK с учетом особенностей мобильного устройства.
- Организовать функционирование приложения, по возможности используя старый код.

Инструкции по выполнению практической работы

Основы работы в Intel XDK

Если вы не изучали курс "Введение в разработку приложений для ОС Android", для выполнения этой работы вам необходимо установить и настроить среду Intel XDK. Особенности установки и настройки этой среды, а также инструкции по созданию и запуску приложений приведены здесь.

Исследование переносимого приложения и переработка интерфейса

Перед началом переноса существующего приложения на мобильную платформу Android необходимо очень хорошо продумать, как именно оно будет выглядеть в новых условиях. Следует учитывать особенности как самой платформы, так и устройств, которые на ней базируются. Необходимо обязательно обратить внимание на следующие факторы:

- Изменение размера окна приложения (как правило, в меньшую сторону).
- Наличие новых возможностей управления приложением. В первую очередь речь идет о том, что у сенсорного экрана любая область является кликабельной, а, значит, в некоторых случаях можно отказаться от выделенных кнопок.

Конечно, если вы портируете небольшое приложение наподобие классического калькулятора, то его вид на мобильном устройстве практически не будет отличаться от десктопных аналогов (см. рис. 8.1):

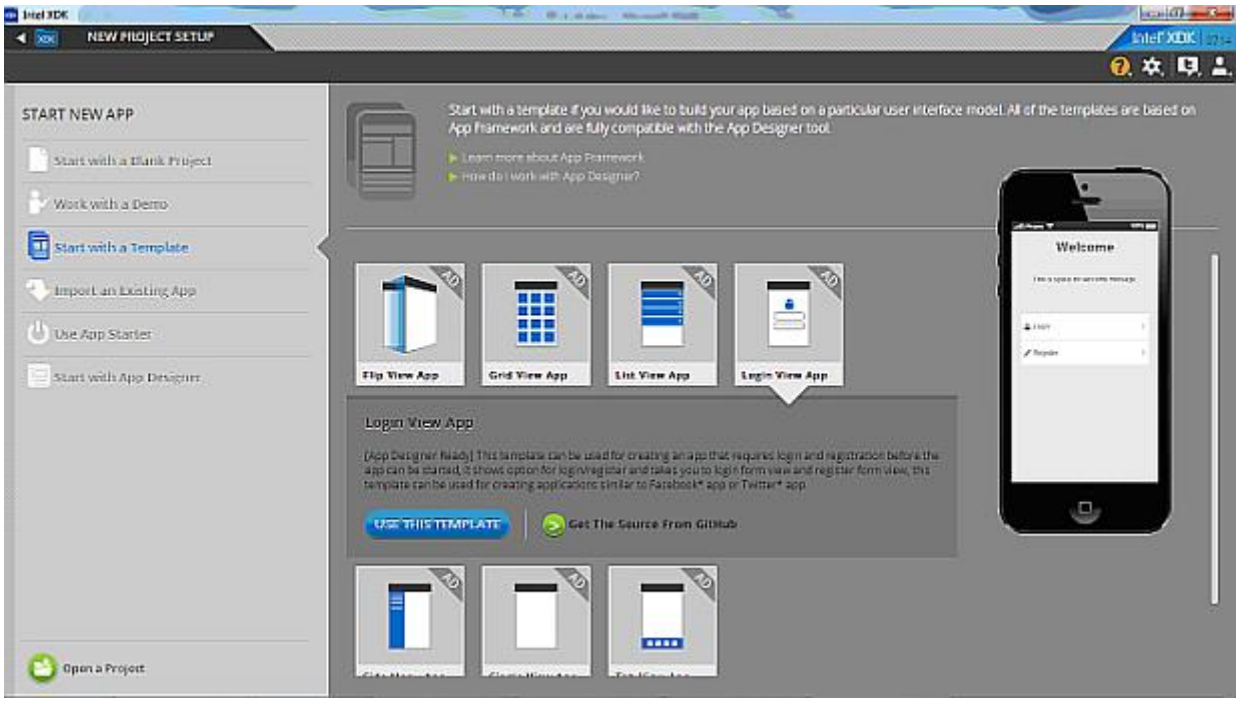
Рис. 8.1. Приложение «Калькулятор» для разных операционных систем. Слева Калькулятор Windows 7, справа бесплатное приложение Shake Calc (скриншот из Google Play)

Однако при внимательном рассмотрении можно заметить существенное отличие: в мобильной версии отсутствует строка меню. Возможность доступа к дополнительным настройкам реализована с использованием возможностей смартфона. Так, для доступа к дополнительным опциям можно переключиться в режим функций, просто проведя пальцем по экрану. Настройки приложения вызываются нажатием на специальную клавишу (аппаратную во многих моделях телефонов).

Если главное окно оригинального приложения больше калькулятора, то при переносе его на мобильное устройство придется основательно переработать интерфейс. Проще всего разработать интерфейс мобильного приложения "с нуля" (здесь мы рекомендуем воспользоваться "Особенности интерфейсов для смартфонов. Принципы юзабилити" данного курса) и уже потом, сравнивая интерфейс исходного и нового приложений, добавить в последнее возможности для наиболее полного соответствия обеих версий.

Перенос интерфейса в среду Intel XDK с учетом особенностей мобильного устройства

Прежде всего необходимо создать новое приложение в Intel XDK. В первой части курса мы рассматривали создание приложений "с чистого листа" и из демонстрационных примеров. После очередного обновления среды (надеемся, что вы их регулярно скачиваете и устанавливаете) появилась новая возможность – создание примера из шаблона. Для этого необходимо при создании нового проекта выбрать пункт меню Start with a Template (см. рис. 8.2).



увеличить изображение
Рис. 8.2. Создание проекта из шаблона

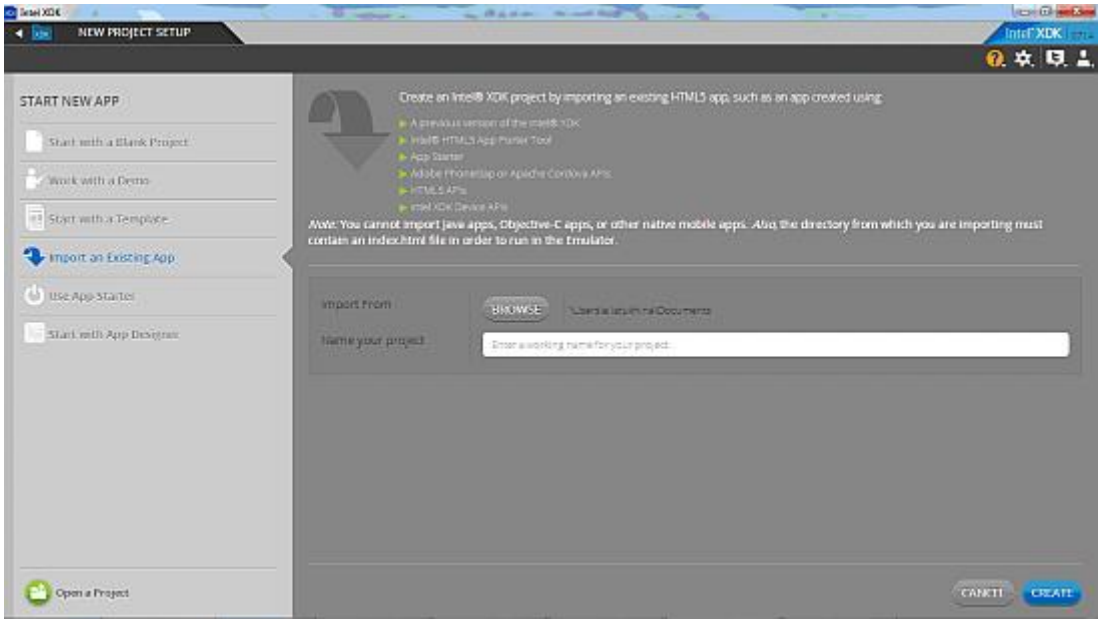
С развитием среды список шаблонов должен увеличиться. В момент написания этой лабораторной работы были доступны следующие семь шаблонов:

- Flip View App – предназначено для небольших приложений, содержащих только две экранные сущности (главную и окно настроек).
- Grid View App подходит для разработки приложений, позволяющих просматривать фотографии и другие объекты.
- List View App – для приложений, содержащих списки с возможностью просмотра подробностей для каждого конкретного элемента (например, почтовые приложения, ленты новостей и т.д.).
- Login View App – для приложений, требующих обязательной предварительной регистрации.

- Side Menu App позволяет разрабатывать приложения, содержащие несколько экранов, при этом переключение между ними осуществляется посредством меню, которое вызывается при помощи жеста вытягивания его из одной из сторон экрана.
- Single View App – для простых одноэкранных приложений.
- Tab View App – приложения, позволяющие переключаться между вкладками как с помощью нажатий на кнопки, так и посредством перелистывания.

Подумайте, какой из перечисленных выше способов наиболее удобен для портируемого приложения, и создайте соответствующий проект.

Если же вам повезло и приложение, которое вы собираетесь перенести на мобильную платформу, уже является HTML5-приложением, мы можете сразу использовать инструмент Import an Existing App (см. рис. 8.3).



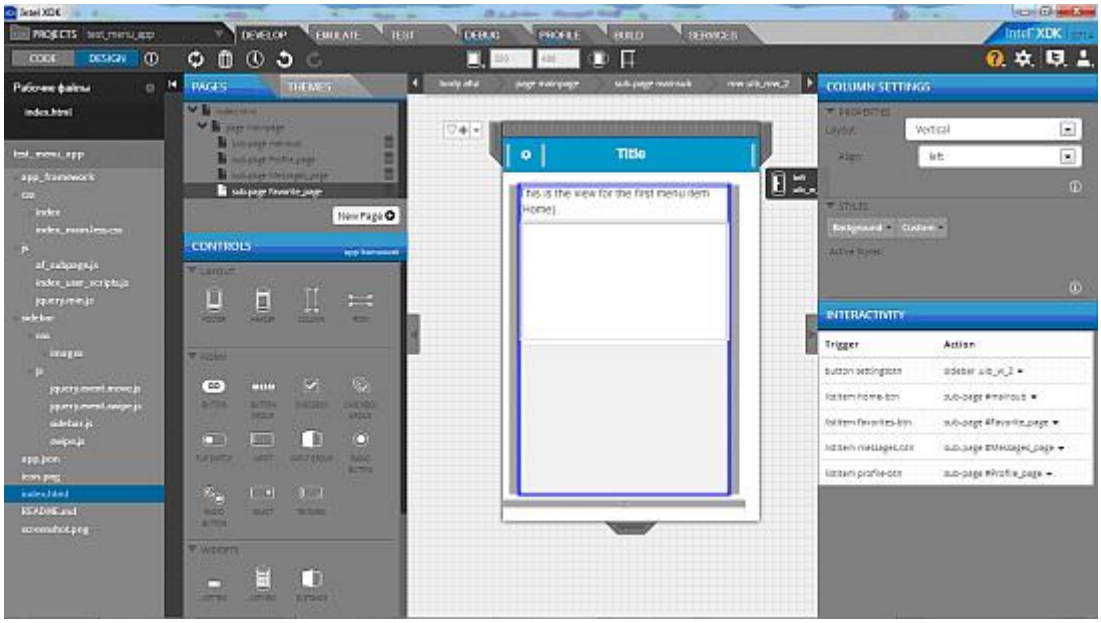
увеличить изображение

Рис. 8.3. Создание проекта из существующего HTML5-приложения

Организация функционирования приложения

После разработки нового интерфейса необходимо исследовать код исходного приложения. Если оно было разработано с соблюдением принципов объектно-ориентированного программирования, то отделение интерфейсных элементов от логики его работы не должно вызвать особых проблем. Если же эти принципы не слишком соблюдались (такое, увы, бывает слишком часто), то могут возникнуть большие сложности с выделением логики работы приложения.

Для использования имеющегося кода в Intel XDK необходимо перевести его на подходящий язык. Поскольку в HTML5-приложениях для описания интерфейса используются HTML5 и CSS, а для обеспечения логики работы скриптовый язык JavaScript, разделенный код необходимо переносить следующим образом. Все, что имеет отношение к интерфейсу, описывается на HTML5 с использованием CSS. Для ускорения процесса портирования можно использовать встроенный редактор дизайна, который имеется в Intel XDK. Для работы в этом режиме необходимо на вкладке Develop перейти из режима CODE в режим DESIGN. В этом режиме можно добавлять управляющие элементы и редактировать их свойства. В разделе PAGES здесь можно переключаться между экранными сущностями, если их несколько, и редактировать каждую из них (см. рис. 8.4).



увеличить изображение
Рис. 8.4. Редактирование проекта в режиме дизайнера

Для переноса логики работы приложения необходимо перевести существующий код на JavaScript. Основные конструкции JavaScript произошли из языка C, поэтому, если исходный код портируемого приложения был написан на языке из C-семейства, перевод его на JavaScript будет гораздо проще. Можно попробовать использовать готовые трансляторы, которые позволяют переводить код с одного языка программирования на другой. Конечно, после автоматической трансляции все равно придется править код, но, по крайней мере, количество трудоемкой рутинной работы значительно сокращается.

В качестве транслятора с языка Java можно использовать проект GWT. Компилятор GWT переведёт код Java-приложения в соответствующий браузеру JavaScript, HTML и CSS. Для перевода кода с других языков можно использовать аналогичные решения.

Задачи для самостоятельной работы

Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения. Попробуйте реализовать эти возможности.

Практическая работа № 2 Использование мобильной связи в приложениях для смартфона

Задачи практической работы

- Разработать приложение, которое может звонить по телефонным номерам.
- Проанализировать работу этого приложения на смартфоне.

Инструкции по выполнению практической работы

Приложение, описанное ниже, позволяет выполня ть телефонные звонки через встроенное *приложение* для звонков, используя при этом собственный *список* контактов. Конечно, то же самое можно сделать вручную или добавить эти номера в адресную книгу телефона. Однако описанный способ может пригодится в некоторых ситуациях, например, в качестве части внутреннего приложения для хранения заранее определенного списка номеров сотрудников компании. В этом случае *пользователь*, используя корпоративное *приложение*, может вызывать нужные номера со своего личного смартфона, не добавляя контакты в адресную книгу. В некоторых случаях таких номеров может быть очень много, и специальное *приложение* позволит использовать рабочий справочник, не смешивая служебную информацию с личной.

Мы создадим *приложение*, содержащее несколько кнопок. Нажатие на кнопку вызывает конкретный телефонный номер, проассоциированный с ней. Для простоты понимания в программе не используются *базы данных*, все нужные сведения "защиты" в код приложения. При желании несложно доработать программу и вынести эту информацию в отдельный *файл*.

Создайте проект SimpleDialer. Рекомендуемые настройки см. на рис. 10.1

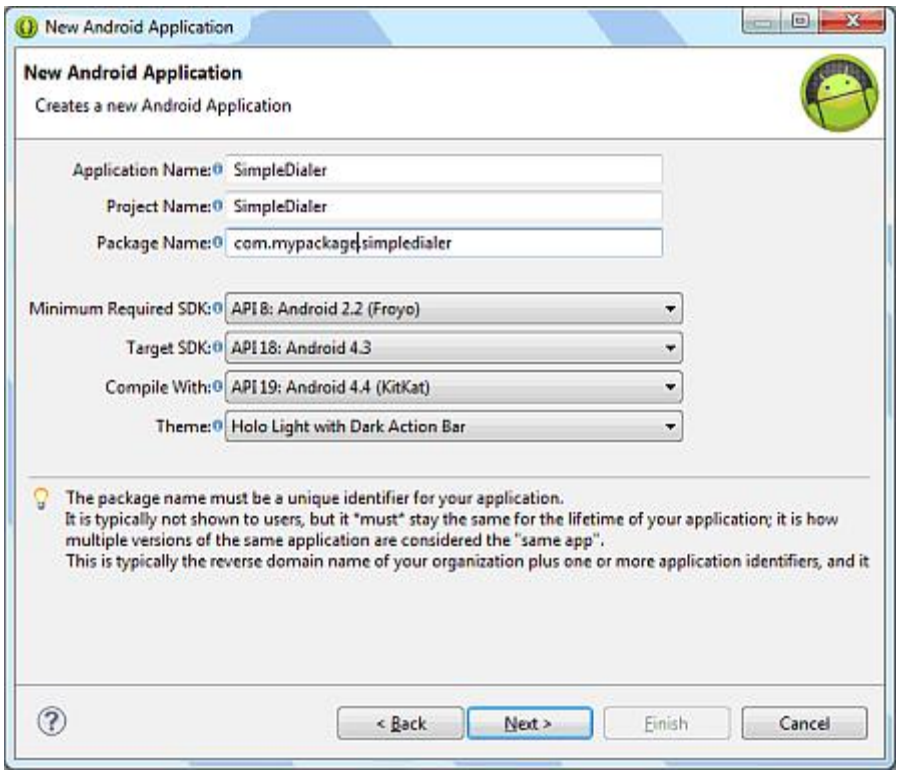


Рис. 10.1. Создание нового приложения

Добавьте следующую строчку в *файл* strings.xml:

```
<string name="main_label">My Friends</string>
```

В файле *layout*.xml создайте объекты TextView и шесть кнопок:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical"
      android:padding="15dip" >
<TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:layout_marginBottom="25dip"
         android:text="@string/main_label"
         android:textSize="22sp" />
<Button
         android:id="@+id/button1"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />
…//здесь добавить описания остальных кнопок
</LinearLayout>
```

Перейдите в *файл* MainActivity.*java*. Создайте переменную для хранения числа записей и два массива строк:

```
private int entries = 6;
private String phoneNum[];
```

```
private String buttonLabels[];
```

Так как кнопки будут реагировать на нажатие, добавьте в заголовок класса `implements OnClickListener` (нужно также добавить `import android.view.View.OnClickListener`;).

В методе `onCreate()` пропишите:

```
phoneNum = new String[entries];
buttonLabels = new String[entries];
```

и создайте шесть кнопок:

```
 Button button1 = (Button)findViewById(R.id.button1);
             button1.setText(buttonLabels[0]);
             button1.setOnClickListener(this);

             Button button2 = (Button)findViewById(R.id.button2);
             button2.setText(buttonLabels[1]);
             button2.setOnClickListener(this)
…
```

Создайте новый метод `lunchDialer()`:

```
public void launchDialer(String number){
             String numberToDial = "tel:"+number;
             startActivity(new Intent(Intent.ACTION_DIAL,
Uri.parse(numberToDial)));
 }
```

Данный метод отвечает за вызов встроенного телефонного приложения, которое позволяет совершать звонки. Он получает номер телефона в качестве параметра и создает специально отформатированную строку `numberToDial`, которая является универсальным идентификатором (`Uniform Resource Identifier`). Далее создается новый интент со стандартным действием `ACTION_DIAL` (звонок), который используется для старта активности, отвечающей за звонки.

Заполните массивы через метод `populateArrays()` (не забудьте вызвать его в начале):

```
public void populateArrays(){
                 phoneNum[0] = "123-456-78-90";
                 phoneNum[1] = "234-567-89-01";
                 phoneNum[2] = "345-678-90-12";
                 phoneNum[3] = "456-789-01-23";
                 phoneNum[4] = "567-890-12-34";
                 phoneNum[5] = "678-901-23-45";
                  buttonLabels[0] = "Иванов Ваня";
```

```
                buttonLabels[1] = "Петров Петя";
                buttonLabels[2] = "Семеныч Сеня";
                buttonLabels[3] = "Кузнецова Катя";
                buttonLabels[4] = "Смирнова Саша";
                buttonLabels[5] = "Попова Полина";
        }
```

Далее опишите метод `onClick()`:

```
@Override
        public void onClick(View v) {
                switch (v.getId()) {
                case R.id.button1:
                        launchDialer(phoneNum[0]);
                        break;
                case R.id.button2:
                        launchDialer(phoneNum[1]);
                        break;
                case R.id.button3:
                        launchDialer(phoneNum[2]);
                        break;
                case R.id.button4:
                        launchDialer(phoneNum[3]);
                        break;
                case R.id.button5:
                        launchDialer(phoneNum[4]);
                        break;
                  case R.id.button6:
                        launchDialer(phoneNum[5]);
                        break;
                }
```

Приложение можно запускать.

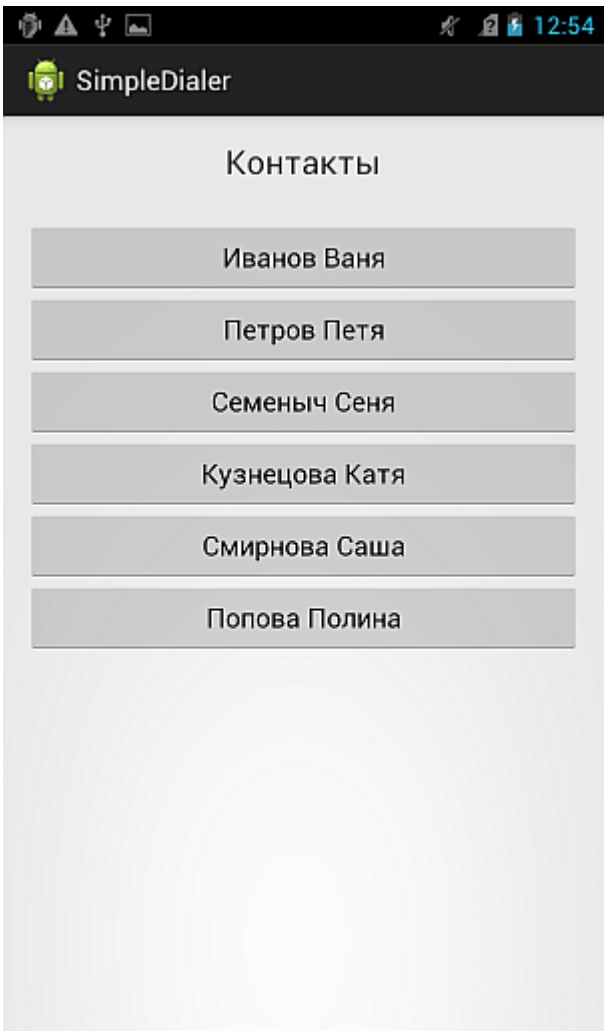


Рис. 10.2. Приложение, запущенное на устройстве

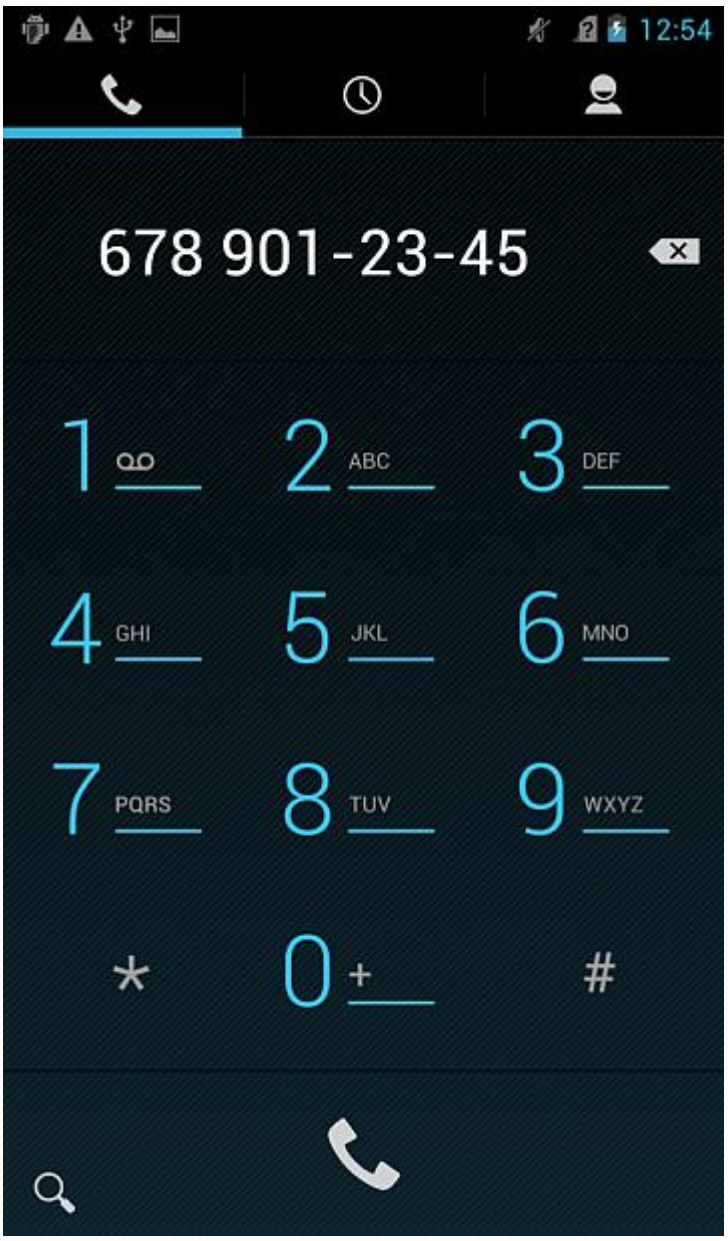


Рис. 10.3. С помощью приложения можно совершать звонки

Задачи для самостоятельной работы

Подумайте, как можно переработать *приложение*, чтобы можно было хранить *произвольное* (не заданное заранее) количество контактов, добавлять и удалять контакты и сопутствующую информацию. Попробуйте реализовать появившиеся идеи. В качестве хранилища данных можно использовать базу данных SQLite. Работа с SQLite описана в первой части курса.

Листинги файлов проекта

```
<?xml version="1.0" encoding="utf-8"?>
   <resources>
      <string name="app_name">SimpleDialer</string>
      <string name="action_settings">Settings</string>
      <string name="hello_world">Hello world!</string>
      <string name="hello">SimpleDialer</string>
      <string name="main_label">My Friends</string>
   </resources>
```

Листинг 10.1. Файл strings.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical"
      android:padding="15dip" >

      <TextView
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_gravity="center"
         android:layout_marginBottom="25dip"
         android:text="@string/main_label"
         android:textSize="22sp" />

      <Button
         android:id="@+id/button1"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

      <Button
         android:id="@+id/button2"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

      <Button
         android:id="@+id/button3"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

      <Button
         android:id="@+id/button4"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

      <Button
         android:id="@+id/button5"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

      <Button
         android:id="@+id/button6"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:textSize="18sp" />

   </LinearLayout>
```

Листинг 10.2. Файл activity_main.xml

```
package mypackage.simpledialer;

   import android.os.Bundle;
   import android.app.Activity;
   import android.view.Menu;
   import android.content.Intent;
   import android.net.Uri;
   import android.view.View;
   import android.view.View.OnClickListener;
```

```
import android.widget.Button;

public class MainActivity extends Activity implements OnClickListener {

	private int entries = 6;
	private String phoneNum[];
	private String buttonLabels[];

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		phoneNum = new String[entries];
		buttonLabels = new String[entries];

		populateArrays();


		Button button1 = (Button)findViewById(R.id.button1);
		button1.setText(buttonLabels[0]);
		button1.setOnClickListener(this);

		Button button2 = (Button)findViewById(R.id.button2);
		button2.setText(buttonLabels[1]);
		button2.setOnClickListener(this);

		Button button3 = (Button)findViewById(R.id.button3);
		button3.setText(buttonLabels[2]);
		button3.setOnClickListener(this);

		Button button4 = (Button)findViewById(R.id.button4);
		button4.setText(buttonLabels[3]);
		button4.setOnClickListener(this);

		Button button5 = (Button)findViewById(R.id.button5);
		button5.setText(buttonLabels[4]);
		button5.setOnClickListener(this);

		Button button6 = (Button)findViewById(R.id.button6);
		button6.setText(buttonLabels[5]);
		button6.setOnClickListener(this);
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}

	public void launchDialer(String number){
		String numberToDial = "tel:"+number;
		startActivity(new Intent(Intent.ACTION_DIAL,
Uri.parse(numberToDial)));
	}
	      phoneNum[0] = "123-456-78-90";
	        phoneNum[1] = "234-567-89-01";
	        phoneNum[2] = "345-678-90-12";
	        phoneNum[3] = "456-789-01-23";
	        phoneNum[4] = "567-890-12-34";
	        phoneNum[5] = "678-901-23-45";
		 buttonLabels[0] = "Иванов Ваня";
		 buttonLabels[1] = "Петров Петя";
		 buttonLabels[2] = "Семеныч Сеня";
```

```
                buttonLabels[3] = "Кузнецова Катя";
                buttonLabels[4] = "Смирнова Саша";
                buttonLabels[5] = "Попова Полина";          }


        @Override
        public void onClick(View v) {
                switch (v.getId()) {

                case R.id.button1:
                        launchDialer(phoneNum[0]);
                        break;

                case R.id.button2:
                        launchDialer(phoneNum[1]);
                        break;

                case R.id.button3:
                        launchDialer(phoneNum[2]);
                        break;

                case R.id.button4:
                        launchDialer(phoneNum[3]);
                        break;

                case R.id.button5:
                        launchDialer(phoneNum[4]);
                        break;

                case R.id.button6:
                        launchDialer(phoneNum[5]);
                        break;
                }
        }
   }
```

Листинг 10.3. Файл MainActivity.java

Практическая работа № 3 Управление воспроизведением аудио

Цель работы

научиться добавлять в приложения, поддерживающие воспроизведения аудио, возможностей:

- управления громкостью и воспроизведением звука с помощью соответствующих кнопок устройства;
- управления аудио фокусом и адекватного реагирования на изменения аудио фокуса, вызванные системой или другими приложениями.

Управление громкостью и воспроизведением

Приложение ориентированное на пользователя должно предоставлять ему привычный способ работы, если дело касается воспроизведения аудио контента, то *пользователь* вправе ожидать возможности управления громкостью звука с помощью клавиш, предусмотренных для этого на самом устройстве. В том числе, если на устройстве досупны кнопки *управляющие* самим процессом воспроизведения звука, то их использование должно вызывать соответствующие действия над аудиопотоком, используемым в приложении.

Для начала необходимо понять какой аудио *поток* будет использовать *приложение*.

Android поддерживает отдельный аудио *поток* для воспроизведения музыки, входящих звонков, предупреждений, напоминаний, системных звуков. Это сделано, чтобы позволить пользователю управлять громкостью каждого потока независимо.

Большинство из этих потоков связывается с системными событиями, таким образом, если *приложение* не является заменой будильника, то аудио будет воспроизводиться в потоке `STREAM_MUSIC` (константа класса `AudioManager`).

По умолчанию кнопки управления громкостью изменяют громкость активного аудио потока, если в данный момент ничего не воспроизводится, регулируется громкость входящего звонка. *Пользователь* игрового приложения или аудиопроигрывателя, нажимая на кнопки управления громкостью, скорей всего ожидает изменения громкости игры или музыки, даже если в данный момент в приложении не воспроизводятся ни какие звуки.

Android предоставляет метод `setVolumeControlStream()` для задания аудио потока, громкость которого будет регулироваться соответствующими кнопками устройства. В метод `onCreate()` активности, в которой выполняется воспроизведение аудио, необходимо добавить строку:

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

На многих устройствах доступны кнопки управления воспроизведением: *play*, *pause*, stop, *skip* и *previous*. Как только *пользователь* нажимает на одну из этих кнопок, система рассылает сообщение о действии `ACTION_MEDIA_BUTTON`. Для обработки таких нажатий в приложении, необходимо в файле манифеста приложения зарегистрировать приемник широковещательных сообщений (BroadcastReceiver) следующим образом:

```
<receiver android:name=".RemoteControlReceiver">
<intent-filter>
<action android:name="android.intent.action.MEDIA_BUTTON"/>
</intent-filter>
</receiver>
</example>
```

В реализации приемника необходимо определять какая кнопка была нажата и вызвала рассылку широковещательных сообщений. Intent-объект содержит эту информацию в ключе `EXTRA_KEY_EVENT`, а *класс* `KeyEvent` содержит набор констант `KEYCODE_MEDIA_*`, для представления каждой возможной медиа кнопки, например, `KEYCODE_MEDIA_PLAY_PAUSE` или `KEYCODE_MEDIA_NEXT` (полный *список* констант можно найти в описании класса `KeyEvent`).

Как получить информацию о нажатии медиа кнопки и выполнить соответствующие действия показывает следующий код:

```
public class RemoteControlReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
if(Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction())) {
KeyEvent event =
(KeyEvent)intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode()) {
// реакция на нажатие кнопки
}
}
}
}
```

Необходимо программно указывать, когда *приложение* может получать информацию о событиях нажатия медиа кнопок.

Сначала необходимо создать экземпляр класса `AudioManager`, который позволит регистрировать и отменять регистрацию приемника событий нажатия медиа кнопок:

```
AudioManager am =
mContext.getSystemService(Context.AUDIO_SERVICE);
```

Далее необходимо зарегистрировать приемник:

```
//начать прослушивание нажатий кнопок
am.registerMediaButtonEventReceiver(RemoteControlReceiver);
```

Зарегистрированный приемник широковещательных сообщений будет единственным приемником всех событий нажатий медиа кнопок, поэтому необходимо отменить регистрацию приемника, как только пропадет неоходимость в нем:

```
//закончить прослушивание нажатий кнопок
am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
```

Обычно приложения отменяют регистрацию большинства своих приемников, когда они становятся неактивными или невидимыми, т.е. при вызове метода `onStop()`. Для приложения, воспроизводящего аудио, обычно важно получать события нажатия кнопок тогда, когда оно невидимо, поэтому лучше регистрировать и снимать регистрацию приемника событий нажатия медиа кнопок, когда *приложение* получает или теряет аудио фокус, соответственно.

Управление аудио фокусом

Для избежания одновременного воспроизведения музыки разными приложениями *Android* использует понятие аудио фокус. Музыку может воспроизводить только то *приложение*, которое в данный момент времени владеет аудио фокусом, поэтому до начала воспроизведения музыки *приложение* должно запросить и получить аудио фокус. Более того оно должно знать, как отслеживать потерю аудио фокуса и соответствующим образом реагировать в случае потери.

Запрос на получение аудио фокуса выполняется вызовом метода `requestAudioFocus()`, который в случае успеха возвращает константу `AUDIOFOCUS_REQUEST_GRANTED`. При этом необходимо указывать какой *поток* используется и какого рода аудио фокус требуется: временный или постоянный. Временный фокус предполагает воспроизведение короткого аудио произведения, например, инструкций *по* навигации. Постоянный фокус предполагает длительное воспроизведение аудио контента, например, прослушивание музыки.

Запрос фокуса необходимо выполнять непосредственно перед началом воспроизведения, например, когда *пользователь* нажимает кнопку *play*. Следующий код демонстрирует *запрос* на постоянный аудио фокус:

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...
//запрос аудио фокуса для воспроизведения
int result = am.requestAudioFocus(afChangeListener,
//используя поток музыки
AudioManager.STREAM_MUSIC,
//запрос постоянного фокуса
AudioManager.AUDIOFOCUS_GAIN);
if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
am.registerMediaButtonEventReceiver(RemoteControlReceiver);
// начало воспроизведения
}
```

Как только заканчивается воспроизведение, необходимо вызвать метод `abandonAudioFocus()`, который сообщает системе, что аудио фокус больше не требуется и отменяет регистрацию соответствующего `AudioManager.OnAudioFocusChangeListener` В случае освобождения временного фокуса, любое приостановленное *приложение* может продолжить воспроизведение.

```
//освобождение аудио фокуса при завершении воспроизведения
am.abandonAudioFocus(afChangeListener);
```

При запросе временного аудио фокуса существует дополнительная *опция*: возможность так называемого "ducking". В обычных условиях *приложение*, теряющее аудио фокус, заглушает воспроизведение, при запросе временного аудио фокуса с возможностью "ducking", работавшему до этого аудио приложению дается возможность лишь приглушить воспроизведение до возвращения фокуса к нему.

```
//запрос аудио фокуса для воспроизведения
int result = am.requestAudioFocus(afChangeListener,
//используя поток музыки
AudioManager.STREAM_MUSIC,
//запрос фокуса
AudioManager.AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK);
if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
// начало воспроизведения
}
```

"Ducking" особенно подходит для приложений, которые используют аудио *поток* периодически, например, для озвучивания рекомендаций водителю.

Приложение может потерять аудио фокус, как ему реагировать на потерю фокуса зависит от способа этой потери. Метод `onAudioFocusChange()` слушателя изменений аудио фокуса, зарегистрированного при запросе аудио фокуса, получает *параметр*, описывающий событие изменения фокуса. Потеря фокуса может быть постоянной или временной с возможностью "ducking" или без нее.

Временная потеря аудио фокуса заставляет *приложение* заглушить воспроизведение аудио потока, но сохраняет состояние воспроизведения. Необходимо следить за изменениями состояния аудио фокуса и быть готовым продолжить воспроизведение после возвращения фокуса.

Если потеря аудио фокуса является постоянной, *приложение* должно корректно завершиться, т.е. остановить воспроизведение, удалить слушателей событий от медиа кнопок и освободить аудио фокус. Для возобновления проигрывания аудио контента необходимо дождаться действия от пользователя, например, нажатия кнопки *play*.

Следующий фрагмент кода описывает приостановку воспроизведения в случае временной потери фокуса, возобновление воспроизведения при возвращении фокуса. В случае постоянной потери выполняется отмена регистрации приемника событий нажатия медиа кнопок и останавливается отслеживание событий изменения фокуса.

```
OnAudioFocusChangeListener afChangeListener = new
OnAudioFocusChangeListener() {
public void onAudioFocusChange(int focusChange) {
if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT){
//приостановить воспроизведение
} else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
//продолжить воспроизведение
} else if (focusChange == AudioManager.AUDIOFOCUS_LOSS) {
am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
am.abandonAudioFocus(afChangeListener);
// прекратить воспроизведение
}
}
};
```

В случае временной потери аудио фокуса с возможностью "ducking" вместо паузы лучше использовать снижение уровня громкости.

"Ducking" это и есть процесс снижения уровня громкости воспроизведения аудио потока, чтобы позволить другому приложению воспроизвести свой аудио *контент*. Следующий фрагмент кода снижает громкость при временной потере фокуса, после возвращения фокуса восстанавливает прежний уровень громкости

```
OnAudioFocusChangeListener afChangeListener = new
OnAudioFocusChangeListener() {
public void onAudioFocusChange(int focusChange) {
if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK) {
// снижение громкости
} else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
// восстановление уровня громкости
}
}
};
```

Взаимодействие с оборудованием для воспроизведения аудио

Пользователи имеют возможность выбора при прослушивании аудио на *Android* устройствах. Большинство устройств имеют встроенный динамик, *выход* для наушников, поддержку A2DP аудио и т.д.

Поведение приложения может зависить от устройства, на которое направлен *вывод* аудио контента. Можно запросить AudioManager определить какое устройство используется для вывода аудио, динамик мобильного устройства, проводные наушники или Bluetooth устройство.

```
if (isBluetoothA2dpOn()) {
//адаптировать вывод для Bluetooth
} else if (isSpeakerphoneOn()) {
//адаптировать вывод для динамика устройства
} else if (isWiredHeadsetOn()) {
// адаптировать вывод для наушников
} else {
//никто его не слышит, аудио все еще играет?
}
```

Когда наушники отсоединяются или Bluetooth устройство становится недоступным, аудио *поток* автоматически перенаправляется на встроенный динамик. Если громкость воспроизведения высокая, то неожиданный шум из динамиков может неприятно удивить, особенно окружающих.

К счастью система рассылает сообщение `ACTION_AUDIO_BECOMING_NOISY`, когда это происходит. Хорошей практикой является привычка регистрировать приемник широковещательных сообщений для получения этого сообщения, каждый раз как *приложение* воспроизводит аудио. В случае с воспроизведением музыки пользователи обычно ожидают приостановки воспроизведения, в случае с играми ожидается существенное понижение уровня громкости.

```
private class NoisyAudioStreamReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
if (AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intent.getAction())) {
// приостановить воспроизведение
}
}
}

private IntentFilter intentFilter =
new IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);
private void startPlayback() {
registerReceiver(myNoisyAudioStreamReceiver, intentFilter);
}
private void stopPlayback() {
unregisterReceiver(myNoisyAudioStreamReceiver);
}
```

Задание:

1. Для тренировки постройте приложение для записи и воспроизведения медиа-контента, добавьте возможности управления громкостью и управления аудио фокусом.
2. Постройте приложение с элементами распознавания речи, пример создания такого приложения: http://www.pandacoder.com/android_speech_recognition/.

Практическая работа № 4. Маркетинг и публикация приложений на Google Play

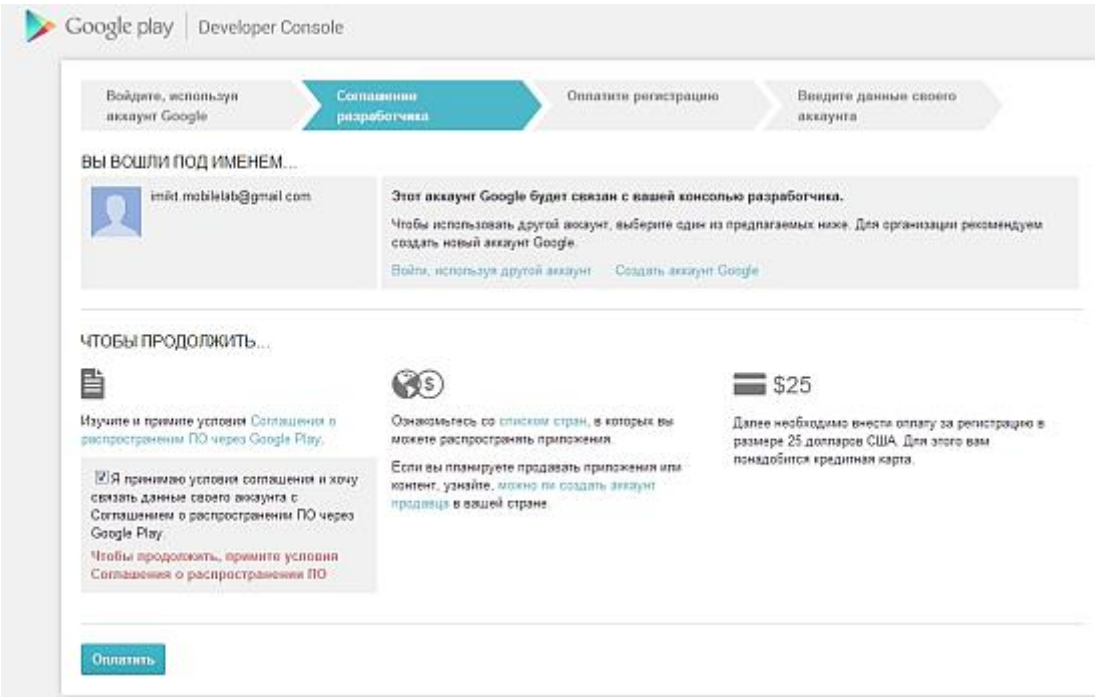
Цель работы:

В данной практической работе мы наглядно отобразим шаги публикации Android приложения на Google Play.

Инструкции по выполнению практической работы

Войдите в Google под своим аккаунтом или создайте новый Google аккаунт.

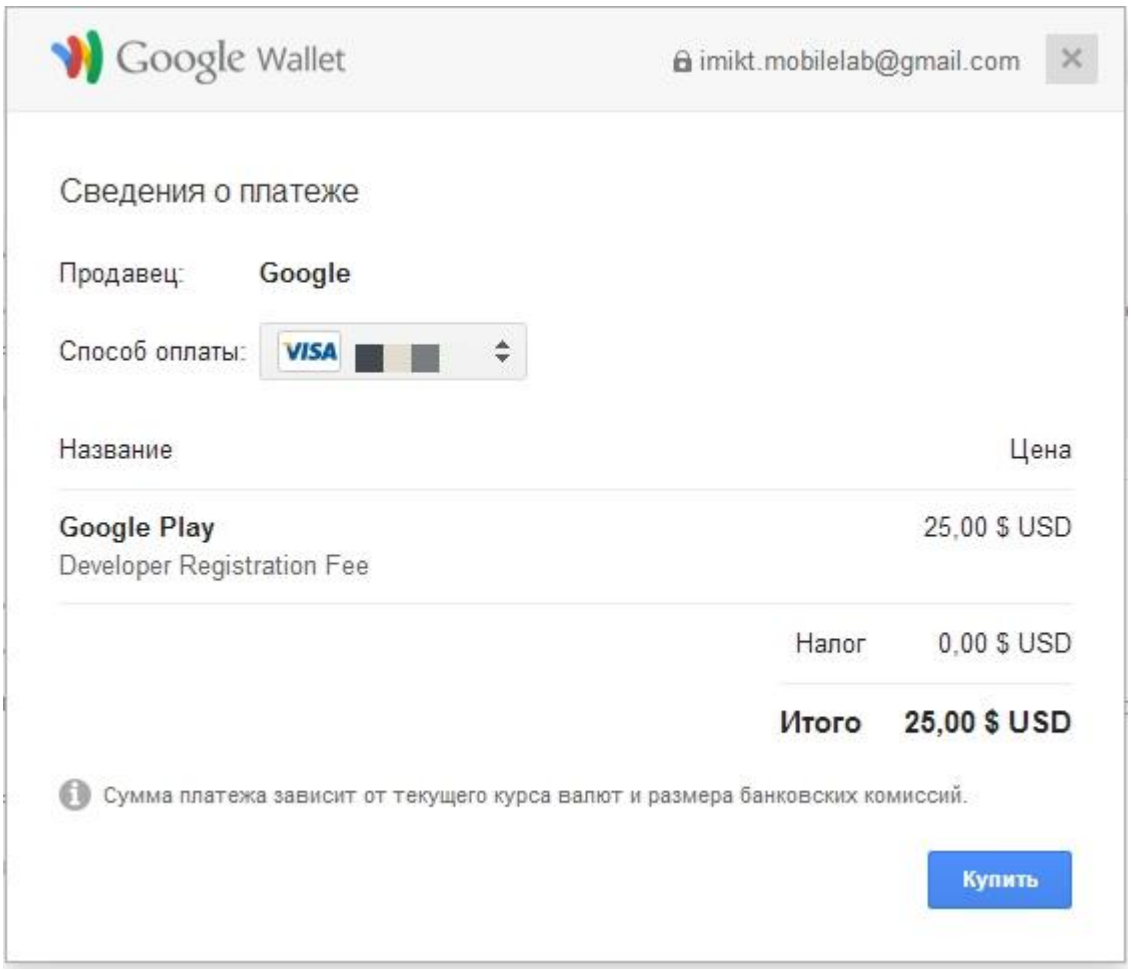
Перейдите *по* ссылке https://play.google.com/apps/publish/signup/ на Google *Play* для последующей регистрации учетной записи разработчика.



увеличить изображение

Обязательно изучите и примите условия Соглашения о распространении *ПО* через Google *Play*.

Уплатить $ 25 в качестве регистрационного сбора, используя Google *Wallet*. Если у вас пока еще нет учетной записи Google *Wallet*, она может быть быстро создана в процессе регистрации.



После уплаты регистрационного сбора вы получите подтверждение на указанный *адрес* электронной почты. Имейте в виду, что обработка платежа может занять до 48 часов. Однако в это время вы можете подготовить apk *файл* приложения и сопутствующие материалы для публикации.

В новом окне введите данные аккаунта разработчика, указав имя, *адрес* электронной почты и телефон.

После совершения всех вышеуказанных действий вы попадете в *Консоль* разработчика, где сможете публиковать и настраивать ваши приложения.

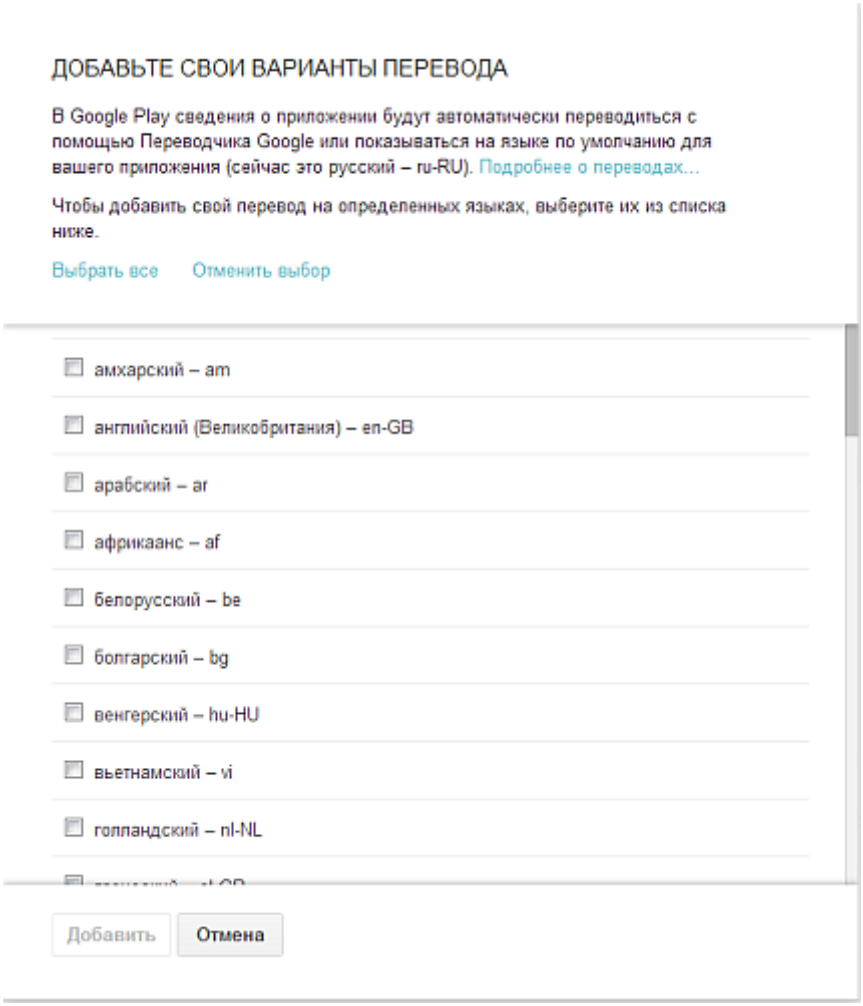
Добавим новое *приложение*.

Выберем язык *по* умолчанию и введем данные о программном продукте:

Введите название приложения;

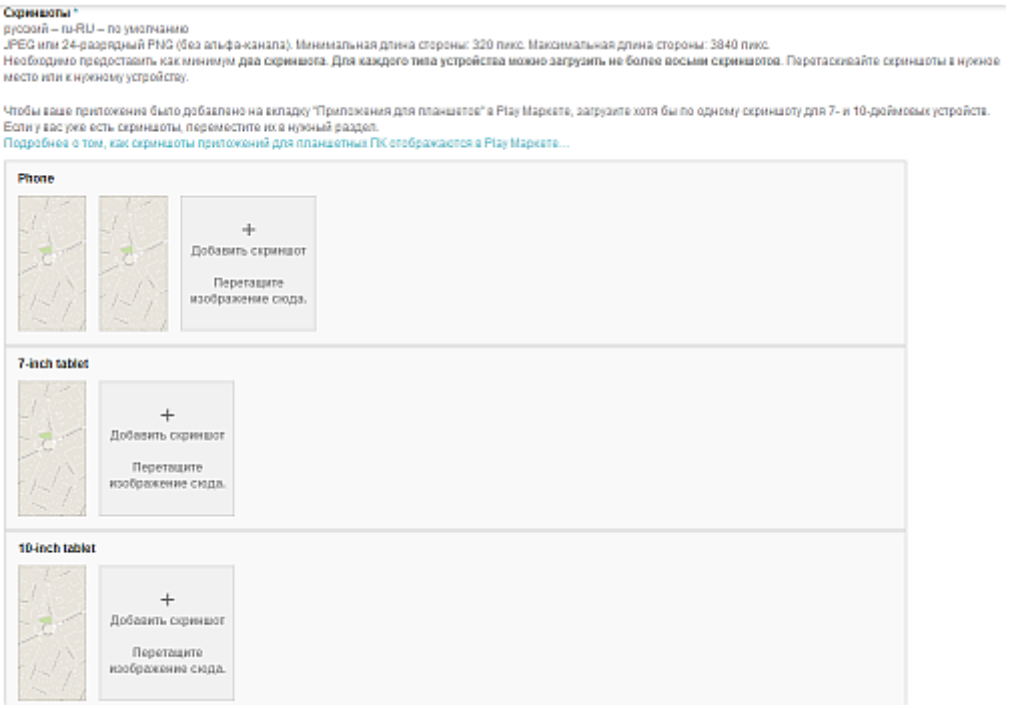
Введите лаконичное описание приложения;

Google *Play* позволяет автоматически переводить описание приложения на другие языки. Не забудьте выбрать языки для перевода, имея в виду страны, в которых будет распространяться ваше *приложение*.



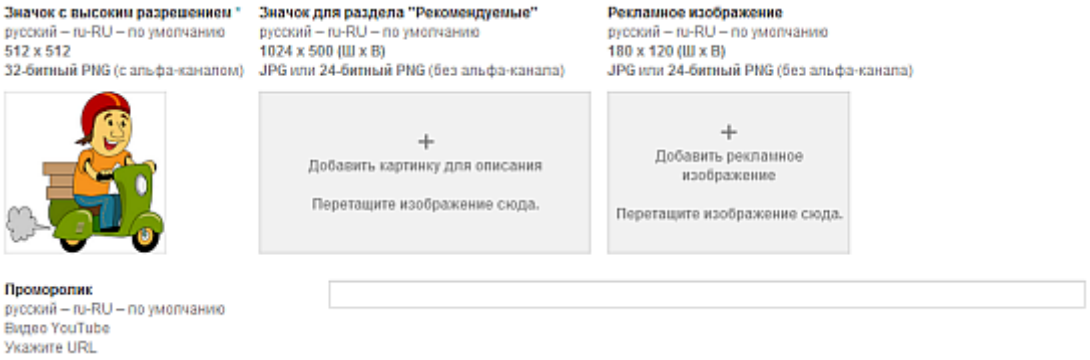
увеличить изображение

После ввода текстовой информации можно приступать к загрузке графических изображений. Необходимо подготовить как *минимум* два скриншота. Для каждого типа устройства можно загрузить не более восьми скриншотов.



увеличить изображение

Также обязательно загрузить значок с высоким разрешением. Характеристики: 512 x 512 пикселей, 32-битный *PNG* с альфа-каналом; максимальный размер 1024 КБ.



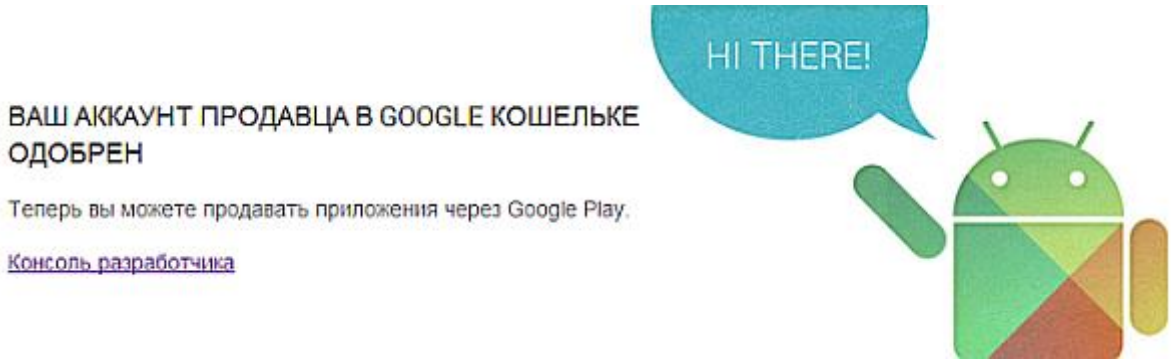
увеличить изображение

Кроме того, вы можете добавить ссылку на промо-ролик приложения.

Укажите свойства приложения. Выберете тип приложения, категорию и возрастные ограничения.

Перейдите на вкладку **Цены и распространение**.

Определитесь, будет ли ваше *приложение* платным или бесплатным. Если вы решили указать цену на *приложение*, то необходимо настроить аккаунт продавца.



Укажите цену приложения. Можно установить цены для других стран вручную или автоматически конвертировать цену *по* умолчанию согласно текущему курсу валют и налоговым ставкам.

Выберете страны, в которых будет доступно ваше *приложение*.

Осталось только загрузить apk *файл* приложения, получит лицензионный *ключ* для защиты своих приложений от несанкционированного распространения и нажать кнопку **Publish**.

Создайте значок, связывающий пользователей с вашей продукцией с помощью Google *Play Badge Generator*.

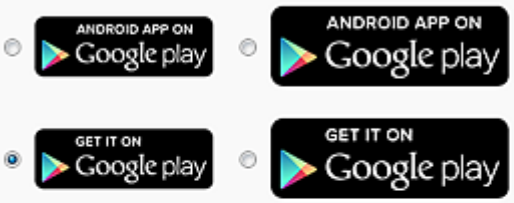
(http://developer.android.com/distribute/googleplay/promote/badges.html)

Language: Русский

Package name: Courier App clear

or

Publisher name: Example, Inc.



Build my badge

Copy and paste this HTML into your web site:

```
<a href="https://play.google.com/store/apps/details?id=Courier App">
  <img alt="Get it on Google Play"
       src="https://developer.android.com/images/brand/ru_generic_rgb_wo_45.png" />
</a>
```

Try it out: