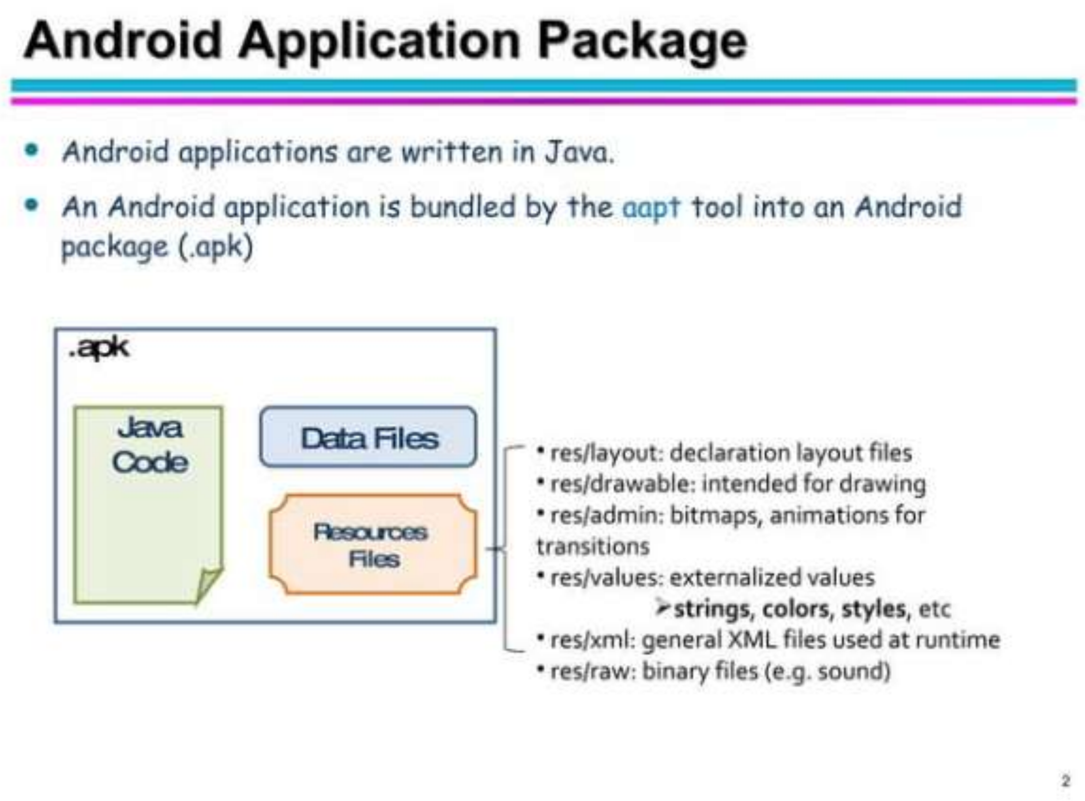
UNIT – III

Application Models for Mobile Application Frameworks



Android Application Package

- Android applications are written in Java.
- An Android application is bundled by the aapt tool into an Android package (.apk)

.apk

Java Code

Data Files

Resources Files

- res/layout: declaration layout files
- res/drawable: intended for drawing
- res/admin: bitmaps, animations for transitions
- res/values: externalized values
  - **strings, colors, styles**, etc
- res/xml: general XML files used at runtime
- res/raw: binary files (e.g. sound)

2

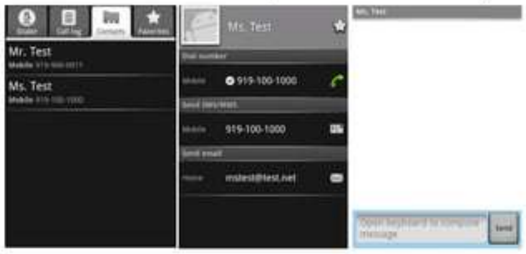
Application Components

- Android applications do not have a single entry point (e.g. no main() function).
- They have essential components that the system can instantiate and run as needed.
- Four basic components

Components	Description
Activity	UI component typically corresponding to one screen
Service	Background process without UI
Broadcast Receiver	Component that responds to broadcast Intents
Content Provider	Component that enables applications to share data

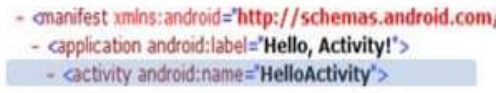
Components - Activity

- An activity is usually a single screen:
  - Implemented as a single class extending Activity.
  - Displays user interface controls (views).
  - Reacts on user input/events.
- An application typically consists of several screens:
  - Each screen is implemented by one activity.
  - Moving to the next screen means starting a new activity.
  - An activity may return a result to the previous activity.



Components - Activity (Cont)

- Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched.
- Created "Activity" must be defined into the application's manifest.

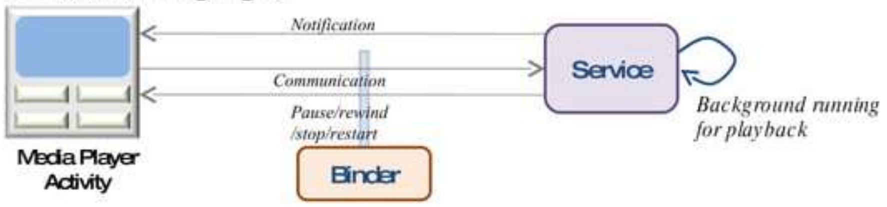


```
- <manifest xmlns:android="http://schemas.android.com/apk/res/android" packag
  - <application android:label="Hello, Activity!">
    - <activity android:name="HelloActivity">
```

Components - Service

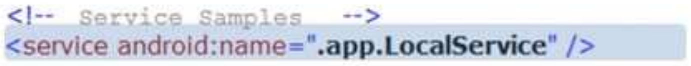
- A service does not have a visual user interface, but rather runs in the background for an indefinite period time.

  Example: music player, network download, etc
- Each service extends the Service base class.
- It is possible to bind to a running service and start the service if it's not already running.
- While connected, it is possible communicate with the service through an interface defined in an AIDL (Android Interface Definition Language).



Components - Service (Cont)

- Adding a "Service" with Android is quite similar than for an "Activity".

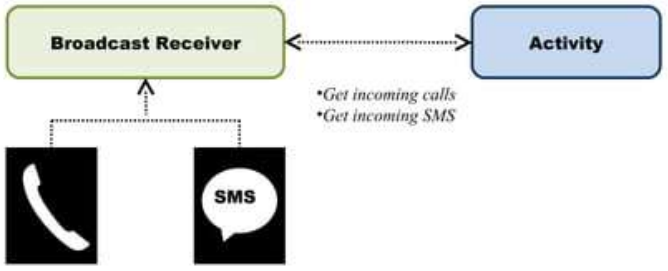


```
<!--  Service Samples   -->
<service android:name=".app.LocalService" />
```

Components - Broadcast Receivers

- A broadcast receiver is a component that receives and reacts to broadcast announcements (Intents).
  - ✓ Many broadcasts originate in system code.

    E.g. announcements that the time zone has changed, that the battery is low, etc.

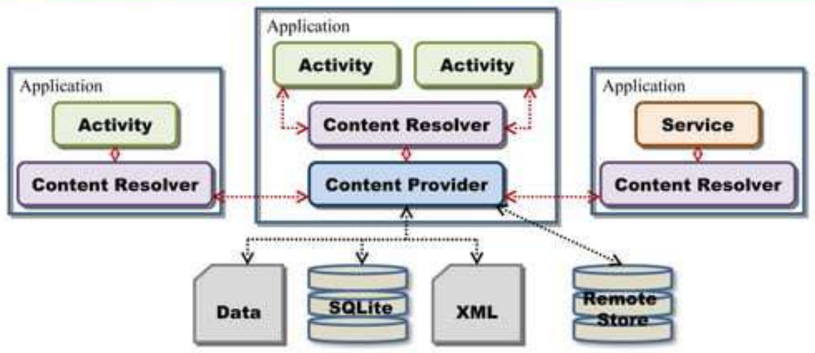


Components - Broadcast Receivers (Cont)

- A broadcast receiver is a component that receives and reacts to broadcast announcements. (Cont)
  - ✓ Applications can also initiate broadcasts.
    E.g. to let other applications know that some data has been downloaded to the device and is available for them to use.

- All receivers extend the ***BroadcastReceiver*** base class.

Components - Content Providers



- A content provider makes a specific set of the application's data available to other applications.
  - ✓ The data can be stored in the file system, in an SQLite, or in any other manner that makes sense.

Components - Content Providers (Cont)

- Using a content provider is the only way to share data between Android applications.
- It extends the ContentProvider bas class and implements a standard set of methods to allow access to a data store.
  - ✓ Querying
  - ✓ Delete, update, and insert data
- Applications do not call these methods directly.
  - ✓ They use a ContentResolver object and call its methods instead.
  - ✓ A ContentResolver can talk to any content provider.
- Content is represented by URI and MIME type.

Intents

- Intents are simple message objects each of which consists of
  - ✓ Action to be performed (MAIN/VIEW/EDIT/PICK/DELETE/DIAL/etc)
  - ✓ Data to operate on (URI)

```
startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("http://www.fhnw.ch"));

startActivity(new Intent(Intent.VIEW_ACTION,
Uri.parse("geo:47.480843,8.211293"));

startActivity(new
Intent(Intent.EDIT_ACTION,Uri.parse("content://contacts/people/1"));
```

Intents (Cont)

- Intent Filters
  - ✓ A component's intent filters in the manifest file inform Android of the kinds of intents the component is able to handle.
  - ✓ An example

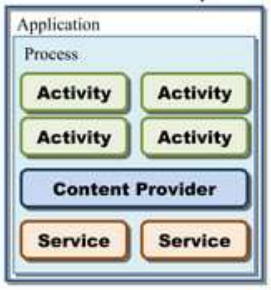
```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

Intents (Cont)

- Intent Filters (Cont)
  - ✓ An example (Cont)
    a. A component can have any number of intent filters, each one declaring a different set of capabilities.
    b. The first filter in the example indicates that the activity is the entry point for the application.
    c. The second filter declares an action that the activity can perform on a particular type of data.

Android Component Model

- An Android application is packaged in a .apk file.
  - ✓ A .apk file is a collection of components.



  - ✓ Components share a Linux process: by default, one process per .apk file.
  - ✓ .apk files are isolated and communicate with each other via Intents or AIDL.
  - ✓ Every component has a managed lifecycle.

The data-view model

This is the model used by the library Swing of Java notably. The view is the interface through which the user interacts with the software. Data are stored separately and can be displayed in different views.
The view may also change the data according to the context, for example, change the text according to the user's language.

User Interface(UI) Design In Android

User Interface (UI) design in Android is a graphical representation of views displayed on a smartphone or tablet. It allows users to interact with the features, functions, and contents of the application.

To design UI, you need no prior programming knowledge, although it is nice to have web developing skills or programming skills.

Every application has a user interface with which users can interact. Android provides various pre-built UI components that allow you to build a GUI for your application. Android also provides other UI modules for special interfaces, such as dialogues, notifications, menu, etc.

Components used for Android application

There are many components for the Android application. Let’s consider a few of them here:

- Main Action Bar (MAR)
- Split Action Bar (SAB)
- Content Area

These play a significant role while developing a complex Android application.

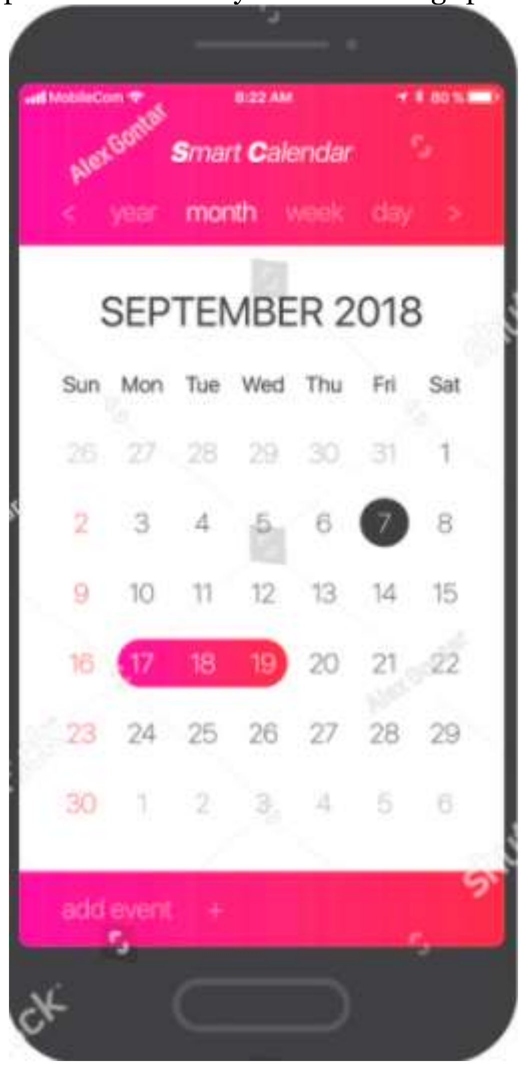
Views components

What is View?

Views are used to customize User Interface (UI) design. A view is considered a building block for a proper User Interface created from the view class.

A view can also be defined as a small rectangular box in Android development that responds to the user inputs, for example, buttons, checkboxes, etc. Basically, a View is what a user sees and interacts with.

View group is an invisible container of other views, such as child view and other view groups. This view class is a super class of all the GUI components in Android. Views play a key role in the development of an application because you need settings present in the application, e.g.,



color, style, themes, etc.

Figure 1.0. A simple UI Design

In the Android studio, we commonly use a view called "edit text and images". Images are classified as widgets used to create an interactive UI component, such as buttons, text fields, etc.

Layout as a subclass of View group

The Layout defines the visual structures of the UI of user application. All elements in the layout are built using a hierarchy of views and view group object.

The layout can be declared through a programming language, or through a simple XML layout file located in the resource layout folder of any project you work on. Android provides a straightforward XML vocabulary that corresponds to the view class and subclasses, such as those present for widget and layout. Due to this, the layout can be treated.

Layouts are kept in the folder called resources. Android studio creates a default XML layout file in the resources layout folder that is extremely useful if you know the basic at the time of compiling.

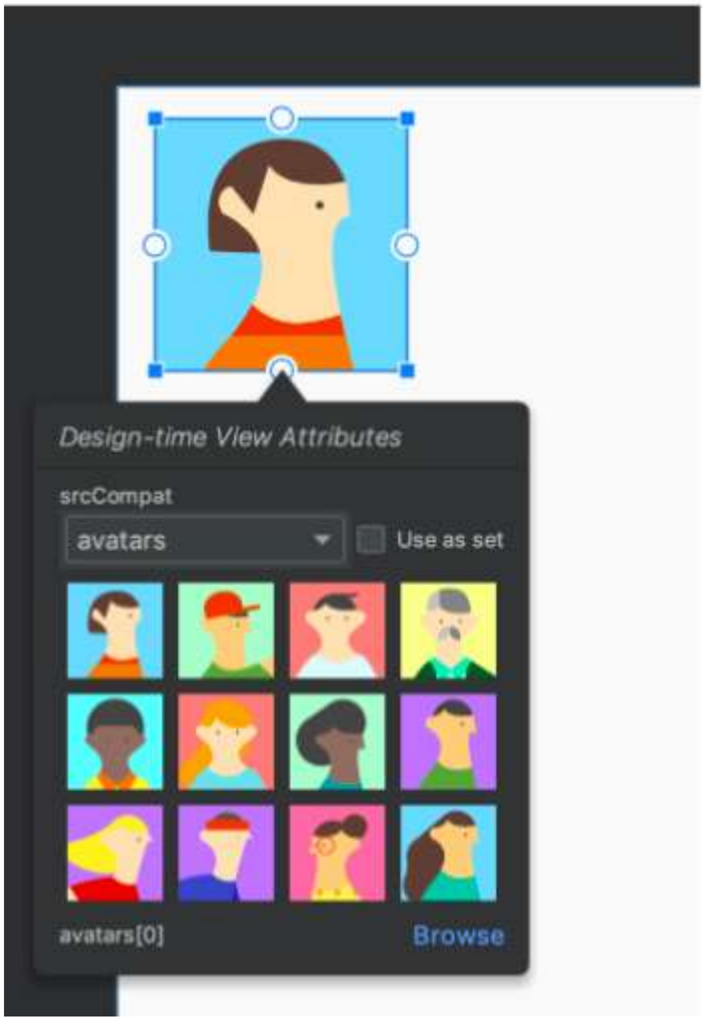


Figure 1.1. A simple layout with imageView

Advantages of XML Layout

There are many advantages of XML layout. These are:

- It is an immensely popular and widely used format.
- It helps provide the UI component from the logic, which in turn provides the flexibility to change one component without affecting the other.
- It is much easier to generate than writing direct code. It allows an easier drag and drop to generate interfaces for the Android application.

Types of layout

Android provides a number of layouts, that provide different views, to use in its applications.

The following are considered the standard layouts or view groups used in Android applications.

- **Linear layout**: It is used to arrange views in a single column or row. It is further divided into a vertical or horizontal linear layout.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
 <Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a Button" />
</LinearLayout>
```

- **Frame layout**: It is a placeholder on the screen used to display a single screen.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical" >
 <Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a Button" />
</FrameLayout>
```

- **Relative layout**: It enables you to specify how child views are positioned to each other.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
>
 <Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a Button" />
</RelativeLayout>
```

- **Table layout**: It groups the views into rows and columns.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
>
 <Button
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a Button" />
</TableLayout>
```

- **Constraint layout**: It provides you with adaptable and flexible ways to create views for your applications.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.andr
oid.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"

 <TextView
android:id="@+id/text"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Hello, I am a TextView" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Unit of measurement

There are a number of measurement units used in Android studio.

- **DP** – Density independent pixel
- **SP** – Scale independent pixel
- **PX** – Pixel

Managing Applications Data

When you develop Android applications, you have a number of options in terms of how you store and manage your data. Depending on the app, you may find yourself using more than one of these for different data tasks.

Internal Storage

You can store data items to the internal storage on your users' devices. The advantage to this is that it does not rely on any external media, but the disadvantage is that some devices have extremely limited amounts of storage space available. Storing to internal memory essentially means saving data files within the internal device directory structure.

The following Java code demonstrates opening a file for data output:

```
//get a FileOutputStream object by passing the file-name
FileOutputStream dataFileOutput = openFileOutput("datafile",
Context.MODE_PRIVATE);
```

When you save files to the internal storage, other applications can't access them and, by default, neither can the user, so it offers good reliability. The **FileOutputStream** write method takes byte parameters, so you can only use this technique to store data items your program can convert to bytes.

SD Cards

Many Android users are dependent on external media storage such as SD cards due to insufficient internal storage. SD cards offer your apps an increased amount of storage space, but they impose additional considerations. For example, you cannot assume that a user will have an SD card or other external media resource. For this reason your programming code needs to carry out checks before storing data in this way. You also need to accommodate the possibility of the user changing their SD card.

Using the **Environment** class, your apps can check the availability of writeable external storage as follows:

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
```

```
{
//can use the external storage
}
```

It's also worth bearing in mind that files saved to an SD card can be accessed by other apps and by the user directly, so the data is more vulnerable to corruption.

Databases

If your app uses more complex data items, you can create a SQLite database to store them using the relational model. When you create a SQLite database for your app, you can access it from any point within your Java code, and it will not be accessible from anywhere else. There are many advantages to using a database, such as the ability to store and execute structured queries.

To build a custom SQLite database from your Java code, you can extend the **SQLiteOpenHelper** class, then define and create your tables inside the "onCreate" method, as follows:

```
public void onCreate(SQLiteDatabase db) {
db.execSQL("CREATE TABLE Item (ItemID INTEGER, ItemName TEXT);");
}
```

This statement creates a database table with two columns in it. The **SQLiteDatabase** class provides the means to manage the data, including query, insert and update methods. One potential downside to using an SQLite database in your Android apps is the amount of processing code required, and the necessary skill set. If you already know your way around SQL, you should have no problems.

Shared Preferences

If you need to store simple data items for your apps, the most straightforward approach is to use Shared Preferences. This is possibly the easiest data management option to implement, but it's only suitable for primitive type items such as numbers and text. Using Shared Preferences, you model your data items as key value pairs. The following code demonstrates acquiring a reference to the **SharedPreferences** object for an app and writing a value to it:

```
//get the preferences, then editor, set a data item
SharedPreferences appPrefs = getSharedPreferences("MyAppPrefs", 0);
SharedPreferences.Editor prefsEd = appPrefs.edit();
prefsEd.putString("dataString", "some string data");
prefsEd.commit();
```

Anything you save to Shared Preferences will still be available the next time your app runs, so it's ideal for saving items such as user preferences and settings. When your app starts up, you can check the Shared Preferences, then present your interface and functionality accordingly. The following code demonstrates retrieving the string data item:

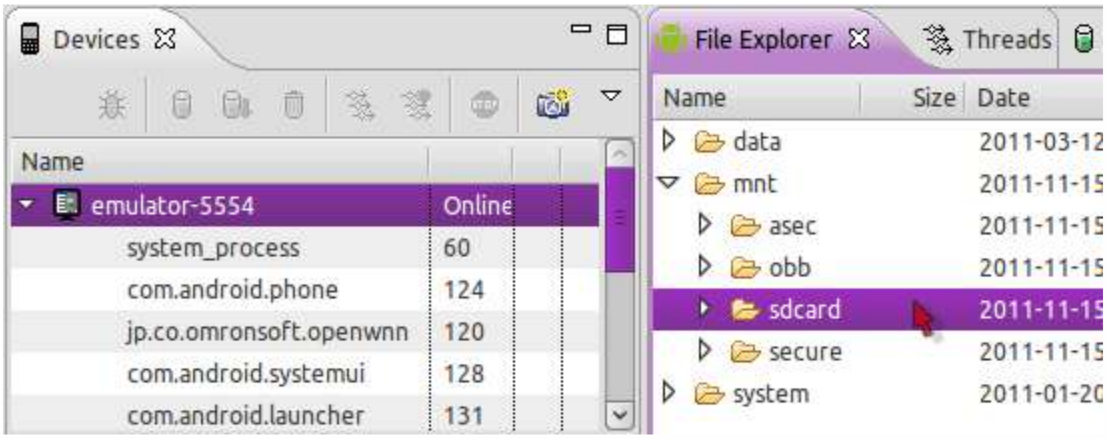
```
//get the preferences then retrieve saved data, specifying a default value
SharedPreferences appPrefs = getSharedPreferences("MyAppPrefs", 0);
String savedData = appPrefs.getString("dataString", "");
```

Web Storage

Since Android devices provide Internet connectivity, your apps can of course use data stored on the Web. Naturally, you need to consider how your applications will cope with restricted or missing connectivity. As with a Web application, this option allows you to store data using virtually any platform or model you prefer. The Java language also benefits from a range of standard libraries for handling data fetched over a network, from sources such as SQL databases and XML documents. The packages **java.net** and **android.net** provide classes for managing networked data.

Conclusion

If you're developing in Eclipse, you can manually control aspects of data storage on virtual devices in the DDMS perspective.



Name			
emulator-5554	Online		
system_process	60		
com.android.phone	124		
jp.co.omronsoft.openwnn	120		
com.android.systemui	128		
com.android.launcher	131		

Name	Size	Date
data		2011-03-12
mnt		2011-11-15
asec		2011-11-15
obb		2011-11-15
sdcard		2011-11-15
secure		2011-11-15
system		2011-01-20

Inevitably, you need to tailor your data management approach to your own Android development projects. The platform is a flexible one, so there's plenty of choice available. When you do implement data management within your apps, make sure you test them on actual devices as well as in the emulator, as there's more potential for error when users access your functionality within their own unique contexts.

Mobile Operating System

A mobile operating system is an operating system that helps to run other application software on mobile devices. It is the same kind of software as the famous computer operating systems like Linux and Windows, but now they are light and simple to some extent.

The operating systems

found on smartphones include Symbian OS, iPhone OS, RIM's BlackBerry, Windows

Mobile, Palm WebOS, Android, and Maemo. Android, WebOS, and Maemo are all derived from Linux

. The iPhone OS originated from BSD and NeXTSTEP, which are related to Unix.

It combines the beauty of computer and hand use devices. It typically contains a cellular built-in modem and SIM tray for telephony and internet connections. If you buy a mobile, the manufacturer company chooses the OS for that specific device.

Popular platforms of the Mobile OS

1. Android OS: The Android operating system

is the most popular operating system

today. It is a mobile OS based on the **Linux Kernel** and **open-source software**. The android operating system was developed by **Google**. The first Android device was launched in **2008**.

2. Bada (Samsung Electronics): Bada is a Samsung mobile operating system that was launched in 2010. The Samsung wave was the first mobile to use the bada operating system. The bada operating system offers many mobile features, such as 3-D graphics, application installation, and multipoint-touch.

3. BlackBerry OS: The BlackBerry operating system

is a mobile operating system developed by **Research In Motion** (RIM). This operating system was designed specifically for BlackBerry handheld devices. This operating system is beneficial for the corporate users because it provides synchronization with Microsoft Exchange, Novell GroupWise email, Lotus Domino, and other business software when used with the BlackBerry Enterprise Server.

4. iPhone OS / iOS: The iOS was developed by the Apple inc for the use on its device. The iOS operating system is the most popular operating system today. It is a very secure operating system. The iOS operating system is not available for any other mobiles.

5. Symbian OS: Symbian operating system is a mobile operating system that provides a high-level of integration with communication. The Symbian operating system is based on the java language. It combines middleware of wireless communications and personal information management (PIM) functionality. The Symbian operating system was developed by **Symbian Ltd** in **1998** for the use of mobile phones. **Nokia** was the first company to release Symbian OS on its mobile phone at that time.

6. Windows Mobile OS: The window mobile OS is a mobile operating system that was developed by **Microsoft**. It was designed for the pocket PCs and smart mobiles.

7. Harmony OS: The harmony operating system is the latest mobile operating system that was developed by Huawei for the use of its devices. It is designed primarily for IoT devices.

8. Palm OS: The palm operating system is a mobile operating system that was developed by **Palm Ltd** for use on personal digital assistants (PADs). It was introduced in **1996**. Palm OS is also known as the **Garnet OS**.

9. WebOS (Palm/HP): The WebOS is a mobile operating system that was developed by **Palm**. It based on the **Linux Kernel**. The HP uses this operating system in its mobile and touchpads.