Unit - 5

Service and User Alerts

Service

- Service is a component which keep an app running in the background to perform long-running operations based on our requirements.
- For Service, we don’t have any user interface and it will run the apps in the background like playing the music in the background or handle network operations when the user in a different app.

Android Service Lifecycle

- In android, the life cycle of service will follow two different paths Started or Bound.

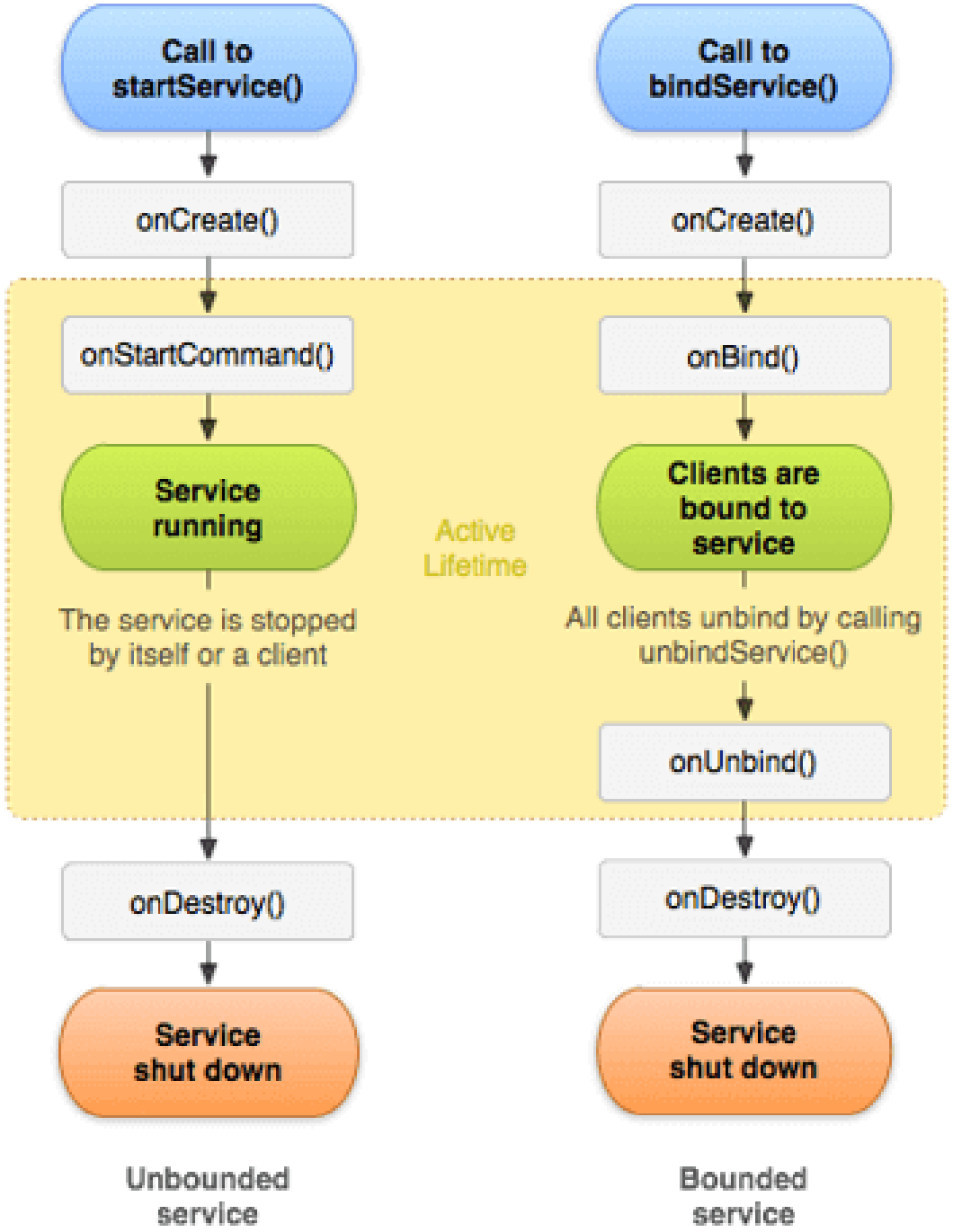
Started Service

- A service is Started when an application component, such as an activity calls startService() method. Once it started, it will run indefinitely in background even if the component that started is destroyed.
- We can stop the Started service by using stopService() method or the service can stop itself by calling stopSelf() method. In android, the Started service component will perform a single operation and it won’t return any result to the caller.

Android Service Lifecycle

Bound Service

- A service is Bound when another application component calls bindService() method. The bound service runs as long as another application component is bound to it.
- We can unbind the service by calling unbindService() method based on our requirements. In android, we can bind multiple components to a single service at once, but the service will be destroyed in case all the components unbind.


Call to startService()
onCreate()
onStartCommand()
Service running
The service is stopped by itself or a client
onDestroy()
Service shut down
Unbounded service
Active Lifetime
Call to bindService()
onCreate()
onBind()
Clients are bound to service
All clients unbind by calling unbindService()
onUnbind()
onDestroy()
Service shut down
Bounded service

- To create a service, we need to create a class that extends a Service base class or one of its existing subclasses.
- During our service implementation, we must need to override some of the callback methods that handle the key aspects of the service lifecycle and provide the functionality that allows our components to bind to the service.

Android Service Callback Methods

onStartCommand()

- The system invokes this method by calling startService() when another component (such as an activity) requests that the service be started.
- When this method executes, the service is started and can run in the background indefinitely.
- If you implement this, it is your responsibility to stop the service when its work is complete by calling stopSelf() or stopService().
- If you only want to provide binding, you don't need to implement this method.

Android Service Callback Methods

onBind()

- The system invokes this method by calling bindService() when another component wants to bind with the service.
- In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an IBinder.
- You must always implement this method; however, if you don't want to allow binding, you should return null.

Android Service Callback Methods

onCreate()

- The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either onStartCommand() or onBind()).
- If the service is already running, this method is not called.

Android Service Callback Methods

onDestroy()

- The system invokes this method when the service is no longer used and is being destroyed.
- Your service should implement this to clean up any resources such as threads, registered listeners, or receivers.
- This is the last call that the service receives.

SMS Manager

- In android, we can send SMS from our android application in two ways either by using SMSManager API or Intents based on our requirements.
- If we use SMSManager API, it will directly send SMS from our application.
- In case if we use Intent with proper action (ACTION_VIEW), it will invoke a built-in SMS app to send SMS from our application.

Android Send SMS using SMSManager API

- In android, to send SMS using SMSManager API we need to write the code like as shown below.

```
SmsManager smgr = SmsManager.getDefault();
smgr.sendTextMessage(MobileNumber,null,Message,null,null);
//destinationAddress, scAddress, text, sentIntent, deliveryIntent
```

- SMSManager API required SEND_SMS permission in our android manifest to send SMS.
- Following is the code snippet to set SEND_SMS permissions in manifest file.
- <uses-permission android:name="android.permission.SEND_SMS"/>

Notification Manager

- Notification is a message which is used to alert the users about some events that happening in our app.
- Generally, the android Notifications will be displayed outside of our app's normal UI and alert the users without interrupting their current activities.
- In android, we can alert the users about our app notifications in different forms like a flash the LED or make sounds or display an icon in the status bar, etc.

Create a Notification in Android

- To create a notification, we need to specify the UI content and required actions with a NotificationCompat.Builder object.
- To display an icon, title and detailed text of notification we need to set the following properties in Builder object.
- setSmallIcon() - It is used to set the small icon for our notification.
- setContentTitle() - It is used to set the title of our notification.
- setContentText() - It is used to set the detailed text to display in notification.

- NotificationCompat.Builder nBuilder

  = new NotificationCompat.Builder(this)

  .setSmallIcon(R.drawable.notification_icon)

  .setContentTitle("Sample notification")

  .setContentText("Hi, Welcome to UTU");

Define the Android Notification Actions

- If we assign an action to the notification, it will allow users to go directly from the notification to an activity of our app.
- We can also add buttons to the notification to perform additional actions such as hang up the call or responding immediately to a text message; this feature is available as of Android 4.1.
- In android, we can define an action inside of notification by using PendingIntent object which contains an Intent that starts an Activity of our app.

- NotificationCompat.Builder nBuilder

  = new NotificationCompat.Builder(this)

  …...

  Intent resultIntent = new Intent(this, MainActivity.class);

  PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,

  resultIntent, 0);

  nBuilder.setContentIntent(pendingIntent);

- A PendingIntent is a reference to a token maintained by the system. Application A can pass a PendingIntent to application B in order to allow application B to execute predefined actions on behalf of application A; regardless of whether application A is still alive.

Issue the Android Notification

- Once we are done with creation of notification, we need to pass a notification to the system by using NotificationManager.notify() method and we need to specify a ID in the notification to use this ID to update a notification later if required

- NotificationCompat.Builder nBuilder
  = new NotificationCompat.Builder(this);
  ....
  int mNotificationId = 999;
  NotificationManager mNotifyMgr =
  (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
  // Builds the notification and issues it.
  mNotifyMgr.notify(mNotificationId, nBuilder.build());

AlarmManager

- It helps us schedule an alarm for a particular time in a particular time period.