Владимирский государственный университет

Методические указания к курсовой работе по дисциплине «Портативные вычислительные системы»

Владимир 2013

Методические указания содержат задания и рекомендации по выполнению курсовой работы, выполняемой в рамках учебного курса «Портативные вычислительные системы», и призвано помочь в освоении общих принципов программирования на платформе Android на примере решения конкретной прикладной исследовательской задачи, состоящей в создании программного продукта справочника.

Пособие предназначено для студентов специальностей: 010300 – «Фундаментальная информатика и информационные технологии», 010400 - «Прикладная математика и информатика», 010500 – «Математическое обеспечение и администрирование информационных систем», а также для преподавателей, проходящих обучение во внутривузовской системе повышения квалификации.

1. Организация курсовой работы

1.1. Цели курсового проектирования

Умение грамотно и эффективно программировать приложения является неотъемлемой частью навыков любого инженера в области ИТ-технологий. При этом основным назначением программирования приложения является структурное представление о работе компонентов программы, установления причинно-следственной связи между различными компонентами программного кода, понимание семантики и синтаксиса программного кода, умение производить отладку приложений, позволяющих визуально представить интересующую тему исследования.

1.2. Тематика курсовой работы

В рамках курсовой работы студенты должны практически освоить общий методологический подход, используемый в программировании портативных устройств на платформе Android, на примере решения конкретной прикладной исследовательской задачи, заключающейся в создании приложения-справочника по теме «Сети». Выполнение работы должно способствовать развитию навыков исследовательской работы, творческого мышления и системного подхода, а также закреплению теоретических знаний, полученных в рамках лекционного курса и в ходе выполнения лабораторных работ. Это применимо как к методам создания пользовательского интерфейса приложения, так и способов их реализации средствами вычислительной техники. Работа выполняется с использованием среды Eclipse SDK с установленным плагином ADT

1.3. Задание на курсовое проектирование

В ходе курсовой работы необходимо разработать пользовательский интерфейс приложения на платформе Android, провести отладку и промежуточное тестирование приложения на эмуляторе и на реальном устройстве. При этом подлежат разработке следующие вопросы:

а) структура программного кода в проекте;
б) схема активностей и служб приложения;
в) структура пользовательского интерфейса приложения;
г) проведение отладки и тестирования на эмуляторе устройства;
д) запуск и отладка приложения на реальном устройстве;
е) оформление расчётно-пояснительной записки по результатам выполнения работы.

Перечень вариантов задания на курсовое проектирование приведена в Приложении А.

Объём расчётно-пояснительной записки составляет 20-50 страниц машинописного текста, включая рисунки, таблицы и приложения. Её оформление должно соответствовать требованиям ГОСТ и ЕСКД.

2. Состав курсовой работы

Основные требования к расчётно-пояснительной записке устанавливают ГОСТ 2.105-95 ЕСКД и ГОСТ 2.106-96 ЕСКД. Для данной курсовой работы рекомендовано следующее содержание расчётно-пояснительной записки:

1. Титульный лист.
2. Формулировка задания на курсовую работу, включающую информацию о параметрах выданного варианта задания (на бланке установленного образца).
3. Описание структуры приложения.
   3.1. Параметры совместимости приложения с различными версиями Android.
   3.2. Схема активностей и служб приложения в виде UML – диаграмм.
   3.3. Службы, используемые при установке приложения.
   3.4. Схема взаимодействия компонентов проекта.
   3.5. Описание графического интерфейса приложения.
4. Описание внешнего вида интерфейса приложения.
5. Описание диаграммы классов Java, используемых в приложении.
6. Результаты отладки и тестирования приложения на различных устройствах (текстовое описание с графиками и диаграммами и описанием ошибок, при установке данного программного продукта, вывод о совместимости приложения).
7. Вывод по курсовой работе с предложениями о возможностях модернизации приложения на основе полученных результатов.
8. Библиографический список.

3. Рекомендации по выполнению работы

3.1. Порядок выполнения

1. Разработка пользовательского интерфейса приложения в соответствии с индивидуальным вариантом задания.
2. Составление схемы активностей и служб приложения:

2.1. Параметры приложения.
2.2. Подключаемые службы.
2.3. Структура процессов.

3. Составление алгоритма работы приложения.
4. Создание приложения и компиляция его в виде apk.файла.
5. Выполнение промежуточного тестирования проекта на эмуляторе в зависимости от варианта задания.
6. Подготовка расчётно-пояснительной записки (с обязательным включением в неё обобщённых результатов тестирования, проиллюстрированных рисунками экрана, демонстрирующими различные режимы работы приложения и его совместимость с различными версиями Android).
7. Демонстрация работы приложения преподавателю и защита курсовой работы.

3.1. Разработка пользовательского интерфейса.

Данный пункт является самым важным при выполнении курсового проекта, так как основой любого приложения на платформе Android является наличие в приложении пользовательского интерфейса. Ввиду того что, данная платформа поддерживает визуализацию объектов и действия над отдельными компонентами приложения (нажатие на кнопку, вызов текстового поля, загрузка анимации и т.д.) необходимо разработать такой интерфейс пользователя, который был доступен для любого человека, пользующегося данным приложением.

3.2. Составление схемы активностей и служб приложения

В рамках курсовой работы после того как создан макет пользовательского интерфейса приложения, необходимо разработать схему активностей и служб данного приложения, для представления о выполняемых приложением функциях и происходящих при этом процессах.

Активности (деятельности) определяют функциональные возможности любого приложения на платформе Android.

Существует четыре типа компонентов, применяемых в среде Android:

- Activity - деятельность;
- Service - служба;
- Content Provider – контент-провайдер;

- Broadcast Receiver – приёмник широковещательных намерений (асинхронных сообщений, активирующих деятельности, службы и приемники широковещательных намерений. Намерение – это объект класса Intent, представляющий собой содержание сообщения.

Разберем основные свойства этих компонентов.

1. Activity(деятельность) представляет собой визуальный интерфейс пользователя, или окно. Окно обычно заполняет весь экран портативного устройства, но может быть меньших размеров, чем экран устройства. Если у приложения несколько окон, то у него будет несколько деятельностей. Каждая деятельность независима от других, при открытии новой деятельности (нового окна) работа предыдущей деятельности приостанавливается.
2. Service (служба). У этого компонента нет графического интерфейса пользователя. Служба выполняется в фоновом режиме до тех пор, пока не завершит свою работу. Другие приложения могут подключаться к службе. После подключения к службе вы можете использовать функции, предоставляемые службой. Приложение также может запустить или остановить службу.
3. Broadcast Receiver. Для получения информации о внешних событиях и реакции на них используется этот компонент. Источником события могут быть службы и другие приложения. Приложение может реагировать на те любые события, которые посчитает важными. У приемников широковещательных намерений нет пользовательского интерфейса, но в ответ на событие они могут запустить деятельность, отобразить окно.
4. Content Provider (контент-провайдер) используется для предоставления доступа к данным программы другим приложениям. Данные программы могут храниться как в файловой системе, так и в базах данных SQLite.

Кроме вышеперечисленных компонентов, необходимо учитывать происходящие в приложении процессы. Операционная система никогда не уничтожит видимый пользователем компонент процесса. Операционная система может завершить процесс с невидимыми деятельностями, которые более не отображаются на экране. Какой процесс будет завершен в первую очередь зависит от его приоритета. У активного процесса критический приоритет, поэтому он не может быть завершен, у видимого и сервисного процесса высокий приоритет, поэтому система, скорее всего его тоже не завершит. Главными кандидатами на завершение являются процессы с

низким приоритетом – фоновые и пустые процессы. Сначала система удалит пустые процессы, а затем – фоновые. Рассмотрим подробнее терминологию процессов.

Существует пять основных видов процессов:

- *Активный процесс* (***Foreground Process***) – процесс, с которым в данный момент работает пользователь. Система считает процесс активным, если процесс выполняет деятельность, с которой работает пользователь, или же процесс выполняет службу, которая использует активная деятельность. Заметьте, что процесс А может выполнять службу, которая используется деятельностью, выполняющуюся процессом Б. С этой деятельностью работает пользователь, поэтому процессы А и Б считаются активными. Также процесс считается активным, если он имеет объект *Service* и выполняется один из методов обратного вызова, который определен в этом объекте. Если процесс имеет объект *Broadcast Receiver* и выполняется его метод обратного вызова для приема намерения, то процесс тоже считается активным. Активные процессы система может удалить только в том случае, если памяти на устройстве осталось так мало, что все активные процессы уже не могут выполняться одновременно.
- *Видимым (**Visible Process**)* называется процесс, не имеющий никаких приоритетных компонентов. Например, процесс А запустил деятельность не на весь экран. Пользователь работает с процессом Б, который также отображает активность. Активность процесса А не имеет фокуса, но все ещё видна пользователю, поэтому процесс А называется видимым. Также видимым считается процесс, выполняющий службу, которая связана с деятельностью, находящейся на переднем плане, но не активна или частично закрыта другой деятельностью.
- *Сервисным (**Service Process**)* называется процесс, в котором выполняется служба и который не относится к активным и видимым. Обычно сервисные процессы не имеют интерфейса, но они выполняют задания, необходимые пользователю, например, сервисным процессом может считаться фоновая работа проигрывателя. Поэтому система старается сохранить сервисные процессы.
- *Фоновым (**Background Process**)* считается процесс, в котором выполняется деятельность, которая в данный момент не видна пользователю. Фоновый процесс может быть уничтожен в любой момент, если понадобилась память активному или видимому процессу.

- *Пустой (**Empty Process**)* процесс вообще не содержит активных компонентов приложения. Система использует такие процессы, как кэш – для более быстрой последующей инициализации компонентов приложения. Такие процессы удаляются в первую очередь.

Представленная терминология должна помочь создать схему активностей и процессов.

3.3. Составление алгоритма работы приложения.

В данном пункте необходимо составить блок схему подключаемых библиотек и классов java в создаваемое приложение с указанием циклических структур и условных операторов. Блок – схема составляется по файлу проекта ***main.java*** (или другой заданный пользователем), который содержится в папке **src** вашего проекта.

3.4. Создание приложения и компиляция его в виде apk.файла.

В этом пункте необходимо указать структуру приложения (окно Package Explorer в Eclipse), добавляемые элементы xml-файлы, подключаемые службы файл *Androidmanifest.xml*, параметры разрешений приложения (*Permission*), если они необходимы и указать параметры скомпилированного файла.

3.5. Выполнение промежуточного тестирования проекта на эмуляторе в зависимости от варианта задания.

Данный пункт должен проиллюстрирован снимками экрана эмулируемого(ых) устройства(в). С указанием возникших ошибок и методов их исправления. Необходимо провести серию тестирований на различных эмуляторах, для того чтобы выявить возможности совместимости создаваемого приложения, в зависимости от индивидуального варианта задания.

Необходимо охватить все характерные сочетания параметров эксперимента. При этом из получаемых параметров для различных устройств должны быть отобраны все качественно различные. Нет необходимости включать в пояснительную записку все графики или рисунки экрана– достаточно ограничиться теми, которые дают наилучшее представление о поведении приложения. В случае если значительное изменение какого-либо параметра в приложении не приводит к заметному изменению характеристик самого приложения, это следует отметить в описании результатов экспериментов.

3.6. Подготовка расчётно-пояснительной записки (с обязательным включением в неё обобщённых результатов тестирования, проиллюстрированных рисунками экрана, демонстрирующими различные режимы работы приложения и его совместимость с различными версиями Android).

Обратите внимание на то, что часть тем является универсальной, совместимой с любой версией Android, а другие ориентированы на конкретную версию, это связано с параметрами приложения и его функциональными возможностями.

Коды приложения и вид графического интерфейса должны быть представлены в приложении и не входят в основную часть пояснительной записки. Пояснительная записка составляет от 20-50 стр. форматированного текста с параметрами : Time New Roman -14pt, отступ для абзаца – 1.25, междустрочный интервал – 1.5, форматирование по ширине. В пояснительной записке должны быть указаны наименования рисунков (Рис.№) и таблиц (Табл.№).

3.7. Порядок защиты курсовой работы

Для успешной защиты курсовой работы, продемонстрировать работу разработанной программы для указанных преподавателем параметров системы, представить расчётно-пояснительную записку и ответить на вопросы. Примерная тематика вопросов:

1. Описать структуру деятельностей приложения на платформе Android.
2. Описать процессы, происходящие в процессе запуска приложения (приоритет процессов, определение критического приоритета процесса, влияние процессов на активность приложения).
3. Изложить суть алгоритма, используемого в приложении, продемонстрировать, модель активности для своего варианта задания.
4. Прокомментировать отдельные фрагменты manifest файла приложения и объяснить роль подключаемых в приложение компонентов.
5. Прокомментировать отдельные фрагменты apk-файла, созданного для проведения численных экспериментов.
6. По полученным снимкам экрана определить параметры совместимости приложения для различных версий Android.
7. Используя снимки экрана предложить условия для модернизации приложения и расширения его функциональных возможностей.

4. Рекомендательный библиографический список

1. Голощапов А.Л. Google Android:системные компоненты и сетевые коммуникации.-СПб.:БХВ-Петербург,2012.-384 с.: ил., ISBN 978-5-9775-0666-3.
2. Голощапов А.Л. Google Android: программирование для мобильных устройств. – СПб.:БХВ-Петербург,2010. – 448с.: ил., ISBN 978-5-9775-0562-8.
3. Голощапов А.Л. Google Android. Создание приложений для смартфонов и планшетных ПК. – СПб.:БХВ-Петербург,2013.
4. http://developer.android.com – официальный сайт Android.
5. http://www.androidjavadoc.com – сайт-справочник по библиотекам Android.
6. Колисниченко Д.Н. Программирование для Android. Самоучитель. – СПб.: БХВ-Петербург, 2012. – 272 с.: ил., ISBN 978-5-9775-0770-7.
7. Бурнет Э. Привет Android! Разработка мобильных приложений. – СПб.: Питер,2012. – 256с.: ил., ISBN 978-5-459-01015-2.
8. Майер Р. Android 2: программирование приложений для планшетных компьютеров и смартфонов: [пер. с англ.] / Рето Майер. – М.: Эксмо,2011. – 672с. – ISBN 978-5-699-50323-0.
9. Хашими С., Коматинени С., Маклин Д. Разработак приложений для Android. – СПб.: Питер,2011. – 736 с.: ил., ISBN 978-5-459-00530-1.

Приложение А

Варианты задания на курсовую работу.

1. Разработка Android-приложения для контроля сетевой активности мобильного устройства.
2. Создание калькулятора для расчета нагрузки сетевого оборудования на платформе Android 4.1 для смартфона .
3. Создание виджета для планшета для версии Android 4.2.
4. Создание справочника по структуре операционной системы Android.
5. Создание справочника по видам сетевых устройств на платформе Android.
6. Создание приложения для передачи мультимедиафайлов по Bluetooth на платформе Android.
7. Файловый менеджер для смартфона на платформе Android 4.1.
8. Приложение для шифрования данных для планшета на платформе Android 4.0.3
9. Графический редактор для смартфона на платформе Android 4.2.
10. Приложение для беспроводного доступа к рабочей станции на платформе Android 4.0.
11. Справочник по составным компонентам архитектуры операционной системы WP8.

Приложение Б

Пример оформления схемы деятельности приложения.

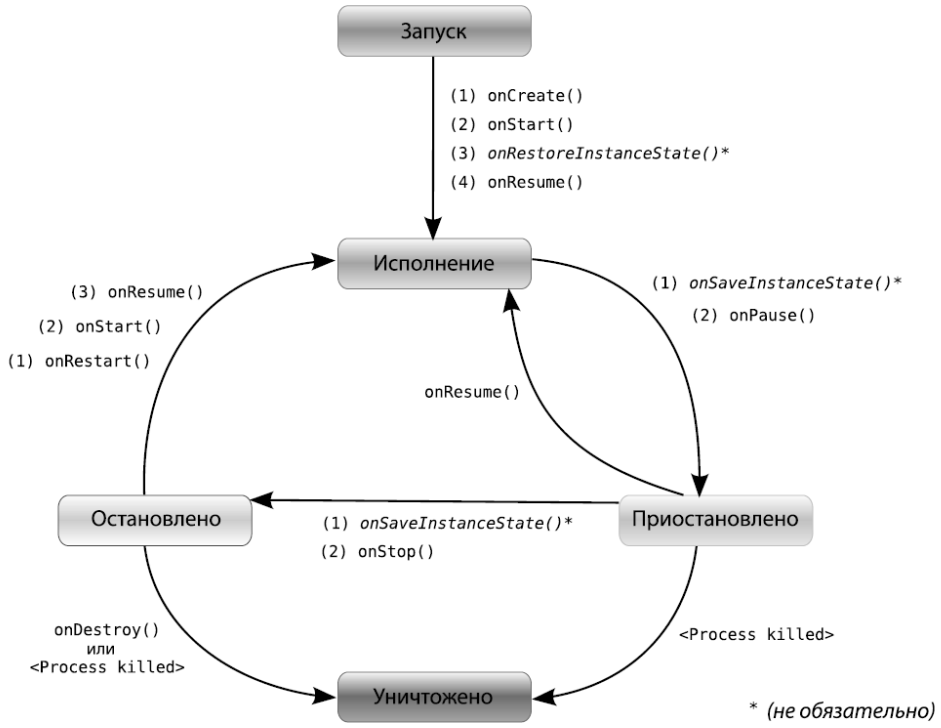

Рис.1

<u>Пример разработки UML –диаграммы приложения</u>.
Для разработки UML – диаграммы можно воспользоваться редактором Microsoft Word, Microsoft Excel или Microsoft Visio или open source – StarUML, ArgoUML, Violet UML Editor, UMLGraph или плагины для Eclipse – Eclipse UML2 Tools, tangible T4 Editor plus UML modeling Tools for Visual Studio (2008/2010), также можно установить дополнение для Eclipse – UML2 sdk(Рис.2)

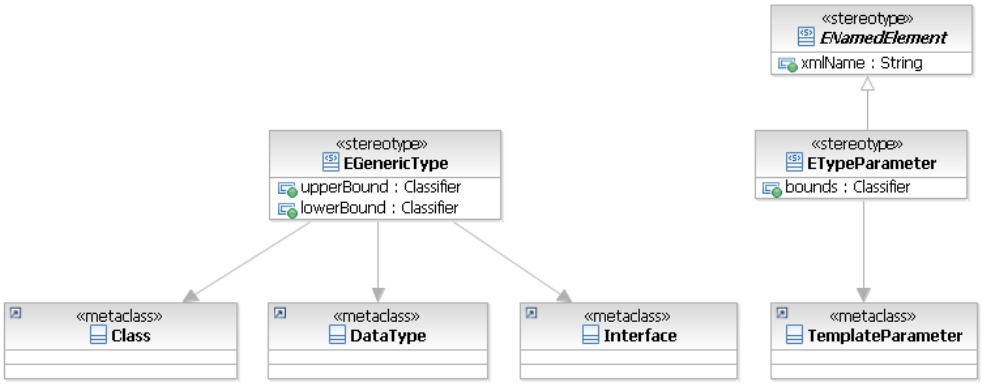

Рис.2.

Приложение В

Пример тестирования и отладки приложения.

Отладка с помощью отладчика

Также можно использовать отладчик Eclipse для установки точки останова, пошагово исполнять программу или просматривать ее состояние. Для этого в файле AndroidManifest.xml нужно добавить команду **android:debuggable="true"**.

```
<application android:icon="@drawable/icon"
    android:label="@string/app_name"
        android:debuggable="true">
        …
```

Сохраните изменения в файле, щелкните правой кнопкой мыши на проекте и выберите команду **Debug As | Android Application**.

Создаём конфигурацию для отладки

Для создания конфигурации по отладке приложения выберите в Eclipse меню **Run | Debug Configurations…**.

В открывшемся диалоговом окне **Debug Configurations** выберите нужный проект из левой части окна. Если в списке приложения нет, то выберите через кнопку **Browse...**.

Поле **Name** в верхней части окна оставляют без изменений, чтобы оно совпадало с именем приложения, но если вы большой оригинал и любите путаться в проектах, то можете задать другое имя.

Щёлкните кнопку **Debug**.

Если эмулятор ещё не запущен, то он запустится, как обычно это происходит при обычном режиме Run. Единственное отличие - при запуске вашей программы появится диалоговое окно **Waiting for Debugger**, которое закроется через несколько секунд.

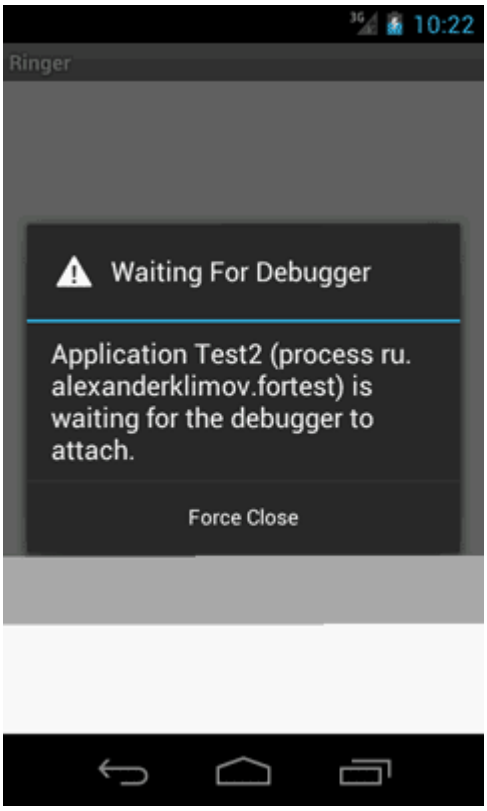

Рис.3. Процесс отладки.

Теперь можно приступать к отладке.

У вам может автоматически открыться новая перспектива **Debug**. Если этого не произошло, то выберите в меню **Window | Open Perspective | Debug**.

Перспектива содержит несколько панелей. Например, в левой верхней части находится панель **Debug**, на которой можно наблюдать за потоками вашей программы.

Справа от этой панели находятся панели **Variables** и **Breakpoints**, позволяющие следить за значениями переменных и управлять точками останова.

В нижней части перспективы можно увидеть панели **Console** и **LogCat.**

Попробуем установить точку останова. Выберите нужную строчку кода и слева от её номера строки щелкните правой кнопкой мыши. Выберите пункт **Toggle Breakpoint** (либо сделать двойной щелчок в этой области). Если прицеливаться вам неохота, то просто установить курсор на нужной строке и нажмите комбинацию клавиш **Ctrl-Shift-B.**

В качестве эксперимента установите точку останова на следующей строке после **setContentView(R.layout.activity_test)**. Например, это может быть строка:

```
textOut = (EditText) findViewById(R.id.textout);
```

Запустите отладку. Это можно сделать через значок жучка на панели инструментов, клавишу F11 или через меню, как было описано в начале данного раздела.

В эмуляторе запустится ваше приложение и затем остановится, когда дойдет до строчки кода, на которую вы поставили точку останова. Строка будет подсвечена зелёным цветом, а само текстовое поле не загрузится на экран (как и остальные элементы, которые идут после этой строки кода).

На панели **Debug** вы увидите различные значки для управления отладкой.

Например, чтобы перейти на следующую строку кода, нажмите на значок **Step Over** или нажмите клавишу F6.

Это базовое описание работы по отладке. Вам следует самостоятельно изучить эти приёмы.

Быстрое отключение журналирования

Настоятельно рекомендуется удалять все вызовы LogCat в готовых приложениях. Если проект очень большой и вызовы журналирования разбросаны по всем местам кода, то ручное удаление (или комментирование) становится утомительным занятием. Многие разработчики используют следующую хитрость - создают обёртку вокруг вызова методов LogCat.

```
public static final boolean isDebug = false;

public final String TAG = "MyLogger";


public void MyLogger(String statement){

        if (isDebug) {

                Log.v(TAG, statement);

        }

}
```

Теперь остаётся только присвоить нужное значение переменной **isDebug** перед созданием готового apk-файла для распространения.

LogCat на устройстве

Рассмотрим пример для просмотра сообщений LogCat на устройстве

Разметка:

```
<?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" >

    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Разметка для элемента списка:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/txtLogString"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Добавляем разрешение в манифест:

```
<uses-permission android:name="android.permission.READ_LOGS" />
```

Код для класса:

```
package ru. proba.test;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;
import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.Context;
import android.graphics.Color;
```

```
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
public class TestActivity extends ListActivity {
        private LogStringAdaptor adaptor = null;
        private ArrayList<String> logarray = null;
        private LogReaderTask logReaderTask = null;
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_test);
                logarray = new ArrayList<String>();
                adaptor  =  new  LogStringAdaptor(this,  R.id.txtLogString,
logarray);

                setListAdapter(adaptor);
                logReaderTask = new LogReaderTask();
                logReaderTask.execute();
        }
        @Override
        protected void onDestroy() {
                logReaderTask.stopTask();
                super.onDestroy();
        }
```

```
	@Override
	protected void onListItemClick(ListView l, View v, int position,
long id) {
		super.onListItemClick(l, v, position, id);

		final AlertDialog.Builder builder = new AlertDialog.Builder(
				TestActivity.this);
		String text = ((String) ((TextView) v).getText());

		builder.setMessage(text);
		builder.show();
	}
	private int getLogColor(String type) {
		int color = Color.BLUE;

		if (type.equals("D")) {
			color = Color.rgb(0, 0, 200);
		} else if (type.equals("W")) {
			color = Color.rgb(128, 0, 0);
		} else if (type.equals("E")) {
			color = Color.rgb(255, 0, 0);
			;
		} else if (type.equals("I")) {
			color = Color.rgb(0, 128, 0);
			;
		}

		return color;
	}
```

```
        private class LogStringAdaptor extends ArrayAdapter<String> {
                private List<String> objects = null;

                public LogStringAdaptor(Context context, int textviewid,
                                List<String> objects) {
                        super(context, textviewid, objects);

                        this.objects = objects;
                }

                @Override
                public int getCount() {
                        return ((null != objects) ? objects.size() : 0);
                }
                @Override
                public long getItemId(int position) {
                        return position;
                }

                @Override
                public String getItem(int position) {
                        return ((null != objects) ? objects.get(position) :
null);
                }

                public View getView(int position, View convertView,
ViewGroup parent) {
                        View view = convertView;

                        if (null == view) {
```

```
                                LayoutInflater    vi    =    (LayoutInflater)
TestActivity.this

        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
                                view = vi.inflate(R.layout.logitem, null);
                        }

                        String data = objects.get(position);

                        if (null != data) {
                                TextView textview = (TextView) view

        .findViewById(R.id.txtLogString);
                                String type = data.substring(0, 1);
                                String line = data.substring(2);

                                textview.setText(line);
                                textview.setTextColor(getLogColor(type));
                        }

                        return view;
                }
        }

        private class LogReaderTask extends AsyncTask<Void, String, Void> {
                private final String[] LOGCAT_CMD = new String[] { "logcat"
};
                private final int BUFFER_SIZE = 1024;

                private boolean isRunning = true;
                private Process logprocess = null;
```

```
                private BufferedReader reader = null;
                private String[] line = null;
                @Override
                protected Void doInBackground(Void... params) {
                        try {
                                logprocess                                        =
Runtime.getRuntime().exec(LOGCAT_CMD);
                        } catch (IOException e) {
                                e.printStackTrace();

                                isRunning = false;
                        }
                        try {
                                reader      =      new      BufferedReader(new
InputStreamReader(
                                                logprocess.getInputStream()),
BUFFER_SIZE);
                        } catch (IllegalArgumentException e) {
                                e.printStackTrace();
                                isRunning = false;
                        }
                        line = new String[1];
                        try {
                                while (isRunning) {
                                        line[0] = reader.readLine();
                                        publishProgress(line);
                                }
                        } catch (IOException e) {
                                e.printStackTrace();
                                isRunning = false;
                        }
```

```
                        return null;
                }
                @Override
                protected void onCancelled() {
                        super.onCancelled();
                }
                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                }
                @Override
                protected void onPostExecute(Void result) {
                        super.onPostExecute(result);
                }
                @Override
                protected void onProgressUpdate(String... values) {
                        super.onProgressUpdate(values);
                        adaptor.add(values[0]);
                }
                public void stopTask() {
                        isRunning = false;
                        logprocess.destroy();
                }
        }
}
```

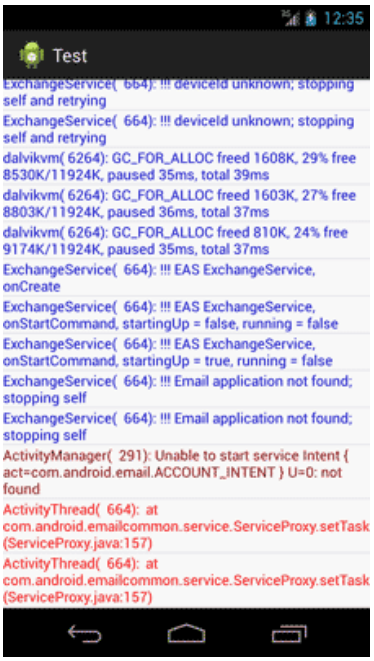

Рис.4. Окно отладчика LogCat на устройстве Android.

Оглавление