XML

-used in Android because it reduces number of lines of code

DVM

-Dalvik Virtual Machine (just like JVM but for Android)

-There is no .class, instead .dex file is created (dalvik executable)

Why not use .class?

-Class files are not economic or efficient

-made for large devices

-we are creating applications for mobile devices and must optimize code

-dvm makes code more compressed

OS

-Linux (kept Kernal) and removed rest

ash

-android shell

-command terminal

-based on linux

SQLite

-database

-smaller version of MySQL

Eclipse IDE vs cmd line (which uses ant)

Ant – Another Neat Tool

What must be in system path?

-tools

-platforms

Annotations:

-provide information for the compiler (to detect errors or suppress warnings)

-compiler-time and deployment-time processing (Software tools can process annotation information to generate code, XML files..etc)

@Override annotation informs the compiler that the element is meant to override an element declared in a superclass

Android Versions:

(old – all tools in tools sub-dir)

v1 – Apple Pie – API level 1

v1.1 – Bannana Breat – API level 2

v1.5 – Cupcake – 3

v1.6 – Donut – 4

v2.0, 2.01, 2.1 – Éclair – 5, 6, 7

(newer versions)

v2.2 – Froyo - 8

v2.3, 2.3.3 - Gingerbread – 9

v3.0, 3., 3.2 – Honeycomb – 11, 12, 13

v4.01, 4.03 – Icecream Sandwich – 14, 15

v4.1 – Jelly Bean – 16

All of applications GUI/resources are written in XML. Why?

-It is simple to use and follow

Do we ever put data into java code?

-In Android we NEVER put data into code

-all data is in XML

What are resources?

-colors, strings, pixils

-these should not be hard-coded into java

-If we change data we must re-compile but we don’t want to do this

-if resources are in different files then we only need to go into those files and obtain those files (values? information?)

-if I sell an application then the user can modify the application by modifying the resource file

AVD-

-android virtual device

Package name

-com.my.cs.job1 // bad package name because each one is a sub-directory

-com.my.first // better

When creating a project:

-ask for minimum sdk level

-this will be the lowest version of android that our application should support (api 5 v2.0)

Sub-directories of the Project:

src

-contains java source code of application

gen

-contains special class R.java (which we must never modify)

-R.java contains all our resource references (size, font, color)

-if we accidently modify this file then our application will not work

-this is an auto-generated file

assets

-contains group of folders and sub-sub-directories into which we place static files

res

-contains information about all our resources

res/drawable

-image files

res/anim

-has xml files for animation (moving)

res/layout

-xml files for gui elements (eg. buttons)

res/menu

-xml files for menu (can create multiple different menus)

res/values

-files that take care of different resources (ie. strings) all represented in xml and can be renamed:

res/values/strings

res/values/arrays

res/values/styles – representing font types

res/values/colors

xml

-if you create extra xml files for your own reason place them into this directory

raw

-can handle any non-xml data (eg. application must contain and display name and age names\_and\_ages)

-can place as many files as I want here

libs

-empty

-we use this often

-if someone created good classes we must put them into .jar files and we put .jar files into libs

bin

-has 2 sub-directories

bin/classes – all classes created in android program (.class files)

bin/classes.dex – all dalvik files are in here (dvm)

Where is R.java?

-gen/my/first/R.java (package name)

Manifest.xml

-auto-generated

-has information about each individual part of your project

-breaks application into parts and explained in this xml file (collection of elements in application)

Context:

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

Context Methods all enable whoever has access to the Context to be able to access application-wide resources. Context, in other words, hooks the component that has a reference to it to the rest of application environment.

Intent

An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

The primary pieces of information in an intent are:

* **action** -- The general action to be performed, such as [**ACTION\_VIEW**](http://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW), [**ACTION\_EDIT**](http://developer.android.com/reference/android/content/Intent.html#ACTION_EDIT), [**ACTION\_MAIN**](http://developer.android.com/reference/android/content/Intent.html#ACTION_MAIN), etc.
* **data** -- The data to operate on, such as a person record in the contacts database, expressed as a [**Uri**](http://developer.android.com/reference/android/net/Uri.html).

There are two primary forms of intents you will use.

**Explicit Intents** have specified a component (via **[setComponent(ComponentName)](http://developer.android.com/reference/android/content/Intent.html" \l "setComponent(android.content.ComponentName))** or **[setClass(Context, Class)](http://developer.android.com/reference/android/content/Intent.html" \l "setClass(android.content.Context, java.lang.Class<?>))**), which provides the exact class to be run. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.

Explicit Intents explicitly defines the component which should be called by the Android system, by using the Java class as identifier.

**Implicit Intents** have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

Implicit Intents do not directly specify the Android components which should be called. They specify the action which should be performed and optionally an URI which should be used for this action

Intents can be used to signal to the Android system that a certain event has occurred. Other components in Android can register to this event and will get notified.

An Intent can also contain data. This data can be used by the receiving component. For example your application can calls via an Intent a browser component. As data it may send the URL to the browser component.

Components:

Action:

A string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The Intent class defines a number of action constants, including these:

Extra

Key-value pairs for additional information that should be delivered to the component handling the intent.

User-generated files include

* **src/** contains the Java packages the developer writes or imports for the application. Each package can have multiple .java files representing different classes.
* **res/layout/** contains the XML files that specify the layout of each screen.
* **res/values/** contains the XML files used as references by other files.
* **res/drawable-hdpi/**, **res/drawable-mdpi/**, and **res/drawable-ldpi/** are directories that contain pictures the application uses. They have high, medium, and low dots-per-inch resolution, respectively.
* **assets/** contains additional nonmedia files the application uses.
* **AndroidManifest.xml** specifies the project to the Android OS.

Autogenerated files include

* **gen/** contains autogenerated code, including the generated class **R.java.**
* **default.properties** contains project settings. Although autogenerated, it should be kept under revision control.

An application's resources include XML files describing the layout, XML files describing values such as strings, labels of UI elements, and additional supporting files such as pictures and sounds. At compile time, references to the resources are gathered into an autogenerated wrapper class called**R.java.** The Android Asset Packaging Tool (aapt) autogenerates this file.

**gen/com/cookbook/simple\_activity/R.java**

the **R.java** class provides a way to reference external resources within Java code

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

The first line is required and standard across all XML files in Android to specify the encoding.

toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);