Android Services

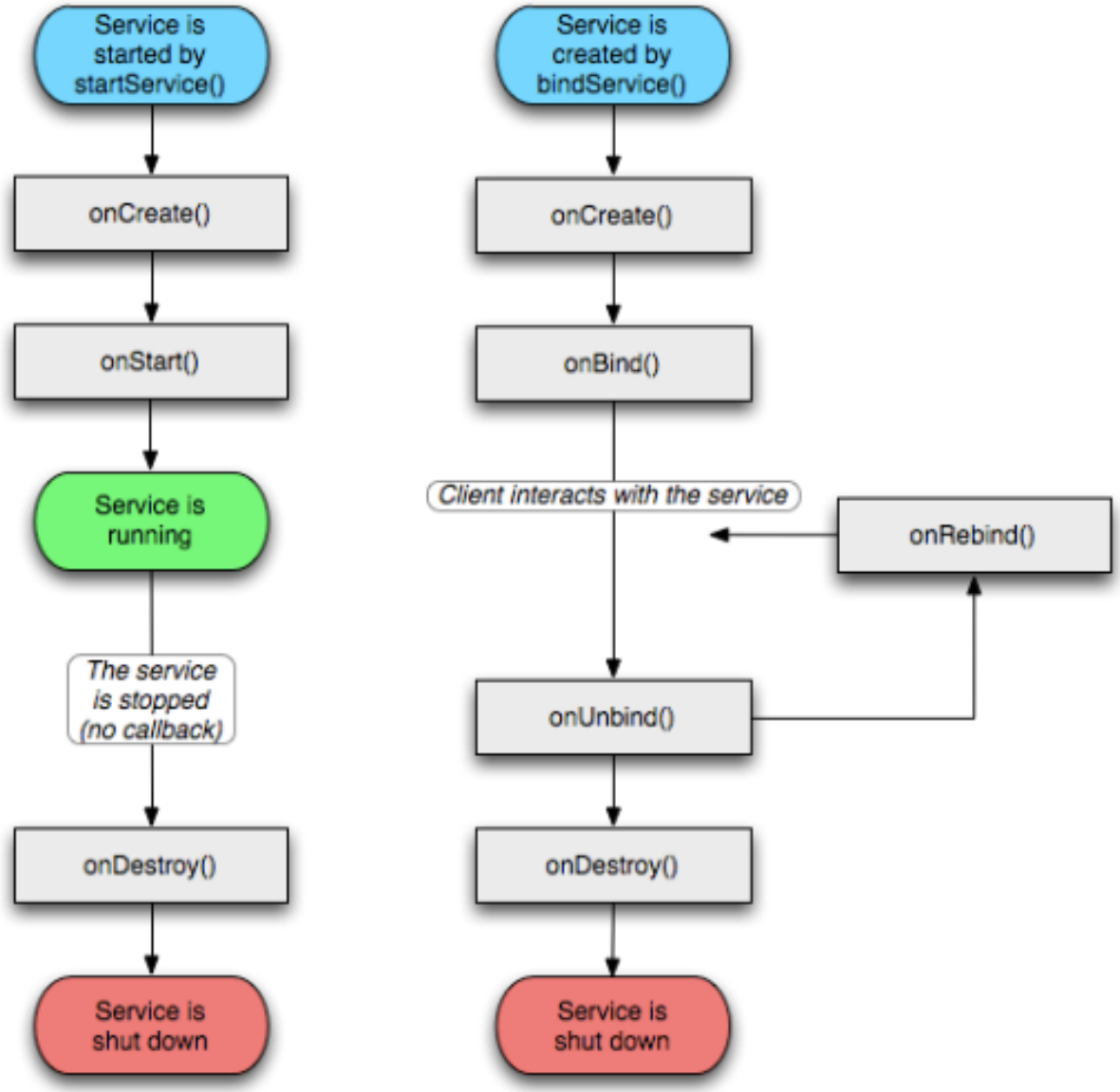
Services

- A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use.
- Services run without a dedicated GUI, but, like Activities and Broadcast Receivers, they still execute in the main thread of the application’s process.
- A Service could be, facility for an application to expose some of its functionality to other applications.
- Started Services receive higher priority than inactive or invisible Activities.
- Android implements several Services including the Location Manager, Media Controller, and the Notification Manager.

What a Service is not?

- A Service is **not** a separate process.
  The Service object itself does not imply it is running in its own process; unless otherwise specified, it runs in the same process as the application it is part of.
- A Service is **not** a thread.
  It is not a means itself to do work off of the main thread (to avoid Application Not Responding errors).

Service Lifecycle



Service Lifecycle

- Clients can also use Context.bindService() to obtain a persistent connection to a service.
- This likewise creates the service if it is not already running (calling onCreate() while doing so), but does not call onStartCommand().
- The client will receive the IBinder object that the service returns from its onBind(Intent) method, allowing the client to then make calls back to the service.

Permissions

- Global access to a service can be enforced when it is declared in its manifest's <service> tag.
- By doing so, other applications will need to declare a corresponding <uses-permission> element in their own manifest to be able to start, stop, or bind to the service.

Creating Service

- To defi ne a Service, create a new class that extends the Service base class.
- Override onBind and onCreate methods.

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
public class MyService extends Service {
        @Override
        public void onCreate() {
        // TODO: Actions to perform when service is created.
        }
        @Override
        public IBinder onBind(Intent intent) {
        // TODO: Replace with service binding implementation.
        return null;
        }
}
```

Creating Service

- We may also override onStart which is called whenever the Service is started with a call to startService.
- Once constructed , a new Service should be registered in the application manifest.

Starting, Controlling, and Interacting with a Service

- To start a Service, call startService; you can either implicitly specify a Service to start using an action against which the Service is registered, or you can explicitly specify the Service using its class.

```
// Implicitly start a Service
startService(new Intent(MyService.MY_ACTION));
// Explicitly start a Service
startService(new Intent(this, MyService.class));
```

Stopping Service

- To stop a Service, use stopService, passing an Intent that defi nes the Service to stop.
- If startService is called on a Service that's already running, the Service's onStart method will be executed again.
- Calls to startService do not nest, so a single call to stopService will terminate it no matter how many times startService has been called.

Stopping Service

```
ComponentName service = startService(new Intent(this,
    BaseballWatch.class));
// Stop a service using the service name.
stopService(new Intent(this, service.getClass()));
```

```
// Stop a service explicitly.
try {
Class serviceClass = Class.forName(service.getClassName());
stopService(new Intent(this, serviceClass));
} catch (ClassNotFoundException
```

Public methods of *android.app.service*

- Application getApplication();
- abstract IBinder onBind(Intent intent);
- void onConfigurationChanged(Configuration newConfig);
- void onCreate();
- void onDestroy();
- void onLowMemory();
- void onRebind(Intent intent);
- void onStart(Intent intent, int startId);
- boolean onUnbind(Intent intent);

Public methods of *android.app.service*

- final void setForeground(boolean isForeground);
- final void stopSelf();

Services in Android

- Android has two types of Services
  - Local services - e.g. Email app
    - Local services are services that are called only by the application that hosts them.
  - Remote services – e.g. Router app
    - Remote services are services that support a Remote Procedure Call (RPC) mechanism.

Local Services

- Local services are services that are started via Context.startService().
- Once started, these types of services will continue to run until a client calls Context.stopService() on the service or the service itself calls stopSelf().

AIDL Services

- Android Interface Definition Language is used as an Interface Definition Language (IDL) to define the interface that will be exposed to clients.
- To build a remote service:
  - Write an AIDL file that defines your interface to clients. The AIDL file uses Java syntax and has an .aidl extension. Use the same package name as the package for your Android Project.
  - Add the AIDL file to your Eclipse project under the src directory. The Android Eclipse plug-in will call the AIDL compiler to generate a Java interface from the AIDL file.
  - Implement a service and return the interface from the onBind() method.
  - Add the service configuration to your AndroidManifest.xml file.

Important points regarding the generated classes:

- The interface we defined in the AIDL file is implemented as an interface in the generated code.
- A static final abstract class named Stub extends android.os.Binder and implements your interface. This class is an abstract class.
- An inner class named Proxy implements the interface that proxies the Stub class.
- The AIDL file must reside in the package where the generated files are supposed to be.

- To implement the service's interface, we need to write a class that extends android.app.Service and implements the IStockQuoteService interface.
- To expose the service to clients, our Class will need to provide an implementation of the onBind() method, and we'll need to add some configuration information to the AndroidManifest.xml file.

Calling the Service from a Client Application

- When a client talks to a service, there must be a protocol or contract between the two. With Android, the contract is AIDL.
- So the first step in consuming a service is to take the service's AIDL file and copy it to your client project.
- When you copy the AIDL file to the client project, the AIDL compiler creates the same interface-definition file that was created when the service was implemented (in the service-implementation project).
- This exposes to the client all of the methods, parameters, and return types on the service.