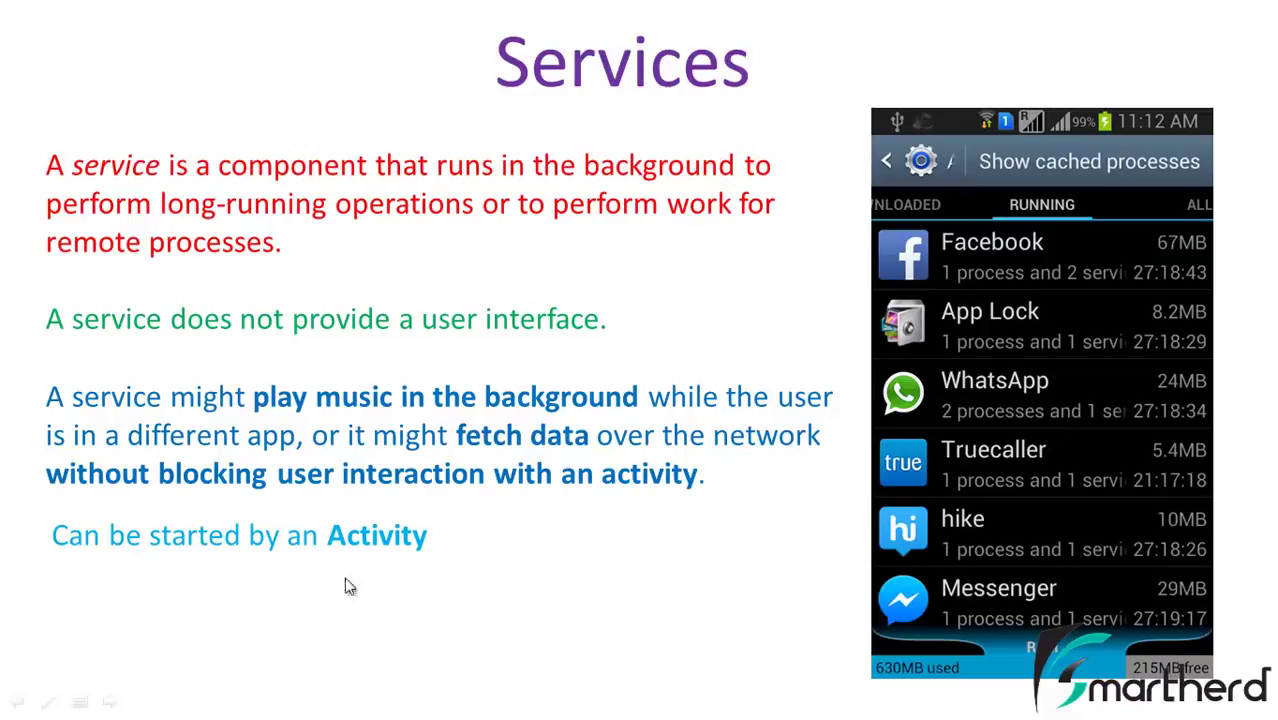
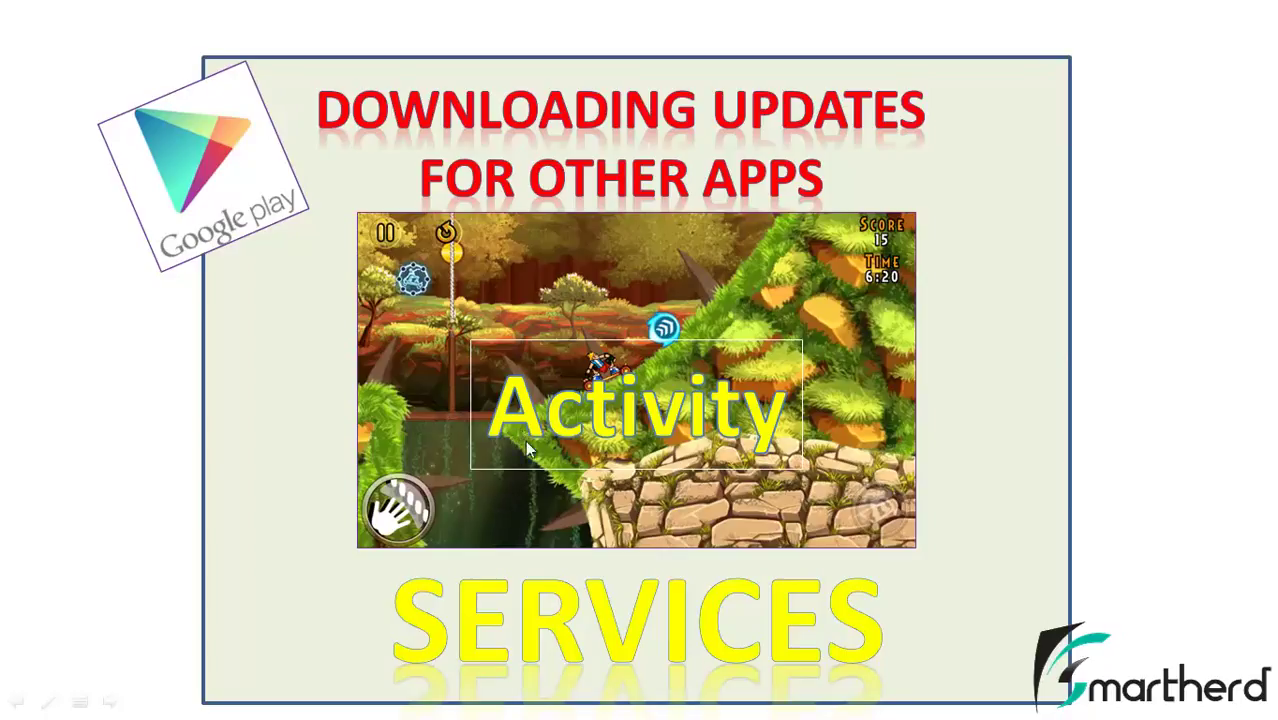
### Que :- What are services?

**Ans:-** A service is a component which runs in the background without direct interaction with the user. Services are used for repetitive and potentially long running operations, i.e., Internet downloads, checking for new data, data processing and the like.

Note:- service is an Android component which should be used only when you don’t have a requirement to display something on user interface. Most of the times, services are used for long running network operations.



Example-



### Que :- What are Types of service?

### Ans:- Platform service and custom services

### Platform service :-

The Android platform provides and runs predefined system services and every Android application can use them, by giving the right permissions. These system services are usually exposed via a specific Manager class. Access to them can be gained via the getSystemService() method. The Context class defines several constants for accessing these services.

Ex:- LocationManager locationManager=(LocationManager)mContext.getSystemService(Context.LOCATION\_SERVICE);

An Android application can, (in addition to consuming the existing Android platform services) define and use custom services. Defining your custom services allows you to design responsive applications.

A service can be:

**Started**

A service is "started" when an application component (such as an activity) starts it by calling [startService()](http://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). Once started, a service can run in the background indefinitely, even if the component that started it, is destroyed. Usually, a started service performs a single operation and does not return a result to the caller. When the operation is done, the service should stop itself.

**Bound**

A service is "bound" when an application component binds to it by calling [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results. A bound service runs only as long as another application component is bound to it.

#### Understanding Started and Bound Service by background music example :-

Suppose, I want to play music in the background, so call startService() method. But I want to get information of the current song is being played, I will bind the service that provides information about the current song.

Example:- service might be playing an audio file and sending data regarding audio start/pause/stop and the time elapsed to the launching Activity component so that the UI can be updated accordingly.

### Services and background processing

**Caution:** A service runs in the main thread of its hosting process—the service does **not** create its own thread and does **not** run in a separate process (unless you specify otherwise). This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work. By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated to user interaction with your activities.

By default, a service runs in the same process as the main thread of the application.

Note:- Any tasks performed in a separate thread from the main thread are essentially performed in the background. Such threads are typically referred to as background or worker threads.

**Note: In order to create separate thread for execution of Service’s task in background, it does not access UI of Activity (Access by UI main thread only).**

### 

### Que :- How to define and start custom services ?

### Ans:- Custom services are started from other Android components, i.e., activities, broadcast receivers and other services.

## Defining custom services

### 1.  Implementation and declaration

A service needs to be declared in the *AndroidManifest.xml* file and the implementing class must extend the Service class or one of its subclasses.

The following code shows an example for a service declaration and its implementation.

**Declaration :-**

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

**Implementation :**-

Traditionally, there is following class we can extend to create a started service:

[Service](http://developer.android.com/reference/android/app/Service.html)

This is the base class for all services. When you extend this class, it's important that you create a new thread in which to do all the service's work, because the service uses your application's main thread, by default, which could slow the performance of any activity in your application is running.

The most important callback methods you should override are:

**Int** [**onStartCommand(Intent i,int flags, int startId)**](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) :-

Called by the system every time a client explicitly starts the service by calling [startService(Intent)](http://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent)). If you implement this, it is your responsibility to stop the service when its work is done, by calling [stopSelf()](http://developer.android.com/reference/android/app/Service.html#stopSelf()) or [stopService()](http://developer.android.com/reference/android/content/Context.html#stopService(android.content.Intent)).

Note:-If you only want to provide binding, you don't need to implement this method.

Where

ReturnType- defines..

**Service restart behavior**

In its onStartCommand() method call, the service returns an int to OS which defines its restart behavior in case the service gets terminated by the Android OS. We can use the constants, the most common options are described by the following -

**Restart options:-**

1. Service.START\_STICKY -- Service is restarted if it gets terminated. Intent data passed to the onStartCommand method is null. Used for services which manages their own state and do not depend on the Intent data.
2. Service.START\_NOT\_STICKY-- If the OS kills the Service it won’t recreate , until the client calls explicitly onStartCommand .
3. Service.START\_REDELIVER\_INTENT-- Similar to Service.START\_STICKY but the original Intent is re-delivered to the onStartCommand method

**Parameters-**

Let’s examine the 3 different arguments passed to onStartCommand():

* Intent – Data to be used by the Service for any sort of asynchronous execution. For example you can pass an image URL that must be downloaded.
* Flag – Representing the history of this start request.

Currently either 0, [START\_FLAG\_REDELIVERY](http://developer.android.com/reference/android/app/Service.html#START_FLAG_REDELIVERY), or [START\_FLAG\_RETRY](http://developer.android.com/reference/android/app/Service.html#START_FLAG_RETRY).

* OS passes [START\_FLAG\_REDELIVERY](http://developer.android.com/reference/android/app/Service.html#START_FLAG_REDELIVERY) if the Intent is a re-delivery of a previously delivered intent, because the service had previously returned [START\_REDELIVER\_INTENT](http://developer.android.com/reference/android/app/Service.html#START_REDELIVER_INTENT) but had been killed before calling [stopSelf(int)](http://developer.android.com/reference/android/app/Service.html#stopSelf(int)) for that Intent.

Constant Value: 1 (0x00000001)

* OS passes  [START\_FLAG\_RETRY](http://developer.android.com/reference/android/app/Service.html#START_FLAG_RETRY). if the Intent is a retry because the original attempt never got to or nothing returned from [onStartCommand(Intent, int, int)](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)).

Constant Value: 2 (0x00000002)

* Start ID – A unique ID provided by the OS for this start request. If the process is terminated and then at a later stage restarted then onStartCommand() will be called with the same start ID. It can be used with [stopSelfResult()](http://developer.android.com/reference/android/app/Service.html#stopSelfResult(int)) which is similar to stopService() but this way you can avoid stopping the Service for old start requests that have not been handled by onStartCommand() yet because maybe during that start request the Service terminated and restarted.

|  |  |
| --- | --- |
|  |  |
|  | A unique integer representing this specific request to start. Use with [stopSelfResult(int)](http://developer.android.com/reference/android/app/Service.html#stopSelfResult(int)) to stop service self which was started by OS with startId.It may be 1 when service started first time, 2-for second time and so on and forth . |

**IBinder** [**onBind(intent i)**](http://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) **:-**

The system calls this method when another component wants to bind with the service by calling [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). In your implementation of this method, you must provide an interface that clients use to communicate with the service, by returning an [IBinder](http://developer.android.com/reference/android/os/IBinder.html) object. You must always implement this method, but if you don't want to allow binding, then you should return null.

Other lifecycle methods of Service class:-

**Void** [**onCreate()**](http://developer.android.com/reference/android/app/Service.html#onCreate())**:-**

The system calls this method when the service is first created, to perform one-time setup background operation (before it calls either [onStartCommand()](http://developer.android.com/reference/android/app/Service.html#onStartCommand(android.content.Intent, int, int)) or [onBind()](http://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent))). If the service is already running, this method is not called.

**Void** [**onDestroy()**](http://developer.android.com/reference/android/app/Service.html#onDestroy()) **:-**

The system calls this method when the service is no longer used and is being destroyed.

**boolean onUnbind(Intent i):-**

Called when all clients have disconnected from a particular interface published by the service.

**Void onRebind(Intent i):-**

Called when new clients have connected to the service, after it had previously been notified that all had disconnected in its [onUnbind(Intent)](http://developer.android.com/reference/android/app/Service.html#onUnbind(android.content.Intent)).



### Extending the Service class:-

**public** **class** MyService **extends** Service {

*@Override*

**public** **int** onStartCommand(Intent intent, **int** flags, **int** startId) {

//TODO do something useful

**return** Service.START\_NOT\_STICKY;

}

*@Override*

**public** IBinder onBind(Intent intent) {

//TODO for communication return IBinder implementation

**return** null;

}

}

### How to Start a service ?

An Android component (service, receiver, activity) can trigger the execution of a service via the startService(intent) method.

// use this to start and trigger a service

Intent i= **new** Intent(context, MyService.**class**);

// potentially add data to the intent

i.putExtra("KEY1", "Value to be used by the service");

context.startService(i);

**How to Stop a service?**

1. You stop a service via the stopService() method. No matter how frequently you called the startService(intent) method, one call to the stopService() method stops the service.
2. A service can terminate itself by calling the stopSelf() method. This is typically done if the service finishes its work.

## 

## Binding services:-

It is a mechanism through which any component (such as Activity) binds to any Bound Service.

**What is** **Bound Service ?**

A bound service is sub class of the [Service](http://developer.android.com/reference/android/app/Service.html) class, whose object allows other applications to bind to it and interact with it.

## Creating a Bound Service

When creating a service that provides binding, you must provide an [IBinder](http://developer.android.com/reference/android/os/IBinder.html) that provides the programming interface(object) that clients can use to interact with the service. There are three ways you can define the interface, and the first way is:

[Extending the Binder class](http://developer.android.com/guide/components/bound-services.html#Binder)

This is the preferred technique when your service is merely a background worker for your own application.

Application components (clients) can bind to a service by calling [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). The Android system then calls the service's [onBind()](http://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) method, which returns an [IBinder](http://developer.android.com/reference/android/os/IBinder.html) for interacting with the service.

The binding is asynchronous. [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)) returns immediately and does not return the [IBinder](http://developer.android.com/reference/android/os/IBinder.html) to the client. To receive the [IBinder](http://developer.android.com/reference/android/os/IBinder.html), the client must create an instance of [ServiceConnection](http://developer.android.com/reference/android/content/ServiceConnection.html) and pass it to [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)). The [ServiceConnection](http://developer.android.com/reference/android/content/ServiceConnection.html) includes a callback method that the system calls to deliver the [IBinder](http://developer.android.com/reference/android/os/IBinder.html).

**Note:** Only activities, services, and content providers can bind to a service—you **cannot** bind to a service from a broadcast receiver.

So, to bind to a service from your client, you must:

1. Implement [ServiceConnection](http://developer.android.com/reference/android/content/ServiceConnection.html) interface.

Your implementation must override two callback methods:

[onServiceConnected()](http://developer.android.com/reference/android/content/ServiceConnection.html#onServiceConnected(android.content.ComponentName, android.os.IBinder))

The system calls this to deliver the [IBinder](http://developer.android.com/reference/android/os/IBinder.html) returned by the service's [onBind()](http://developer.android.com/reference/android/app/Service.html#onBind(android.content.Intent)) method.

[onServiceDisconnected()](http://developer.android.com/reference/android/content/ServiceConnection.html#onServiceDisconnected(android.content.ComponentName))

The Android system calls this when the connection to the service is unexpectedly lost, such as when the service has crashed or has been killed. This is not called when the client unbinds.

1. Call [bindService()](http://developer.android.com/reference/android/content/Context.html#bindService(android.content.Intent, android.content.ServiceConnection, int)), passing the [ServiceConnection](http://developer.android.com/reference/android/content/ServiceConnection.html) implementation.
2. When the system calls your [onServiceConnected()](http://developer.android.com/reference/android/content/ServiceConnection.html#onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback method, OS passes an IBinder object as an argument of [onServiceConnected()](http://developer.android.com/reference/android/content/ServiceConnection.html#onServiceConnected(android.content.ComponentName, android.os.IBinder)).
3. To disconnect from the service, call [unbindService()](http://developer.android.com/reference/android/content/Context.html#unbindService(android.content.ServiceConnection)).

#### public abstract void onServiceConnected ([ComponentName](http://developer.android.com/reference/android/content/ComponentName.html) name, [IBinder](http://developer.android.com/reference/android/os/IBinder.html) service)

Called when a connection to the Service has been established, with the [IBinder](http://developer.android.com/reference/android/os/IBinder.html) of the communication channel to the Service.

##### Parameters

|  |  |
| --- | --- |
| *name* | The concrete component name of the service that has been connected. |
| *service* | The IBinder of the Service's communication channel, which you can now make calls on. |

**public** **class** LocalService **extends** Service {

MediaPlayer myPlayer;

@Override

**public** **void** onCreate() {

// **TODO** Auto-generated method stub

**super**.onCreate();

Toast.*makeText*(**this**,"onCreate() executed...", Toast.*LENGTH\_SHORT*).show();

myPlayer = MediaPlayer.*create*(**this**, R.raw.*guzarish*);

myPlayer.setLooping(**false**); // Set looping

}

@Override

**public** **void** onDestroy() {

Toast.*makeText*(**this**, "Service Stopped", Toast.*LENGTH\_LONG*).show();

myPlayer.stop();

}

IBinder i=**new** LocalBinder();

@Override

**public** IBinder onBind(Intent arg0) {

Toast.*makeText*(LocalService.**this**, "Service binds", Toast.*LENGTH\_LONG*).show();

Runnable r=**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

// Toast.makeText(MyService.this, "Service Started", Toast.LENGTH\_LONG).show();

myPlayer.start();

}

};

**new** Thread(r).start();

**return** i;

}

**public** **class** LocalBinder **extends** Binder

{

LocalService getService()

{

**return** LocalService.**this**;

}

}

**public** String getInfo()

{

**int** d=myPlayer.getDuration();

**return** String.*valueOf*(d);

}

}

The LocalBinder provides the getService() method for clients to retrieve the current instance of LocalService. This allows clients to call public methods in the service. For example, clients can call getInfo() from the service.

Here's an activity that binds to LocalService and calls getInfo() when a button is clicked:

**public** **class** MainActivity **extends** Activity {

LocalService mService;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

}

**public** **void** bindService(View v)

{

Intent intent=**new** Intent(**this**,LocalService.**class**);

bindService(intent,sc,Context.*BIND\_AUTO\_CREATE*);

}

**public** **void** unBindService(View v)

{

unbindService(sc);

}

**public** **void** getInfo(View v)

{

String d=mService.getInfo();

Toast.*makeText*(**this**,"Duration of Song= "+d, Toast.*LENGTH\_SHORT*).show();

}

ServiceConnection sc=**new** ServiceConnection() {

@Override

**public** **void** onServiceDisconnected(ComponentName arg0) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** onServiceConnected(ComponentName arg0, IBinder ib) {

// **TODO** Auto-generated method stub

LocalBinder lb=(LocalBinder) ib;

mService=lb.getService();

}

};

}

The above sample shows how the client binds to the service using an implementation of [ServiceConnection](http://developer.android.com/reference/android/content/ServiceConnection.html) and the [onServiceConnected()](http://developer.android.com/reference/android/content/ServiceConnection.html#onServiceConnected(android.content.ComponentName, android.os.IBinder)) callback.

**CAUTION-**

To ensure your app is secure, always use an explicit intent when starting a Service and do not declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain what service will respond to the intent, and user cannot see which service starts.

NOTE- Beginning with Android 5.0(API level 21), the System throws an exception if we call bindService() with an implicit intent.