Univérsité Mohamed V
***Faculté des sciences***
Département d’informatique

***2017/2018***

Module:
**Mobile and cloud computing**

# Services in Android

***Starring***

# IntentService

***Featurring***

# Intent

***Pr. Oussama REDA***

# Services

# in

# Android

*Starring*

**IntentService**

*Featuring*

**Intent**

- A Service is an application component that **runs in the background** for an **indefinite** period of time, **not interacting with** the **user.**

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- A Service is **not** a separate process. Unless otherwise specified, it **runs in the same process as the application it is part of.**

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- 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).

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- A Service is **not** a separate process. The Service object itself does not imply it is running in its own process; <u>unless otherwise specified</u>, 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).

If a **service** *and* the respective **activity** are ***run on the same thread*** ***(Default behaviour)***, then the <u>**activity will become unresponsive**</u> when the service is being executed

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- 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).

⬇

If a **service** *and* the respective **activity** are ***run on the same thread*** ***(Default behaviour)***, then the **activity will become unresponsive** when the service is being executed

If so, then why bother working with services ?

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- A Service is **not** a separate process. The Service object itself does not imply it is running in its own process; <u>unless otherwise specified</u>, 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).

If a **service** *and* the respective **activity** are ***run on the same thread*** ***(Default behaviour)***, then the **<u>activity will become unresponsive</u>** when the service is being executed

If so, then why bother working with services ?

Because

- A Service is an application component that runs in the background for an **indefinite** period of time, not interacting with the user.
- 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).

If a **service** *and* the respective **activity** are ***run on the same thread*** ***(Default behaviour)***, then the **activity will become unresponsive** when the service is being executed

If so, then why bother working with services ?

Because

- Services have higher priority than inactive Activities, so less likely to be killed
- If killed, they can be configured to re-run automatically (when resources available)

## Android: Services

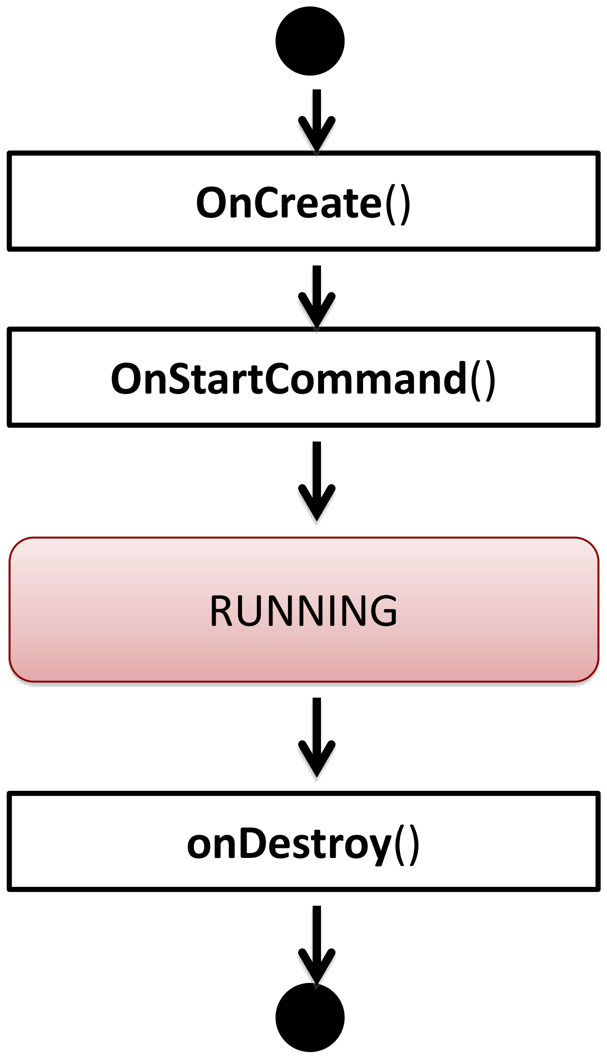
A **Service** is an application that can perform *long-running operations in background* and *does not provide a user interface*.

- **Activity** → UI, can be disposed when it loses visibility

- **Service** → No UI, disposed when it terminates or when it is terminated by other components

A Service provides a robust environment for background tasks …

# Android: Service Lifetime

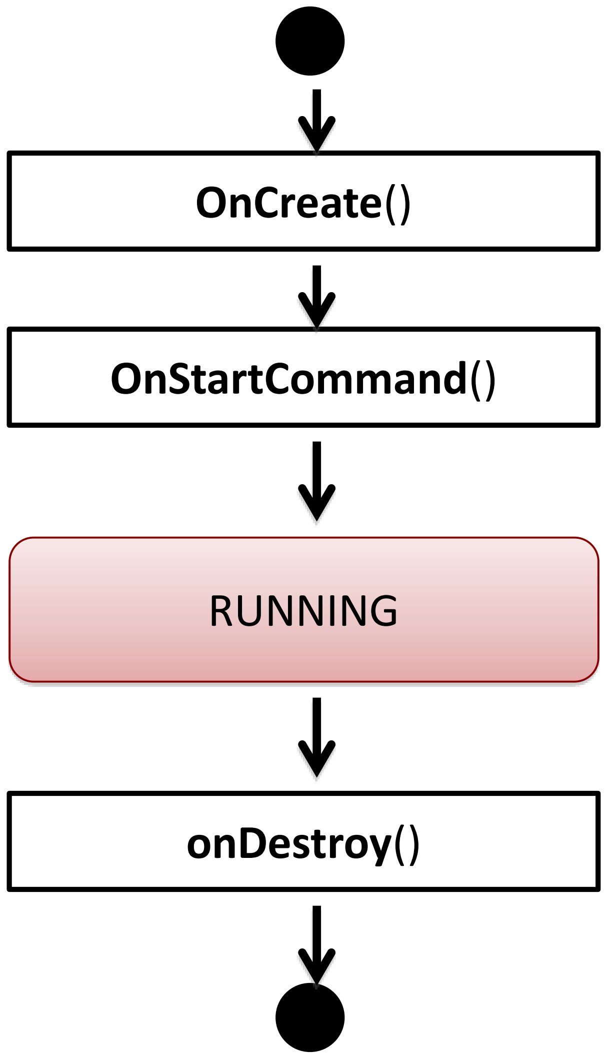


Two Types of **Services**:

1. **Local** Services: Start-stop lifecycle as the one shown.

2. **Remote/Bound** Services: Bound to application components. Allow interactions with them, send requests, get results, IPC facilities.

# Android: Service Lifetime



Two Types of **Services**:

1. **Local** Services: Start-stop lifecycle as the one shown.

2~~.~~ ~~**Remote/Bound** Services:~~

~~Bound to application components.~~

~~Allow interactions with them, send requests, get results, IPC facilities.~~

- A Service is started when an application component starts it by calling **startService(**Intent**)**.

- Once started, a Service can run in **background**, even if the component that started it is destroyed.

- *Termination* of a Service:

  1. **selfStop**() → self-termination of the service

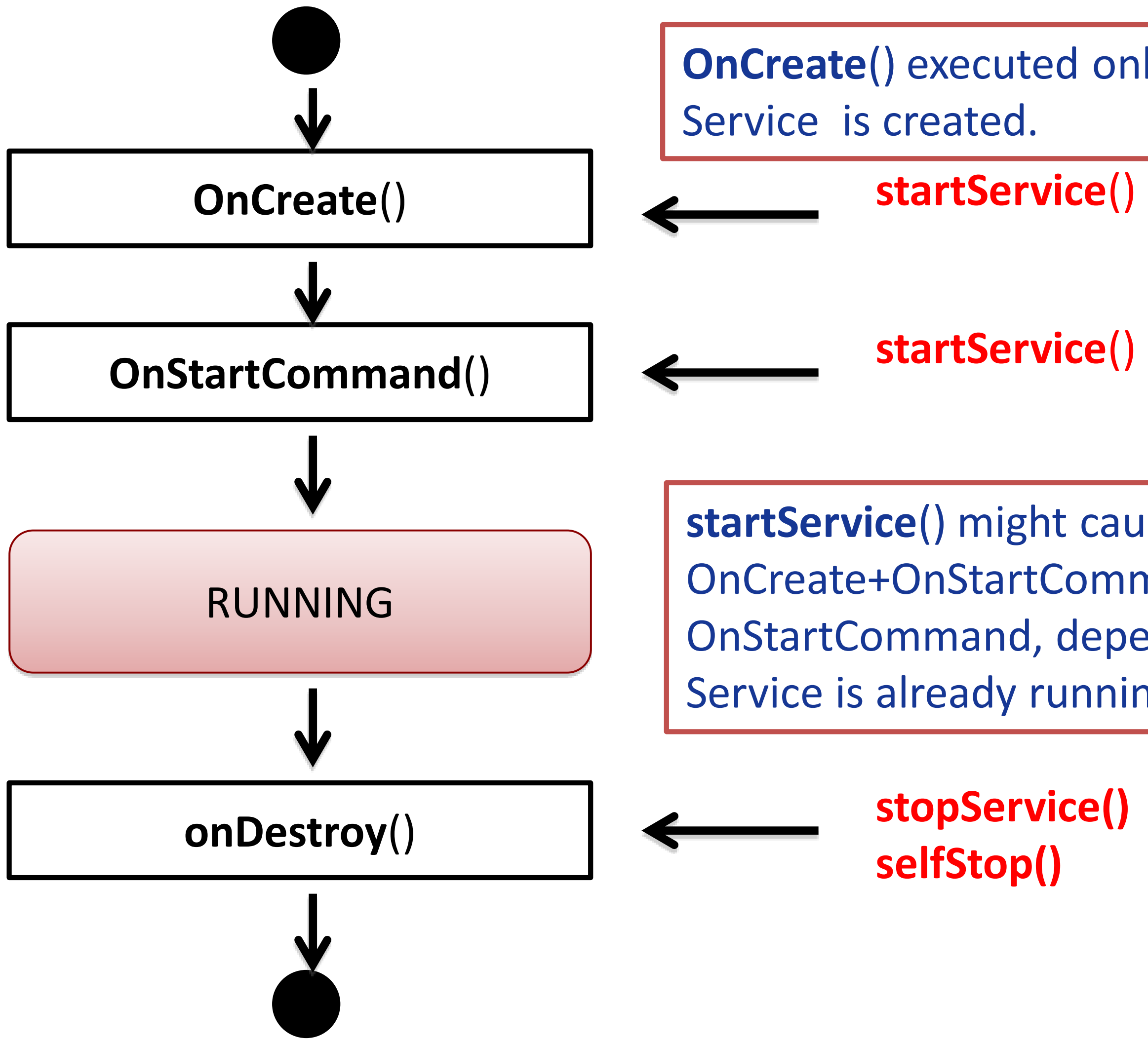
  2. **stopService**(Intent) → terminated by others

  3. System-decided termination (i.e. memory shortage)

- Each service class must have a corresponding  declaration in its package's **AndroidManifest.xml**  under **<application>**

```
<service android:name=”.MyService” />
```

# Android: Service Lifetime



**OnCreate**() executed only once when the Service is created.

**startService**() might cause the execution of OnCreate+OnStartCommand, or only of OnStartCommand, depending whether the Service is already running …

# Android: Services

COMMON **MISTAKES**

- A **Service** provides only a **robust environment** where **to host separate threads** of our application.

  - A Service is not a separate process.
  - A Service is not a separate Thread (i.e. it runs in the main thread of the application that hosts it).
  - A Service does nothing except executing what listed in the **OnCreate**() and **OnStartCommand**() methods.

# Services

- Services can be started with **Context.startService()** in the main thread of the application's process.

  - **CPU intensive tasks** must be **offloaded** to **background threads** using Thread or AsyncTask .

- To start a service:

  **startService**(**new** Intent(getBaseContext(), MyService.**class**));

- To stop a service:

  **stopService**(**new** Intent(getBaseContext(), MyService.**class**))

  or

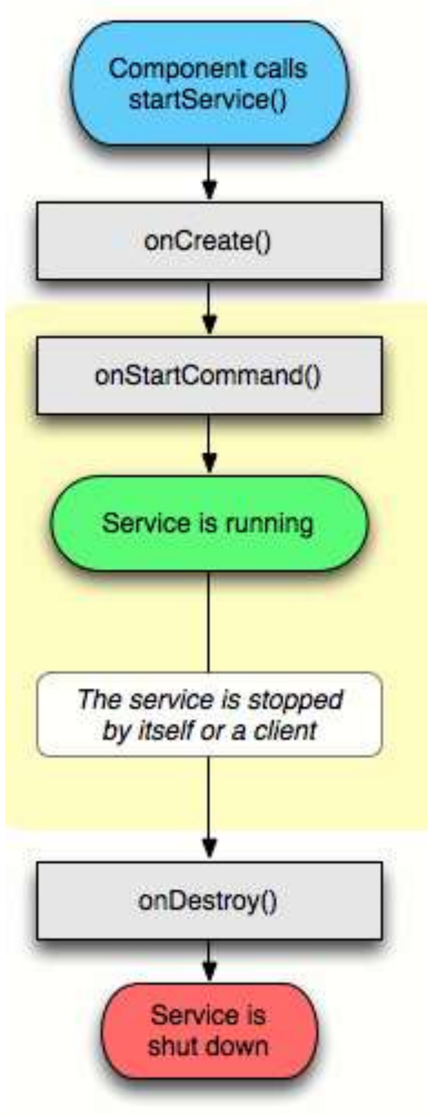
  **stopSelf**()

The written services class should extend Service class and implement three methods

- **public void** ***onCreate***() { ... }
- **public int** ***onStartCommand***(Intent intent, **int** flags, **int** startId) { ... }
- **public void** ***onDestroy***() { ... }

# Services Lifecycle

- A Service has three lifecycle methods:
  - 1. void onCreate()
  - 2. void onStartCommand()
  - 3. void onDestroy()
- onStartCommand() is invoked when a service is explicitly started using startService() method
- onDestroy() is invoked when a service is stopped using stopService() or stopSelf() methods



## onStartCommand

- **Called whenever the Service is started with startService call**
  - May be executed several times in Service's lifetime!
  - Controls how system will respond if Service restarted
  - Runs from main GUI thread, so **standard pattern** is to **create a new Thread** from **onStartCommand** to perform processing and stop Service when complete

# Creating a Service

```
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 int onStartCommand(Intent intent, int flags, int startId) {
                // TODO Launch a background thread to do processing.
        return  Service.START_STICKY;  //  }
      //   (START_STICKY) means the service will run indefinitely until explicitly stopped

       @Override
        public void onDestroy () {
                // TODO: Actions to perform when service is ended.
        }
}
```

## Starting a Service

```
startService(myIntent) ;
```

## Starting a Service

```
Intent  myIntent =  new Intent(this, MyService.class);



startService(myIntent) ;
```

## Starting a Service

```
Intent  myIntent =  new Intent(this, MyService.class);

myIntent.putExtra( key, value );

startService(myIntent) ;
```

## Stopping a Service

**Call stopService**

```
stopService(new Intent(this, service.getClass()));

stopService(new Intent(this, service.Class));
```

## Services (Recipe)

- Declare services in manifest
- Service Lifecycle:
  - onCreate, onStartCommand, onDestroy
- Can start services by passing in an intent similar to starting an activity
- Must stop service before starting up another instance
  - Best to start service in onCreate/onResume and stop in onPause

# IntentService

- Subclass Service, then override:
  - onStartCommand() -- called when startService() is called.  Then you can call stopSelf() or stopService()
  - ~~onBind() -- called when bindService() is called.  Returns an IBinder (or null if you don't want to be bound).~~
  - onCreate() -- called before above methods.
  - onDestroy() -- called when about to be shut down.
- There are two classes you can subclass:
  - Service: you need to create a new thread, since it is not created by default.
  - **IntentService**

    **This uses a worker thread to perform the requests, and all you need to do is override**

    **onHandleIntent(){...}**

## Services using IntentService class

- To easily create a service that runs a task asynchronously and terminates itself when it is done, you can use the IntentService class
- The IntentService class is a base class for Service that handles asynchronous requests on demand
- It is started just like a normal service; and it executes its task within a worker thread and terminates itself when the task is completed
- // Create a class that extends IntentService class instead of Service class
- **public class** MyIntentService **extends IntentService** { }
- // create a constructor and call superclass with the name of the intent
  // service as a string
- **public** MyIntentService() { **super**(“MyIntentServiceName”); }
- // onHandleIntent() is executed on a worker thread
- **protected void onHandleIntent(Intent intent) { … }**

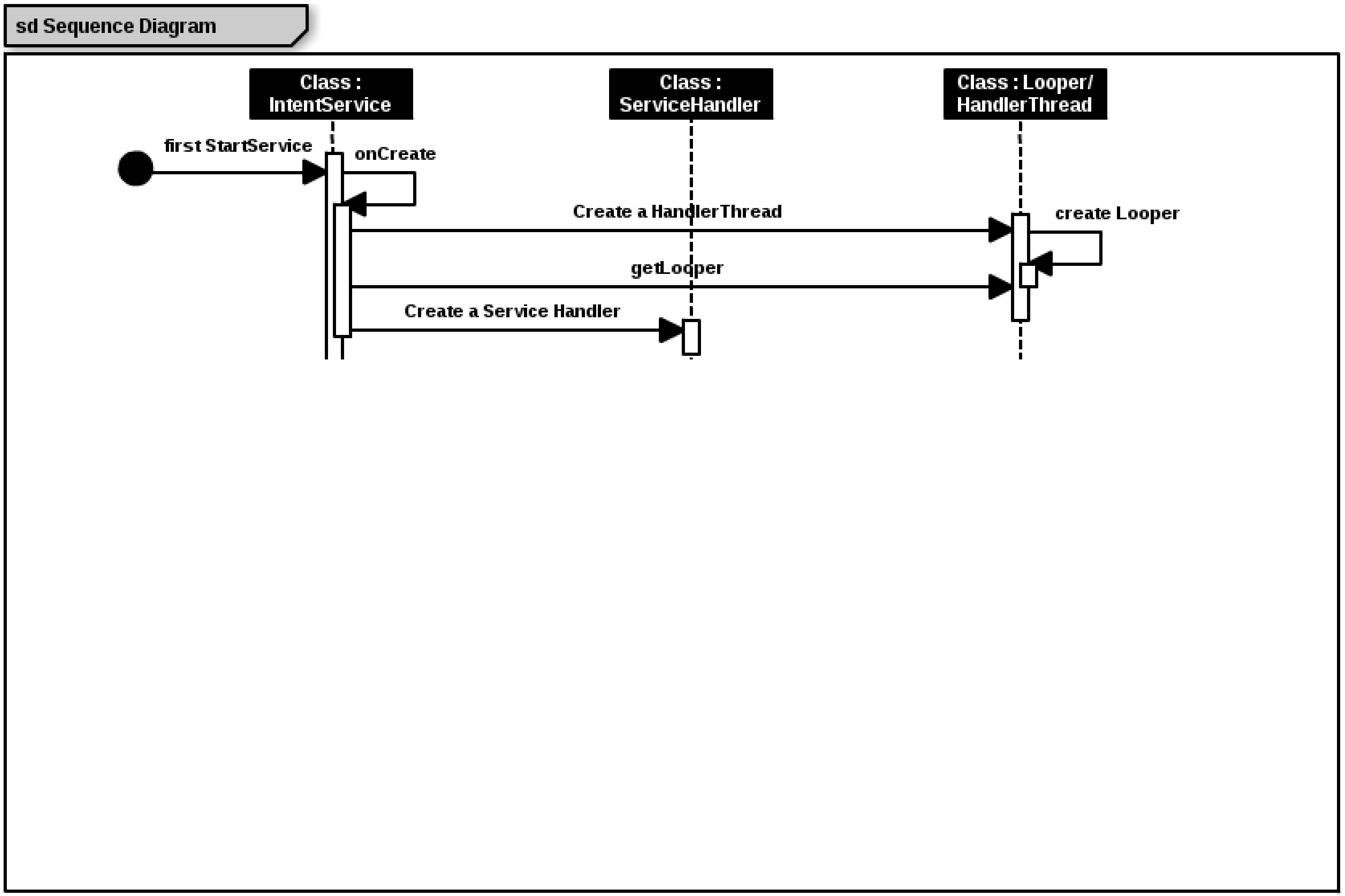
## Services using IntentService class

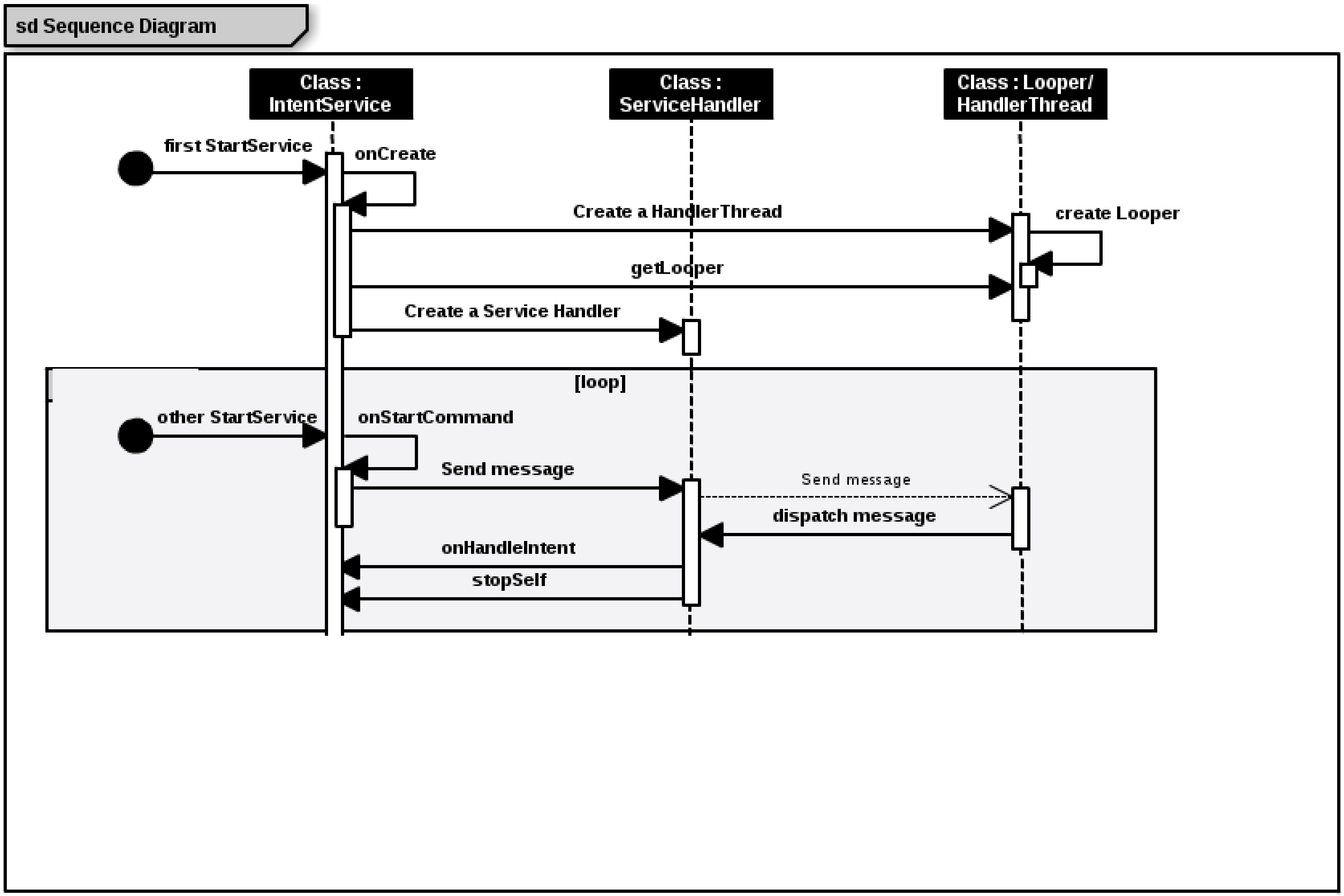
- The IntentService class does the following:
- Creates a default worker thread that executes all intents delivered to onStartCommand() separate from your application's main thread.
- Creates a work queue that passes one intent at a time to your onHandleIntent() implementation, so you never have to worry about multi-threading.
- Stops the service after all start requests have been handled, so you never have to call stopSelf().
- Provides a default implementation of onStartCommand() that sends the intent to the work queue and then to your onHandleIntent() implementation.

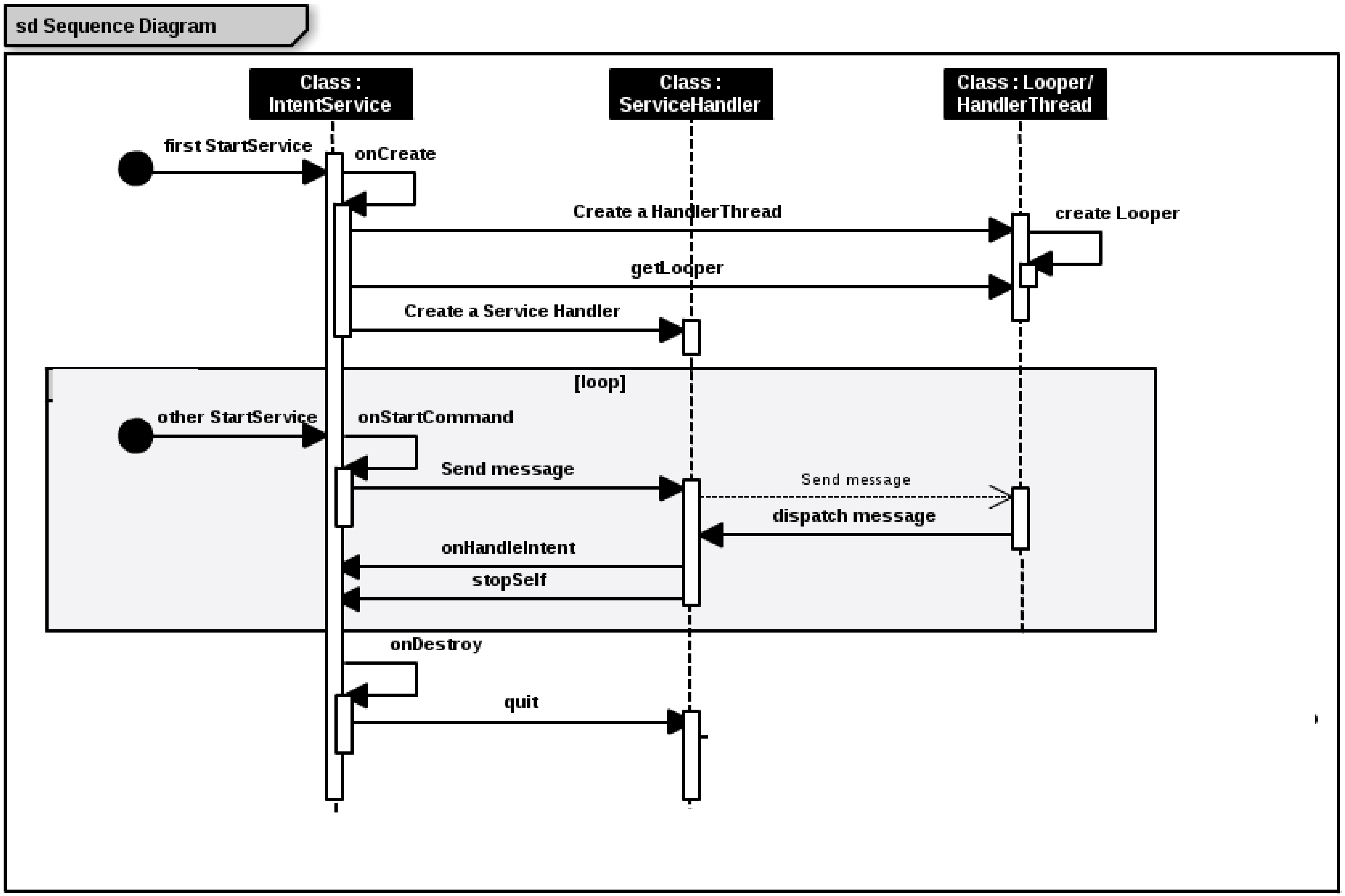
- **All you have to do is handle onHandleIntent().**

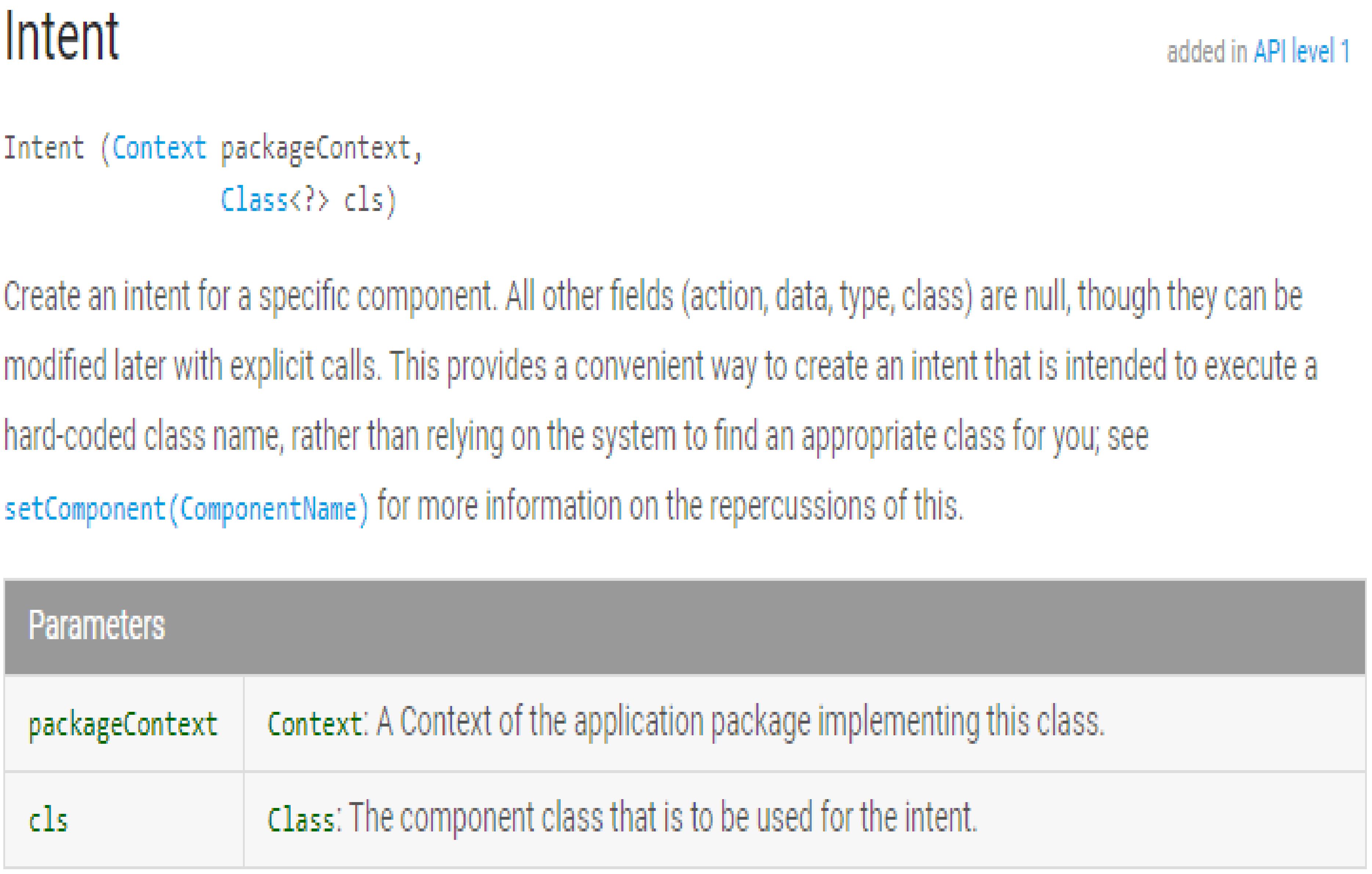
# Services using IntentService class

```
public class HelloIntentService extends IntentService {


 // A constructor is required, and must call the super IntentService(String)
 // constructor with a name for the worker thread.
  public HelloIntentService() {  super("HelloIntentService");  }



  // The IntentService calls this method from the default worker thread with the
 // intent that started the service. When this method returns, IntentService stops
 //the service, as appropriate.
  @Override
  protected void onHandleIntent(Intent intent) {
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
      long endTime = System.currentTimeMillis() + 5*1000;
      Thread.sleep(endTime -System.currentTimeMillis);
   }

}
```


sd Sequence Diagram
Class : IntentService
Class : ServiceHandler
Class : Looper/ HandlerThread
first StartService
onCreate
Create a HandlerThread
create Looper
getLooper
Create a Service Handler


sd Sequence Diagram
Class : IntentService
Class : ServiceHandler
Class : Looper/ HandlerThread
first StartService
onCreate
Create a HandlerThread
create Looper
getLooper
Create a Service Handler
[loop]
other StartService
onStartCommand
Send message
Send message
dispatch message
onHandleIntent
stopSelf


sd Sequence Diagram
Class : IntentService
Class : ServiceHandler
Class : Looper/ HandlerThread
first StartService
onCreate
Create a HandlerThread
create Looper
getLooper
Create a Service Handler
[loop]
other StartService
onStartCommand
Send message
Send message
dispatch message
onHandleIntent
stopSelf
onDestroy
quit



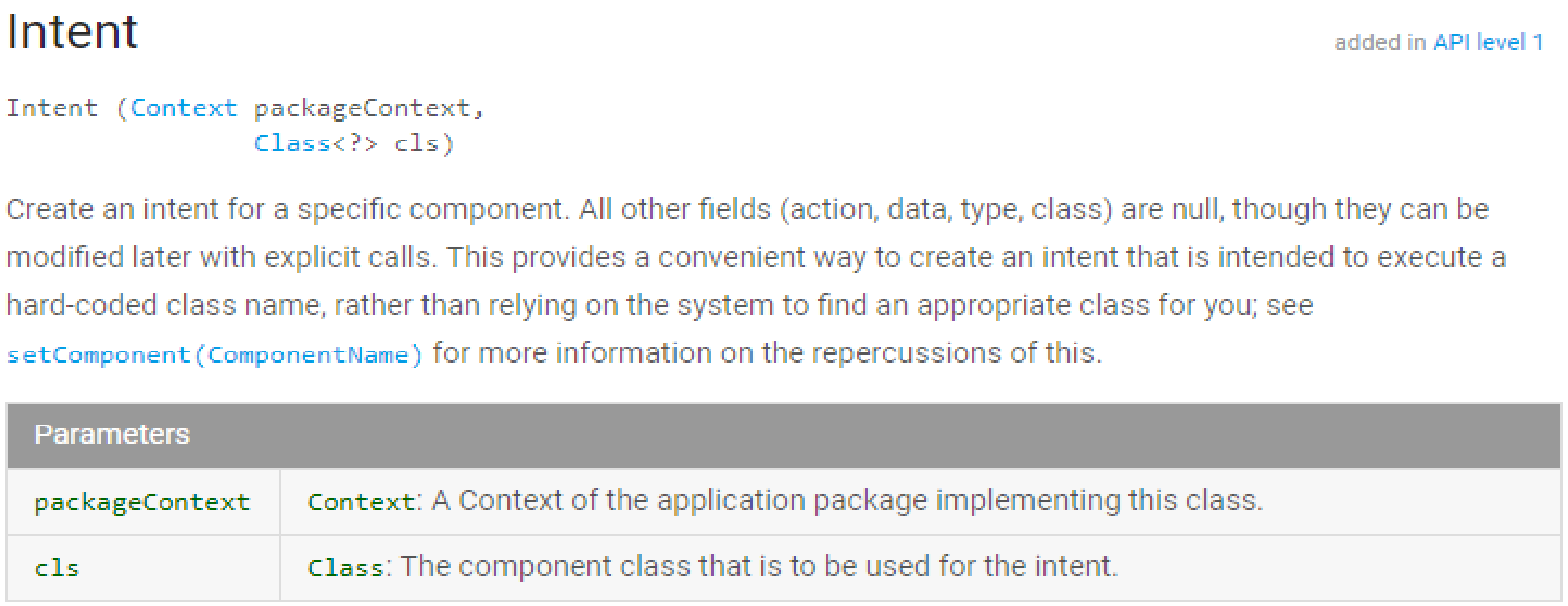
## Intent

added in API level 1

```
Intent (Context packageContext,
                Class<?> cls)
```

Create an intent for a specific component. All other fields (action, data, type, class) are null, though they can be modified later with explicit calls. This provides a convenient way to create an intent that is intended to execute a hard-coded class name, rather than relying on the system to find an appropriate class for you; see `setComponent(ComponentName)` for more information on the repercussions of this.

| Parameters | |
|---|---|
| `packageContext` | `Context`: A Context of the application package implementing this class. |
| `cls` | `Class`: The component class that is to be used for the intent. |



## Intent

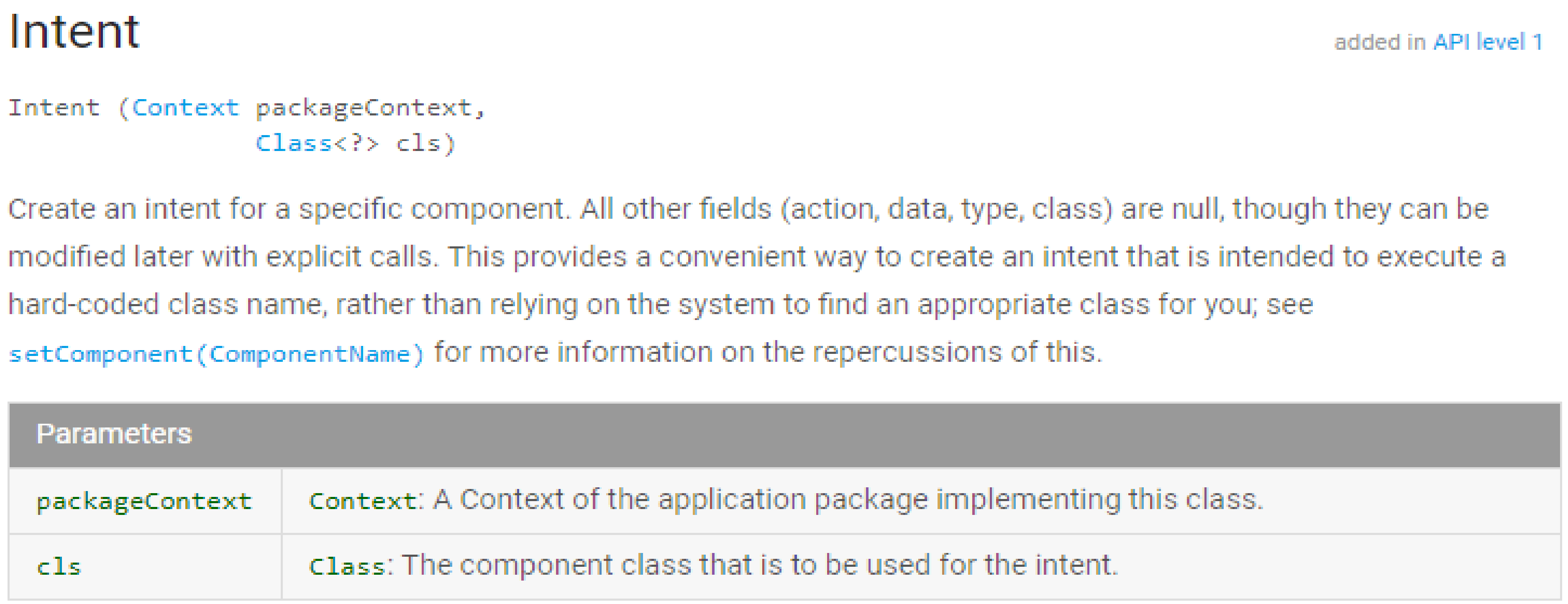
added in API level 1

```
Intent (Context packageContext,
                Class<?> cls)
```

Create an intent for a specific component. All other fields (action, data, type, class) are null, though they can be modified later with explicit calls. This provides a convenient way to create an intent that is intended to execute a hard-coded class name, rather than relying on the system to find an appropriate class for you; see `setComponent(ComponentName)` for more information on the repercussions of this.

| Parameters | |
|---|---|
| `packageContext` | `Context`: A Context of the application package implementing this class. |
| `cls` | `Class`: The component class that is to be used for the intent. |

```
    }

    private void triggerIntentService(int primeToFind) {
        Intent intent = new Intent(?????, ?????);


    }
}
```



## Intent

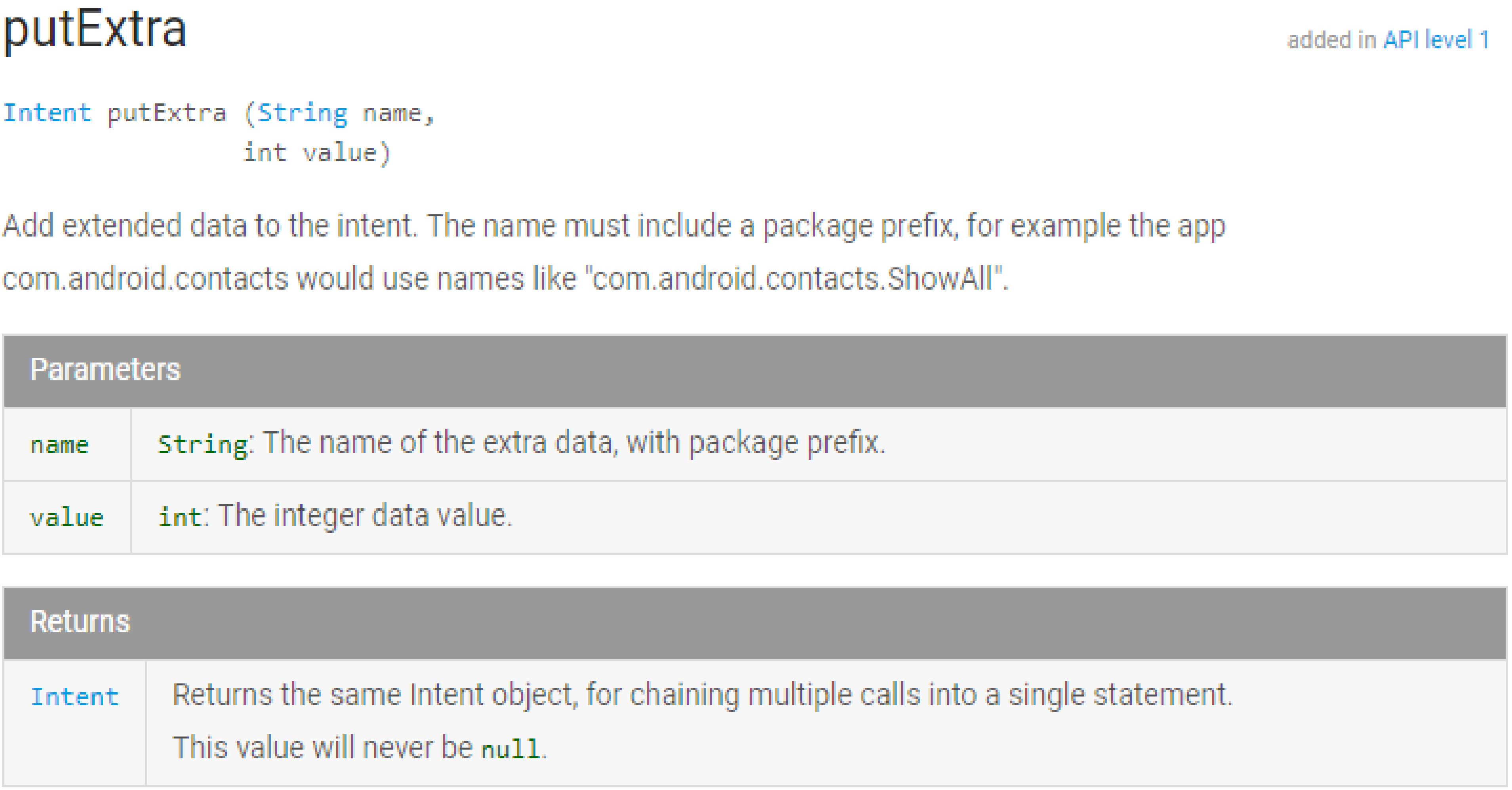
added in API level 1

```
Intent (Context packageContext,
                Class<?> cls)
```

Create an intent for a specific component. All other fields (action, data, type, class) are null, though they can be modified later with explicit calls. This provides a convenient way to create an intent that is intended to execute a hard-coded class name, rather than relying on the system to find an appropriate class for you; see `setComponent(ComponentName)` for more information on the repercussions of this.

| Parameters | |
|---|---|
| `packageContext` | `Context`: A Context of the application package implementing this class. |
| `cls` | `Class`: The component class that is to be used for the intent. |

```
    }

    private void triggerIntentService(int primeToFind) {
        Intent intent = new Intent(Context , Service component);


    }
}
```



## putExtra

added in API level 1

```
Intent putExtra (String name,
                int value)
```

Add extended data to the intent. The name must include a package prefix, for example the app com.android.contacts would use names like "com.android.contacts.ShowAll".

| Parameters | |
|---|---|
| `name` | `String`: The name of the extra data, with package prefix. |
| `value` | `int`: The integer data value. |

| Returns | |
|---|---|
| `Intent` | Returns the same Intent object, for chaining multiple calls into a single statement. This value will never be `null`. |

```
    private void triggerIntentService(int primeToFind) {
        Intent intent = new Intent(Context , Service component););
        intent.putExtra(Name, Parameter to be sent to the service);

    }
}
```

## End of Lecture

Université Mohamed V
***Faculté des sciences***
Département d'informatique

**2017/2018**

Module:
**Mobile and cloud computing**

# Services in Android

*Starring*

**IntentService**

*Featuring*

**Intent**

***Pr. Oussama REDA***