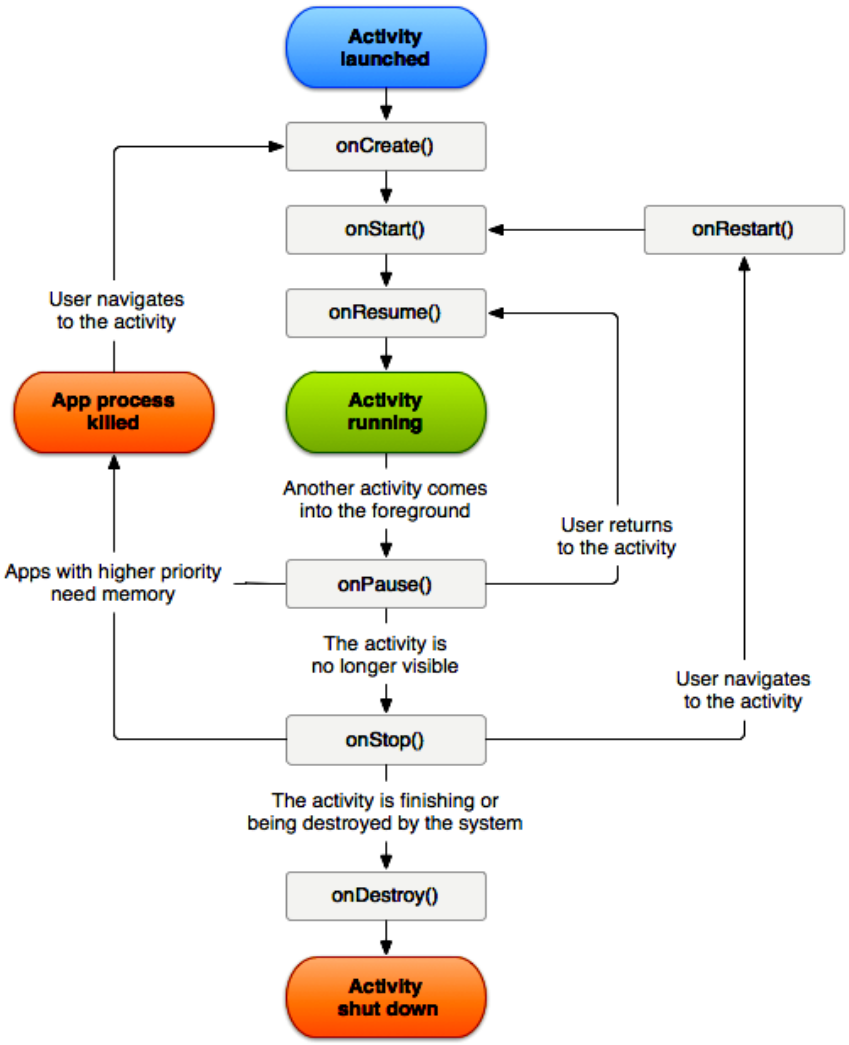
UNIT – II
SIMPLE ANDROID APPLICATION DEVELOPMENT

ACTIVITY IN ANDROID

An Android activity is one screen of the Android app's user interface. In other words, building block of the user interface is the activity. Activity class is a pre-defined class in Android and every application which has UI must inherit it to create window. An Android app may contain one or more activities, meaning one or more screens. The Android app starts by showing the main activity, and from there the app may make it possible to open additional activities.An activity provides the window in which the app draws its UI.

LIFE CYCLE OF AN ACTIVITY

Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class. The 7 lifecycle method of Activity describes how activity will behave at different states. Below figure shows the life cycle of an activity and the various stages it goes through — from when the activity is started until it ends.



Android Activity Lifecycle methods

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen.

Method	Description
onCreate	Called when activity is first created. Used to initialize the activity, for example create the user interface.
onStart	Called when activity is becoming visible to the user.
onResume	Called if the activity get visible again and the user starts interacting with the activity again. Used to initialize fields, register listeners, bind to services, etc.
onPause	Called once another activity gets into the foreground. Always called before the *activity* is not visible anymore. Used to release resources or save application data. For example you unregister listeners, intent receivers, unbind from services or remove system service listeners.
onStop	Called once the activity is no longer visible. Time or CPU intensive shut-down operations, such as writing information to a database should be down in the onStop() method.
onRestart	Called after your activity is stopped, prior to start.
onDestroy	Called before the activity is destroyed.

INTENT

Android application components can connect to other Android applications. This connection is done using an Intent object. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.An intent can contain data via a Bundle. This data can be used by the receiving component.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Types of Android Intents

Android supports explicit and implicit intents. An application can define the target component directly in the intent (*explicit intent*) or ask the Android system to evaluate registered components based on the intent data(*implicit intents*). Below Figure shows how an

intent is used when starting an activity. When the Intent object names a specific activity component explicitly, the system immediately starts that component.

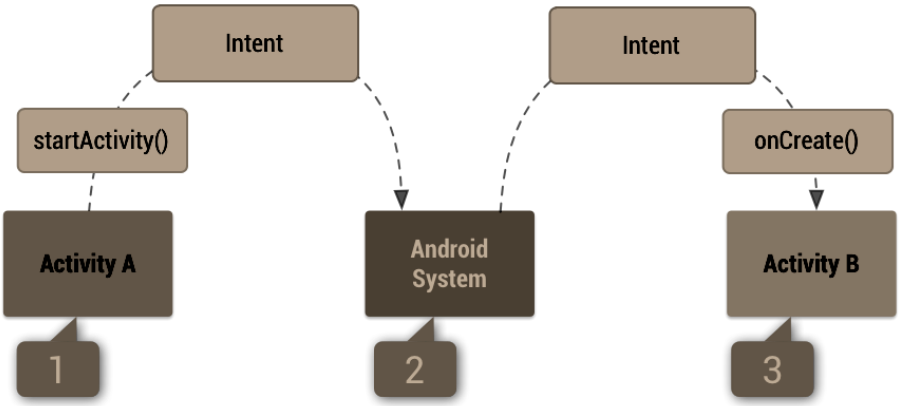


Fig:How an implicit intent is delivered through the system to start another activity:
[1] *Activity A* creates an Intent with an action description and passes it to startActivity().
[2] The Android System searches all apps for an intent filter that matches the intent. When a match is found,
[3] the system starts the matching activity (*Activity B*) by invoking its onCreate() method and passing it the Intent.

1) Implicit Intent

Implicit intents specify the action which should be performed and optionally data which provides content for the action. If an implicit intent is sent to the Android system, it searches for all components which are registered for the specific action and the fitting data type. If only one component is found, Android starts this component directly. If several components are identified by the Android system, the user will get a selection dialog and can decide which component should be used for the intent. This process is called **intent resolution**. Intent filters are used for this which is specified in AndroidManifest.xml

For example, the following tells the Android system to view a webpage. All installed web browsers should be registered to the corresponding intent data via an intent filter. Intent resolution data are action,data(both uri &type) and category

For example, you may write the following code to view the webpage.

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.javatpoint.com"));
startActivity(intent);
```

An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. Likewise, if you do *not* declare any intent filters for an activity, then it can be started only with an explicit intent.

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked. Explicit intents explicitly define the component which should be called by the Android system, by using the Java class as identifier. Explicit intents are typically used within an application as the classes in an application are controlled by the application developer. The following shows how to create an explicit intent and send it to the Android system to start an activity.

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

LINKING ACTIVITIES USING INTENT

Activities are linked using Intent class

The Intent class

It is a datastructure that represent two things:

1. An operation to be performed
2. an event that has occurred

The Intent class defines no less than six constructors.

- Intent()
- Intent(Intent o)
- Intent(String action)
- Intent(String action, Uri uri)
- Intent(Context packageContext, Class cls)
- Intent(String action, Uri uri, Context packageContext, Class cls)

Intent fields are

- **Action** – Represent action to be performed

  Eg: **ACTION_CALL -** Perform a call to someone specified by the data.
  ACTION_DIAL - Dial a number as specified by the data.
  ACTION_EDIT – Displays data to edit
  ACTION_SYNC - Perform a data synchronization with server.
  ACTION_MAIN - Start as a main entry point, does not expect to receive data.

  Setting intent action

  There are several ways:

  Eg: Intent i=new Intent(Intent.ACTION_DIAL);

  Or

  Intent i=new Intent();
  i.setAction(Intent. ACTION_DIAL);

- **Data**

  Data associated with the intent which is formatted as Uniform Resource Identifier(URI)

Setting Intent data

```
Intent i=new Intent();
i.setAction(Intent. ACTION_DIAL);
i.setData(Uri.parse("tel:+1555555555555"));
```

or

```
Intent i=new Intent(Intent. ACTION_DIAL, Uri.parse("tel:+1555555555555"));
```

- **Category** –

  Gives additional information about the components that can handle the intent

  Eg: CATEGORY_BROWSABLE- can be invoked by a browser to display data references by a URI

  CATEGORY_LAUNCHER- Can be the initial activity of a task and is listed in top level app launcher

- **Type** – Specifies the MIME type of the intent data

  Eg:image/text/html

- **Component** – specifies the component that should receive this intent.This is used when there is exactly one component that should receive the intent.

  Eg: Intent i=new Intent(getApplicationContext(),Second.class);

- **Extras** – Additional information associated with intent

  Eg: Intent i=new Intent();

  i.putExtra("Msg","Hello");

- **Flags** –specifies how intent should be handled

Application Context

It is an instance which is the singleton and can be accessed in an activity via getApplicationContext(). This context is tied to the lifecycle of an application. The application context can be used where you need a context whose lifecycle is separate from the current context or when you are passing a context beyond the scope of an activity.

Using permissions

Inside manifest file add appropriate tag as below.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

DATA PASSING BETWEEN ACTIVITIES USING INTENT

One of the most important aspect in developing android application is **passing data** between activities. Usually an Android app is made by several activities that work together.

These activities are pieces of code independent each other and we need to **exchange information** between them.

This method is the most elegant one because it uses all the android features. **Intent** in android is an action description. ***Intent*** supports three ways to pass data:

- **Direct**: put our data into intents directly
- **Bundle**: create a bundle and set the data here
- **Parcelable**: It is a way of "serializing" our object.

To analyse these methods let's suppose we have two activities: one that allows user to enter data (***EditActivity***) and the other one that shows to the user the data (***ViewActivity***). We have to pass this data between our activities. Let's suppose, moreover, that we want to pass the name, surname and the email.

Passing data: Direct

Intent has several method called ***putExtra(String name, …..)*** that allows us to save inside the Intent our information. Reading the API Doc we know that we can add string, longs, CharSequence, int and so on.It behaves like a Map where there's the key and the value. In the caller (***EditActivity***) we can pass data in this way:

```
Intent i = new Intent(EditActivity.this, ViewActivity.class);
i.putExtra("name", edtName.getText().toString();
i.putExtra("surname", edtSurname.getText().toString();
i.putExtra("email", edtEmail.getText().toString();
startActivity(i);
```

Where "name", "surname" and "email" are the keys. In other word first we create the intent (line 1) defining the caller activity (EditActivity) and the destination activity (ViewActivity), then we use **putExtra** to add our data. In the destination activity to retrieve the data sent we have:

```
protected void onCreate(Bundle savedInstanceState) {
Intent i = getIntent();
String name = i.getStringExtra("name"));
String surname = i.getStringExtra("surname"));
String email = i.getStringExtra("email"));
. . .
}
```

As we can see this is very easy way to pass data and can be used when we have primitives or very simple object to share.

Passing data: Bundle

Android has a class called **Bundle** where we can store our data and supports several data types like strings, chars, boolean, integer and so on. Instead of using the **Intent**,as data container, we store our info directly into the **bundle** and then we save the bundle into the

Intent. This approach is similar to the one described above. In this case in the caller activity we have:

```
Intent i = new Intent(EditActivity.this, ViewActivity.class);
Bundle b = new Bundle();
b.putString("name", edtName.getText().toString());
b.putString("surname", edtSurname.getText().toString());
b.putString("email", edtEmail.getText().toString());
i.putExtra("personBdl", b);
startActivity(i);
```

Notice that , we don't store our data into the Intent but into the Bundle, while in the destination activity:

```
Intent i = getIntent();
Bundle b = i.getBundleExtra("personBdl");
String name = b.getString("name");
String surname = b.getString("surname");
String email = b.getString("email");
```

Passing data: Parcelable

This is the most elegant way to pass data between activities . ***Parcelable*** in android is an interface and every object that wants to be passed across different activities using intent has to implement this interface. It is something like Serializable in Java. This interface has two method that we have to implements: ***describeContent***() that returns an ***int*** and ***writeToParcel(Parcel dest, int flags)*** that returns a void. More over a class that implements this interface must have a static field called ***CREATOR*** that is used by the OS to recreate the object.This strategy is very useful when we need to pass complex objects and of course we can't use the first two strategies. Let's suppose we have a class called ***Person*** that contains three attributes :*name*, *surname* and *email*. Now if we want to pass this class it must implement the Parcelable interface like that:

```
public class Person implements Parcelable {
private String name;
private String surname;
private String email;
// Get and Set methods

@Override
public int describeContents() {
return hashCode();
}
@Override
public void writeToParcel(Parcel dest, int flags) {
```

```
dest.writeString(name);
dest.writeString(surname);
dest.writeString(email);
}
// We reconstruct the object reading from the Parcel data
public Person(Parcel p) {
name = p.readString();
surname = p.readString();
email = p.readString();
}
public Person() {}
// We need to add a Creator
public static final Parcelable.Creator&lt ;
person&gt;
CREATOR = new Parcelable.Creator&lt;
person&gt;() {
@Override
public Person createFromParcel(Parcel parcel) {
return new Person(parcel);
}
@Override
public Person[] newArray(int size) {
return new Person[size];
}
};
```

Let's analyze the code shown above. The first thing we did is to declare that our class implements ***Parcelable***, so we have to override two method as stated above. The first one is ***describeContents***() where we simply return the hashcode of our class. The second is ***writeToParcel***. In this method we write the attribute values into the Parcel class. We use the Parcel methods to store our information. The order used to write them isn't important, but we have to use the same order when we have to read it. Once we have declared that our class is Parcelable, we have to implement a static field called ***CREATOR***. This field is used by OS to "unparcel" or "deserialize" our class. This static field looks like that public static final Parcelable.Creator<person> CREATOR = new Parcelable.Creator< person>() {

```
@Override
public Person createFromParcel(Parcel parcel) {
return new Person(parcel);
}
@Override
public Person[] newArray(int size) {
```

```
return new Person[size];
}};
```

There are two other methods to override the most important one is ***createFromParcel*** where we receive a parcel and we have to build up our Object, in our case Person. We call in this method a constructor, we made, that read data from the Parcel object:

```
// We reconstruct the object reading from the Parcel data
        public Person(Parcel p) {
        name = p.readString();
        surname = p.readString();
        email = p.readString();
        }
```

It is import to notice that the order used to read data is the same used to write them. Now we simply pass data like that:

```
        Intent i = new Intent(EditActivity.this, ViewActivity.class);
        Person p = new Person();
        p.setName(edtName.getText().toString());
        p.setSurname(edtSurname.getText().toString());
        p.setEmail(edtEmail.getText().toString());
        i.putExtra("myPers", p);
        startActivity(i);
```

As you notice we simply put our object Person into the Intent. When we receive the data we have:

```
        Bundle b = i.getExtras();
        Person p = (Person) b.getParcelable("myPers");
        String name = p.getName();
        String surname = p.getSurname();
        String email = p.getEmail();
```

Android Application Components

Android application components are basic, but essential, building blocks of any android application. Each component is an entry point through which the system or user can enter your app. Some components depend on other. They are loosely coupled with Application manifest file. In Manifest file, we describe how are they being used and interacting with other components. Each component type has a distinct purpose and distinct lifecycle that describes how the component was created and destroyed.

There are 4 main components of any android application. They are

1. Activities
2. Services
3. Broadcast Receivers
4. Content Providers

1. Activities

An activity is single screen with user interface. For example, A activity can have user interface for signup process, Another activity can have user interface for login process. Another activity can show all the emails, like gmail app showing all the mails in activity. All activity are independent from each other i.e. they do not depend on each other to show something. They are standalone. However, one activity can start another activity when required. But, it does not mean they are depended to the caller activity. They can be called from anywhere. It also represents entry point for any android application.

An activity provides following features :

1. It keeps track of the what the user currently using..i.e. what is on the screen right now. Activity does so that system keeps running the process that activity needs to be on the screen.

2. It keeps track of the previously visited activities or other things so that user may return to it whenever needed. It also store the state of the activity before closing the activity so that it can restore the state whenever needed.

3. It also provides a way for apps to implement user flows between each other, and for the system to coordinate these flows.

If any application has more than one activity, one of them is made launcher activity. Launcher activity is launched when application is started. An activity is implemented as subclass of Activity class. You can create an activity as below

```
public class MainActivity extends Activity {

}
```

2. Services

A service is component that runs in background to perform long running operations or to perform work for remote processes. For example, A service may play music in the background while the use is using a different application. A service may upload file to server from the server while the use is using a different application. A service does not provide user interface. You can declare the service as below.

```
public class MyService extends Service {
}
```

Generally Android Service are of 2 types.

- **Started Service (Unbounded)**

This type of service is created and called by Android Activities. There is no 2 way communication between Android Activity and Service. The Activity just starts the service and does not care about the status of the service. The Service will finish it's work and automatically stops when finish it's job.

- **Bound Service (Bounded)**

This type of Android Service is for 2 way communication. Suppose an Android Activity has started a bounded service, then Activity can be notified about the status by the service.

3. Broadcast Receivers

A broadcast receiver is a component that receive information, broadcasted by other applications or by the system, and take some actions accordingly. For example, An app can schedule an alarm to notify the user at any specified time. It's same as we set an alarm to notify at a specific time. Another example is notify the user when battery is low. Although the user do not show user interface, it can create status bar notifications to alert the user to take a specific action. A broadcast receiver is implemented as subclass of BroadcastReceiver.

```
public class MyReceiver  extends  BroadcastReceiver {
  public void onReceive(context,intent){}
}
```

4. Content Providers

Content providers make the content of any file system, SQLite database, or on any other persistent storage, available to any other applications. i.e. you can use content providers to get a specific set of data in any application. Means it acts as bridge between the data and the application. you can also access many native databases using content providers. For example, you can access contacts using ContactManager. You must have proper permissions to access data of other applications. You can implement Content Providers by subclassing ContentProvider as below –

```
public class MyProvider extends  ContentProvider {
  public void onCreate(){}
}
```

Let's take an example: If you want to search a contact in your contact database (Like: name, number etc), then you can use Content Provider for this purpose. You can say Content provider is the pointer in your application which points to a specific database from other application.