ID2212 Network Programming with Java
Lecture 14

An Overview of the Android Programming

Hooman Peiro Sajjad
KTH/ICT/SCS
HT 2016

References

- **http://developer.android.com/training/index.html**
- **Developing Android Apps by Google (online course)**
- **Course ID2216 Developing Mobile Applications**
  - **Offered by the Communication Systems department**
  - **Course responsible: Associate Professor Konrad Tollmar**

Outline

- **Mobile Web Apps vs. Native Apps**
- **Android Platform**
- **Android Programming**
- **User Experience**

Mobile Web App / Native App

Mobile Web Apps (1/2)

- Accessing **browser-based internet services** from a **handheld mobile device**

- **Core technologies**: HTML, CSS and JavaScript

Mobile Web Apps (2/2)

- **Advantages***:
  - Cross-platform compatibility
  - Cheaper and easier to maintain
  - Simple and ubiquitous access

- **Disadvantages**:
  - Requires customization across different browser versions
  - Limited access to mobile's hardware and software
  - Generally requires internet connection

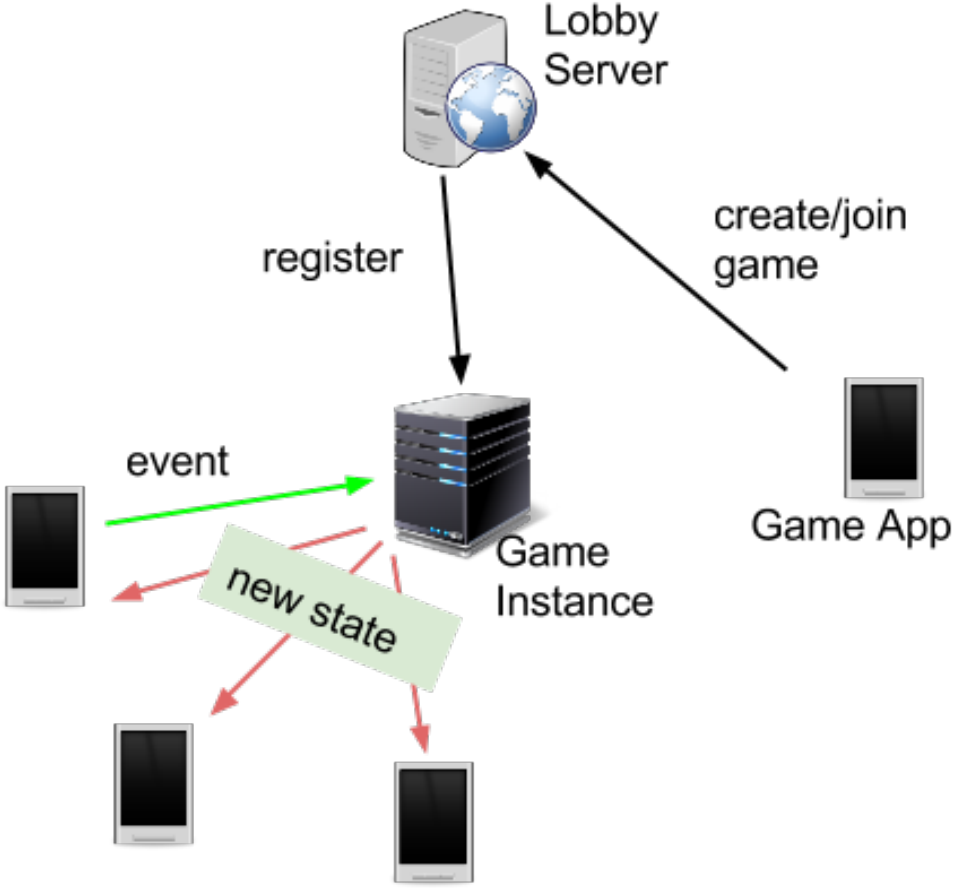
* www.lionbridge.com: Mobile Web Apps vs. Native Apps: How to Make the Right Choice

Mobile Native Apps

- Built specifically for a particular device and operating system

- **Advantages**:
  - Leverage the device specific hardware and software
  - Work offline
  - Better visibility in app stores, making money immediately
- **Disadvantages**:
  - Different versions of the app for different platforms
  - Keeping apps up to date is costly
  - Content publishers have to share information about their subscribers with the app store

A Sample Architecture of a Mobile Application



Overview of Android Platform

- **Android Operating System**
- **Application Lifecycle**

Android Operating System (1/2)

- **Google's Linux based open-source OS that includes:**
  - **Linux kernel** optimized for mobile and embedded devices
  - **Open-source application development libraries** such as SQLite, OpenGL, and a media manager
  - **A runtime** to host and execute Android applications, including **Android Runtime** (**ART**)
  - **An application framework** to expose system services to the application layer, including the window manager and location manager, databases, telephony and sensors
  - **A user interface framework** used to host and launch applications
  - A set of core **pre-installed applications**

Android Operating System (2/2)

Dalvik Virtual Machine (1/2)

- The **process virtual machine** (VM) in Google's Android operating system
- **Runs the apps** on Android devices.
- Programs are commonly written in **Java** and compiled to **bytecode**.

Dalvik Virtual Machine (2/2)

- Then converted from **Java Virtual Machine-compatible** .class files to **Dalvik-compatible** .dex (Dalvik Executable) files before installation on a device.

- In **Android 5,** a new virtual machine – **Android Runtime** (**ART**) – replaced Dalvik as the platform default.

Application Lifecycle (1/2)

- Android applications have **limited control over their own lifecycle**.
- Each application **runs in its own process**.
- Applications have **different priorities**.

Application Lifecycle (2/2)

- Android can **kill applications without warning**, to free resources for higher-priority applications.
- An application's priority is equal to that of its **highest-priority component**.
- It's important to structure the application to ensure that it has the right priority for the work it's doing.

Application Lifecycle: Application States (1/3)

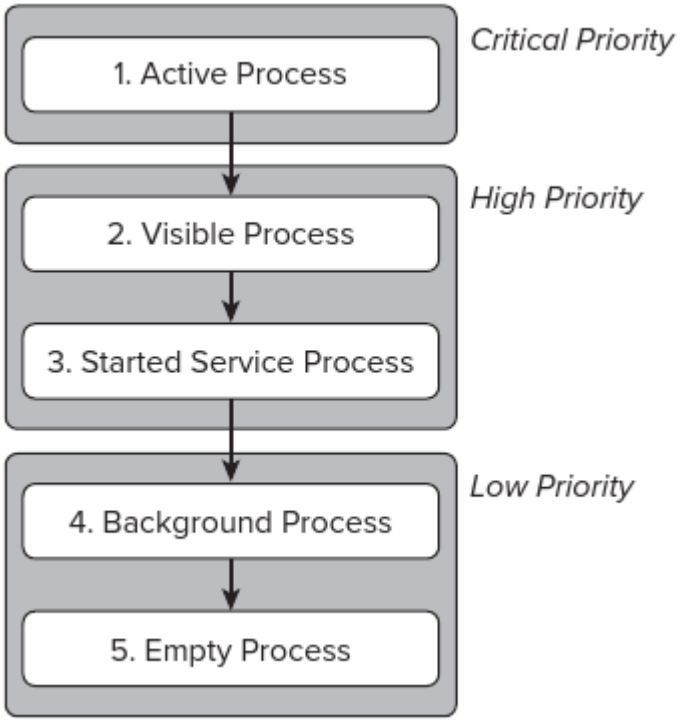


Figure taken from "Professional Android 4 Application Development (3rd Edition)".

Application Lifecycle: Application States (2/3)

- **Active**: includes application components the user is interacting with.
- **Visible**: those activities which aren't in the foreground but still can affect what the user sees on screen.
- **Started service**: processes hosting services.
- **Background**: processes hosting Activities which aren't visible and don't have any running services.
- **Empty**: a process having no active application component

Application Lifecycle: Application States (3/3)

And now let's watch an example!

Android Programming

- **Android SDK**
- **Application Model and Components**
- **Processes and Threads**
- **Permissions**
- **Networking**
- **Location**

Development Environment

- First, **download Android Studio.** It includes:
  - Integrated Development Environment (IDE)
  - Android SDK Tools
  - Android Platform-tools
  - The latest Android platform
  - The latest Android system image for the emulator

- Notice that, you need to have **JDK** installed beforehand.

Android SDK (1/3)

- Provides the **API libraries** and **developer tools** necessary to **build, test,** and **debug apps** for Android.

- Includes:
  - **Build Tools:** all the tools required to compile and build the app.
  - **SDK Tools:** Contains main tools for debugging and testing, plus other utilities that are required to develop an app.

Android SDK (2/3)

- **SDK Platform-tools:** Contains platform-dependent tools for developing and debugging your application.
- **Documentation:** the latest documentation for the Android platform APIs.
- **SDK Platform:** It includes an android.jar file with a fully compliant Android library.
- **System Images:** Required system images for the Android emulator.
- **Google APIs:** APIs which adds special Google features to your apps.

Android SDK (3/3)

- **Android support:** a set of code libraries that provide backward-compatible versions of Android framework APIs as well as features that are only available through the library APIs
- **Google Play Billing:** Provides the static libraries and samples that allow you to integrate billing services in your app with Google Play.
- **Google Play Licensing:** Provides the static libraries and samples that allow you to perform license verification for your app when distributing with Google Play.

Application Model and Components

- Every Android application consists of some **loosely coupled components** and the **application manifest**.

- The manifest defines **application's metadata** and the **components bindings**.

Application Components

- **Activities & UI design elements:** The application's presentation layer.
- **Services:** components that run in the background to perform long-running operations.
- **Intents:** a powerful inter-application message passing framework.
- **Broadcast Receivers:** Intent listeners *(not covered in this lecture)*
- ***Content Provider:*** manages a shared set of application data *(not covered in this lecture)*

Application Manifest

- Every Android project includes a **manifest file**.
- Defines **the structure and metadata** of the application, its **components** and **requirements**.
- **AndroidManifest.xml**

Manifest Example

```
<application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".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>
```

Activities

- Each activity represents a **screen** that an application can present to its users.
- To create an activity, you must create a **subclass of Activity**.
- **Implement callback methods** inherited from Activity class.
- Two important callback methods:
  - **onCreate**(): called when creating the activity.
  - **onPause**(): indicates that the user might be leaving.

Activities: Example

```
public class MainActivity extends Activity {

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

Activity Lifecycle

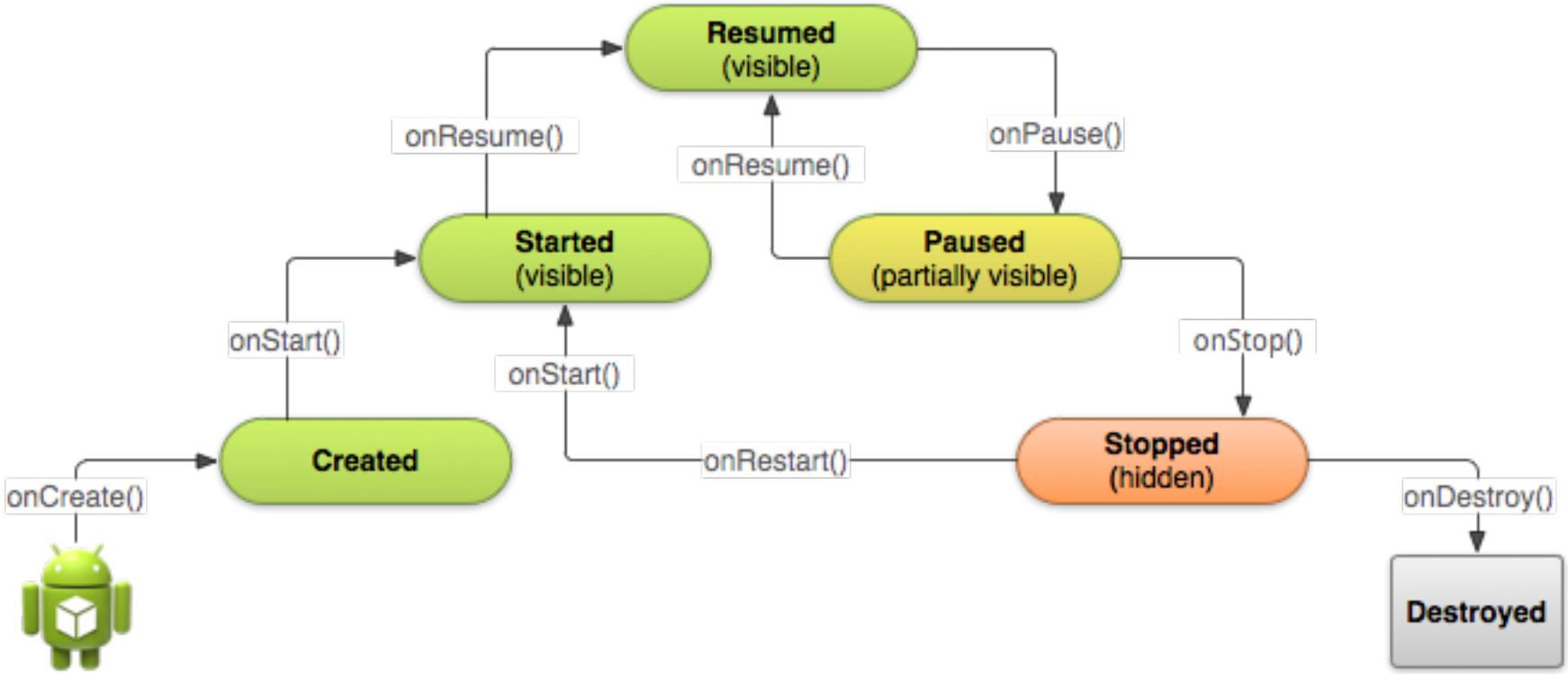


Figure taken from "developer.android.com/training/basics/activity-lifecycle/starting.html".

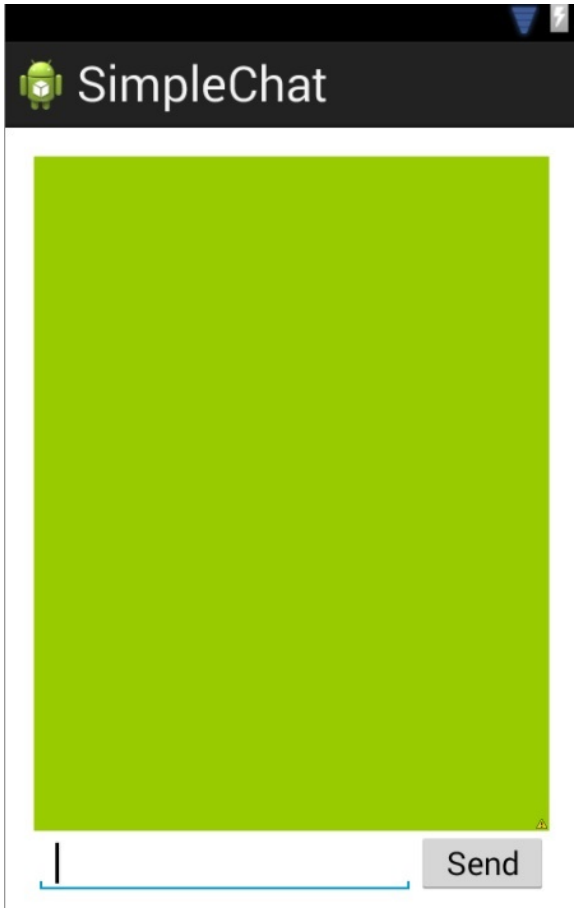
User Interface Design

Some UI terminologies in Android:

- **Views:** the base class for all visual interface elements.
- **View Groups:** extensions of the View class that can contain multiple child Views.
- **Fragments:** A Fragment represents a behavior or a portion of user interface in an Activity. Fragments have their own lifecycle, state, and back stack.
- **Activities:** represents the window or screen being displayed. To display a UI, you assign a View to an Activity.

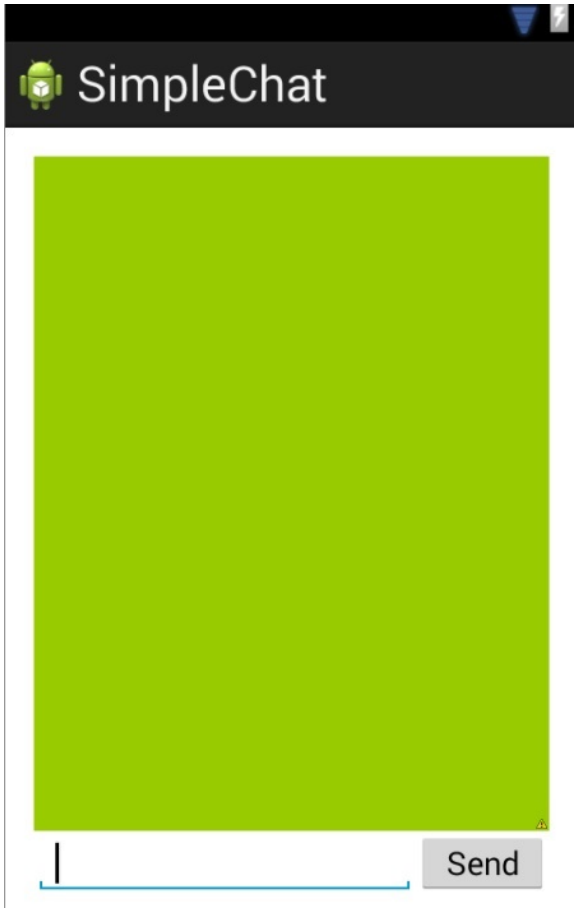
User Interface: Example (1/2)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="top"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/chatTextView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@+id/sendButton"
        android:layout_alignParentTop="true"
        android:layout_gravity="top"
        android:background="@android:color/holo_green_light" />
```



User Interface: Example (2/2)

```
<Button
        android:id="@+id/sendButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/chatTextInput"
        android:layout_alignRight="@+id/chatTextView"
        android:layout_alignTop="@+id/chatTextInput"
        android:layout_toRightOf="@+id/chatTextInput"
        android:text="@string/send" />

    <EditText
        android:id="@+id/chatTextInput"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/chatTextView"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="16dp"
        android:ems="10"
        android:inputType="textMultiLine" >

        <requestFocus />
    </EditText>

</RelativeLayout>
```



Intent

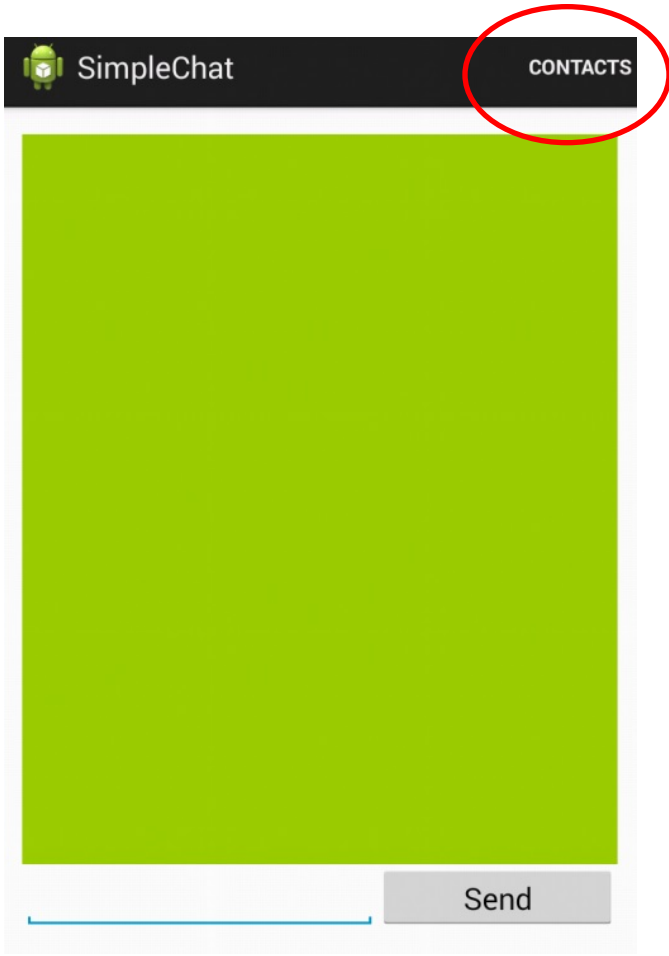
Intents work as **a message-passing** mechanism both **within and between applications**.

Using Intents you can:

- **Explicitly**, start a particular Service or Activity using its class name
- **Implicitly**, start an Activity or Service by requesting an action on a piece of data
- Broadcast the occurrence of an event

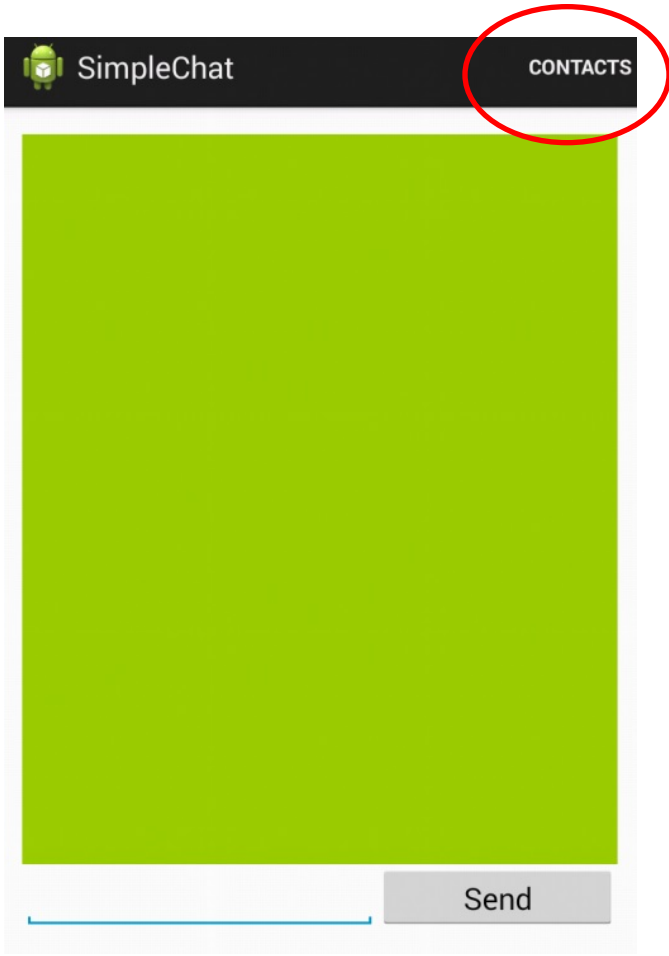
Explicit Start of a New Activity

```
Intent intent = new Intent(MainActivity.this,
                           MyContactsActivity.class);

startActivityForResult(intent, PICK_CONTACT);
```



Implicit Start of a New Activity

```
Intent intent = new Intent(Intent.ACTION_PICK,
                           Contacts.CONTENT_URI);

startActivityForResult(intent, PICK_CONTACT);
```



Service (1/3)

- A Service is an application component that can perform **long-running operations** in the **background** and does not provide a user interface.
- A service can run in the background to perform work even while the user is in a different application.
- A component **can bind to a service to interact with it** and even perform inter-process communication (IPC).
- A service might handle **network transactions, play music, perform file I/O**

Service (2/3)

- To create a service, you must create **a subclass of Service.**

- You need to **override the callback methods** to control the behavior of the service:
  - **onStartCommand**(): when another component requests the service to start.
  - **onBind**(): when another component wants to bind with the service
  - **onCreate**(): when the service is first created.
  - **onDestroy**(): when the service is no longer used and is being destroyed.

Service (3/3)

Declare the service in the manifest

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Processes

- By default, **all components** of the same application run in **the same process**.
- You can define **in the manifest,** that **different components** of the same application are to be run in **different processes**.

android:process="*string*"

Threads (1/2)

- When an application is launched, the system creates a **thread of execution for the application,** called **"main.“**
- The **main thread** is called **UI thread** because: it interacts the **Android UI components**.
- **Performing long operations** such as network access or database queries will **block the whole UI.**

Threads (2/2)

- The **Andoid UI toolkit** is **not thread-safe**, so do not manipulate your UI from a worker thread.
- **Remember these two rules:**
  1. Do not block the UI thread
  2. Do not access the Android UI toolkit from outside the UI thread

Threads: Example (1/3)

An example of **wrong implementation**:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

Worker thread is updating ImageView which is not thread-safe.

Threads: AsyncTask (1/3)

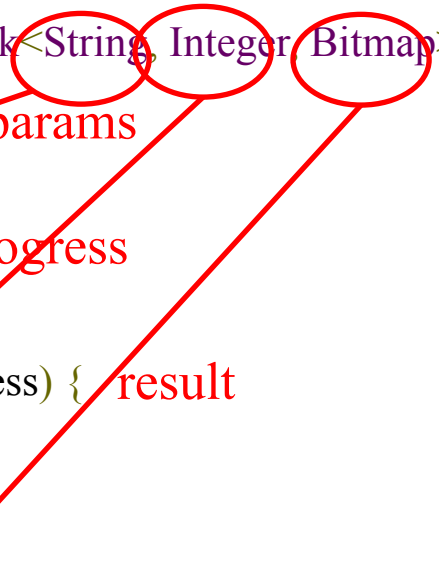
- Correct implementation using **AsyncTask**
- AsyncTask performs the **blocking operations in a worker thread** and then **publishes the results on the UI thread**.
- you must **subclass AsyncTask** and implement the **doInBackground**() callback method.
- To **update the UI**, you should implement **onPostExecute**()

Threads: AsyncTask (2/3)

```
public void onClick(View v) {

    new DownloadImageTask().execute("http://example.com/image.png");

}

private class DownloadImageTask extends AsyncTask<String, Integer, Bitmap> {

    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }
    protected void onProgressUpdate(Integer… progress) {
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Threads: AsyncTask (3/3)

```
private class DownloadImageTask extends AsyncTask<String, Integer, Bitmap> {

    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    protected void onProgressUpdate(Integer… progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```



Permissions

- A basic Android application has no permission associated with it by default, so it cannot access data on the device.
- To make use of protected features of the device, you must give the related to permissions to your application.
- Permissions must be added to **AndroidManifest.xml**. Example:

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

Network and Internet Connectivity (1/4)

- There are **different network technologies** with different speed, reliability and cost:
  - **Wi-Fi, GPRS, 3G, LTE** and so on
- Application can manage these connections to ensure the **efficiency** and **responsiveness**
- Networking in Android is handled via **ConnectivityManager**.
- Changes in network connectivity are broadcasted by Android to Intents.

Network and Internet Connectivity (2/4)

To utilize the network connectivity, following **user permissions** are required:

- **INTERNET**: Allows applications to open network sockets.
- **ACCESS_NETWORK_STATE**: Allows applications to access information about networks.

Network and Internet Connectivity (3/4)

To check if the network is connected:

```
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
  NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
  if (networkInfo != null && networkInfo.isConnected()) {
     // do network operations
  } else {
     // display error
  }
```

Network and Internet Connectivity (4/4)

- The **NetworkInfo** object includes the type of the network connection which is available.

- **getType**() returns the network connection type:

  - **TYPE_MOBILE**
  - **TYPE_WIFI**
  - **TYPE_WIMAX**
  - **TYPE_ETHERNET**
  - **TYPE_BLUETOOTH**

Location (1/4)

- The central component of the location framework is the **LocationManager** system service.
- Using **Google Maps Android API**, you can add maps to your app based on Google Maps data.
- The application can acquire the user location utilizing **GPS** and Android's **Network Location Provider**.

Location (2/4)

- **Network Location Provider:**
  - Determines location through cell tower and Wi-Fi signals
  - Works indoors and outdoors
  - Responds faster
  - Less battery power

- **GPS:**
  - Most accurate
  - Only works outdoor
  - Consumes battery quickly
  - Slow

Location (3/4)

You need to request user permission for either:

•**ACCESS_FINE_LOCATION**: Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.

•**ACCESS_COARSE_LOCATION**: Allows an app to access approximate location derived from network location sources such as cell towers and Wi-Fi.

Location (4/4)

- Getting user location in Android works by means of **callback**.
- First, acquire a reference to the system Location Manager

```
LocationManager locationManager =
        (LocationManager)this.getSystemService(
                                Context.LOCATION_SERVICE);
```

Location: Define a Listener

- Define a listener that responds to location updates

```
LocationListener locationListener = new LocationListener() {

   public void onLocationChanged(Location location) {
     makeUseOfNewLocation(location);
   }

   public void onStatusChanged(String provider, int status,
                               Bundle extras) {}

   public void onProviderEnabled(String provider) {}

   public void onProviderDisabled(String provider) {}
  };
```

Location : Register the Listener

- Register the listener with the Location Manager to receive location updates

```
locationManager.requestLocationUpdates(
        LocationManager.NETWORK_PROVIDER, 0,
        0, locationListener);
```

Location: Last Known Location

- If you need to Get the **last known location** for the quick location information:

```
String locationProvider =
   LocationManager.NETWORK_PROVIDER;
   // Or use LocationManager.GPS_PROVIDER
Location lastKnownLocation =
   locationManager.getLastKnownLocation(locationProvider
   );
```

User Experience

- A **high quality app** is more probable to have **higher user ratings, better rankings, more downloads.**
- Improve **stability** and **eliminate bugs**
- Improve **UI responsiveness**, a slow and unresponsive UI will disappoint the users.
- Improve **the ease of use**

User Experience

- **High quality User Interface**
- Having the **right set of features**
- You can find many good suggestions and best practices to improve your application following the link:

http://developer.android.com/training/index.html