**Министерство образования Республики Беларусь**

**Учреждение образования**
**«Гомельский государственный технический**
**университет имени П. О. Сухого»**

**Кафедра «Информационные технологии»**

**И. Л. Стефановский, Т. С. Семенченя**

# РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ANDROID

**ПРАКТИКУМ**
**по выполнению лабораторных работ**
**по одноименной дисциплине для студентов**
**специальности 1-40 05 01 «Информационные**
**системы и технологии (по направлениям)»**
**дневной и заочной форм обучения**

**Гомель 2021**

УДК 004.45(075.8)
ББК 32.973.2я73
С79

*Рекомендовано научно-методическим советом*
*факультета автоматизированных и информационных систем*
*ГГТУ им. П. О. Сухого*
(*протокол № 10 от 04.05.2020 г.*)

Рецензент: зав. каф. «Промышленная электроника» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *Ю. В. Крышнев*

**Стефановский, И. Л.**

С79 Разработка мобильных приложений для Android : практикум по выполнению лаборатор. работ по одноим. дисциплине для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» днев. и заоч. форм обучения / И. Л. Стефановский, Т. С. Семенченя. – Гомель : ГГТУ им. П. О. Сухого, 2021. – 53 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Практикум знакомит студентов с архитектурой, разработкой мобильных Android-приложений, а также содержит материал для выполнения лабораторных занятий.

Для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения.

**УДК 004.45(075.8)**
**ББК 32.973.2я73**

## Оглавление

# ВВЕДЕНИЕ

Целью дисциплины «Разработка мобильных приложений для *Android*» является формирование у студентов теоретических знаний о современных мобильных устройствах на платформе *Android*, приемах, методах и технологиях разработки программ для этих устройств, обучение студентов основным принципам программирования мобильных систем; формирование практических навыков создания современных мобильных приложений; формирование у студентов теоретических знаний и практических навыков в области разработки мобильных приложений.

Основными задачами дисциплины являются следующие:

– усвоение основных понятий платформы *Android*;

– усвоение языковых средств, используемых для создания мобильных приложений.

– овладение навыками работы с основными инструментальными средствами конструирования и создания прикладных программных продуктов для платформы *Android* различной сложности, используя различные технологии;

– приобретение студентами практических навыков решения задач с использованием современных методов программирования;

– обучение студентов самостоятельной работе и хорошей ориентации в области технологий и программных комплексов.

Лабораторный практикум предназначен для выполнения лабораторных работ по курсу «Разработка мобильных приложений для *Android*» для специальности 1-40 05 01 "Информационные системы и технологии (по направлениям)".

Целями данного практикума являются:

– изучение студентами методов и инструментальных средств разработки мобильных приложений;

– освоение основных приемов и технологий разработки мобильных приложений;

– изучение методов эффективного программирования взаимодействия мобильных приложений с удаленными системами и другими средствами обеспечения информационной поддержки решаемых задач;

– приобретение студентами практических навыков по разработке, адаптации мобильных приложений.

–

## Лабораторная работа № 1. Введение в программирование для платформы *Android*

**Цель работы**: изучить основы построения *Android* приложения, создание проекта *Android*. Построение макета пользовательского интерфейса. Запуск на устройстве.

### Теоретические сведения

ОС *Android* – операционная система для мобильных телефонов, планшетных компьютеров и нетбуков, основанная на ядре *Linux*. Изначально разрабатывалась компанией *Android Inc*., которую затем купила компания *Google*. Впоследствии, компания *Google* инициировала создание альянса компаний *Open Handset Alliance* (*OHA*), занимающегося поддержкой и дальнейшим развитием платформы. Первая версия ОС *Google Android* вышла в сентябре 2008 года. В конце 2010 года ОС *Android* стала самой продаваемой ОС для смартфонов.

ОС *Android* – это набор открытого программного обеспечения для мобильных устройств от компании *Google*, в состав которого входит операционная система и комплект базовых межплатформенных приложений [1]. Приложения для ОС *Android* являются программами в байт-коде для виртуальной машины *Dalvik Virtual Machine*, которая является частью мобильной платформы *Android*.

Для разработки приложений под платформу *Android* используется набор инструментов и библиотек *API – Android SDK* предназначенный для компьютеров с архитектурой процессора *x*86 под операционными системами *Windows XP*, *Windows Vista*, *Windows* 7, *Mac OS X* (10.4.8 или выше) и *Linux*. Для разработки требуется среда исполнения *Java Runtime Environment* (*JRE*), комплект разработчика *Java Development Kit* (*JDK*), среда разработки *Eclipse* [2] и *Android SDK* [3].

Разработку приложений для ОС *Android* можно вести на языке *Java*. Для этой цели существует плагин для среды разработки *Eclipse* – «*Android Development Tools*» (*ADT*). Кроме того, существуют плагины, облегчающие разработку *Android*-приложений в средах разработки *IntelliJ IDEA* и *NetBeans IDE*. *MonoDroid SDK* позволяет писать для ОС *Android* на *C#* и других языках. На рис. 1 представлена диаграмма основных компонентов операционной системы *Android*.

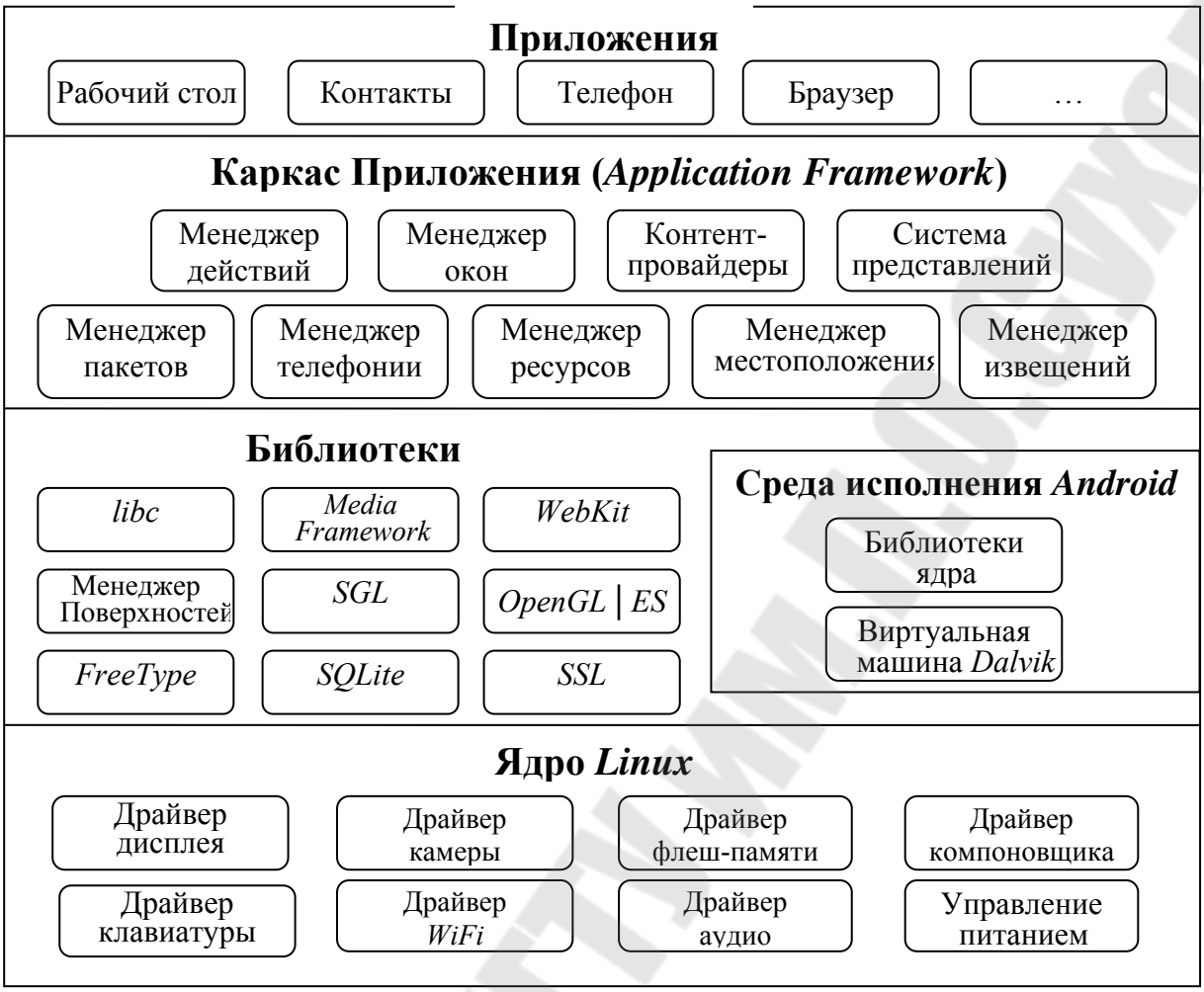


*Рис. 1.1.* Основные компоненты ОС *Android*

## Задание на лабораторную работу

Разработать *Android* приложение. Приложение должно содержать одно текстовое поле и кнопку. При нажатии на кнопку выводить в текстовое поле фамилию студента. Выполнить запуск приложения на эмуляторе.

## Контрольные вопросы

1. Введение в программирование для платформы Android.
2. Виды программ, средства и технологии разработки, структура программы.
3. Основы построения Android приложения.
4. Создание проекта Android.
5. Построение макета пользовательского интерфейса.
6. Запуск на устройстве.

7. Подключение устройства.
8. Настройка устройства для разработки.
9. Добавление значка.
10. Добавление ресурсов в проект.
11. Создание строковых ресурсов.
12. Среды разработки.

## Лабораторная работа № 2. Использование шаблона проектирование MVC в разработке Android приложений

**Цель работы:** изучить архитектуру "Модель-Представление-Контроллер" в *Android*.

### Теоретические сведения

Приложения *Android* строятся на базе архитектуры, называемой «Модель-Представление-Контроллер», или сокращенно *MVC* (*Model-View-Controller*). Согласно канонам *MVC*, каждый объект приложения должен быть объектом модели, объектом представления или объектом контроллера.

*Объект модели* содержит данные приложения и «бизнес-логику». Классы модели обычно проектируются для моделирования сущностей, с которыми имеет дело приложение — пользователь, продукт в магазине, фотография на сервере, вопрос «да/нет» и т. д. Объекты модели ничего не знают о пользовательском интерфейсе; их единственной целью является хранение и управление данными.

Уровень модели *GeoQuiz* состоит из класса *TrueFalse*.

*Объекты представлений* умеют отображать себя на экране и реагировать на ввод пользователя — например, касания. Простейшее правило: если вы видите что-то на экране — значит, это представление. *Android* предоставляет широкий набор настраиваемых классов представлений.

Создадим *layout*-файл, например ***myscreen.xml*** (рис. 2.1). Для этого выделим папку *res*/*layout* в нашем проекте и нажмем кнопку создания нового файла

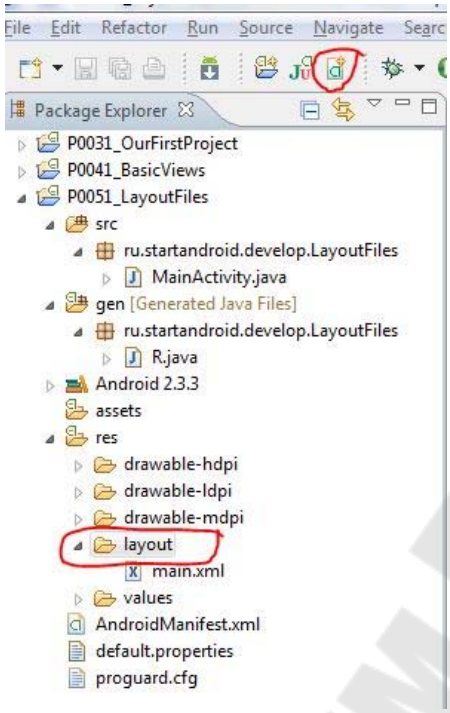


*Рис. 2.1.* Создание *layout*-файла

Откроется визард (рис. 2.2):

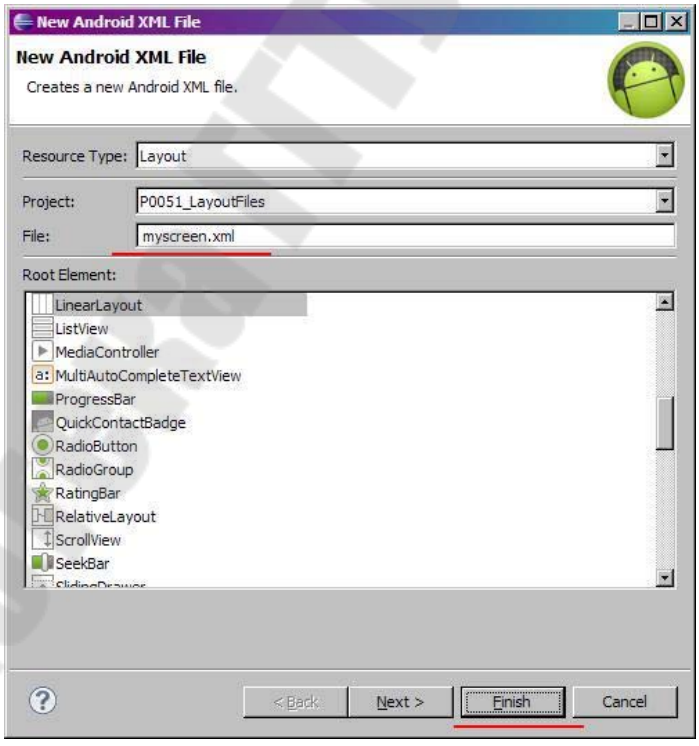


*Рис. 2.2.* Визард создания *layout*-файла

В поле *File* вводим имя файла: *myscreen.xml* и жмем *Finish*.

Новый *layout*-файл должен сразу открыться. Добавим *TextView* и через *Properties* изменим его текст на: «Этот экран описан не в *main.xml*, а в *myscreen.xml*» (рис. 2.3).

В приложениях *Android* классы моделей обычно создаются разработчиком для конкретной задачи. Все объекты модели в вашем приложении составляют его *уровень модели*.

При этом *Eclipse* будет подчеркивать этот текст желтым цветом и ругаться примерно так: *[I18N] Hardcodedstring "...", shoulduse @stringresource*.

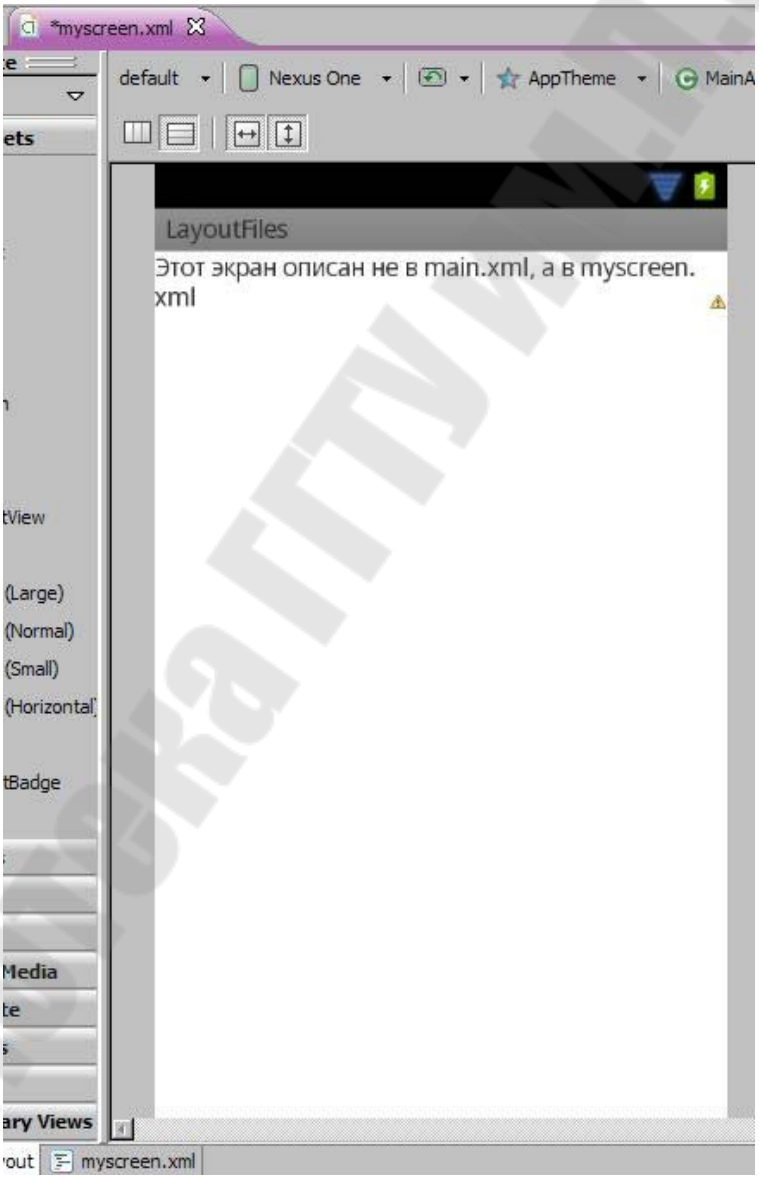


*Рис. 2.3.* Вид *layout* в среде разработке

Обязательно сохраняем. Чтобы в *R.java* появилась новая константа для этого файла – *R.layout.myscreen*.

Теперь настроим так, чтобы *Activity* использовало новый файл *myscreen.xml*, а не *main.xml*. Откроем *MainActivity.java* и поменяем аргумент метода *setContentView*. Замените «*R.layout.main*», на «*R.layout.myscreen*» (*ID* нового *layout*-файла).

Сохраняем, запускаем приложение. Видим (рис. 2.4), что теперь оно отображает содержимое из *myscreen.xml*, т.к. мы явно ему это указали в методе *setContentView*, который выполняется при создании (*onCreate*) *Activity*

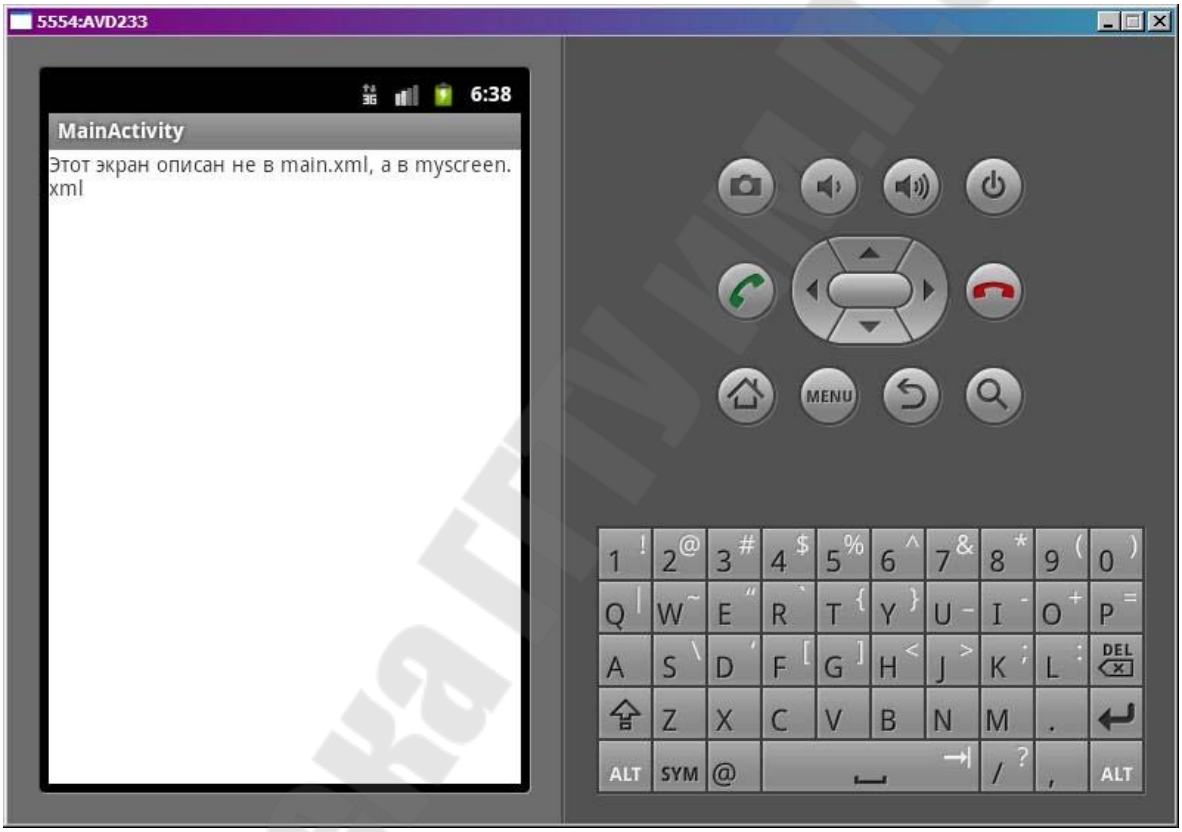


*Рис. 2.4*. Визард создания *layout*-файла

## Задание на лабораторную работу

Разработать *Android* приложение с использованием архитектуры "Модель-Представление-Контроллер", выполняющее ввод данных, вывод и редактирование в соответствии с вариантом. Для выполнения каждого пункта задания (*a-d*) использовать отдельную *Activity* и модель. Выполнить запуск приложения на эмуляторе.

## Варианты заданий

1. ***Student***: *id*, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон,

Создать массив объектов. Вывести:

*a*) список студентов заданного факультета;

*b*) списки студентов для каждого факультета и курса;

*c*) список студентов, родившихся после заданного года;

*d*) список учебной группы.

2. ***Customer***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной

карточки, Номер банковского счета.

Создать массив объектов. Вывести:

*a*) список покупателей в алфавитном порядке;

*b*) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. ***Patient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.

Создать массив объектов. Вывести:

*a*) список пациентов, имеющих данный диагноз;

*b*) список пациентов, номер медицинской карты у которых находится в заданном интервале.

4. ***Abiturient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

*a*) список абитуриентов, имеющих неудовлетворительные оценки;

*b*) список абитуриентов, средний балл у которых выше заданного;

*c*) выбрать заданное число *n* абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

5. ***Book***: *id*, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Переплет.

Создать массив объектов. Вывести:

*a*) список книг заданного автора;

*b*) список книг, выпущенных заданным издательством;

*c*) список книг, выпущенных после заданного года.

6. ***House***: *id*, Номер квартиры, Площадь, Этаж, Количество комнат, Улица,

Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

*a*) список квартир, имеющих заданное число комнат;

*b*) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;

*c*) список квартир, имеющих площадь, превосходящую заданную.

7. ***Phone***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки,

Дебет, Кредит, Время городских и междугородных разговоров.

Создать массив объектов. Вывести:

*a*) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;

*b*) сведения об абонентах, которые пользовались междугородной связью;

*c*) сведения об абонентах в алфавитном порядке.

8. ***Car***: *id*, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.

Создать массив объектов. Вывести:

*a*) список автомобилей заданной марки;

*b*) список автомобилей заданной модели, которые эксплуатируются больше *n* лет;

*c*) список автомобилей заданного года выпуска, цена которых больше указанной.

9. ***Product***: *id*, Наименование, *UPC*, Производитель, Цена, Срок хранения, Количество.

Создать массив объектов. Вывести:

*a*) список товаров для заданного наименования;

*b*) список товаров для заданного наименования, цена которых не превосходит заданную;

*c*) список товаров, срок хранения которых больше заданного.

10. ***Train***: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс).

Создать массив объектов. Вывести:

*a*) список поездов, следующих до заданного пункта назначения;

*b*) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

*c*) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

11. ***Bus***: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

*a*) список автобусов для заданного номера маршрута;

*b*) список автобусов, которые эксплуатируются больше 10 лет;

*c*) список автобусов, пробег у которых больше 100000 км.

12. ***Airlines***: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.

Создать массив объектов. Вывести:

*a*) список рейсов для заданного пункта назначения;

*b*) список рейсов для заданного дня недели;

*c*) список рейсов для заданного дня недели, время вылета для которых больше заданного.

## Контрольные вопросы

1. Использование шаблона проектирование *MVC* в разработке *Android* приложений. Архитектура "Модель-Представление-Контроллер" и *Android*.
2. Преимущества *MVC*.
3. Обновление уровня представления.
4. Обновление уровня контроллера.
5. Иерархия представлений.
6. Атрибуты виджетов.
7. Объекты *View*.
8. Ссылки на ресурсы в *XML*.
9. Добавление слушателя для компонентов.
10. Анонимные внутренние классы.

## Лабораторная работа № 3. Жизненный цикл *Activity*

**Цель работы**: изучить жизненный цикл *Activity*.

### Теоретические сведения

Управление жизненным циклом операций путем реализации методов обратного вызова имеет важное значение при разработке надежных и гибких приложений. На жизненный цикл операции напрямую влияют его связи с другими операциями, его задачами и стеком переходов назад.

Существует всего три состояния операции:

*Возобновлена*

Операция выполняется на переднем плане экрана и отображается для пользователя. (Это состояние также иногда называется «Выполняется».)

*Приостановлена*

На переднем фоне выполняется другая операция, которая отображается для пользователя, однако эта операция по-прежнему не скрыта. То есть поверх текущей операции отображается другая операция, частично прозрачная или не занимающая полностью весь экран. Приостановленная операция полностью активна (объект *Activity* по-прежнему находится в памяти, в нем сохраняются все сведения о состоянии и информация об элементах, и он также остается связанным с диспетчером окон), однако в случае острой нехватки памяти система может завершить ее.

*Остановлена*

Операция полностью перекрывается другой операцией (теперь она выполняется в фоновом режиме). Остановленная операция по-прежнему активна (объект *Activity* по-прежнему находится в памяти, в нем сохраняются все сведения о состоянии и информация об элементах, но объект больше *не* связан с диспетчером окон). Однако операция больше не видна пользователю, и в случае нехватки памяти система может завершить ее.

Если операция приостановлена или полностью остановлена, система может очистить ее из памяти путем завершения самой операции (с помощью метода *finish*()) или просто завершить ее процесс. В случае повторного открытия операции (после ее завершения) ее потребуется создать полностью.

При переходе операции из одного вышеописанного состояния в другое, уведомления об этом реализуются через различные методы обратного вызова. Все методы обратного вызова представляют собой привязки, которые можно переопределить для выполнения подходящего действия при изменении состояния операции.

Примечание. При реализации этих методов жизненного цикла всегда вызывайте реализацию суперкласса, прежде чем выполнять какие-либо действия, как показано в примерах выше.

Указанная ниже базовая операция включает каждый из основных методов жизненного цикла:

```
public class ExampleActivity extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// The activity is being created.
}
@Override
protected void onStart() {
super.onStart();
// The activity is about to become visible.
}
@Override
protected void onResume() {
super.onResume();
// The activity has become visible (it is now "resumed").
}
@Override
protected void onPause() {
super.onPause();
// Another activity is taking focus (this activity is about to be
"paused").
}
@Override
protected void onStop() {
super.onStop();
// The activity is no longer visible (it is now "stopped")
}
@Override
```

```
protected void onDestroy() {
super.onDestroy();
// The activity is about to be destroyed.
}
}
```

Вместе все эти методы определяют весь жизненный цикл операции. С помощью реализации этих методов можно отслеживать три вложенных цикла в жизненном цикле операции.

Весь жизненный цикл операции происходит между вызовом метода *onCreate*() и вызовом метода *onDestroy*(). Ваша операция должна выполнить настройку «глобального» состояния (например, определение макета) в методе *onCreate*(), а затем освободить все оставшиеся в *onDestroy*() ресурсы. Например, если в вашей операции имеется поток, выполняющийся в фоновом режиме, для загрузки данных по сети, операция может создать такой поток в методе *onCreate*(), а затем остановить его в методе *onDestroy*().

Видимый жизненный цикл операции происходит между вызовами методов *onStart*() и *onStop*(). В течение этого времени операция отображается на экране, где пользователь может взаимодействовать с ней. Например, метод *onStop*() вызывается в случае, когда запускается новая операция, а текущая больше не отображается. В промежутке между вызовами этих двух методов можно сохранить ресурсы, необходимые для отображения операции для пользователя. Например, можно зарегистрировать объект *BroadcastReceiver* в методе *onStart*() для отслеживания изменений, влияющих на пользовательский интерфейс, а затем отменить его регистрацию в методе *onStop*(), когда пользователь больше не видит отображаемого. В течение всего жизненного цикла операции система может несколько раз вызывать методы *onStart*() и *onStop*(), поскольку операция то отображается для пользователя, то скрывается от него.

Жизненный цикл операции, выполняемый на переднем плане, происходит между вызовами методов *onResume*() и *onPause*(). В течение этого времени операция выполняется на фоне всех прочих операций и отображается для пользователя. Операция может часто уходить в фоновый режим и выходить из него – например, метод *onPause*() вызывается при переходе устройства в спящий режим или при появлении диалогового окна. Поскольку переход в это состояние может выполняться довольно часто, код в этих двух методах должен

быть легким, чтобы не допустить медленных переходов и не заставлять пользователя ждать.

Указывается, может ли система в любое время завершить процесс, содержащий операцию, *после возвращения метода* без выполнения другой строки кода операции. Для трех методов в этом столбце указано «Да»: (*onPause*(), *onStop*() и *onDestroy*()). Поскольку метод *onPause*() является первым из этих трех после создания операции, метод *onPause*() является последним, который гарантированно будет вызван перед тем, как процесс *можно будет* завершить; если системе потребуется срочно восстановить память в случае аварийной ситуации, то методы *onStop*() и *onDestroy*() вызвать не удастся. Поэтому следует воспользоваться *onPause*(), чтобы записать критически важные данные (такие как правки пользователя) в хранилище постоянных данных. Однако следует внимательно подходить к выбору информации, которую необходимо сохранить во время выполнения метода *onPause*(), поскольку любая блокировка процедур в этом методе может вызвать блокирование перехода к следующей операции и тормозить работу пользователя.

Методы, для которых в столбце Завершаемый указано «Нет», защищают процесс, содержащий операцию, от завершения сразу с момента их вызова. Поэтому завершить операцию можно в период между возвратом *onPause*() и вызовом *onResume*(). Его снова не удастся завершить, пока снова не будет вызван и возвращен *onPause*().

Примечание. Операцию, которую технически невозможно завершить в соответствии с определением, по-прежнему может завершить система, однако это может произойти только в чрезвычайных ситуациях, когда нет другой возможности.

## Задание на лабораторную работу

Разработать *Android* приложение, выполняющее ввод данных, вывод и редактирование в соответствии с вариантом. При запуске программы выполнять чтение данных из файла. При окончании работы программы сохранять данные в файл. Из изменении состояния жизненного цикла *Activity* записывать информацию в журнал. Выполнить запуск приложения на эмуляторе.

## Варианты заданий

1. ***Student***: *id*, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон,

Создать массив объектов. Вывести:

*a*) список студентов заданного факультета;

*b*) списки студентов для каждого факультета и курса;

*c*) список студентов, родившихся после заданного года;

*d*) список учебной группы.

2. ***Customer***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета.

Создать массив объектов. Вывести:

*a*) список покупателей в алфавитном порядке;

*b*) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. ***Patient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.

Создать массив объектов. Вывести:

*a*) список пациентов, имеющих данный диагноз;

*b*) список пациентов, номер медицинской карты у которых находится в заданном интервале.

4. ***Abiturient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

*a*) список абитуриентов, имеющих неудовлетворительные оценки;

*b*) список абитуриентов, средний балл у которых выше заданного;

*c*) выбрать заданное число *n* абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

5. ***Book***: *id*, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Переплет.

Создать массив объектов. Вывести:

*a*) список книг заданного автора;

*b*) список книг, выпущенных заданным издательством;

*c*) список книг, выпущенных после заданного года.

6. ***House***: *id*, Номер квартиры, Площадь, Этаж, Количество комнат, Улица,

Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

*a*) список квартир, имеющих заданное число комнат;

*b*) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;

*c*) список квартир, имеющих площадь, превосходящую заданную.

7. ***Phone***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки,

Дебет, Кредит, Время городских и междугородных разговоров.

Создать массив объектов. Вывести:

*a*) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;

*b*) сведения об абонентах, которые пользовались междугородной связью;

*c*) сведения об абонентах в алфавитном порядке.

8. ***Car***: *id*, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.

Создать массив объектов. Вывести:

*a*) список автомобилей заданной марки;

*b*) список автомобилей заданной модели, которые эксплуатируются больше *n* лет;

*c*) список автомобилей заданного года выпуска, цена которых больше указанной.

9. ***Product***: *id*, Наименование, *UPC*, Производитель, Цена, Срок хранения, Количество.

Создать массив объектов. Вывести:

*a*) список товаров для заданного наименования;

*b*) список товаров для заданного наименования, цена которых не превосходит заданную;

*c*) список товаров, срок хранения которых больше заданного.

10. ***Train***: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс).

Создать массив объектов. Вывести:

*a*) список поездов, следующих до заданного пункта назначения;

*b*) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

*c*) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

11. ***Bus***: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

*a*) список автобусов для заданного номера маршрута;

*b*) список автобусов, которые эксплуатируются больше 10 лет;

*c*) список автобусов, пробег у которых больше 100000 км.

12. ***Airlines***: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.

Создать массив объектов. Вывести:

*a*) список рейсов для заданного пункта назначения;

*b*) список рейсов для заданного дня недели;

*c*) список рейсов для заданного дня недели, время вылета для которых больше заданного.

## Контрольные вопросы

1. Жизненный цикл *Activity*.
2. Регистрация событий жизненного цикла *Activity*.
3. Создание сообщений в журнале.
4. Использование *LogCat*.
5. Повороты и жизненный цикл активности.
6. Конфигурации устройств и альтернативные ресурсы.
7. Создание макета для альбомной ориентации.
8. Сохранение данных между поворотами.
9. Переопределение *onSaveInstanceState*(*Bundle*).

## Лабораторная работа № 4 .Интерфейс пользователя в *Android*

**Цель работы:** изучить разметки и использование элементов управления (*Button*, *TextView*, *EditText*, *ListView*) в *Android*.

## Теоретические сведения

Описание интерфейса пользователя выполняется в виде *layout* файлов.

***Layout*-файл в виде *XML***

Открыв файл *main.xml* (рис. 4.1), вы видите его визуальное представление. Т.е. некий предпросмотр, как это будет выглядеть на экране. Снизу вы можете видеть две вкладки –***GraphicalLayout*** и ***main.xml***. Откройте *main.xml*.


*Рис. 4.1*. Файл main.xml

Мы видим достаточно читабельное *xml*-описание всех *View* нашего *layout*-файла. Названия *xml*-элементов - это классы *View*-элементов, *xml*-атрибуты - это параметры *View*-элементов, т.е. все те параметры, что мы меняем через вкладку *Properties*.

Также вы можете вносить изменения прямо сюда и изменения будут отображаться в *GraphicalLayout*. Например изменим текст у *TextView*. Вместо ссылки на константу, вставим свой текст «Какой-то текст» (рис. 4.2).

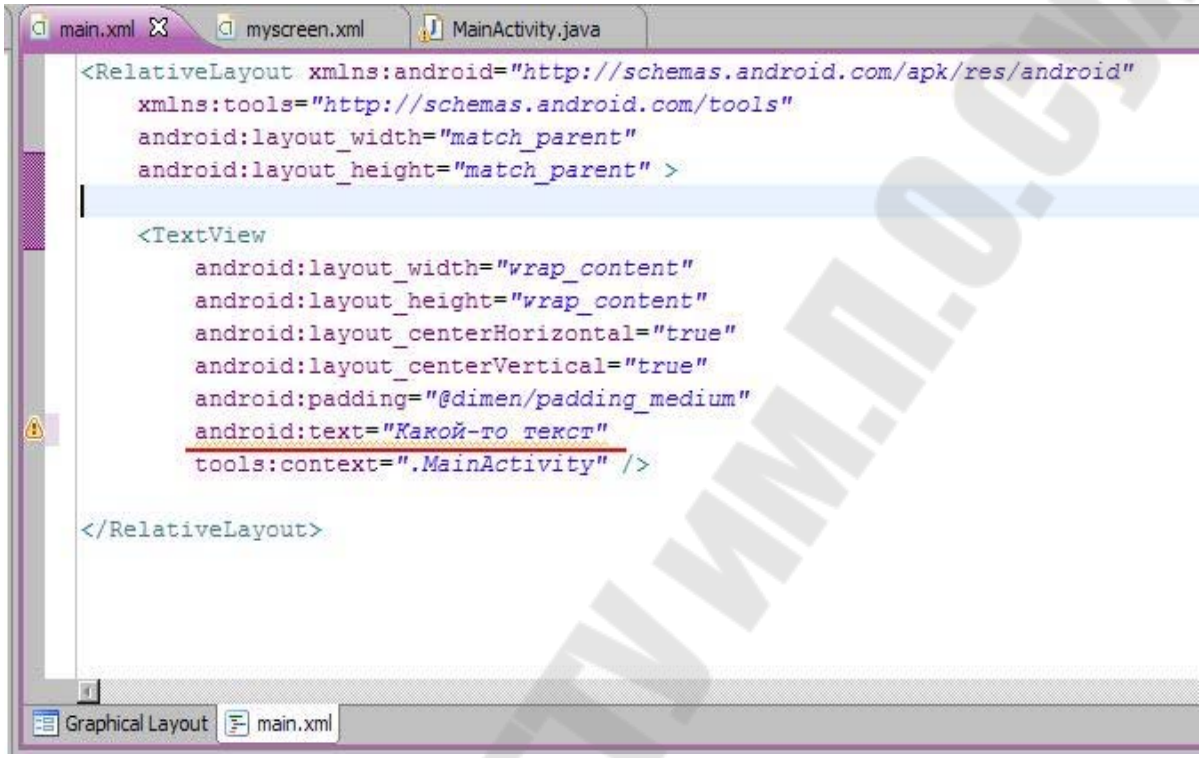


*Рис. 4.2.* Измененный *Layout* файл

Сохраняем. Открываем *GraphicalLayout* и наблюдаем изменения (рис. 4.3). Обычно дают содержание *layout*-файлов именно в *xml* виде. Это удобно – вы можете просто скопировать фрагмент и использовать, и не надо вручную добавлять *View*-элементы, бегать по *Properties* и настраивать все руками.

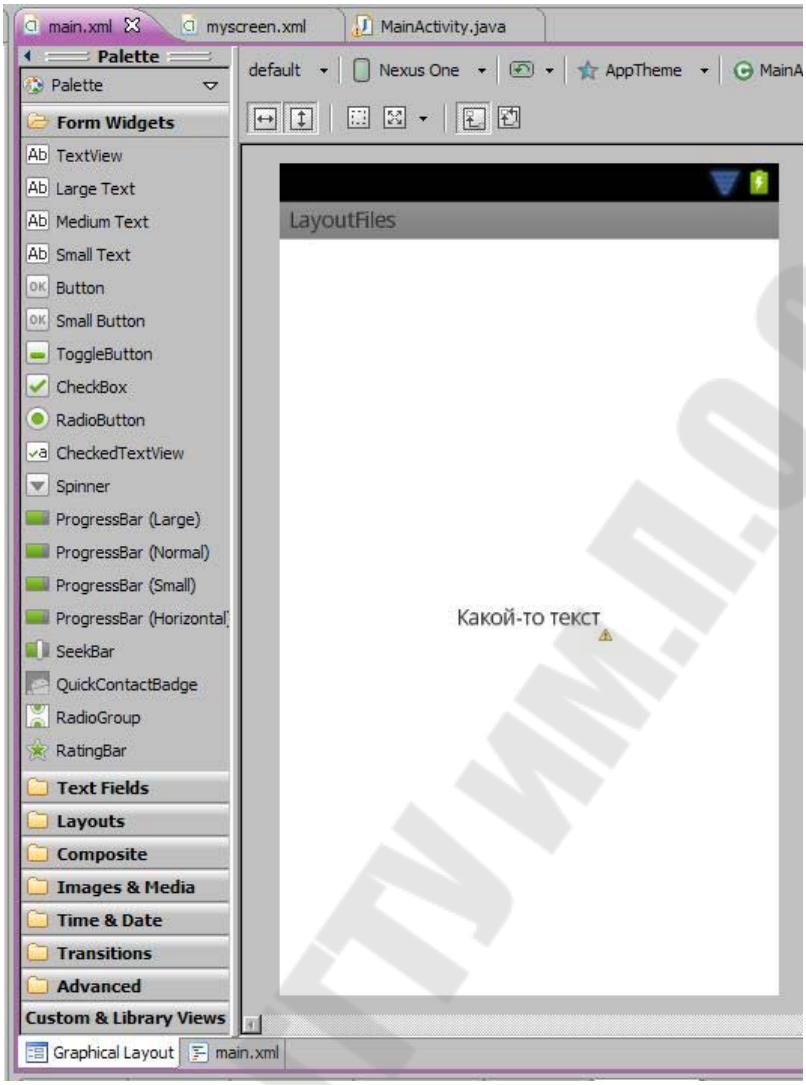


*Рис. 4.3.* Графическое представление разработанного *Layout* файла

### ***Layout*-файл при смене ориентации экрана**

По умолчанию мы настраиваем *layout*-файл под вертикальную ориентацию экрана. Но что будет если мы повернем смартфон и включится горизонтальная ориентация ? Давайте смотреть.

Изменим *myscreen.xml*. Добавим вертикальный ряд кнопок и изменим надпись.

*Xml*-код (вы можете скопировать его и вставить в ваш файл):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="match_parent"
```

```
android:layout_height="match_parent">
<TextView
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Вертикальная ориентация экрана">
</TextView>
<LinearLayout
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:id="@+id/linearLayout1"
android:orientation="vertical">
<Button
android:text="Button1"
android:id="@+id/button1"
android:layout_height="100dp"
android:layout_width="100dp">
</Button>
<Button
android:text="Button2"
android:id="@+id/button2"
android:layout_height="100dp"
android:layout_width="100dp">
</Button>
<Button
android:text="Button3"
android:id="@+id/button3"
android:layout_height="100dp"
android:layout_width="100dp">
</Button>
<Button
android:text="Button4"
android:id="@+id/button4"
android:layout_height="100dp"
android:layout_width="100dp">
</Button>
</LinearLayout>
</LinearLayout>
```

## Задание на лабораторную работу

Добавить в приложение, разработанное в лабораторной работе №2 элементы управления (*Button*, *TextView*, *EditText*, *ListView*). Выполнить запуск приложения на эмуляторе.

## Варианты заданий

1. ***Student***: *id*, Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон,

Создать массив объектов. Вывести:

*a*) список студентов заданного факультета;

*b*) списки студентов для каждого факультета и курса;

*c*) список студентов, родившихся после заданного года;

*d*) список учебной группы.

2. ***Customer***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, Номер банковского счета.

Создать массив объектов. Вывести:

*a*) список покупателей в алфавитном порядке;

*b*) список покупателей, у которых номер кредитной карточки находится в заданном интервале.

3. ***Patient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Номер медицинской карты, Диагноз.

Создать массив объектов. Вывести:

*a*) список пациентов, имеющих данный диагноз;

*b*) список пациентов, номер медицинской карты у которых находится в заданном интервале.

4. ***Abiturient***: *id*, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки.

Создать массив объектов. Вывести:

*a*) список абитуриентов, имеющих неудовлетворительные оценки;

*b*) список абитуриентов, средний балл у которых выше заданного;

*c*) выбрать заданное число *n* абитуриентов, имеющих самый высокий средний балл (вывести также полный список абитуриентов, имеющих полупроходной балл).

5. ***Book***: *id*, Название, Автор(ы), Издательство, Год издания, Количество страниц, Цена, Переплет.

Создать массив объектов. Вывести:

*a*) список книг заданного автора;

*b*) список книг, выпущенных заданным издательством;

*c*) список книг, выпущенных после заданного года.

6. ***House***: *id*, Номер квартиры, Площадь, Этаж, Количество комнат, Улица,

Тип здания, Срок эксплуатации.

Создать массив объектов. Вывести:

*a*) список квартир, имеющих заданное число комнат;

*b*) список квартир, имеющих заданное число комнат и расположенных на этаже, который находится в заданном промежутке;

*c*) список квартир, имеющих площадь, превосходящую заданную.

7. ***Phone***: *id*, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки,

Дебет, Кредит, Время городских и междугородных разговоров.

Создать массив объектов. Вывести:

*a*) сведения об абонентах, у которых время внутригородских разговоров превышает заданное;

*b*) сведения об абонентах, которые пользовались междугородной связью;

*c*) сведения об абонентах в алфавитном порядке.

8. ***Car***: *id*, Марка, Модель, Год выпуска, Цвет, Цена, Регистрационный номер.

Создать массив объектов. Вывести:

*a*) список автомобилей заданной марки;

*b*) список автомобилей заданной модели, которые эксплуатируются больше *n* лет;

*c*) список автомобилей заданного года выпуска, цена которых больше указанной.

9. ***Product***: *id*, Наименование, *UPC*, Производитель, Цена, Срок хранения, Количество.

Создать массив объектов. Вывести:

*a*) список товаров для заданного наименования;

*b*) список товаров для заданного наименования, цена которых не превосходит заданную;

*c*) список товаров, срок хранения которых больше заданного.

10. ***Train***: Пункт назначения, Номер поезда, Время отправления, Число мест (общих, купе, плацкарт, люкс).

Создать массив объектов. Вывести:

*a*) список поездов, следующих до заданного пункта назначения;

*b*) список поездов, следующих до заданного пункта назначения и отправляющихся после заданного часа;

*c*) список поездов, отправляющихся до заданного пункта назначения и имеющих общие места.

11. ***Bus***: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

*a*) список автобусов для заданного номера маршрута;

*b*) список автобусов, которые эксплуатируются больше 10 лет;

*c*) список автобусов, пробег у которых больше 100000 км.

12. ***Airlines***: Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.

Создать массив объектов. Вывести:

*a*) список рейсов для заданного пункта назначения;

*b*) список рейсов для заданного дня недели;

*c*) список рейсов для заданного дня недели, время вылета для которых больше заданного.

## Контрольные вопросы

1. «Простые» разметки Layout.
2. Элементы управления (Button, TextView, EditText).
3. «Сложные» разметки Layout.
4. Элемент управления (ListView).
5. Низкоуровневый интерфейс пользователя в Android.
6. Обработка действий пользователя.

## Лабораторная работа № 5. Двухмерная и трехмерная графика. Мультимедиа-возможности в *Android*.»

**Цель работы:** изучить программирование двухмерной и трехмерной графики в *Android*.

### Теоретические сведения

Для начала рассмотрим обычное 2*D* рисование.

Для рисования используется объект *Canvas*. *Canvas* является лишь инструментом для рисования. А весь результат сохраняется на *Bitmap*. Мы не можем напрямую попросить *Bitmap* нарисовать на себе линию или круг, поэтому канва выступает посредником и помогает нам нарисовать то, что нужно.

В этом е разберем два способа получения доступа к *Canvas*.

Первый способ – через наследника *View* класса. Нам нужно просто переопределить его метод *onDraw* и он даст нам доступ к канве. Кода тут минимум и все предельно просто. Но есть недостаток – все рисование выполняется в основном потоке. Это прокатит, если у вас статичная картинка или не слишком динамичная анимация.

Второй способ – через *SurfaceView*. Этот способ подойдет, если планируете рисовать что-то тяжелое и динамичное. Под рисование здесь будет выделен отдельный поток. Это уже немного посложнее в реализации, чем первый способ.

В *onCreate* мы в метод *setContentView* передаем не *layout*-файл, как обычно, а свой *view*-компонент *DrawView*. Он будет занимать все содержимое *Activity*.

Класс *DrawView* является наследником *View* и переопределяет его метод *onDraw*. А этот метод дает нам доступ к объекту *Canvas*. Пока что не будем рисовать ничего особенного, а просто закрасим все зеленым цветом с помощью метода *drawColor*.

Собственно, все. Готово первое приложение, которое что-то рисует на экране. Все сохраняем, запускаем. Экран зеленый, как мы и просили.

Метод *onDraw* был вызван системой, когда возникла необходимость прорисовать *View*-компонент на экране. Это также произойдет, например, если выключить-включить экран. Попробуйте поставить в *onDraw* лог и посмотреть результат.

Если вам надо, чтобы на канве была какая-то анимация, необходимо самим постоянно вызывать перерисовку экрана, когда ваши изменения готовы к отображению. Для этого используется метод *invalidate*. Вызываете его и он в свою очередь вызовет *onDraw*. Также есть реализации метода *invalidate*, которые позволяет перерисовать не весь компонент, а только его часть, указав координаты:

```
package ru.startandroid.develop.p1411canvasview;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(new DrawView(this));
 }

 class DrawView extends View {
  public DrawView(Context context) {
   super(context);
  }

  @Override
  protected void onDraw(Canvas canvas) {
   canvas.drawColor(Color.GREEN);
  }
  }
 }
```

## Задание на лабораторную работу

Разработать приложение *Android*, выполняющее следующее задание. Выполнить запуск приложения на эмуляторе.

1. Создать классы *Point* и *Line*. Объявить массив из *n* объектов класса *Point*. Для объекта класса *Line* определить, какие из объектов *Point* лежат на одной стороне от прямой линии и какие на другой. Реализовать ввод данных для объекта *Line* и случайное задание данных для объекта *Point*.

2. Создать классы *Point* и *Line*. Объявить массив из *n* объектов класса *Point* и определить в методе, какая из точек находится дальше всех от прямой линии.

3. Создать класс *Triangle* и класс *Point*. Объявить массив из *n* объектов класса *Point*, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.

4. Определить класс *Rectangle* и класс *Point*. Объявить массив из *n* объектов класса *Point*. Написать функцию, определяющую, какая из точек лежит снаружи, а какая – внутри прямоугольника.

5. Реализовать полиморфизм на основе абстрактного класса *AnyFigure* и его методов. Вывести координаты точки, треугольника, тетраэдра.

6. Определить класс *Line* для прямых линий, проходящих через точки $A(x1, y1)$ и $B(x2, y2)$. Создать массив объектов класса *Line*. Определить, используя функции, какие из прямых линий пересекаются, а какие совпадают. Нарисовать все пересекающиеся прямые.

7. Определить классы *Triangle* и *NAngle*. Определить, какой из *m*-введенных *n*-угольников имеет наибольшую площадь.

8. Задать движение по экрану строк (одна за другой) из массива строк. Направление движения по экрану и значение каждой строки выбираются случайным образом.

9. Задать составление строки из символов, появляющихся из разных углов в окне приложения и выстраивающихся друг за другом. Процесс должен циклически повторяться.

10. Задать движение окружности по экрану так, чтобы при касании границы окружность отражалась от нее с эффектом упругого сжатия.

11. Изобразить в окне приложения приближающийся издали шар, удаляющийся шар. Шар должен двигаться с постоянной скоростью.

12. Изобразить в окне приложения отрезок, вращающийся в плоскости экрана вокруг одной из своих концевых точек. Цвет прямой должен изменяться при переходе от одного положения к другом.

13. Создать класс *Triangle* и класс *Point*. Объявить массив из *n* объектов класса *Point*, написать функцию, определяющую, какая из точек лежит внутри, а какая – снаружи треугольника.

14. Определить класс *Rectangle* и класс *Point*. Объявить массив из *n* объектов класса *Point*. Написать функцию, определяющую, какая из точек лежит снаружи, а какая – внутри прямоугольника.

15. Реализовать полиморфизм на основе абстрактного класса *AnyFigure* и его методов. Вывести координаты точки, треугольника, тетраэдра.

### Контрольные вопросы

1. Программирование двухмерной и трехмерной графики.
2. Основы программирования мультимедиа в *Android* (аудио).
3. Основы программирования мультимедиа в *Android* (видео).
4. Включение активности и разрешений камеры в манифест.
5. Использование *API* камеры.
6. Открытие и освобождение камеры.
7. Реализация обратных вызовов камеры.

## Лабораторная работа № 6. Фоновые задачи и службы

**Цель работы:** изучить программирование фоновых задач и служб в *Android*.

### Теоретические сведения

Мы рассматривали, как в приложении можно выполнять тяжелые задачи. Мы вводили их в отдельный поток и использовали *Handler* для обратной связи и обновления экрана. Создатели *Android* решили, что эти механизмы стоит выделить в отдельный класс – *AsyncTask*. То естьего цель – это выполнение тяжелых задач и передача в *UI*-поток результатов работы. Но при этом нам не надо задумываться о создании *Handler* и нового потока.

Чтобы работать с *AsyncTask*, надо создать класс-наследник и в нем прописать свою реализацию необходимых нам методов. Рассмотрим три метода:

*doInBackground* – будет выполнен в новом потоке, здесь решаем все свои тяжелые задачи. Т.к. поток не основной - не имеет доступа к *UI*.

*onPreExecute* – выполняется перед *doInBackground*, имеет доступ к *UI*

*onPostExecute* – выполняется после *doInBackground* (не срабатывает в случае, если *AsyncTask* был отменен - об этом в следующих ах), имеет доступ к *UI*

Сделаем простое приложение, в котором используем вышеуказанные методы. Выведем на экран слово *Begin* в методе *onPreExecute*, эмулируем тяжелый код в методе *doInBackground*, выведем на экран слово *End* в методе *onPostExecute*.

Создадим проект.

Файл ***strings.xml***:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">AsyncTask</string>
<string name="start">Start</string>
</resources>
```

Файл ***main.xml***:

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

```
 android:id="@+id/tvInfo"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="">
</TextView>
</LinearLayout>
```

По нажатию кнопки будем стартовать задачу, в *TextView* выводить информацию. *ProgressBar* покажет, что приложение не "висит" во время выполнения.

Файл ***MainActivity.java***:

```
package ru.startandroid.develop.p0861asynctask;

import java.util.concurrent.TimeUnit;
import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {
  MyTask mt;
  TextView tvInfo;
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvInfo = (TextView) findViewById(R.id.tvInfo);
  }
  public void onclick(View v) {
    mt = new MyTask();
    mt.execute();
  }
  class MyTask extends AsyncTask<Void, Void, Void> {
    @Override
    protected void onPreExecute() {
      super.onPreExecute();
      tvInfo.setText("Begin");
    }
```

```
@Override
protected Void doInBackground(Void... params) {
  try {
    TimeUnit.SECONDS.sleep(2);
  } catch (InterruptedException e) {
    e.printStackTrace();
  }
  return null;
}
@Override
protected void onPostExecute(Void result) {
  super.onPostExecute(result);
  tvInfo.setText("End");
 }
}}
```

Все сохраним и запустим приложение. Жмем ***Start***, появляется текст *Begin* (рис. 6.1).


*Рис. 6.1.* Запуск фоновой задачи

В методе *onclick* мы создаем объект *MyTask* и запускаем его методом *execute*. *MyTask* – это наш класс, наследующий *AsyncTask*. В нем мы прописываем свой код. В методе *onPreExecute* выводим текст *Begin*. В методе *doInBackground* эмулируем тяжелый код - делаем паузу на две секунды.В методе *onPostExecute* выводим текст *End*. *ProgressBar* работает, значит, основной поток (отвечающий за *UI*) – свободен. Через две секунды появляется текст *End* (рис. 6.2).

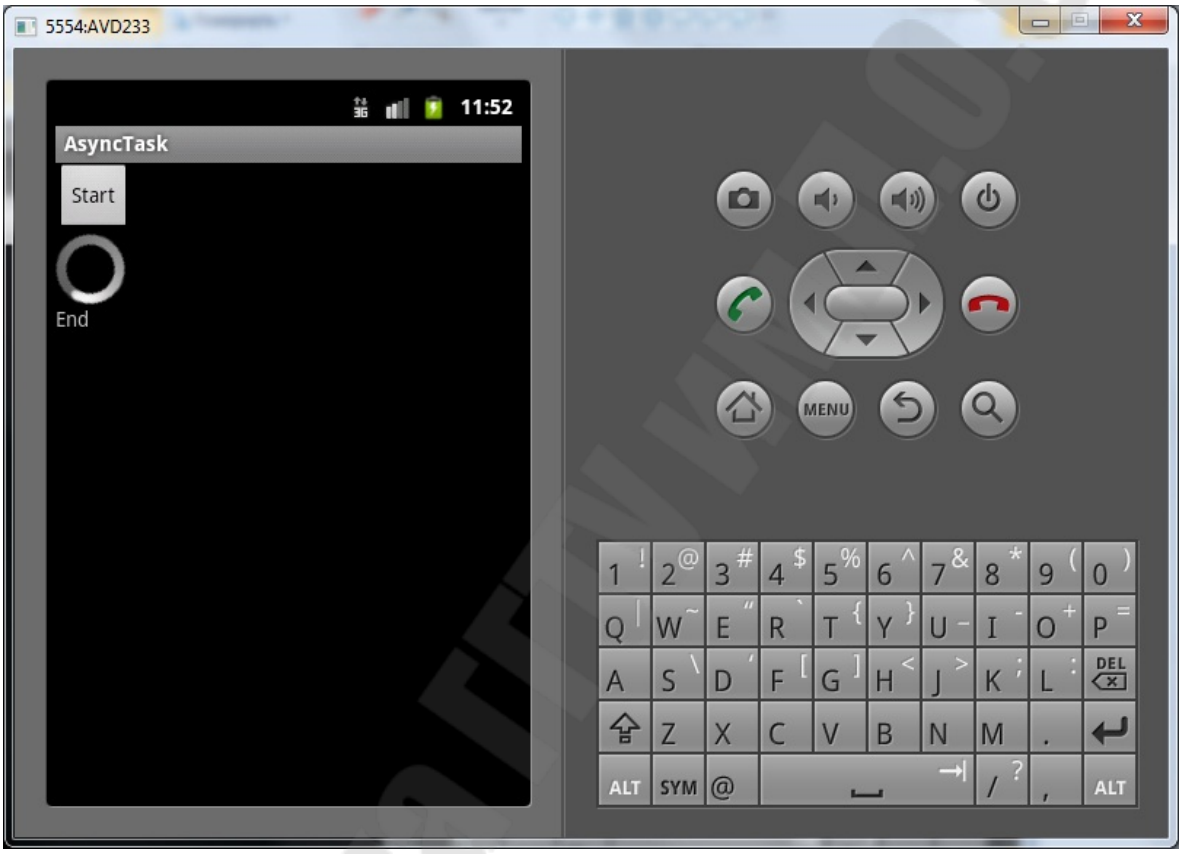


*Рис. 6.2.* Запуск фоновой задачи (появляется текст *End*)

## Задание на лабораторную работу

Добавить в приложение, разработанное в лабораторной работе № 4 фоновую службу, выполняющее периодическое сохранение данных приложения в файл. Выполнить запуск приложения на эмуляторе.

## Контрольные вопросы

1. Основы сетевой поддержки.
2. Разрешение на работу с сетью.
3. Использование *AsyncTask* для выполнения в фоновом потоке.

4. Главный программный поток.
5. Создание фонового потока.
6. Сообщения и обработчики сообщений.
7. Создание фоновых служб.
8. Жизненный цикл службы.
9. Сигналы.

## Лабораторная работа № 7. *Internet* коммуникации в устройствах *Android*

**Цель работы:** изучить программирование *Internet* коммуникаций в устройствах *Android*.

### Теоретические сведения

*Android* устройства могут предоставить нам данные по нашему текущему местоположению. Это, конечно, очень удобно и вовсю используется для, например, пользования картой, получения актуальной для вашей местности информации (прогноз погоды), всевозможных чекинов и пр. Реализация этого всего вполне проста. Мы реализуем слушателя на провайдера и получаем данные. На данный момент есть два провайдера: *GPS* и *Network*.

Напишем простое приложение, которое будет запрашивать и отображать координаты.

В ***strings.xml*** добавим строки:

```
    <string name="provider_gps">GPS</string>
<string name="provider_network">Network</string>
<string name="location_settings">Location settings</string>
```

Экран ***main.xml***:

```
    <?xml version="1.0" encoding="utf-8"?>
<LinearLayout
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"
android:orientation="vertical"
```

```
 android:padding="5dp">
<TextView
 android:id="@+id/tvTitleGPS"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/provider_gps"
 android:textSize="30sp">
</TextView>
<TextView
 android:id="@+id/tvEnabledGPS"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
</TextView>
<TextView
 android:id="@+id/tvStatusGPS"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
</TextView>
<TextView
 android:id="@+id/tvLocationGPS"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
</TextView>
<TextView
 android:id="@+id/tvTitleNet"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginTop="10dp"
 android:text="@string/provider_network"
 android:textSize="30sp">
</TextView>
<TextView
 android:id="@+id/tvEnabledNet"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
```

```
</TextView>
<TextView
 android:id="@+id/tvStatusNet"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
</TextView>
<TextView
 android:id="@+id/tvLocationNet"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textSize="24sp">
</TextView>
<Button
 android:id="@+id/btnLocationSettings"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_marginTop="10dp"
 android:onClick="onClickLocationSettings"
 android:text="@string/location_settings">
</Button>
</LinearLayout>
```

Несколько *TextView*, в которые мы будем выводить данные, и кнопка для открытия настроек местоположения.

***MainActivity.java***:

```
package ru.startandroid.develop.p1381location;

import java.util.Date;
import android.app.Activity;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
```

```
public class MainActivity extends Activity {
  TextView tvEnabledGPS;
  TextView tvStatusGPS;
  TextView tvLocationGPS;
  TextView tvEnabledNet;
  TextView tvStatusNet;
  TextView tvLocationNet;
  private LocationManager locationManager;
  StringBuilder sbGPS = new StringBuilder();
  StringBuilder sbNet = new StringBuilder();
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tvEnabledGPS = (TextView) findViewById(R.id.tvEnabledGPS);
    tvStatusGPS = (TextView) findViewById(R.id.tvStatusGPS);
    tvLocationGPS = (TextView) findViewById(R.id.tvLocationGPS);
    tvEnabledNet = (TextView) findViewById(R.id.tvEnabledNet);
    tvStatusNet = (TextView) findViewById(R.id.tvStatusNet);
    tvLocationNet = (TextView) findViewById(R.id.tvLocationNet);
    locationManager = (LocationManager) getSystemSer-
vice(LOCATION_SERVICE);
  }
  @Override
  protected void onResume() {
    super.onResume();
locationMan-
ager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        1000 * 10, 10, locationListener);
    locationManager.requestLocationUpdates(
        LocationManager.NETWORK_PROVIDER, 1000 * 10, 10,
        locationListener);
    checkEnabled();
  }
  @Override
  protected void onPause() {
    super.onPause();
    locationManager.removeUpdates(locationListener);
  }
```

```
  private LocationListener locationListener = new LocationListener()
{
    @Override
    public void onLocationChanged(Location location) {
      showLocation(location);
    }
    @Override
    public void onProviderDisabled(String provider) {
      checkEnabled();
    }
    @Override
    public void onProviderEnabled(String provider) {
      checkEnabled();
      showLoca-
tion(locationManager.getLastKnownLocation(provider));
    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) {
      if (provider.equals(LocationManager.GPS_PROVIDER)) {
        tvStatusGPS.setText("Status: " + String.valueOf(status));
      } else if (pro-
vider.equals(LocationManager.NETWORK_PROVIDER)) {
        tvStatusNet.setText("Status: " + String.valueOf(status));
      }
    }
  };
  private void showLocation(Location location) {
    if (location == null)
      return;
    if (loca-
tion.getProvider().equals(LocationManager.GPS_PROVIDER)) {
      tvLocationGPS.setText(formatLocation(location));
    } else if (location.getProvider().equals(
        LocationManager.NETWORK_PROVIDER)) {
      tvLocationNet.setText(formatLocation(location));
    }
  }
  private String formatLocation(Location location) {
```

```
    if (location == null)
      return "";
    return String.format(
        "Coordinates: lat = %1$.4f, lon = %2$.4f, time = %3$tF
%3$tT",
        location.getLatitude(), location.getLongitude(), new Date(
            location.getTime()));
  }
  private void checkEnabled() {
    tvEnabledGPS.setText("Enabled: "
        + locationManager
            .isProviderEnabled(LocationManager.GPS_PROVIDER));
    tvEnabledNet.setText("Enabled: "
        + locationManager

.isProviderEnabled(LocationManager.NETWORK_PROVIDER));
  }
  public void onClickLocationSettings(View view) {
    startActivity(new Intent(
an-
droid.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS
));
  };
}
```

В *onCreate* определяем *TextView*-компоненты и получаем *LocationManager,* через который и будем работать.

В *onResume* вешаем слушателя с помощью метода *requestLocationUpdates*. На вход передаем*:*

– Тип провайдера*: GPS_PROVIDER* или *NETWORK_PROVIDER* минимальное время *(*в миллисекундах*)* между получением данных. Я укажу здесь *10* секунд, мне этого вполне хватит. Если хотите получать координаты без задержек – передавайте *0*. Но учитывайте, что это только минимальное время. Реальное ожидание может быть дольше.

– Минимальное расстояние *(*в метрах*)*. Т.е. если ваше местоположение изменилось на указанное кол-во метров, то вам придут новые координаты.

– Слушатель, объект *locationListener,* который рассмотрим ниже.

Также здесь обновляем на экране информацию о включенности провайдеров.

В *onPause* отключаем слушателя методом *removeUpdates*.

*LocationListener* – слушатель, реализует интерфейс *LocationListener* с методами:

– *onLocationChanged* – новые данные о местоположении, объект *Location*. Здесь мы вызываем свой метод *showLocation*, который на экране отобразит данные о местоположении.

– *onProviderDisabled* – указанный провайдер был отключен юзером. В этом методе вызываем свой метод *checkEnabled*, который на экране обновит текущие статусы провайдеров.

– *onProviderEnabled* – указанный провайдер был включен юзером. Тут также вызываем *checkEnabled*. Далее методом *getLastKnownLocation* (он может вернуть *null*) запрашиваем последнее доступное местоположение от включенного провайдера и отображаем его. Оно может быть вполне актуальным, если вы до этого использовали какое-либо приложение с определением местоположения.

– *onStatusChanged* – изменился статус указанного провайдера. В поле *status* могут быть значения *OUT_OF_SERVICE* (данные будут недоступны долгое время), *TEMPORARILY_UNAVAILABLE* (данные временно недоступны), *AVAILABLE* (все ок, данные доступны). В этом методе мы просто выводим новый статус на экран.

Провайдеры включаются и отключаются в настройках системы. Тем самым, просто определяется доступен ли провайдер для получения от него координат. Чуть позже увидим, как можно отправить юзера в эти настройки. Программное включение/выключение провайдеров через стандартные методы недоступно.

Далее идут свои методы.

– *showLocation* на вход берет *Location*, определяет его провайдера методом *getProvider* и отображает координаты в соответствующем текстовом поле.

– *formatLocation* на вход берет *Location*, читает из него данные и форматирует из них строку. Какие данные он берет: *getLatitude* – широта, *getLongitude* – долгота, *getTime* – время определения.

– *checkEnabled* определяет включены или выключены провайдеры методом *isProviderEnabled* и отображает эту инфу на экране.

Метод *onClickLocationSettings* срабатывает по нажатию кнопки *Location settings* и открывает настройки, чтобы пользователь мог включить или выключить провайдер. Для этого используется *Intent* с *action = ACTION_LOCATION_SOURCE_SETTINGS.*

Осталось в манифесте прописать разрешение на определение координат - *ACCESS_FINE_LOCATION*, которое позволит нам использовать и *Network* и *GPS*. Также существует разрешение *ACCESS_COARSE_LOCATION*, но оно дает доступ только к *Network*-провайдеру.

С кодом все, давайте смотреть, что получилось. Все сохраняем и запускаем приложение.

На планшете сейчас выключен *GPS*, выключен *WiFi*, вставлена симка и выключен мобильный интернет. Запускаем приложение*:*

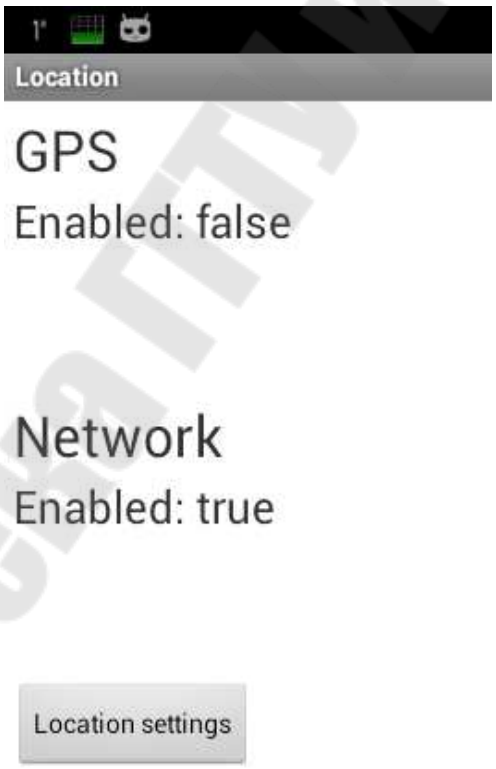


*Рис. 7.1.* Запуск приложения геолокации

*GPS* выключен, *Network* включен. Проходит секунд 15-20 и данные с *Network* поступают в эмулятор.

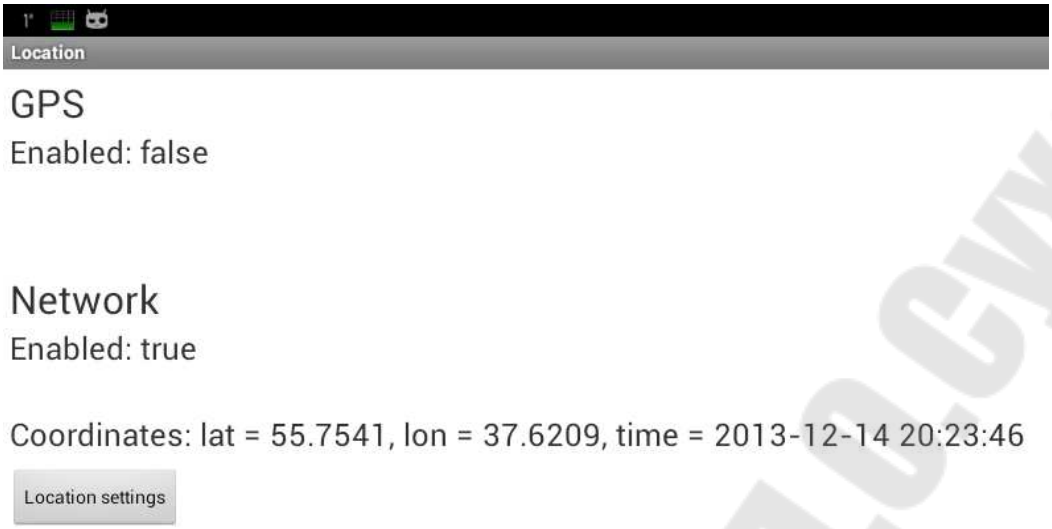


*Рис. 7.2.* Результат геолокации

Видим широту, долготу и время.

## Задание на лабораторную работу

Добавить в приложение, разработанное в лабораторной работе №6 функционал, выполняющий определение ближайшего объекта, способного обеспечить предоставление сервиса, требуемого приложению, в соответствии с логикой его работы (для приложения «книги» ближайшего книжного магазина, приложения «расписание поездов» ближайшего железнодорожного вокзала). Данные о доступных объектах подгружать из файла. Выполнить запуск приложения на эмуляторе.

## Контрольные вопросы

1. Использование *Internet* коммуникаций в устройствах *Android* (*HTTP*).
2. Использование *Internet* коммуникаций в устройствах *Android* (*FTP*).
3. Использование *Internet* коммуникаций в устройствах *Android* (*e-mail*).
4. Использование *Internet* коммуникаций в устройствах *Android* (сокеты).
5. Форматы данных(*html*, *xml*, *json*).
6. Распространение и публикация приложений.
7. Отслеживание местоположения устройства (*GeoLocation*).

## Лабораторная работа № 8. Обращение с данными и их долговременное хранение

**Цель работы:** изучить программирование долговременного хранения данных мобильного приложения.

### Теоретические сведения

Для долговременного хранения данных в Андроид существует БД *SQLite*. Это база данных с таблицами и запросами - все как в обычных БД. Для начала, немного теории по взаимодействию приложения и БД.

В приложении, при подключении к БД мы указываем имя БД и версию. При этом могут возникнуть следующие ситуации:

1. БД не существует. Это может быть, например, в случае первичной установки программы. В этом случае приложение должно само создать БД и все таблицы в ней. И далее оно уже работает с только что созданной БД.

2. БД существует, но ее версия устарела. Это может быть в случае обновления программы. Например, новой версии программы нужны дополнительные поля в старых таблицах или новые таблицы. В этом случае приложение должно апдейтить существующие таблицы и создать новые, если это необходимо.

3. БД существует, и ее версия актуальна. В этом случае приложение успешно подключается к БД и работает.

Как вы понимаете, фраза "приложение должно" равнозначна фразе "разработчик должен", т.е. это наша задача. Для обработки описанных выше ситуаций нам надо создать класс, являющийся наследником для *SQLiteOpenHelper*. Назовем его *DBHelper*. Этот класс предоставит нам методы для создания или обновления БД в случаях ее отсутствия или устаревания:

– *onCreate* - метод, который будет вызван, если БД, к которой мы хотим подключиться – не существует

– *onUpgrade* - будет вызван в случае, если мы пытаемся подключиться к БД более новой версии, чем существующая.

Давайте накидаем простое приложение – справочник контактов, которое будет хранить имя и *email*. Вводить данные будем на экране приложения, а для отображения информации используем логи. Обычно для этого используется *List* (список) – но мы эту тему пока не зна-

ем. Да и не хочется перегружать приложение. Главное – освоить приемы работы с БД.

Нарисуем экран для ввода записей и очистки таблицы. Открываем ***main.xml*** и пишем:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical">
<LinearLayout
 android:id="@+id/linearLayout1"
 android:layout_width="match_parent"
 android:layout_height="wrap_content">
<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Name"
 android:layout_marginLeft="5dp"
 android:layout_marginRight="5dp">
</TextView>
<EditText
 android:id="@+id/etName"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_weight="1">
<requestFocus>
</requestFocus>
</EditText>
</LinearLayout>
<LinearLayout
 android:id="@+id/linearLayout3"
 android:layout_width="match_parent"
 android:layout_height="wrap_content">
<TextView
 android:id="@+id/textView2"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Email"
```

```
 android:layout_marginLeft="5dp"
 android:layout_marginRight="5dp">
</TextView>
<EditText
 android:id="@+id/etEmail"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_weight="1">
</EditText>
</LinearLayout>
<LinearLayout
 android:id="@+id/linearLayout2"
 android:layout_width="match_parent"
 android:layout_height="wrap_content">
<Button
 android:id="@+id/btnAdd"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Add">
</Button>
<Button
 android:id="@+id/btnRead"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Read">
</Button>
<Button
 android:id="@+id/btnClear"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Clear">
</Button>
</LinearLayout>
</LinearLayout>
```

Пара полей для ввода и кнопки добавления записи, вывода существующих записей и очистки таблицы.

Открываем ***MainActivity.java*** и пишем:

```
package ru.startandroid.develop.p0341simplesqlite;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity implements OnClickLis-
tener {

 final String LOG_TAG = "myLogs";

 Button btnAdd, btnRead, btnClear;
 EditText etName, etEmail;

 DBHelper dbHelper;

 /** Called when the activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   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);
  etName = (EditText) findViewById(R.id.etName);
```

```
    etEmail = (EditText) findViewById(R.id.etEmail);
    // создаем объект для создания и управления версиями БД
    dbHelper = new DBHelper(this);
  }


  @Override
  public void onClick(View v) {

    // создаем объект для данных
    ContentValues cv = new ContentValues();

    // получаем данные из полей ввода
    String name = etName.getText().toString();
    String email = etEmail.getText().toString();
    // подключаемся к БД
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    switch (v.getId()) {
    case R.id.btnAdd:
      Log.d(LOG_TAG, "--- Insert in mytable: ---");
      // подготовим данные для вставки в виде пар: наименование
столбца - значение
      cv.put("name", name);
      cv.put("email", email);
      // вставляем запись и получаем ее ID
      long rowID = db.insert("mytable", null, cv);
      Log.d(LOG_TAG, "row inserted, ID = " + rowID);
      break;
    case R.id.btnRead:
      Log.d(LOG_TAG, "--- Rows in mytable: ---");
      // делаем запрос всех данных из таблицы mytable, получаем
Cursor
      Cursor c = db.query("mytable", null, null, null, null, null, null);
      // ставим позицию курсора на первую строку выборки
      // если в выборке нет строк, вернется false
      if (c.moveToFirst()) {
        // определяем номера столбцов по имени в выборке
        int idColIndex = c.getColumnIndex("id");
        int nameColIndex = c.getColumnIndex("name");
```

```
        int emailColIndex = c.getColumnIndex("email");
        do {
          // получаем значения по номерам столбцов и пишем все в
лог
          Log.d(LOG_TAG,
              "ID = " + c.getInt(idColIndex) +
              ", name = " + c.getString(nameColIndex) +
              ", email = " + c.getString(emailColIndex));
          // переход на следующую строку
          // а если следующей нет (текущая - последняя), то false -
выходим из цикла
        } while (c.moveToNext());
      } else
        Log.d(LOG_TAG, "0 rows");
      c.close();
      break;
    case R.id.btnClear:
      Log.d(LOG_TAG, "--- Clear mytable: ---");
      // удаляем все записи
      int clearCount = db.delete("mytable", null, null);
      Log.d(LOG_TAG, "deleted rows count = " + clearCount);
      break;
    }
    // закрываем подключение к БД
    dbHelper.close();
  }
    class DBHelper extends SQLiteOpenHelper {
    public DBHelper(Context context) {
      // конструктор суперкласса
      super(context, "myDB", null, 1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
      Log.d(LOG_TAG, "--- onCreate database ---");
      // создаем таблицу с полями
      db.execSQL("create table mytable ("
          + "id integer primary key autoincrement,"
          + "name text,"
          + "email text" + ");");
```

```
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
    }
  }
}
```

В методе *Activity – onCreate* мы определяем объекты, присваиваем обработчики и создаем объект *dbHelper* класса *DBHelper* для управления БД. Сам класс будет описан ниже.

Далее смотрим метод *Activity – onClick,* в котором мы обрабатываем нажатия на кнопки.

Класс *ContentValues* используется для указания полей таблицы и значений, которые мы в эти поля будем вставлять. Мы создаем объект *cv,* и позже его используем. Далее мы записываем в переменные значения из полей ввода. Затем, с помощью метода *getWritableDatabase* подключаемся к БД и получаем объект *SQLiteDatabase*. Он позволит нам работать с БД. Мы будем использовать его методы *insert* – вставка записи, *query* – чтение, *delete* – удаление. У них много разных параметров на вход, но мы пока используем самый минимум.

Далее смотрим, какая кнопка была нажата:

– *btnAdd* – добавление записи в таблицу *mytable*. Мы заполняем объект *cv* парами: имя поля и значение. И (при вставке записи в таблицу) в указанные поля будут вставлены соответствующие значения. Мы заполняем поля *name* и *email*. *id* у нас заполнится автоматически (*primary key autoincrement*). Вызываем метод *insert* – передаем ему имя таблицы и объект *cv* с вставляемыми значениями. Второй аргумент метода используется, при вставке в таблицу пустой строки. Нам это сейчас не нужно, поэтому передаем *null*. Метод *insert* возвращает *ID* вставленной строки, мы его сохраняем в *rowID* и выводим в лог.

– *btnRead* – чтение всех записей из таблицы *mytable*. Для чтения используется метод *query*. На вход ему подается имя таблицы, список запрашиваемых полей, условия выборки, группировка, сортировка. Т.к. нам нужны все данные во всех полях без сортировок и группировок - мы используем везде *null*. Только имя таблицы указываем. Метод возвращает нам объект класса *Cursor*. Его можно рассматривать как таблицу с данными. Метод *moveToFirst* – делает первую запись в *Cursor* активной и заодно проверяет, есть ли вообще записи в нем (т.е.

выбралось ли что-либо в методе *query*). Далее мы получаем порядковые номера столбцов в *Cursor* по их именам с помощью метода *getColumnIndex*. Эти номера потом используем для чтения данных в методах *getInt* и *getString* и выводим данные в лог. С помощью метода *moveToNext* мы перебираем все строки в *Cursor* пока не добираемся до последней. Если же записей не было, то выводим в лог соответствующее сообщение – *0 rows*. В конце закрываем курсор (освобождаем занимаемые им ресурсы) методом *close*, т.к. далее мы его нигде не используем.

– *btnClear* – очистка таблицы. Метод *delete* удаляет записи. На вход передаем имя таблицы и *null* в качестве условий для удаления, а значит удалится все. Метод возвращает кол-во удаленных записей.

После этого закрываем соединение с БД методом *close*.

Класс *DBHelper* является вложенным в *MainActivity* и описан в конце кода. Как я уже писал выше, этот класс должен наследовать класс *SQLiteOpenHelper*.

В конструкторе мы вызываем конструктор суперкласса и передаем ему:

– *context* – контекст;

– *mydb* – название базы данных;

– *null* – объект для работы с курсорами, нам пока не нужен, поэтому *null*.

В методе *onCreate* этого класса мы используем метод *execSQL* объекта *SQLiteDatabase* для выполнения *SQL*-запроса, который создает таблицу. Напомню – этот метод вызывается, если БД не существует и ее надо создавать. По запросу видно, что мы создаем таблицу *mytable* с полями *id*, *name* и *email*.

Метод *onUpgrade* пока не заполняем, т.к. используем одну версию БД и менять ее не планируем.

## Задание на лабораторную работу

Добавить в приложение, разработанное в лабораторной работе №7 функционал, выполняющий сохранение всех данных приложения в БД *Android* и выборку их при необходимости. Выполнить запуск приложения на эмуляторе.

## Контрольные вопросы

1. Долговременное хранение данных мобильного приложения.

2. Локальные базы данных и SQLite.
3. Хранение серий и позиций в базе данных.
4. Запрос списка серий из базы данных.
5. Вывод списка серий с использованием CursorAdapter.
6. Создание новых серий.
7. Работа с существующими сериями.

## Литература


1. Майер, Р. *Android* 4. Программирование приложений для планшетных компьютеров и смартфонов: [перевод с английского] / Рето Майер. – Москва: Эксмо, 2014. – 814 с. – (Мировой компьютерный бестселлер).

2. Харди, Б. Программирование под *Android* / Брайан Харди, Билл Филлипс; пер. с англ. Е. Матвеев. – Санкт-Петербург [и др.] : Питер, 2014. - 592 с. - (Для профессионалов).

3. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на *Java*2. Книга 1. Графика, *JavaBeans*, интерфейс пользователя. - М.: Бином, 2003.

4. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на *Java*2. Книга 2. Распределенные приложения. – М.: Бином, 2003.

5. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на *Java*2. Книга 3. Корпоративные системы, сервлеты, *JSP*, *web*-сервисы. - М.: Бином, 2003.

6. Дейтел Х. М., Дейтел П. Д. Как программировать на *Java*. Книга 1. Основы программирования. - М.: Бином, 2003. Дейтел Х. М., Дейтел П. Д. Как программировать на *Java*. Книга 2. Файлы, сети, базы данных. – М.: Бином, 2006.

7. Ноутон П., Шилдт Г. *Java*2. Наиболее полное руководство. - СПб.: *BHV*, 2001.

8. Биллиг В.А. Основы программирования на *Java*/ - М.: Изд-во «Интернет-университет информационных технологий – ИНТУИТ.ру», 2006. – 488 с.

9. Шилдт Г.С. *Java*: Учебный курс. – СПб.:Питер, 2002. – 512с.

10. Шилдт Г.С. Полный справочник по *Java*. – М.: Издательский дом «Вильямс», 2004. – 752 с.

**Стефановский** Игорь Леонидович
**Семенченя** Татьяна Сергеевна

# РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ ДЛЯ ANDROID

**Практикум**
**по выполнению лабораторных работ**
**по одноименной дисциплине для студентов**
**специальности 1-40 05 01 «Информационные**
**системы и технологии (по направлениям)»**
**дневной и заочной форм обучения**

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 27.03.21.

Рег. № 49Е.
http://www.gstu.by