Android Platform

Basics and Application Development

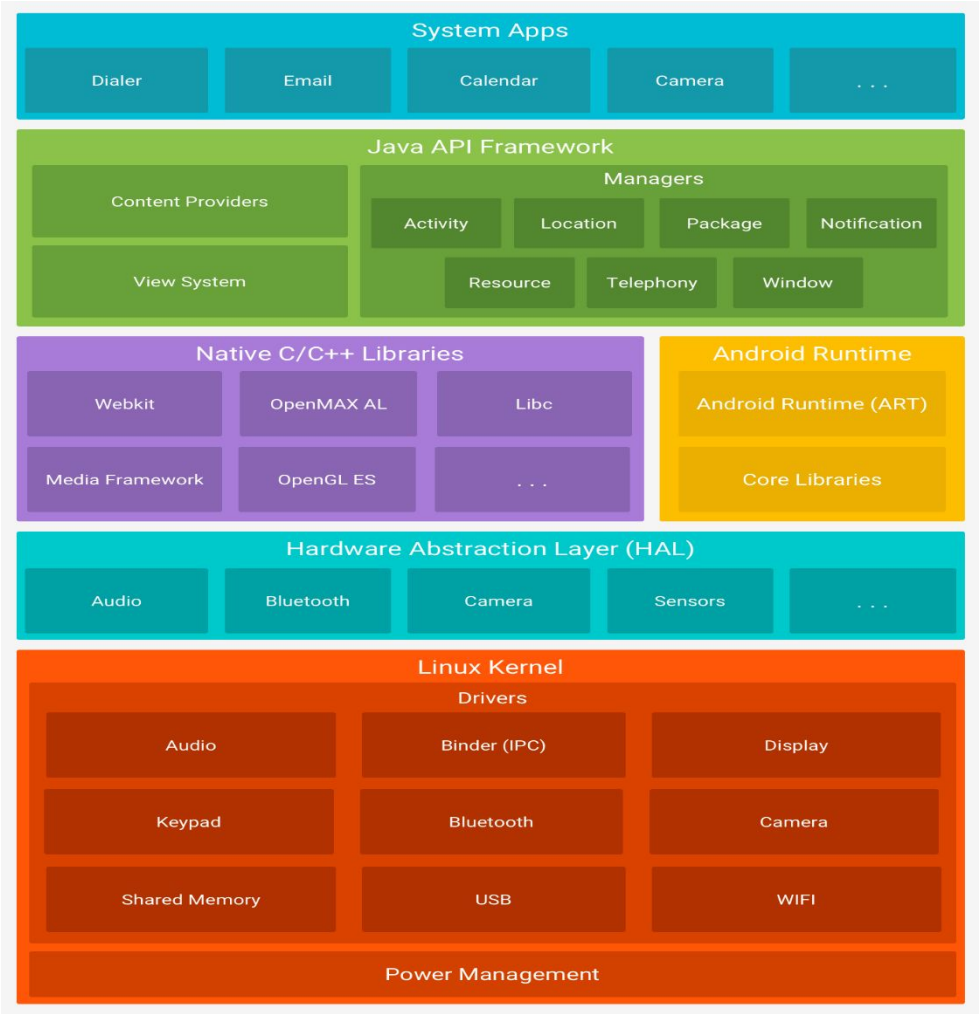
- Platform Architecture
- Applications development overview
- Main concepts and components
- Framework review
- New features of Android platform

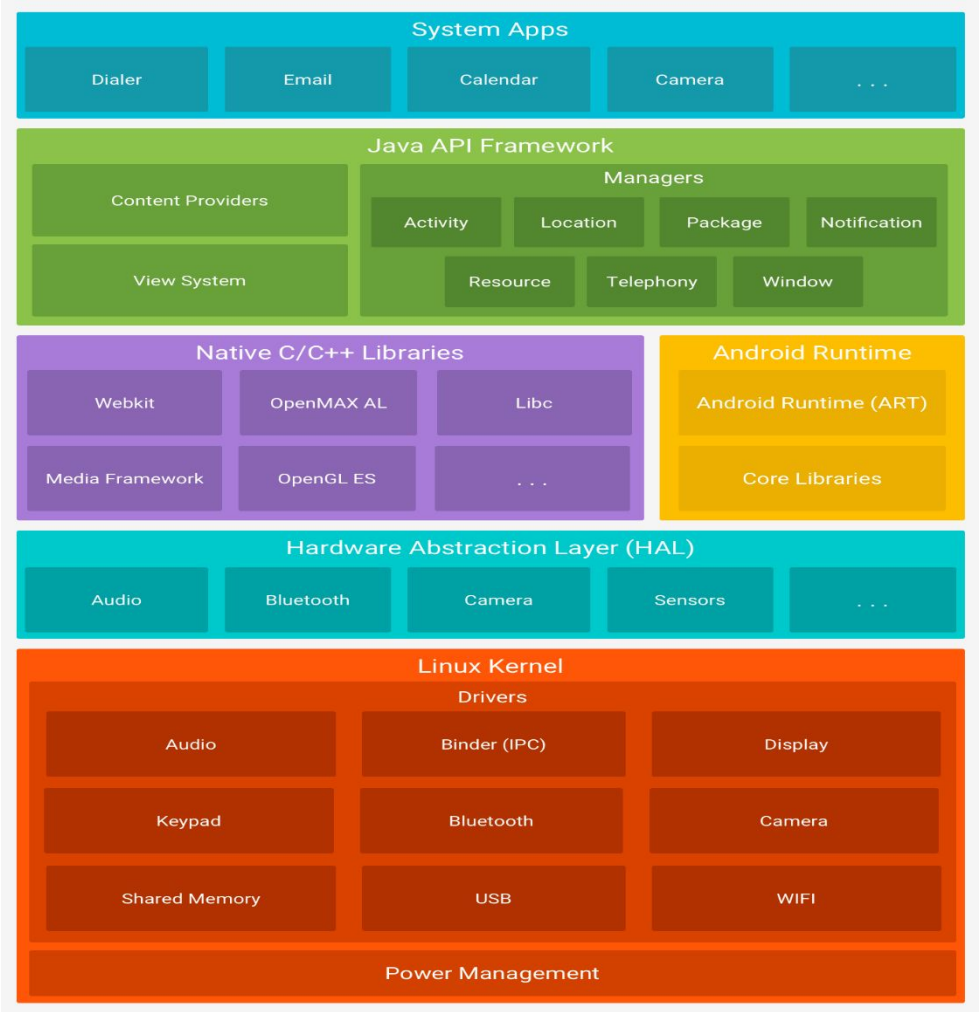
Platform Architecture

Architecture

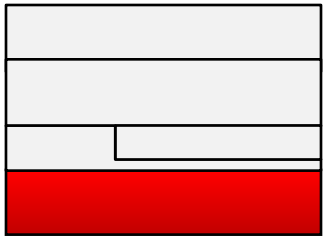
- Linux kernel
- Native libraries
- Runtime environment
- Application framework
- Applications

System Architecture

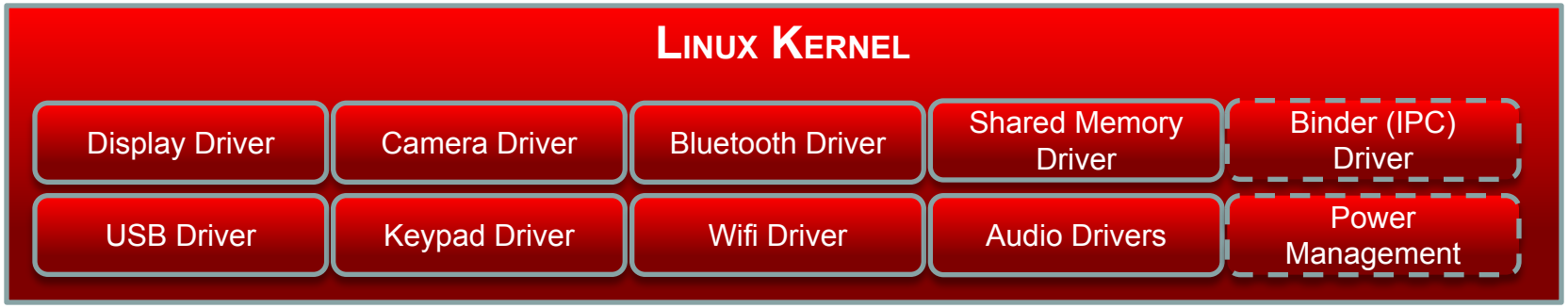




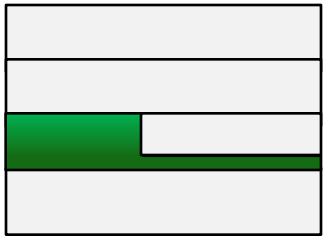
Linux Kernel 2.6/3.4



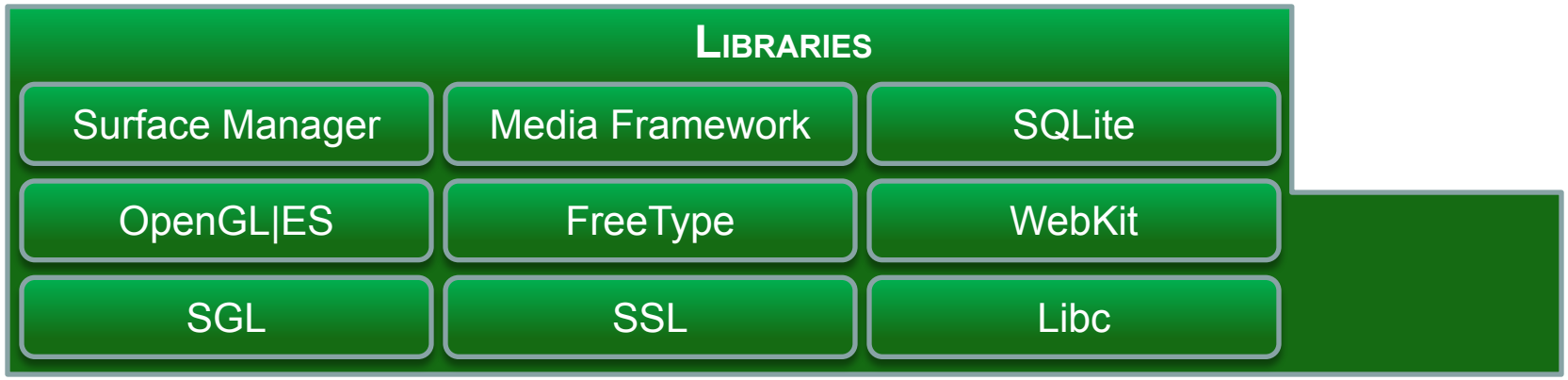
- Open source
- Improved memory manager
- Own process and threads management
- Proven driver model
- Loadable modules support



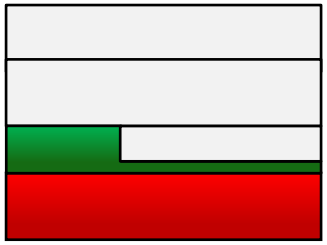
Native Libraries



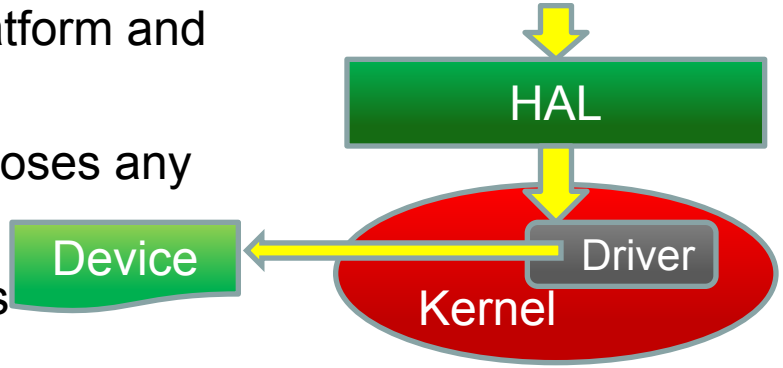
- Bionic (Libc)
- Function libraries
- Native servers
- Hardware Abstraction Layer (HAL)



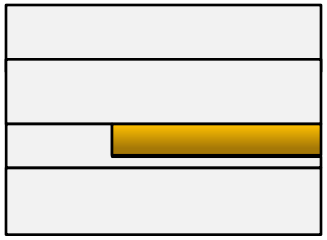
Hardware Abstraction Layer



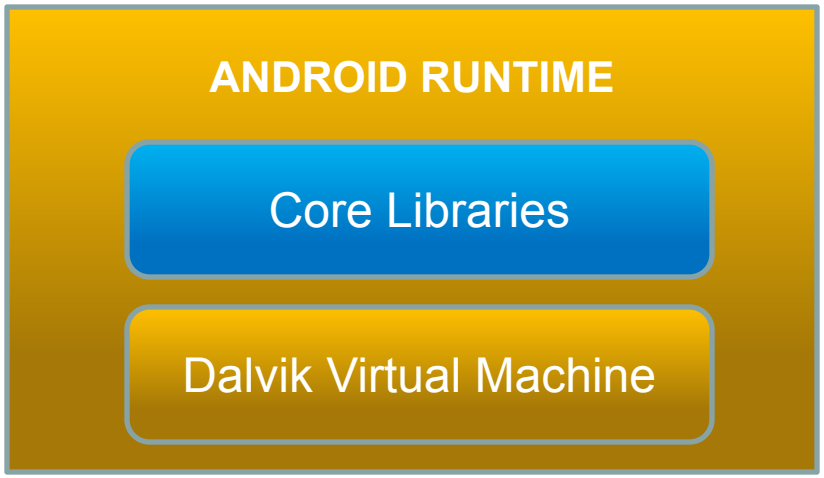
- User space C/C++ library layer
- Defines the interface that Android hardware “drivers” have to implement
- Not all components have standardized kernel driver interfaces
- Separates the logic of Android platform and the hardware interface
- Kernel drivers are GPL which exposes any proprietary IP
- Android has specific requirements for hardware drivers



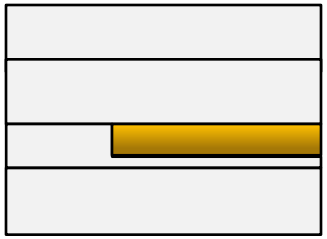
Android Runtime



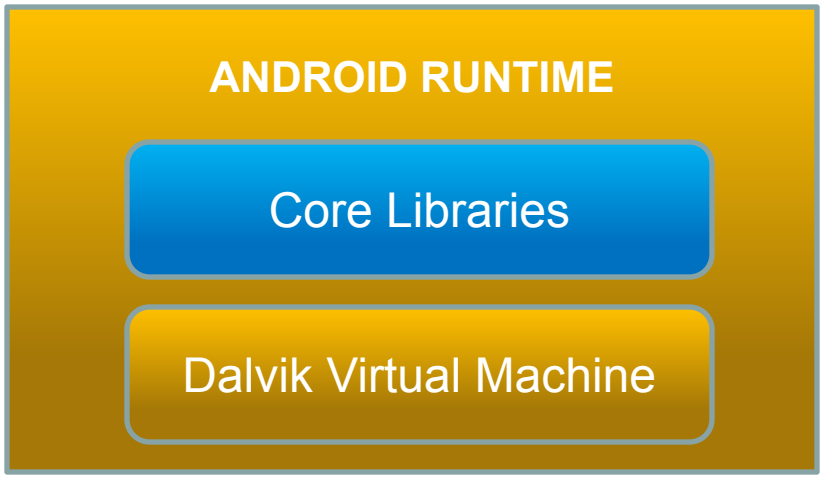
- Android has own implementation of VM (Dalvik)
  - Provides application portability and runtime consistency
  - Runs ARM-optimized byte code (Dalvik EXecutable, .dex)
  - Java .class/.jar files are transformed to .dex at build time
  - Supports multiple virtual machine processes per device
  - Efficiency use memory at runtime



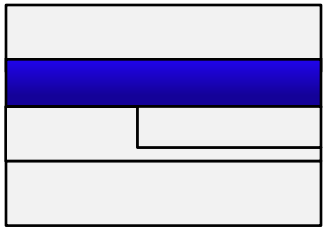
Core Libraries



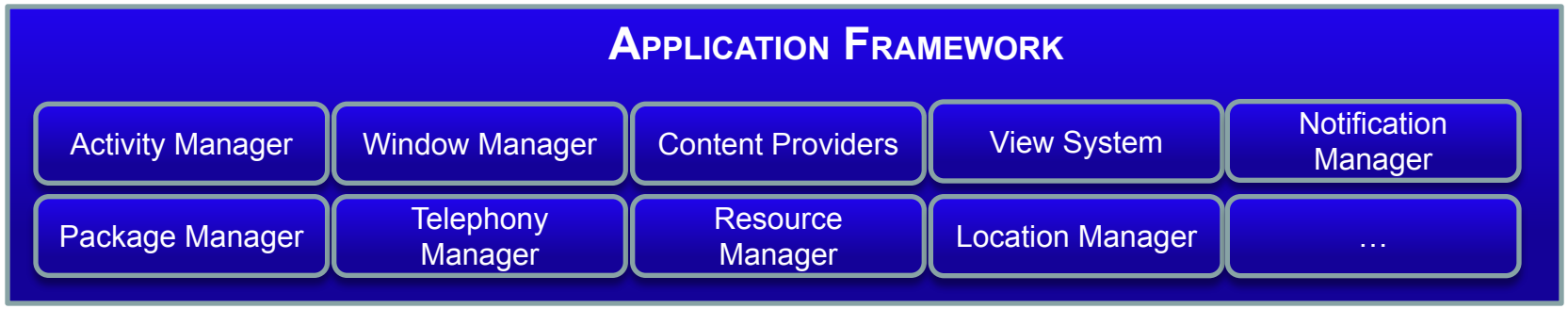
- Core API of Java5 provides a powerful, simple and familiar development platform
  - Data structures (java.net)
  - File access (java.io)
  - Network access (java.net)
  - Utilities (java.util)



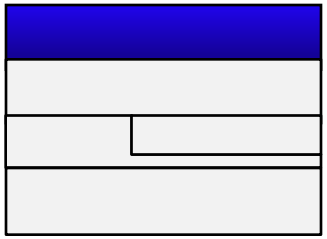
Application Framework



- Core platform functionality
  - Activity Manager, Package Manager, Window Manager, Resource Manager
  - Content Providers
  - View System
- Hardware services
  - Telephony, Location, Bluetooth, Wi-Fi, USB, Sensor



Applications



- The Android platform has a variety of Java applications
  - Home - displays applications shortcuts, widgets, supports custom wall paper.
  - Phone - supports regular telephony functions.
  - Web Browser - WebKit-based browser that supports HTML, XHTML, CSS and JavaScript.
  - Email - provides access to email servers via POP3, IMAP4, and SMTP.
  - Media Player - enables managing, importing, and playback of media content.
  - Alarm Clock, Calculator, Calendar, Camera, Contacts, IM, MMS, Settings, Voice Dialer, and others.

Applications development overview

Applications

- Environment
- Eclipse and Ant Projects
- Project Structure
- JIT. Basic information
- Debugging (Eclipse Memory Analyzer)

Environment

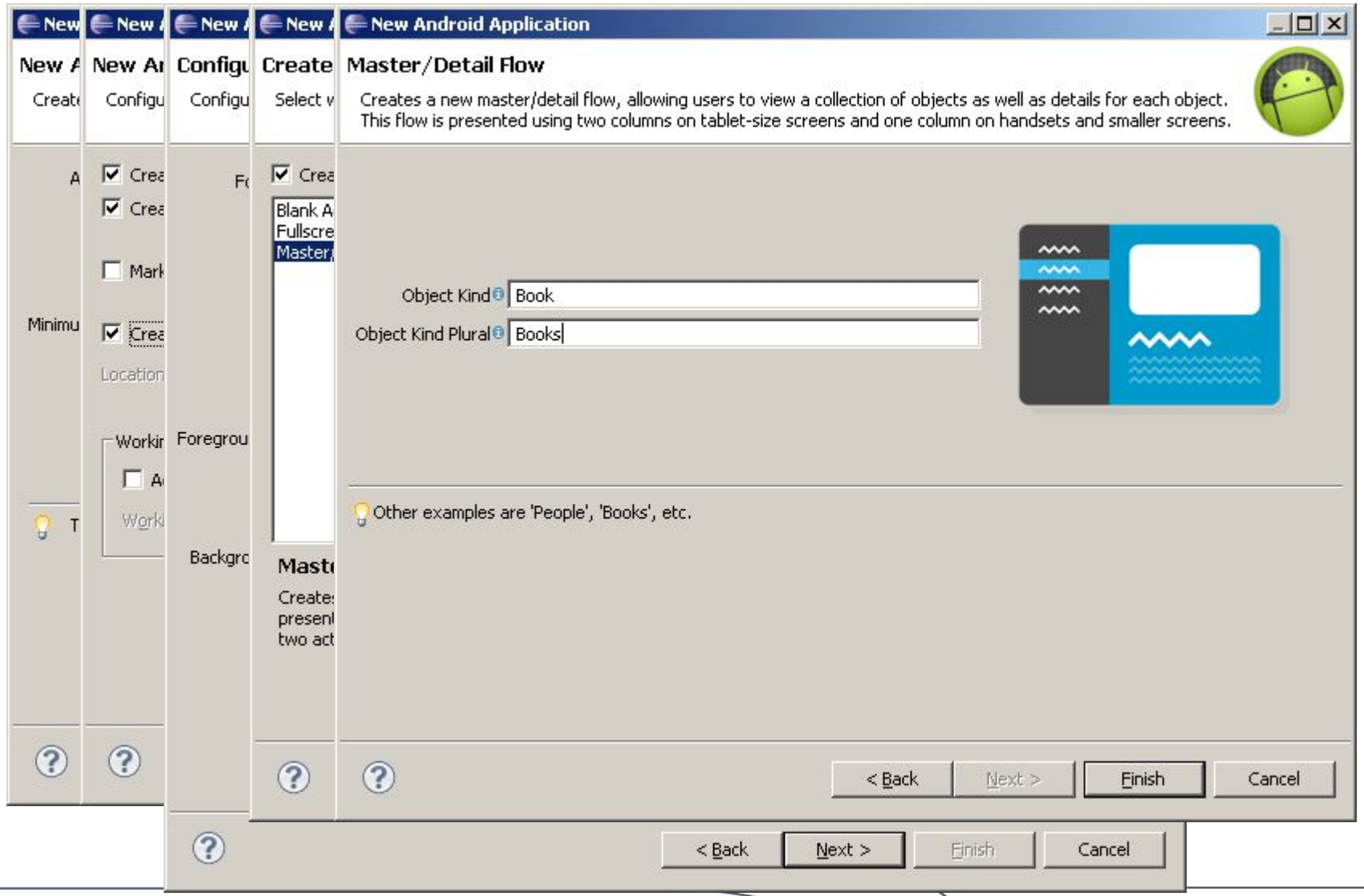
- JDK5 or JDK6
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Android SDK
  - http://developer.android.com/sdk/index.html
- Eclipse IDE 3.4 or higher
  - http://www.eclipse.org/downloads
- Android development tools (ADT) as Eclipse plug-in
  - http://developer.android.com/sdk/installing.html
- Apache Ant
  - http://ant.apache.org

  - Detailed setup introduction:
    http://developer.android.com/sdk/index.html

Installing procedure

- Get and install JDK5/6
- Get and install the ADT bundle (Eclipse+ADT)
- Alternative – get the new Android Studio IDE (based on IntelliJ IDEA)
- Use an existing Eclipse IDE
  - Get Android SDK Tools
  - Run Eclipse
  - Install ADT thru Help/Software Updates/Add Site/Archive menu
  - Specify SDK thru Window/Preferences/Android/SDK Location menu

Eclipse Project



Ant project

- Android/android-sdk/tools/android.bat

- android.bat create project --target 14 --name ExampleApplication --path ./ExampleApplication --activity MainActivity --package com.myexample.exampleapplication

Project Structure

- TestApplication/src/com/myexample/exampleapplication/MainActivity.java
  - The main Activity of the project
- TestApplication/gen/com/myexample/exampleapplication/R.java
  - The project dynamically generated resource file
- TestApplication/Android-x.x/android.jar
  - Reference to used Android SDK library
- TestApplication/AndroidManifest.xml
  - The project manifest

Project Structure (contd.)

- TestApplication/res/drawable-*dpi/ic_launcher.jpg
  - The project images
- TestApplication/res/layout/main.xml
  - The main activity UI layout
- TestApplication/res/values/strings.xml
  - The project string resources
- TestApplication/res/values-xx/strings.xml
  - The project localized string resources

MainActivity.java

```
package com.myexample.exampleapplication;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY. */
package com.myexample.exampleapplication;

public final class R {
 public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class menu {
        public static final int main=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050002;
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
...
}
```

./Android-xx/Android.jar

- Provides:
  - J2SE5 classes
  - Android-specific classes
  - 3-rd party classes
- Can’t be extended by application developer

AndroidManifest.xml

- Is XML file that is required file for every application
- Lists all of the components of your application
- Describes capabilities and behaviors of each component
- Contains Intent Filters which describe where and when each Activity can be started
- Describes global values for an application and permissions requested by/for the application from Android environment

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.myexample.exampleapplication"
    android:versionCode="1" android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="15" />

    <application
        android:allowBackup="true" android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity
            android:name="com.myexample.exampleapplication.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Project resources

- ./res/drawable/ should contain drawable resources like images
- The name of resource IDs are defined by resources names in Java notation. Be sure that all of your resources have correct names

./res/layout/main.xml

```
<RelativeLayout
   xmlns:android=http://schemas.android.com/apk/res/android
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />

</RelativeLayout>
```

./res/values/strings.xml

- Describes text resources
- Typically contains string values

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">ExampleApplication</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>

</resources>
```

Debugging

- Android Debug Bridge (ADB)
- Dalvik Debug Monitor Server (DDMS)
- Logcat
- Traceview
- Eclipse Memory Analyzer

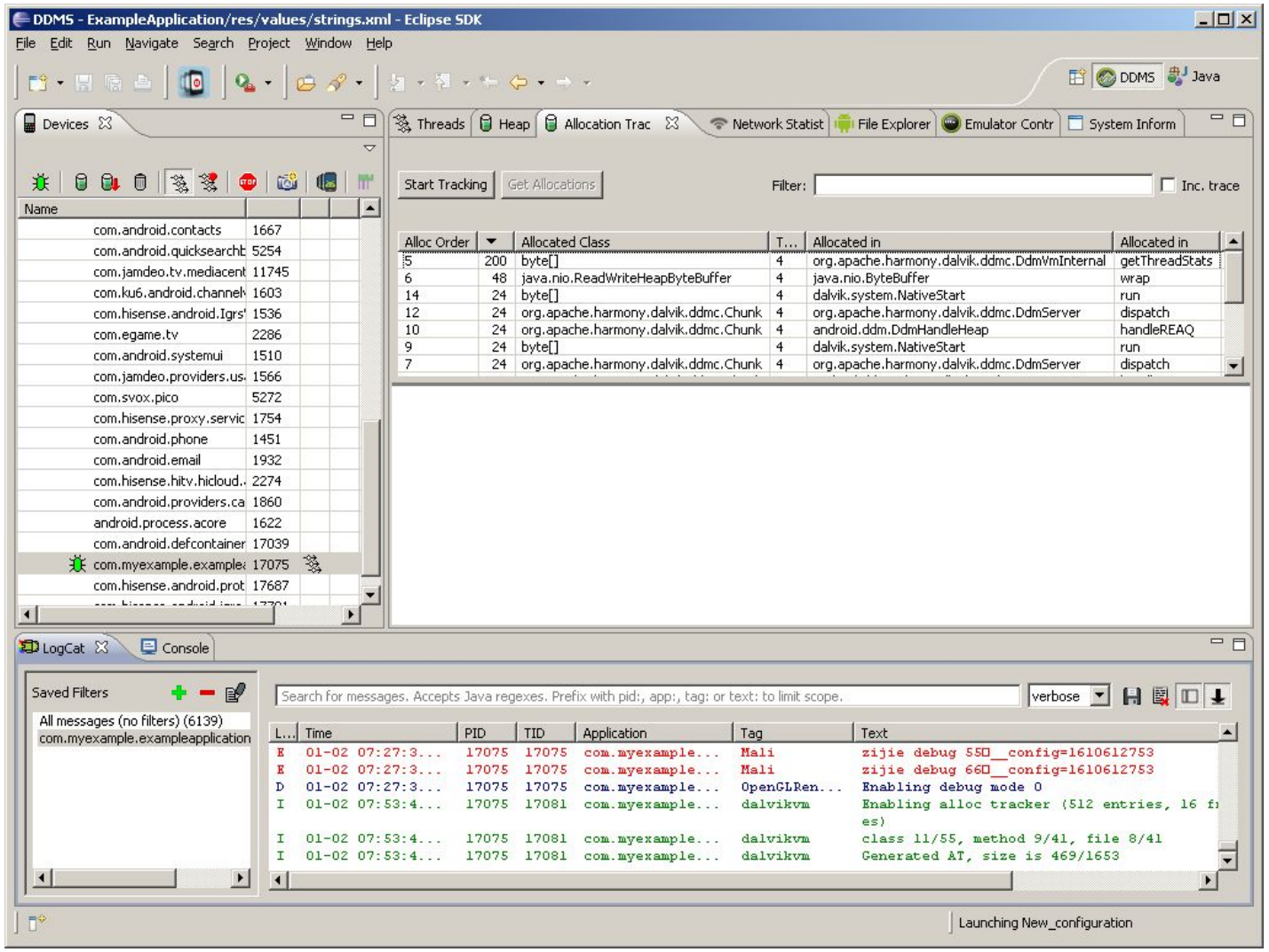
Debugging: ADB

- Device management
  - adb devices
  - adb connect 192.168.1.10
  - Adb disconnect 192.168.1.10
- Moving files and directories
  - adb push D:\TestProject\bin\TestProject.apk /sdcard
  - adb pull /sdcard/test.log D:\tmp
- Shell
  - adb shell
  - adb shell ps
- Port forwarding
  - adb forward tcp:5139 tcp:5139

Debugging: DDMS

- Thread and heap information
- Process information
- SMS and incoming calls spoofing
- Location data spoofing
- Screen capture
- Port-forwarding service
- LogCat

Debugging: DDMS



Debugging: Logcat

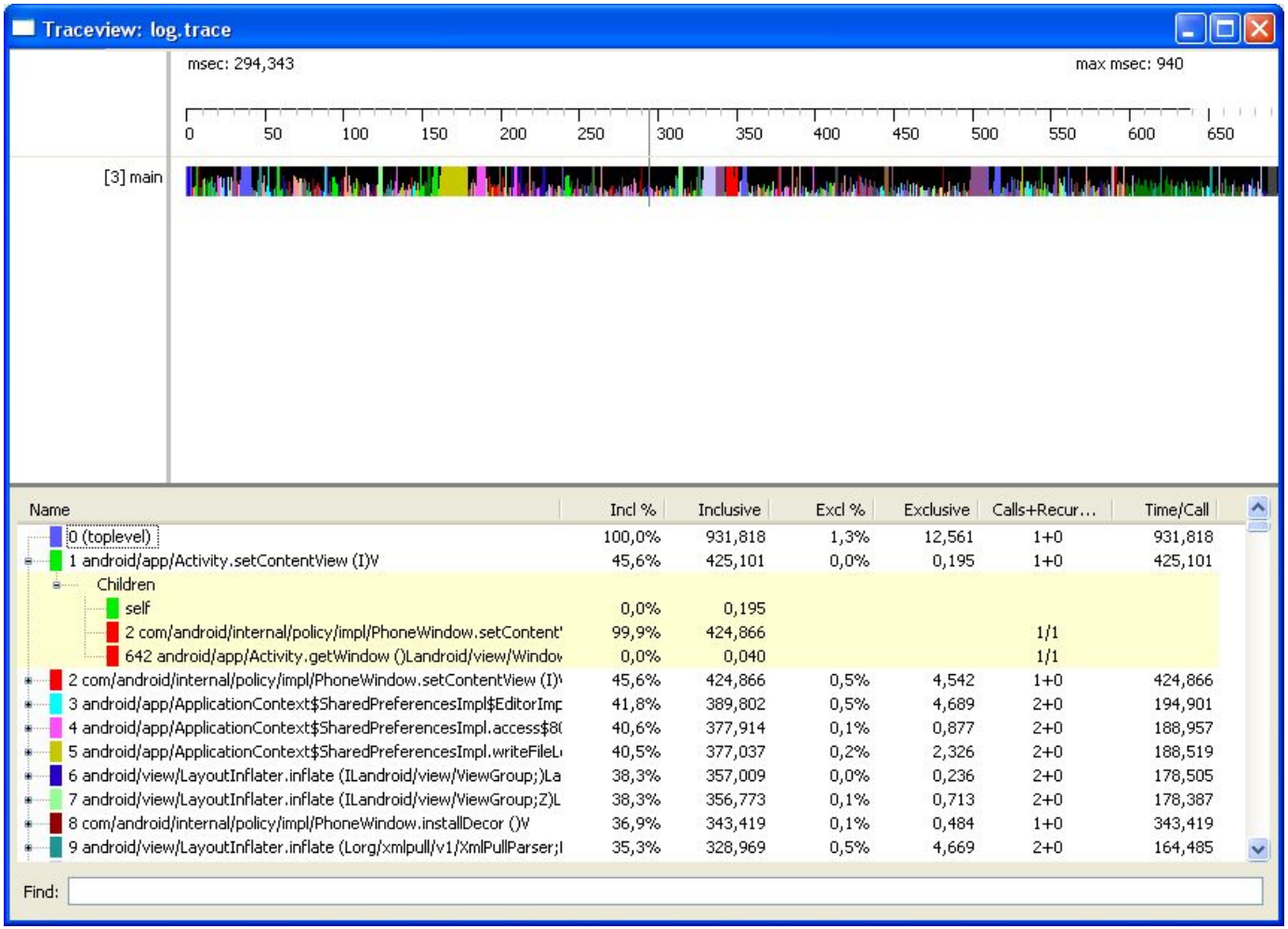
- Logs device messages
  - adb logcat
  - DDMS – Device/Run LogCat menu
  - Eclipse – Window/Show view/LogCat menu
- *Log* class is used instead of *System.out* to print messages
  - int Log.d(String tag, String msg)
  - Log.e(), Log.d(), Log.v(), Log.i() and Log.w() are generally used

Debugging: Traceview

- Graphical tool to view application traces
- Trace file *.trace* is used as input
  - traceview.bat log.trace
- Linear piece of code is used to get *.trace* file

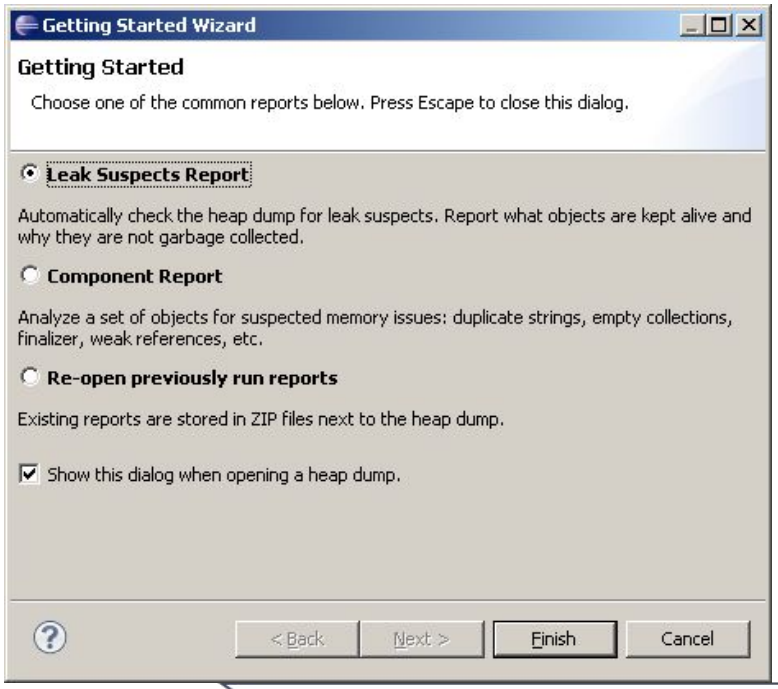
```
// start tracing to "/sdcard/log.trace"
Debug.startMethodTracing("log");


...


// stop tracing
Debug.stopMethodTracing();
```

Debugging: Traceview



Debugging: Eclipse Memory Analyzer

- Install from http://download.eclipse.org/mat/1.3/update-site/.
- What it can do:
  - Get a heap dump
  - Find memory leaks
  - Analyze Java collection usage
  - Detect duplicated/conflicting libs/classes
- Convert to Sun format
  - hprof-conf android.hprof sun.hprof
- Create a heap dump using the Dump HPROF file button in the DDMS *Perspective*



Practice

(Slides 1-7 from ContentProvider_UI_Adapter presentation)

Main concepts and components

- Basics of application development
- Thread interaction mechanisms
- Data storages
- UI building blocks

Main concepts and components

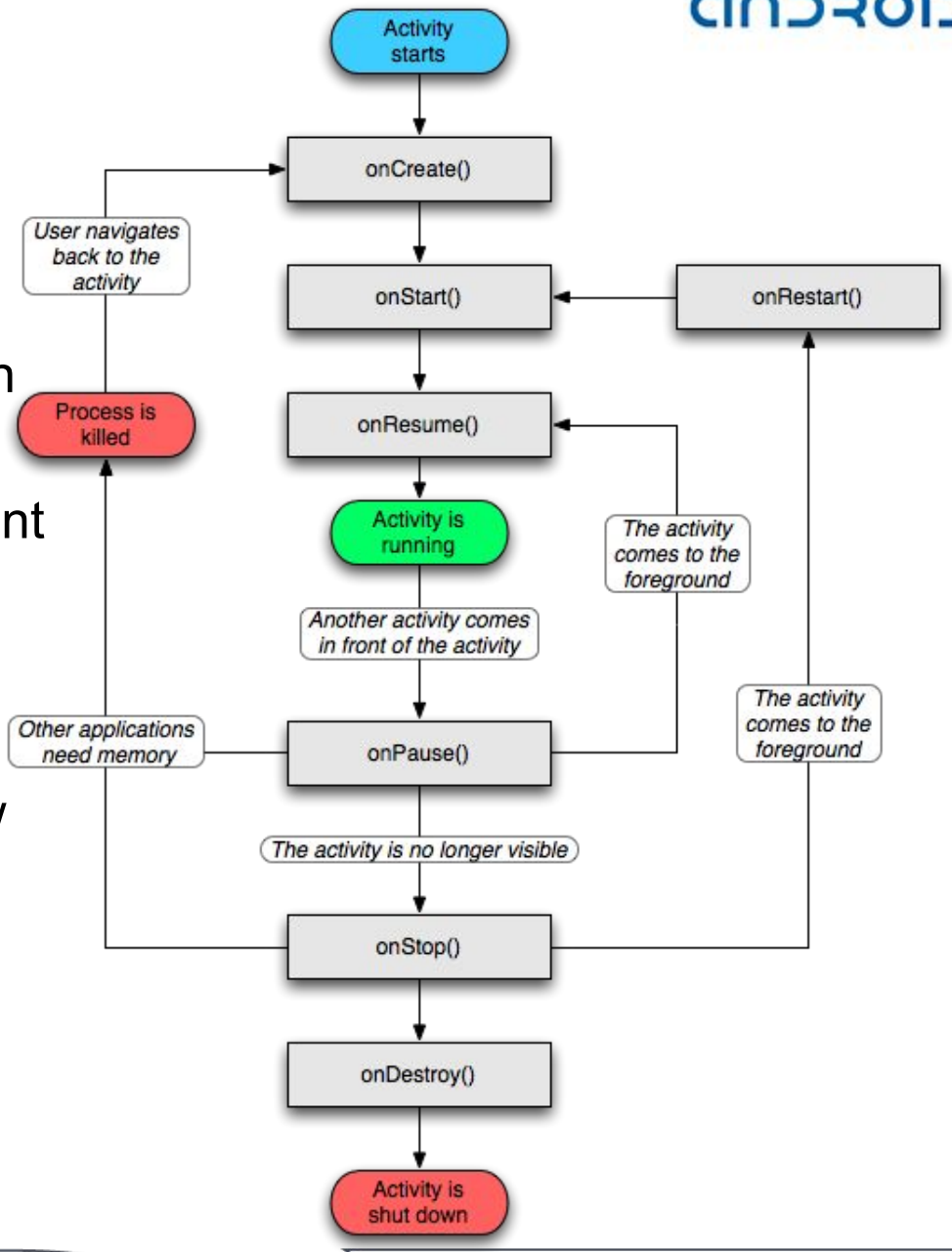
Basics of Application Development

Basics of application development

- Activities
- Tasks and Back Stack
- Intents
- Services
- Broadcast Receivers
- Content Providers
- Applications
- Differences between core and user apps (no other suitable place for this item)
- Security (Security part of the training)

Activities

- Single focused thing which can interact with user
- Each activity is independent of the others
- Takes care of creating a UI window
- The content of the window is provided by a views hierarchy

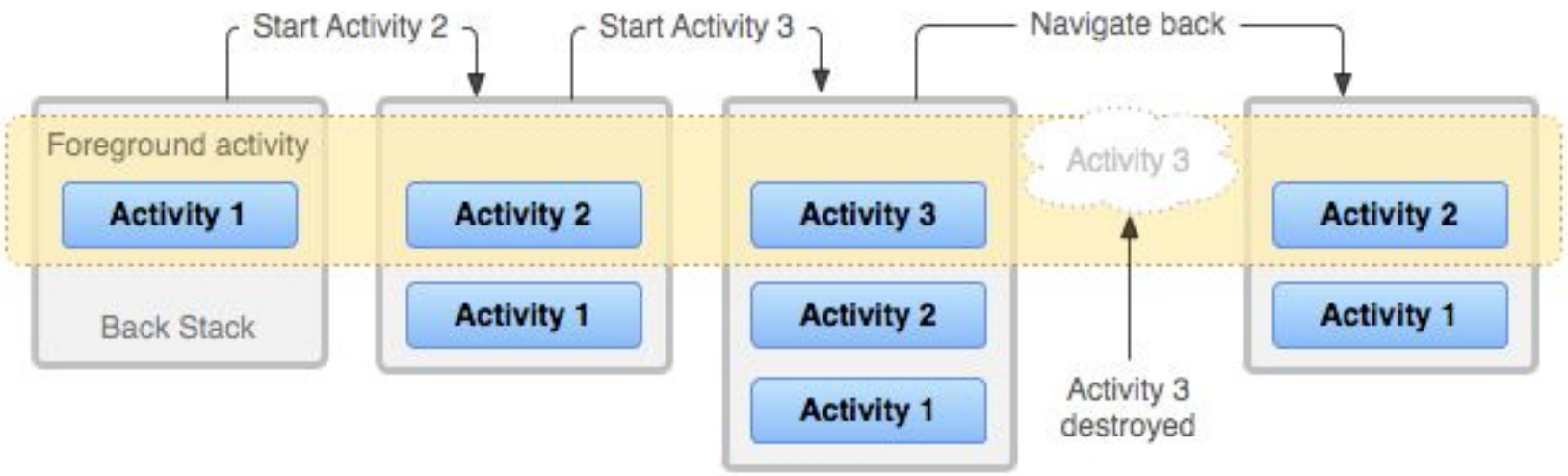


Activities

```
public class TestActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // useful code
    }

    /** Called when an activity is going into the background */
    @Override
    protected void onPause() {
        // useful code
        super.onPause();
    }
}
```

Tasks and Back Stack

- Task is a collection of activities user interacts with when performing a certain job. Activities inside task are organized as a stack.
- Default system mechanism for task management

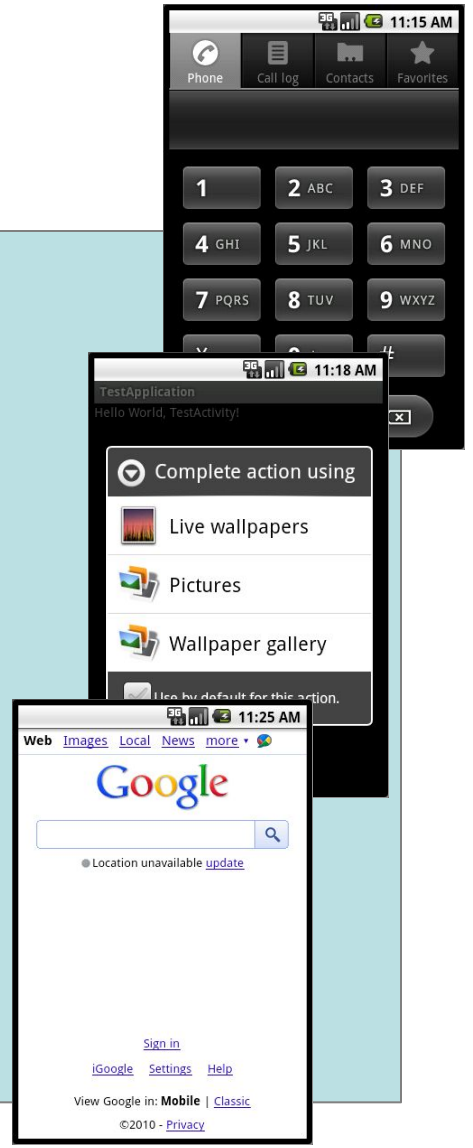


Intents

- A passive data structure holding an abstract description of an operation to be performed
- May contain the following information:
  - Component name
  - Action
  - Data
  - Category
  - Extras
  - Flags
- Can be used to:
  - Launch an activity
  - Communicate with a service
  - Send a broadcast to all interested broadcast receivers

Intents

```
startActivity(new Intent(
    android.content.Intent.ACTION_DIAL,
    null
));


startActivity(new Intent(
    android.content.Intent.ACTION_SET_WALLPAPER,
    null
));


startActivity(new Intent(
    android.content.Intent.ACTION_VIEW,
    Uri.parse("http://google.com")
));
```



Intents

- System defines receiving component depending on the contents of Intent object, so intents are divided into 2 categories:
  - Explicit – define target in component name field.
  - Implicit – the field for the component name is blank. Contents of Intent object is compared with intent filters of system components. Intent filter is usually created in AndroidManifest.xml as a part of appropriate component, but can be instantiated dynamically inside component's callback methods.

```
<!-- Intent filter example -->
<!- component is activity, service or broadcast receiver -->
<activity>
  <intent-filter>
    <action android:name="com.sample.project.SHOW_CURRENT" />
    <data android:mimeType="image/*" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

Intents (contd.)

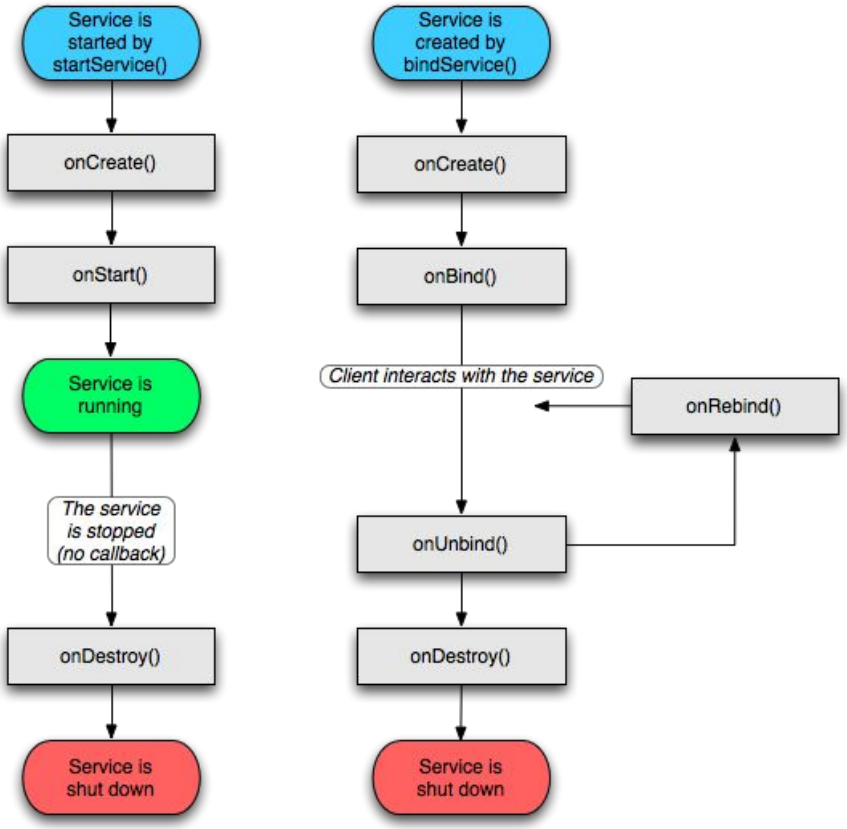
- Explicit intents

```
/* Explicit intents example */

Intent intent = new Intent(getContext(), SecondaryActivity.class);

    or

Intent intent = new Intent();
Intent.setClass(getContext(), SecondaryActivity.class);
```

- Implicit intents

```
/* Implicit intents example */

Intent intent = new Intent("com.sample.project.SHOW_CURRENT");
```

Services



Services

- Application component without UI
- Extends *android.app.Service*
- By default runs in the same process as the caller, and doesn't create its own thread
- Can take two forms: started and bound
- Communication scheme can vary: receive intents from the clients, use messenger to exchange messages, or utilize remote procedure calls scheme

Services

- Have distinctive life-cycle states. Can start multiple times, but stop just once
- Can connect with UI, trigger events, Notification Manager, etc.
- Component can connect to Services in it's own process or another process
  - *bindService(String className, ServiceConnection conn)*
- System will try to keep processes with running services alive for as long as possible

Broadcast Receivers

- Component designed to respond to broadcast Intents
- Implementing an Broadcast Receiver involves:
  - Extending *android.content.BroadcastReceiver*
  - Declaring the component within application manifest file or via registration API
  - Requesting proper permission in manifest file
- When *onReceive()* returns, receiver is inactive and may deleted from memory
- So anything that requires asynchronous operation is not available in *onReceive()*

Content Providers and Content Resolvers

- Application components which are intended to support data sharing model
- Content Resolver provides access to all Content Providers
  - resolves requests from clients by directing them to content provider with a distinct authority
  - includes the CRUD (create, read, update, delete) methods corresponding to the abstract methods (insert, delete, query, update) in the Content Provider class
- Content Providers give an abstraction from the underlying data source (i.e. a SQLite database)
  - offers a standard interface that connects data in one process with code running in another process
  - interface for publishing and consuming data, based around a simple URI addressing model using the content:// schema
- Works across processes

Content Providers

- All content is represented by URIs
  - Convenience methods mean clients don't need to know syntax
  - ContentProviders own URIs based on authority
  - ContentProviders are responsible for mapping URIs they own to a MIME type
  - content://contacts/people is the URI that would return a list of all contact names on the device

Applications

- *android.app.Application* maintains global application state
- Can be provided non-system implementation in *AndroidManifest.xml* for <*application*> tag

```
<application
    android:icon = "@drawable/icon"
    android:label = "@string/app_name"
    android:name = ".TestApplicationImpl"
>
...
</application>
```

Security

- Permissions
- Signing Applications

Basic Security Regulations

- Each Android application runs in its own process
- Security is enforced at the process level through standard Linux facilities, such as user and group IDs
- Additional security features are provided through a "permission" mechanism
- Application, by default, doesn’t have permissions to perform operations that would impact other applications, the operating system, or the user

Permissions

- Basic application has no permissions thus it can't do anything that would adversely impact the user experience or any data on the device
- To use protected device features application should declare permissions in the manifest file
- Requested permissions are granted at the install time

Permissions

- Permission may be enforced at a number of places during program's operation:
  - At the time of a call into the system, to prevent an application from executing certain functions
  - When starting an activity, to prevent applications from launching activities of other applications
  - Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you
  - When accessing and operating on a content provider
  - Binding or starting a service

Signing Applications

- All Android applications (.apk files) must be signed with a certificate
- The certificate identifies the author of the application
- The certificate does not need to be signed by a certificate authority
- To sign application *keytool* and *jarsigner* are used

Main concepts and components

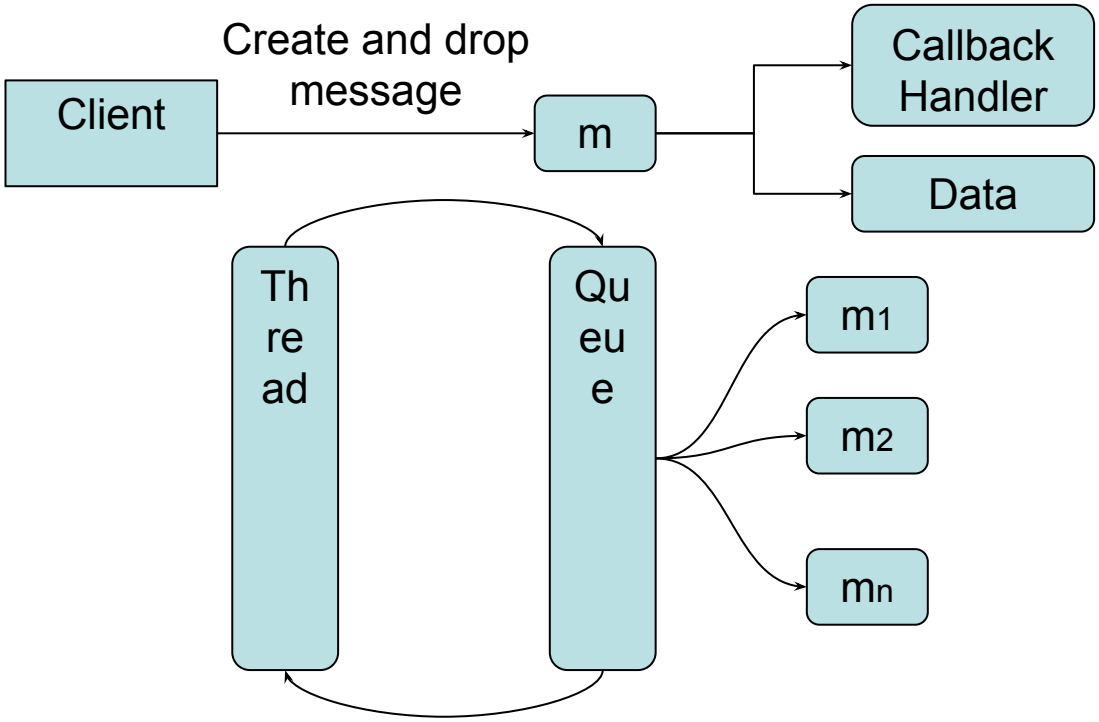
Thread interaction mechanisms

Platform helper classes

- Handler, Looper, Message Queue
- Messenger
- Parcelable classes, Bundle
- AsyncTask

Handlers mechanism

- Handler is an instrument to drop a message on the queue attached to the thread on which handler object was instantiated so that the message can be processed at a later point in time.
- The dropped message has an internal reference to the handler that dropped it.



Messenger

- When you need to perform IPC between service and client, using a Messenger for your interface is simpler than implementing it with AIDL
- Create a Messenger based on the IBinder returned by the service and send a message using send()
- Using a Messenger allows the service to handle only one call at a time
- For multi-threading services, use AIDL

Parcelable

- Is similar to Serializable, but faster
- Allows data to be transferred between different processes/threads
- Your classes will be flattened inside a message container called a Parcel to facilitate high performance inter process communication
- describeContents() and writeToParcel() should be implemented

AsyncTask

- Separates processing of long-term operations of the results representation on thread level
- Runs only once on UI thread by execute() method, can be cancelled using cancel() method
- Generic structure:

```
/**
 *Params – the type of  parameters sent to the task
 *Progress – the type of the progress units published
 *Result – the type of the result of the background computation
 */
class TestAsyncTask extends AsyncTask  < Params, Progress,
Result>{
    /**
     * Don't call these methods manually!
     */
    protected void onPreExecute(){…}
    protected Result doInBackground ( Params arg){…}
    protected void onProgressUpdate( Progress){…}
    protected void onPostExecute( Result result){…}

}
```

AsyncTask

- Member fields of AsyncTask subclass can be set inside its constructor, onPreExecute() and doInBackground() methods.
- AsyncTask state is accessible via AsyncTask.Status class. Possible values:
  - FINISHED – onPostExecute(Result) has finished
  - PENDING – the task hasn't been executed yet
  - RUNNING – the task is running

Practice

(Complete MediaPlayer exercises)

Main concepts and components

Data Storages

Data Storages

- Preferences
  - A lightweight mechanism to store and retrieve key/value pairs of primitive data types
- Files
  - You can store your files on the device or on a removable storage
- Databases
  - The Android API provides SQLite support
- Content Providers
  - Store and retrieve data and make it accessible to all applications. This is the way to share data across applications
- Assets
- Copy and Paste framework

Data Storages: Preferences

- *Context.getSharedPreferences()* to share them with other components in the same application
- *Activity.getPreferences()* to have them private to the calling activity
- You can not share preferences across applications

Data Storages: Preferences Using

```
SharedPreferences settings = context.getSharedPreferences("prefs", 0);
boolean vMode = settings.getBoolean("viewMode", false);

--------------------------------------------------------------------------

SharedPreferences settings = activity.getPreferences(0);
boolean vMode = settings.getBoolean("viewMode", false);

--------------------------------------------------------------------------

SharedPreferences settings = getSharedPreferences("prefs", 0);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("viewMode", false);
editor.remove("viewMode");
// Don't forget to commit your edits!
editor.commit();
```

Data Storages: Files

- Android provides an access to the file system and files
  - Standard Java API: *java.io.FileInputStream* and *java.io.FileOutputStream*
  - Android specific API: *Context.openFileInput()* and *Context.openFileOutput()*
  - Android Resources: *Resources.openRawResource()*
  - Android Assets: *AssetManager.open()*
- You can't share files across applications
  - Except */sdcard* directory

Data Storages: Files Using

```
try {
    final String TESTSTRING = new String("Hello Android");
    FileOutputStream fOut = openFileOutput("samplefile.txt",
    MODE_WORLD_WRITABLE);
    OutputStreamWriter osw = new OutputStreamWriter(fOut);

    // Write the string to the file
    osw.write(TESTSTRING);
    osw.close();
} catch (IOException ioe) {
    Log.e("tag", Log. getStackTraceString(ioe));
}
```

File will be created on /data/data/<package_name>/files/samplefile.txt

Data Storages: Files Using

```
try {
    final String TESTSTRING = new String("Hello Android");
    char[] inputBuffer = new char[TESTSTRING.length()];
    FileInputStream fIn = openFileInput("samplefile.txt");
    InputStreamReader isr = new InputStreamReader(fIn);

    isr.read(inputBuffer);
    String readString = new String(inputBuffer);
    boolean isTheSame = TESTSTRING.equals(readString);
    Log.i("File Reading stuff", "success = " + isTheSame);
} catch (IOException ioe) {
    Log.e("tag", Log. getStackTraceString(ioe));
}
```

Data Storages: Data Bases

- Android provides an access to SQLite-based DBs
- Platform specific API *android.database.sqlite.** is used instead of standard JDBC mechanism
- All application DBs are accessible to any application class, but not outside application
- To direct access to DBs *sqlite3* tool is provided
- DBs are stored as files on the file system

Data Storages: Data Bases Using

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE …");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE …");
        onCreate(db);
    }
}

SQLiteDatabase db = new DatabaseHelper(getContext()).getReadableDatabase();
SQLiteDatabase db = new DatabaseHelper(getContext()).getWritableDatabase();
```

Data Storages: Content Providers

- This is the only way to share data across applications
- Android already has providers *android.provider.** for common data types
  - *ContactsContract* – contact information
  - *Browser* – browser bookmarks and searches
  - *MediaStore* – all media content
  - *Settings* – global system-level settings

Data Storages: Content Providers

- ContentResolver provides access to ContentProvider by URI
- URI is defined by RFC 2396
- Typed Content URI: content://<authority>/<path>/<ID>
  - content – schema
  - authority – identifier of content provider
  - path – what data is requested
  - ID – what record is requested

Example: *content://contacts/people/23*

content://contacts/people?id=23

Content Providers vs. SQL requests

- Storing your data in a database is one good way to persist your data, but databases created are **visible** only to the application that created them
- If you need to share data between applications, you need to use the content provider model as recommended in Android

Data Storages: Content Providers Using

```
// data query

String[] projection = new String[] {
    "_id",
    "name",
    "number",
};
Uri mContacts = Uri.parse("content://contacts/people");
Cursor managedCursor = this
    .managedQuery(mContacts, projection, "_id=?", {"23"}, "name
   ASC");
Cursor queryCursor = getContentResolver()
    .query(mContacts, projection, "_id=?", {"23"}, "name ASC");
```

Data Storages: Content Providers Using

```
// data update

ContentValues values = new ContentValues();
values.put("name", "David");
Uri mContacts = Uri.parse("content://contacts/people/");
int rows = getContentResolver().update(mContacts, values, "_id=?",
    {"23"});
```

Data Storages: Content Providers Using

```
// data insert

ContentValues values = new ContentValues();
values.put("name", "David");
Uri mContacts = Uri.parse("content://contacts/people");
Uri uri = getContentResolver().insert(mContacts, values);
```

Data Storages: Content Providers Using

```
// data delete

Uri mContacts = Uri.parse("content://contacts/people");
int rows = getContentResolver().delete(mContacts, "_id=?", {"23"});
```

Data Storages: Assets

- Assets behave like a file system, they can be listed, iterated over. Assets go into the ./assets directory in the project root and can contain any files.
- System doesn’t generate resID for assets contents.

Data Storages: Copy and Paste

- Provides functionality for copy and paste operations within and between Android applications. Supports text strings, URIs and Intents.
- Clipboard is a container storing only one Clip Data object at any time. Clip Data object holds ClipDescription (metadata related to the copied object) and any number of ClipData.Item objects.
- Interaction with the clipboard is organized through Clipboard manager object.

```
ClipboardManager clipboard =
(ClipboardManager)getSystemService(CLIPBOARD_SERVICE);

// Creating new clip data item

ClipData firstClip = ClipData.newURI(getContentResolver, "URI",
userURI);

Clipboard.setPrimaryClip(firstClip);

// get data

ClipData.Item item = clipboard.getPrimaryClip.getItemAt(0);
```

Practice

(Complete the ContentProvider_UI_Adapter presentation)

Main concepts and components

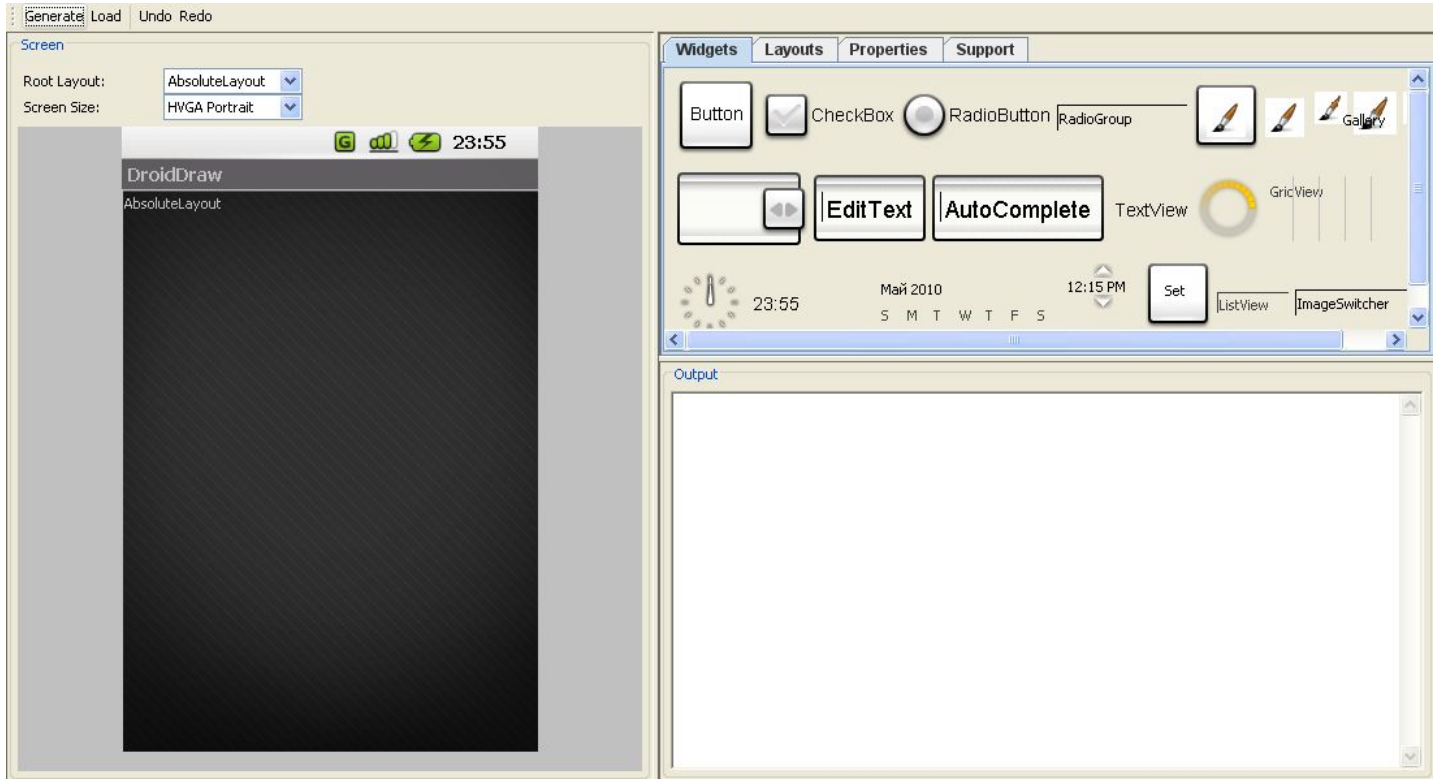
User Interface

User Interface

- Visual Editors
- UI Components (typical UI widgets attributes. Merging and including UI xml)
- App Widgets
- Live Wallpapers
- Drawing (screen sizes support)
- Examples (Custom View, SurfaceView, GLSurfaceView)

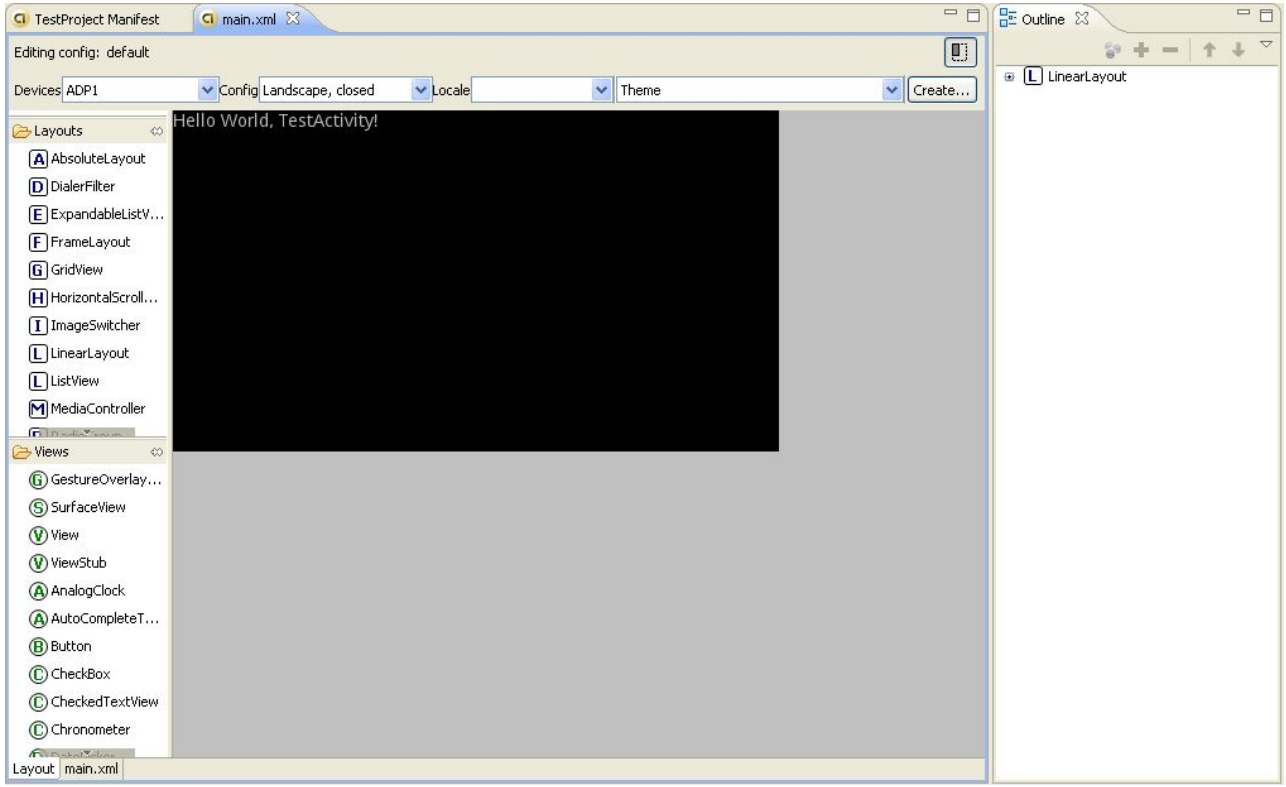
Visual Editors

- DroidDraw is the interactive UI designer



Visual Editors

- Eclipse ADT embedded UI designer



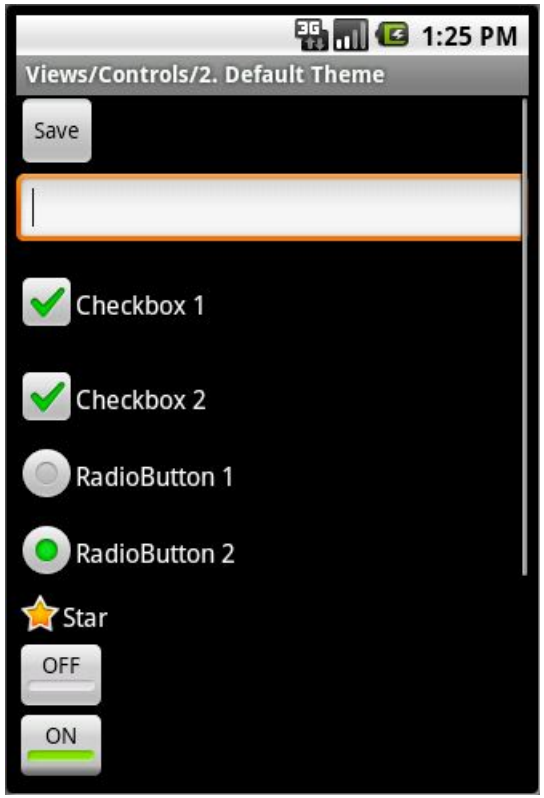
UI Components

- View and Widgets
- Fragments
- ViewGroup and Layouts
- AdapterView
- Floating dialogs
- Menus
- Notifications
- Custom components

UI Components: View And Widgets

- *android.view.View* objects are the basic units of UI expression on the Android platform
- The View class serves as the base for subclasses called “widgets”, which offer fully implemented UI objects: buttons, text fields, etc
- *View* occupies a rectangular area on the screen and is responsible for drawing, event handling, focus change, scrolling and key/gesture interactions

UI Components: Widgets

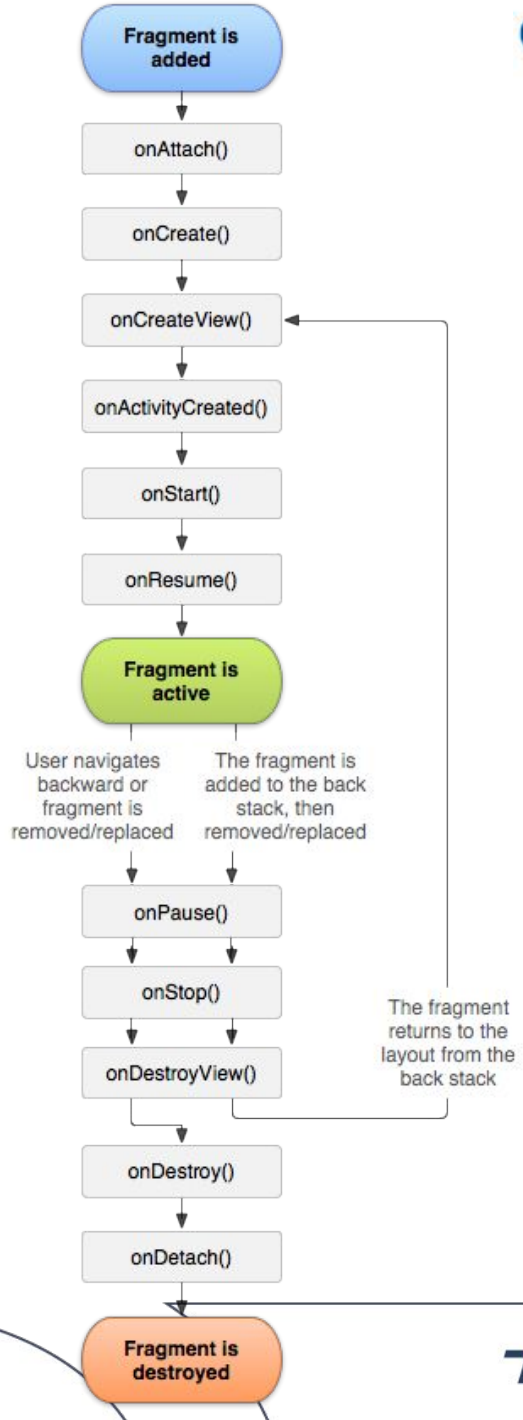


Fragments

- What's the matter?
  1. Complex Activity code for heavy-weight UI
  2. Handling identical piece of UI code in different Activities
  3. Mix of business logic and UI
- Old-school solution
  1. Extend View to implement complex UI
  2. Implement business logic as part of View code.
- Fragments
  1. Has it's own life-cycle
  2. Constructed as a reusable component
  3. Supported starting 3.0+(API11).
  4. Back-ported to 1.6+(API4) as Google Support Library

Fragments cont.

- onAttach() is called once Fragment is attached to its Activity
- onCreateView() defines what View will be inflated



Fragments cont.

- XML definition

```
<fragment
    android:id="@+id/foo_fragment"
    android:name="com.foo.FooFragment"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

- Java definition

```
FooFragment foo =
    Fragment.instantiate(context, "com.foo.FooFragment");
```

Fragments cont.

- Inserted Fragments.
  1. In general it works.
  2. This feature is NOT supported using XML and won't work correctly.

  To use inserted Fragments you should define a placeholder (FrameLayout) in parent XML

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

  Replace it in the Java code by an instance of your fragment.

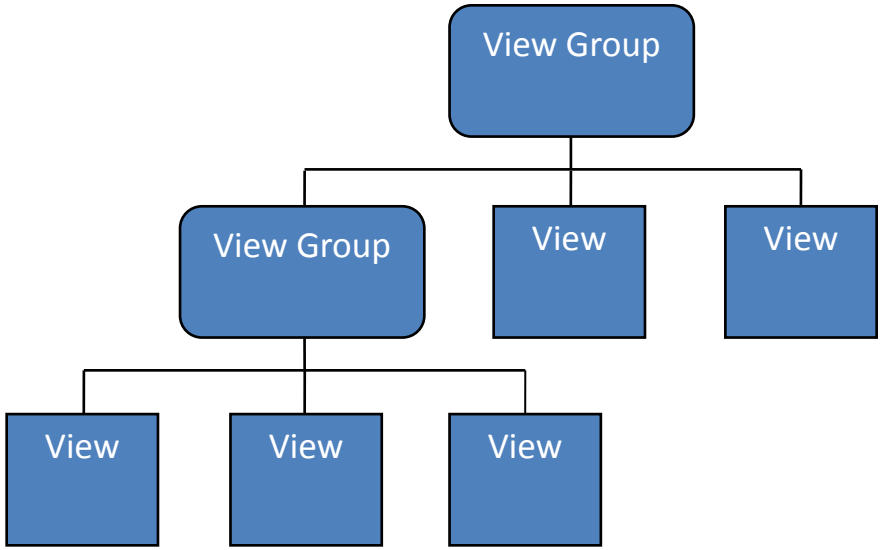
```
FragmentTransaction t =
  getSupportFragmentManager().beginTransaction();
  t.replace(R.id.inserted_fragment, new FooFragment());
  t.commit();
```

Practice

(Complete the Fragmets_adb_sqlite3 presentation)

UI Components: ViewGroup

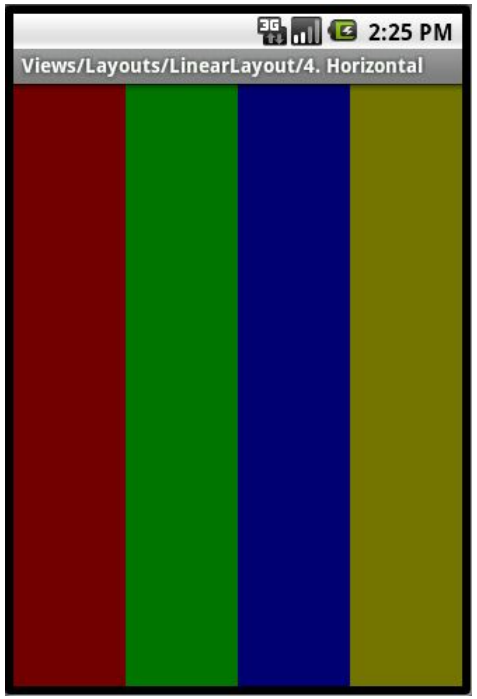
- Special view that can contain other views (children)
- The view group is the base class for layouts and views containers
- UI architecture is a hierarchy of View and ViewGroup nodes

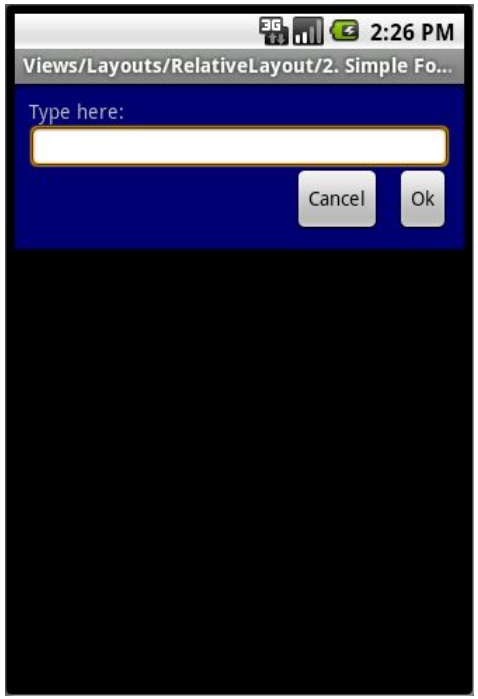


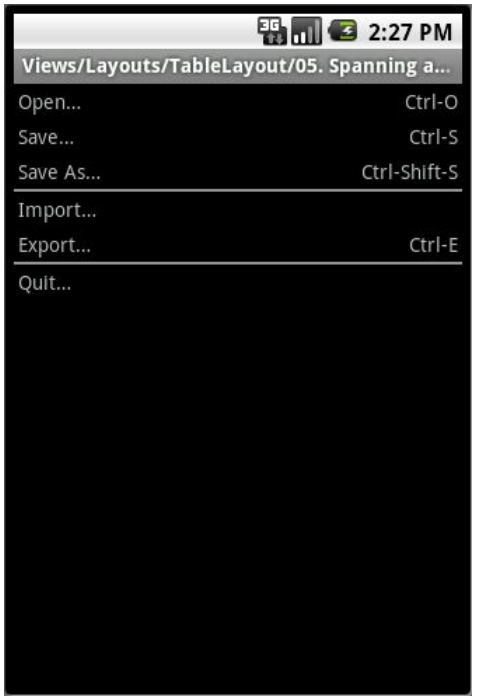
UI Components: Layout

- Layout is based on ViewGroup
- Layout defines a position and parameters of children inside
- Standard layouts:
  - LinearLayout – children are disposed in line (single column or row)
  - AbsoluteLayout – position of children is specified in absolute units
  - RelativeLayout – position of children is specified relative to layout borders and others
  - FrameLayout – children are drawn in a stack, with the most recently added child on top
  - TableLayout – arranges children into rows and columns
  - AdapterView<T extends Adapter> – children are determined by an Adapter

UI Components: Layout



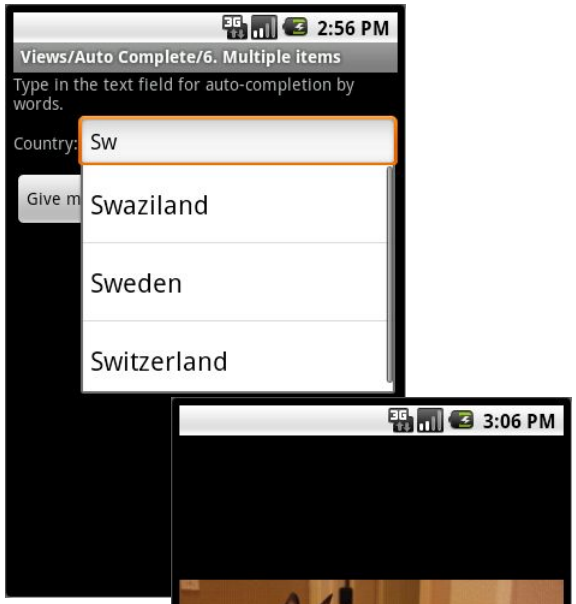


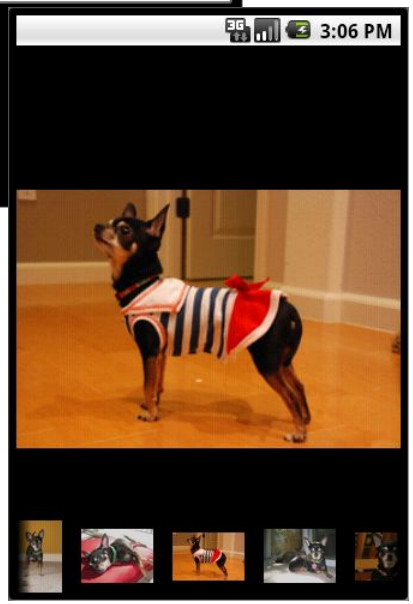


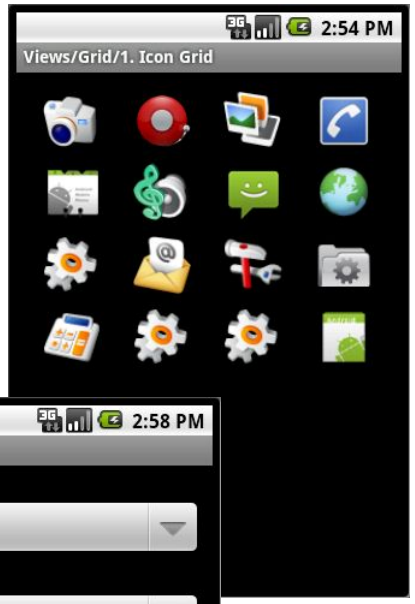
UI Components: AdapterView

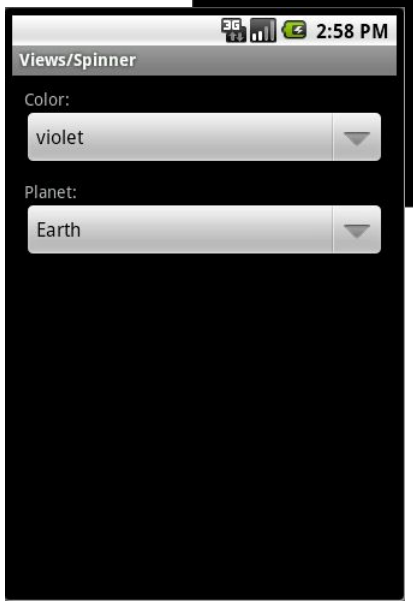
- AdapterView is a view whose children are determined by an Adapter class
- Adapter object is data source for AdapterView
- AdapterView gets data from Adapter by the next methods:
  - getCount()
  - getItem(int position)
- Adapter is responsible for providing a View to display data at a particular position
  - getView(int position, View convertView, ViewGroup parent)
- Adapter provides notifications if data is changed

UI Components: AdapterView





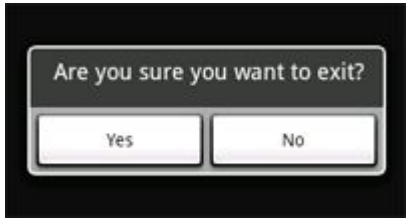


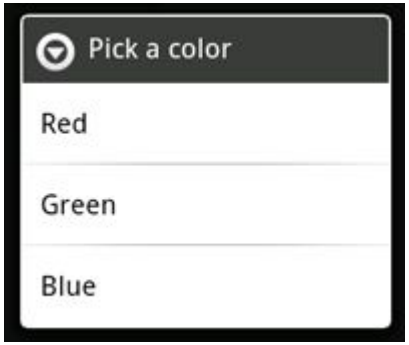


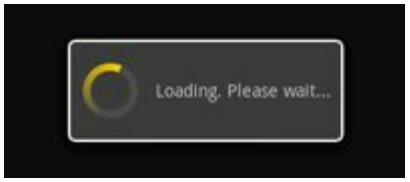
UI Components: Floating Dialogs

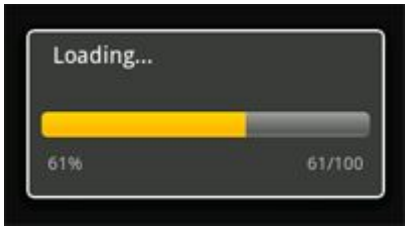
- Small window that appears in front of the current Activity
- Activity loses focus and the dialog accepts all user interaction
- Normally used for notifications and short activities
- Android supports the following types of Dialog objects
  - AlertDialog – manage few buttons, and/or a list of selectable items that can include checkboxes or radio buttons
  - ProgressDialog – displays a progress wheel or progress bar
  - DatePickerDialog – allows to select a date
  - TimePickerDialog – allow to select a time

UI Components: Floating Dialogs







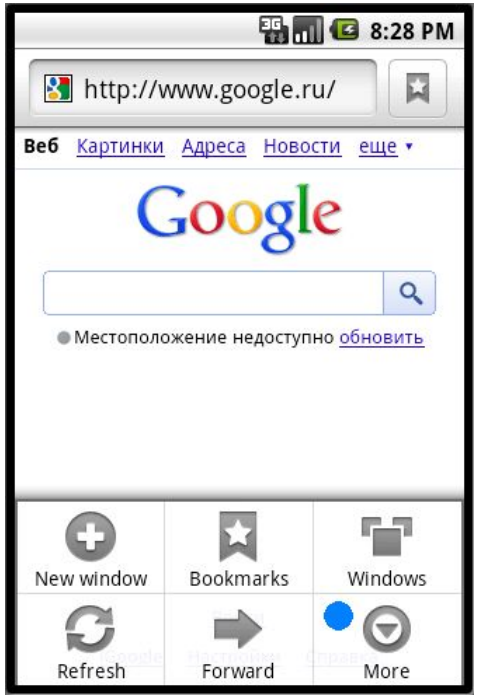


UI Components: Menu

- Part of any application to reveal its functions and settings
- Android offers an easy programming interface to provide standardized application menus
- There are three fundamental types of application menus:
  - Options Menu – primary set of menu items
  - Context Menu – floating list of menu items that appears by long-click on a View
  - Submenu – floating list of menu items that is revealed by an item in the Options Menu or a Context Menu

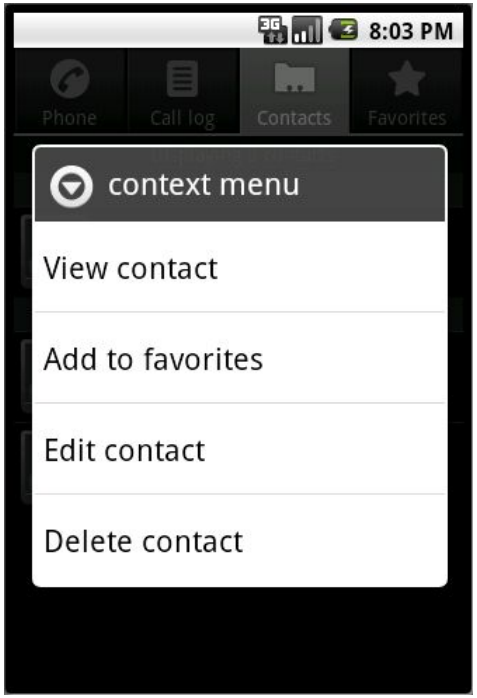
UI Components: Options Menu

- Is opened by MENU button
- Displays no more than the first six menu items
- “More” menu item is added if more than six items are added
- *onCreateOptionsMenu()* callback is called from Activity for the first time when menu is opened
- Options menu items support icons for the first six items and shortcut keys for others



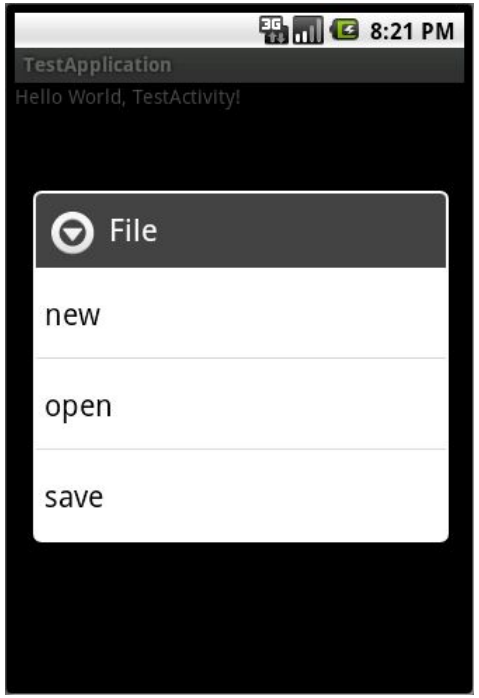
UI Components: Context Menu

- Conceptually, it is similar to “right-click” PC menu
- Is displayed by “long-click”
- Can be registered to any View object
- *onCreateContextMenu()* and *onContextItemSelected()* callbacks are called from Activity
- Context menu items do not support icons or shortcut keys



UI Components: Submenu

- Submenus are added for organizing topics and including extra menu functionality
- A submenu can be added within any menu, except another submenu
- *addSubMenu()* adds a submenu to an existing *Menu*
- Callbacks for items selected in a submenu are made to the parent menu callback method

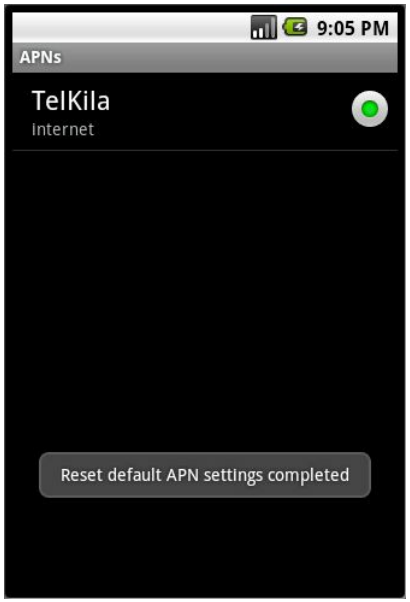


UI Components: Notifications

- Notification tasks can be achieved using a different technique:
  - Toast Notification – brief messages that come from the background
  - Status Bar Notification – persistent reminders that come from the background and request the user response
  - Dialog Notification – Activity-related notifications

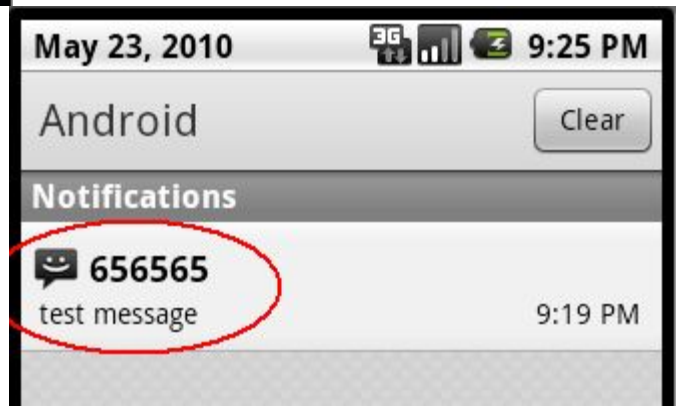
UI Components: Toast Notifications

- A message that pops up on the window surface
- Fills only the amount of space required for the message
- Current activity remains visible and interactive
- The notification automatically fades in and out, and does not accept interaction events
- Can be created and displayed from an Activity or Service



UI Components: Status Bar Notifications

- Adds an icon to the system status bar and an expanded message in the “Notifications” window



UI Components: Status Bar Notifications

- When the expanded message is selected, Android fires an *Intent* that is defined by the notification (usually an *Activity* is launched)
- The notification can be configured to alert the user with a sound, a vibration, and flashing lights on the device
- A background *Service* to interact with user should create a status bar notification that will launch the *Activity*
- To create a notification, two classes: *Notification* and *NotificationManager* are used

UI Components: Custom Components

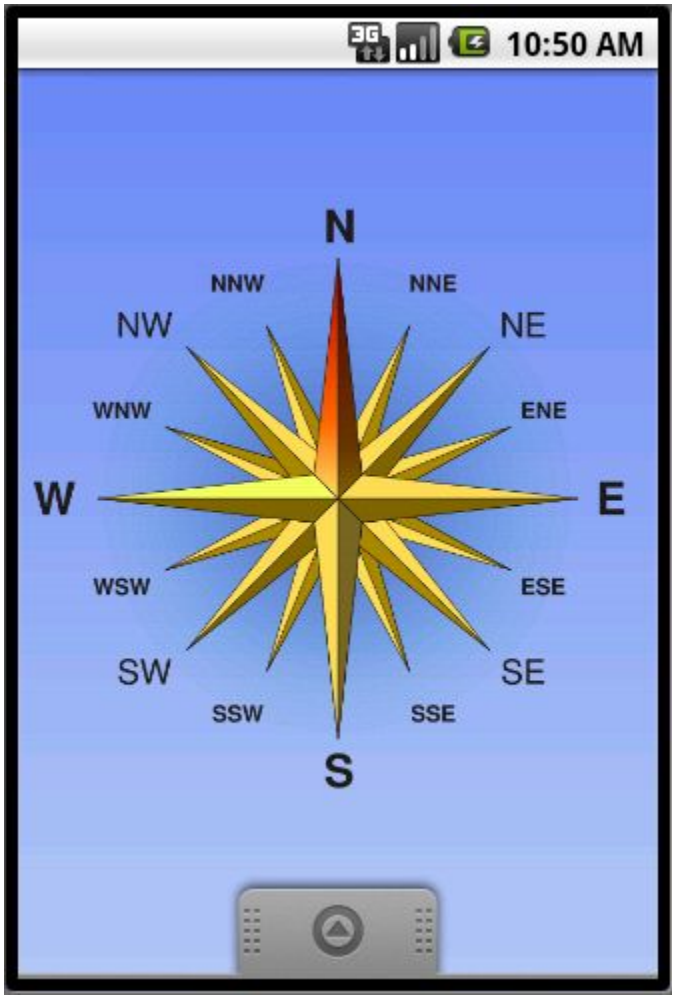
- Custom widget based on *View* class
  - Extend existing widget such as *TextView*, *ProgressBar, ImageView etc* and override its methods
  - Extend the View class and implement void *onDraw(Canvas canvas)* method and other callback methods if necessary
- *SurfaceView*-based implementation
- Open GL-based (*GLSurfaceView*) implementation

App Widgets

- Miniature application views that can be embedded in other applications and receive periodic updates
- It can be published using an App Widget provider
- An application component that is able to hold other App Widgets is called an App Widget host (Home Screen)

Live Wallpapers

- Interactive backgrounds on the home screens
- Similar to a normal Android application and has access to all the facilities of the platform
- A live wallpaper is very similar to a regular Android service, except *onCreateEngine()* method
- The engine is responsible for handling the lifecycle and drawing of a wallpaper



Drawing

- The *Canvas* class provides the "draw" calls
- To draw something, you need 4 basic components:
  - *Bitmap* to hold the pixels
  - *Canvas* to host the draw calls (writing into the bitmap)
  - Drawing primitives – Rect, Path, Bitmap, etc
  - Paint – to describe the colors and styles for the drawing

Domain-specific items

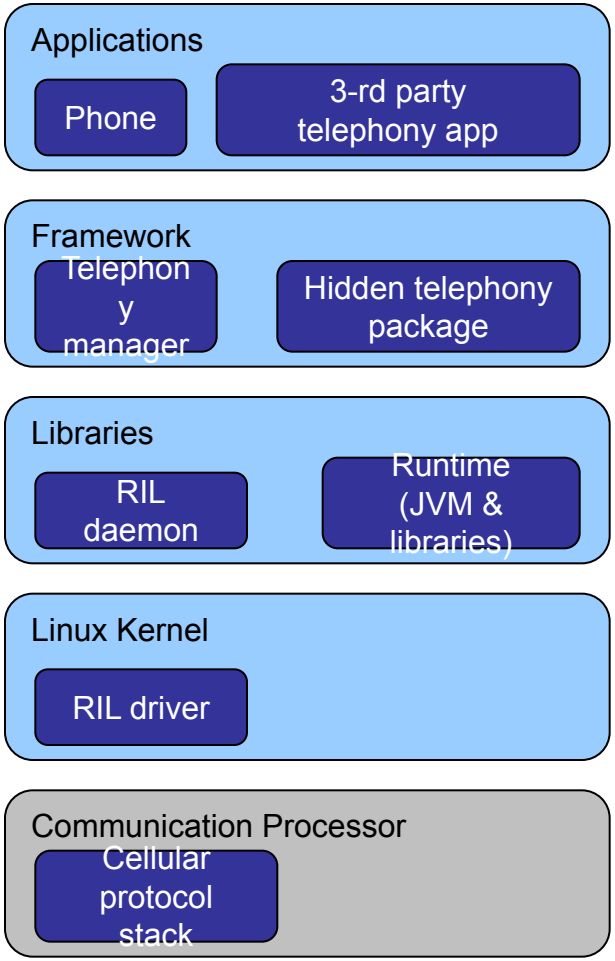
Framework Review

- Network
- Telephony
- Media framework (media player, jet player, camera, NFC)
- Web applications development (V8 Java Script support)
- Working with sensors
- Location
- 3-rd Party Components
- Examples

Network

- java.net
  - Standard Java5 network API
- android.net
  - Network state
  - DHCP information
  - UNIX sockets
  - Android proxy settings
  - URL parsers and builders
- org.apache.http
  - Apache HTTP 4.x client

Telephony Stack



Telephony Stack

- Phone – platform application to make calls
- Telephony Manager – provides a telephony API for user applications
  - android.telephony
  - android.telephony.gsm
  - android.telephony.cdma
- RIL daemon
  - Internal framework module communicates via UNIX domain sockets with the RIL
  - Daemon communicates using AT commands with either the RIL kernel driver
- RIL driver – pipe that forwards AT commands to the baseband processor via the appropriate hardware interface

Media framework

- Media player
- Jet player
- Camera
- NFC (Near Field Communication)

Web applications development

- You can make your web content available to users in two ways: in a traditional web browser and in an Android application, by including a WebView in the layout.
- Add WebView to the application

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

- Load Web page

```
   WebView myWebView = (WebView) findViewById(R.id.webview);
   myWebView.loadUrl("http://www.example.com");
```

Web applications development (contd.)

- If JavaScript is planned to use, enable it

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```

- Binding JavaScript code to Android code (call addJavascriptInterface())

```
 public class WebAppInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

Web applications development (contd.)

- Navigate Web page history using `goForward()` and `goBack()`
- You can debug your JavaScript using the console JavaScript APIs, which output messages to logcat

```
console.log("Hello World");
```

- Best practices:
  - Redirect mobile devices to a dedicated mobile version of your web site
  - Use a valid markup DOCTYPE that's appropriate for mobile devices
  - Use viewport meta data to properly resize your web page
  - Avoid multiple file requests
  - Use a vertical linear layout

Working with sensors

- The Android platform supports three categories of sensors:
  - Motion sensors
  - Environmental sensors
  - Position sensors
- Sensors can be accessed using Android sensor framework
  - Determine which sensors are available
  - Determine capabilities
  - Acquire raw data
  - Register event listeners
- Core classes are: `SensorManager`, `Sensor`, `SensorEvent`, `SensorEventListener`

Location

- Two frameworks can be used:
  - Android framework location APIs (android.location package)
  - Google Location Services API (part of Google Play Services)
- Android framework way:
  - Request an instance of LocationManager from the system
  - Query for the list of all LocationProviders for the last known user location
  - Register/unregister for periodic updates of the user's current location
- Google Play services:
  - Set up the Google Play services SDK
  - Use common well-described approaches
  - Retrieve the current location using LocationClient
  - Subscribe to location updates
  - Create and monitor geofences (geographic areas as locations of interest)
  - Detect user’s current activity and use this information in your app

3-rd Party Components

- Apache HTTP Client – powerful HTTP client connections
- JUnit – Java testing framework
- JSON – JavaScript Object Notation
- XmlPull – XML Pull parsing

ANDROID NDK

NDK setup

- On Win PC be sure Cygwin is properly installed
- Download an archive from http://developer.android.com/tools/sdk/ndk/index.html
- Use Cygwin console to work with NDK
- NDK samples are located at $NDK/samples and can be build by:
  - ./ndk-build -C samples/<PROJECT_NAME>

Android app with native code

- Create an empty Eclipse project (Android app)
- Create wrapper-class CalcWrapper.java
- Load native library by name WITHOUT ‘lib’ prefix and ‘.so’ suffix before the first usage:

```
static {
    System.loadLibrary(“calc-jni");
}
```

- Create prototypes of native functions in this class:

```
public static native int getSum(int a, int b);
```

- Compile your project and go to $YOUR_PROJECT/bin/classes
- Generate C header:

```
javah -jni com.example.jnitestapp.CalcWrapper
```

- Copy generated functions names to a project with native code /jni/calc-jni.c and implement them

Android app with native code

- Compile NDK project (libcalc-jni.so)
- Create new folder $YOUR_PROJECT/libs/armeabi inside
- Put *.so files: libcalc-jni.so, for example to this folder.
- Recompile $YOUR_PROJECT
  - Check that resulted *.apk file has ./lib/armeabi/libcalc-jni.so inside

Pure native apps

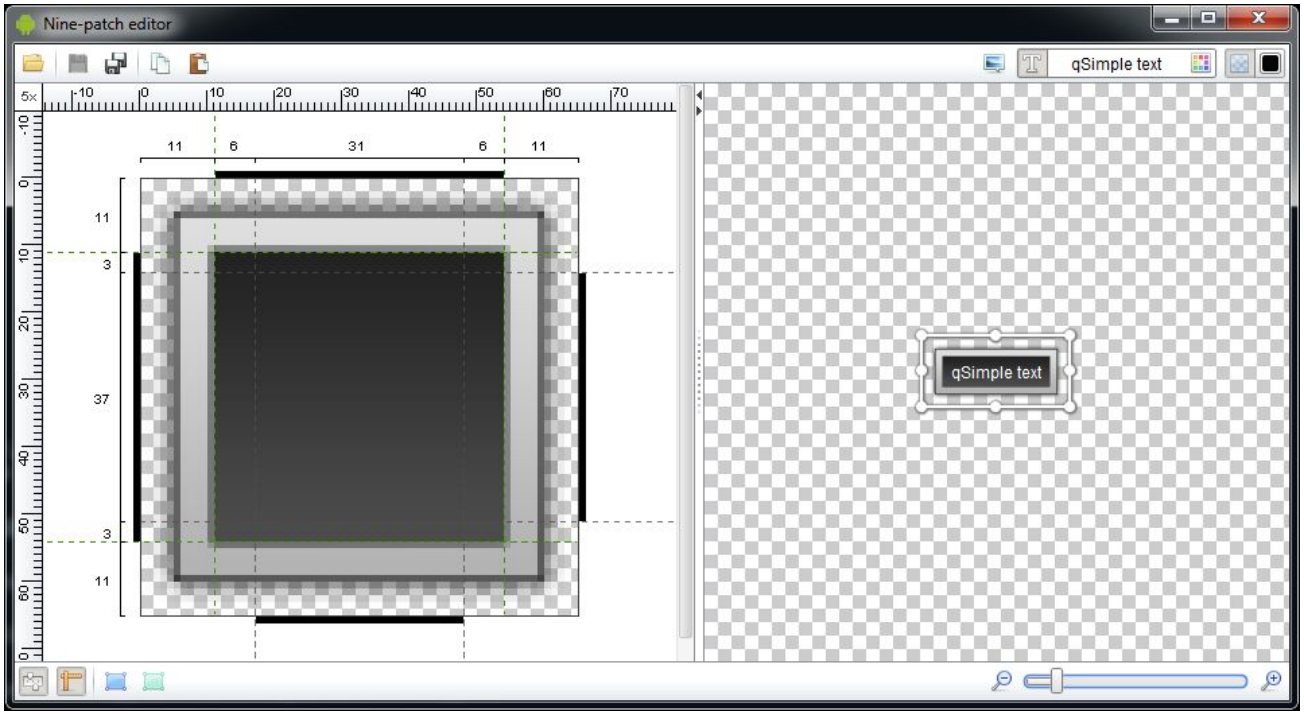
- http://developer.android.com/reference/android/app/NativeActivity.html
- Sample code is in $NDK/samples/native-activity
- AndroidManifest.xml should contain:

```
<activity android:name="android.app.NativeActivity">
 <meta-data android:name="android.app.lib_name"
            android:value="native-activity" />
</activity>
```

- native-activity here is a name of *.so what defined in Android.mk
- Build native code:
  - ./ndk-build -C samples/native-activity
- Create Eclipse Project based off native-activity source code.
- Compile and run.

9 patch editor

- Useful editor based off Android SDK
- http://weblookandfeel.com/downloads/ninepatch-demo.jar



OpenGL ES open source engines

- AndEngine
  - Lightweight and powerful Java engine with loadable extensions
  - Supports OpenGL ES 1.0/2.0
  - Extensions: Box2D, Multiplayer, SVG textures, Live wallpapers.
  - Provides a lot of usage examples
  - Used for games: Bunny Shooter, Greedy Spiders, Face Costume, etc
  - https://github.com/nicolasgramlich/AndEngine

System Services

- Getting of the system services is made by name using `Context.getSystemService(String name)`
- There are many services accessed from user apps:

❑WindowManager
❑LayoutInflater
❑ActivityManager
❑PowerManager
❑AlarmManager
❑NotificationManager
❑KeyguardManager
❑LocationManager
❑SearchManager
❑Vibrator
❑ConnectivityManager
❑WifiManager
❑InputMethodManager
❑DownloadManager

System Services. LocationManager.

It's used for accessing to the system location services

- Add permissions ACCESS_COARSE_LOCATION and/or ACCESS_FINE_LOCATION to AndroidManifest.xml
- Implement LocationListener interface

```
private class InternalLocationListener implements
  LocationListener
{
@Override
public void onLocationChanged(Location location) {
    // get location here and do a job
}
```

System services. LocationManager.

- Get system service and location updates listener

```
LocationManager lm = getSystemService(Context.LOCATION_SERVICE);
InternalLocationListener mListener = new InternalLocationListener();
```

- Sign in for location updates

```
lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 15000,
   0F, mListener, Looper.getMainLooper());
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 15000, 0F,
   mListener, Looper.getMainLooper());
```

Features

- Fragments
- Loaders
- Calendar API
- Rich UI Components
- Google Cloud Messaging
- Support Library

Loaders

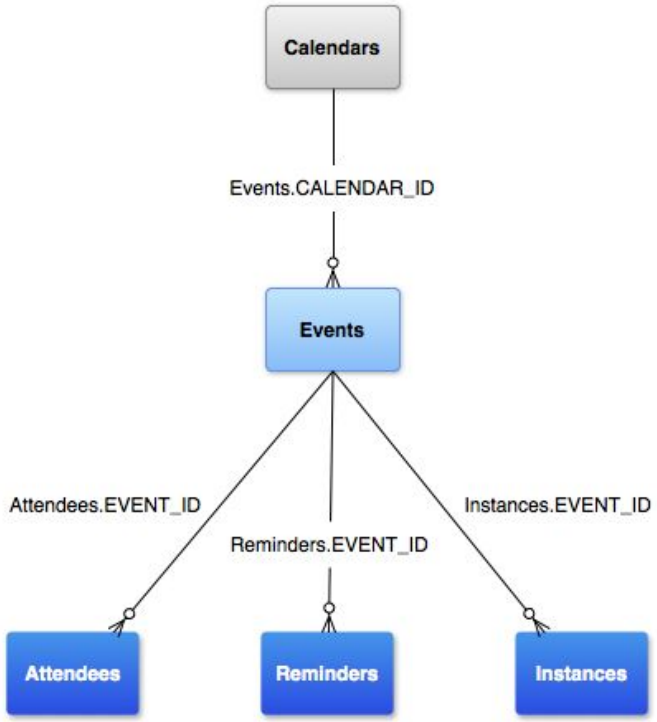
- Async data loading for Activities/Fragments.
- Introduced since 3.0+ (API11) and Support Library 4+
- Monitor data and deliver new data once it available.
- Can be reconnected to the previously created record set without data re-query.

Loaders usage

- Get LoaderManager
  `Activity.getLoaderManager()`
- Init Loader
  `LoaderManager.initLoader(int id, Bundle args, LoaderCallbacks<D> callback)`
- Implement LoaderCallbacks
  - `onCreateLoader(int id, Bundle args)` to create Loader instance
  - `onLoadFinished(Loader<Cursor> loader, Cursor data)` to initialize UI by loaded data
  - `onLoaderReset(Loader<Cursor> loader)` to release UI bacause of data unavailability.
- Dev guide:
  http://developer.android.com/guide/components/loaders.html

Calendar API

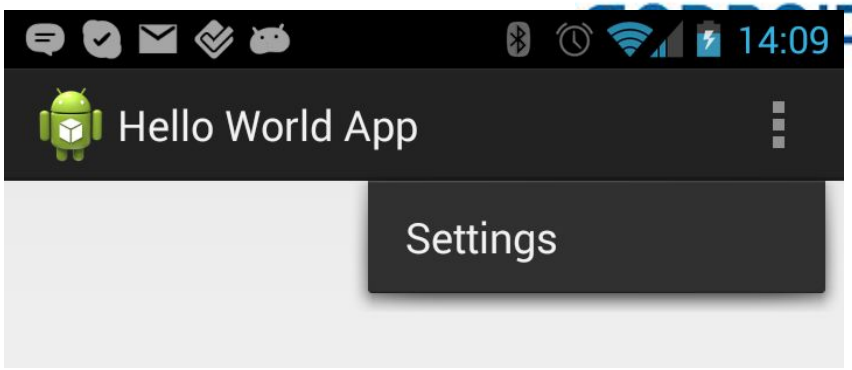
- Allows you to perform query, insert, update, and delete operations on calendars, events, attendees, reminders, and so on
- Calendar Provider data model
- A user can have multiple calendars



Rich UI::Action Bar

- Action Bar was introduced since 3.0+ (API11)
- Action Bar is a replacement of a classic ‘Options Menu’
- ‘Options Menu’ still can be available in compatibility mode
  - Looks ugly on some devices like tablets.

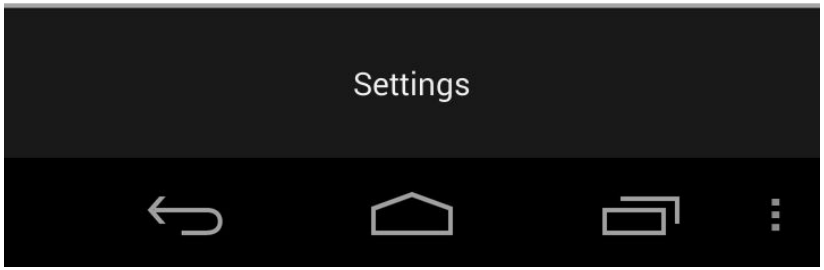
Rich UI::Action Bar



- AndroidManifest.xml

```
<uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

<application android:theme="@style/AppTheme">
...
</application>
```

Rich UI::Action Bar:compatibility mode



- AndroidManifest.xml

- Define SDK version as 10.

```
<uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="10" />
```

- Defile a style with no title bar

```
<application
   android:theme="@android:style/Theme.Light.NoTitleBar.Fullscreen">
...
</application>
```

Google Cloud Messaging (GCM)

- Sends messages from your server to your Android clients.
- Gets client messages back.
- Message size is up to 4K
- Android device should have a proper BroadcastReceiver
- Requires a Google account on 2.2+ devices.
  A Google account isn't required since 4.0.4+ devices.

GCM cont.

- Register your project at Google Console
- Enable GCM
- Obtain an API key

GCM client

- Check Google API availability

```
GooglePlayServicesUtil.isGooglePlayServicesAvailable(Context)
```

- Register your app

```
GoogleCloudMessaging gcm;
. . .
gcm.register("YOUR-SENDER-ID");
```

- To receive messages implement WakefulBroadcastReceiver
- To send messages just call

```
gcm.send("YOUR-SENDER-ID" + "@gcm.googleapis.com",
messageId, dataBundle);
```

GCM 3rd party server

- GCM supports HTTP and CCS connection servers
- Message streaming
  - HTTP supports only cloud-to-device downstreaming
  - CSS supports upstreaming and downstreaming
- Async messaging
  - 3rd party server sends HTTP POST to the cloud and blocks until response
  - CSS sends/receives messages asynchronously using a persistent connection to the cloud
- JSON usage
  - JSON message is sent as HTP POST for 3rd party HTTP server
  - JSON is encapsulated in XMPP messages

Support library

- Practically, some of useful and popular API introduced in 3.0+ and 4.0+ platforms are good to use on all platform versions.
- Support Library as part of Android SDK is targeted to do this.
- There are several versions: 4, 7, 13 and 18. Each new version is backward compatible with the previous ones.
  1. You don’t need to include all of them to your project.
  2. Be sure you have the same library JAR in all included projects.

3-rd party SDKs

Keep in mind a license type!!!

- Volley Framework
- UI SDKs
  - Action Bar Sherlock
  - Sliding Menu

Volley Framework

- Created to solve two main every day goals
  1. Net requests/responses execution and caching. Basically used for JSON and XML formats.
  2. Image loading and caching
- Manages a pool of threads for net requests. Priorities can be changed.
- Checks cache hits/misses
- Optimizes a net traffic to speed up an app.
- Unified API for old (Apache HTTP Client) and new Android platforms (URLConnection)
- Video presentation: http://www.youtube.com/watch?v=yhv8l9F44qo

UI SDK::Action Bar Sherlock

- Developed to provide a modern and rich Action Bar functionality for platforms since 2.2+(API10).
- Classic Action Bar functions + themes and well customization.
  - Google has back-ported common Action Bar functionality in Support Library 7.
- Web site: http://actionbarsherlock.com/

Known Android issues

- Apps starting
- Persistent Notifications

Apps starting

- Security policy was significantly changed since 3.1+ (API12)
- Apps can’t be started automatically by system Intents like ACTION_BOOT_COMPLETED.
- To get this behavior user has to launch an application manually the first time.
- User has to repeat this procedure if app was stopped manually (‘Force stop’)
- For intents defined by app itself FLAG_INCLUDE_STOPPED_PACKAGES should be set to avoid this problem.

Persistent Notifications

- Since 4.3+ (API18) a persistent notification is shown in Notification bar for all services called startForeground()
- Users are really annoying about that.
- Dianne Hackborn explanation: https://plus.google.com/105051985738280261832/posts/MTinJWdNL8t

Practice

(Complete the Android 4.x LocationApp presentation)

Q&A

Sources

- http://www.openhandsetalliance.com
- http://developer.android.com
- http://source.android.com
- Architecture
  - http://developer.android.com/guide/index.html
  - http://kernel.org
  - http://sites.google.com/site/io/dalvik-vm-internals
- Applications
- Application Components
  - http://developer.android.com/guide/topics/fundamentals.html

Sources

- User Interface
  - http://droiddraw.org
- Data Storages
  - http://developer.android.com/guide/topics/providers/content-providers.html
- Framework Review
  - http://hc.apache.org/httpcomponents-client
  - http://junit.org
  - http://www.json.org
  - http://www.xmlpull.org

Sources

- Security
  - http://java.sun.com/javase/6/docs/technotes/tools
- Android Tools Review
  - http://www.sqlite.org
  - http://tools.android.com