# Android Development

# Conditionals

There's a nicer way to write that series of `if`/`else if`/`else` statements in Kotlin, using the `when` statement, which is like the `switch` statement in other languages.

```
when (numberOfFish) {
    0  -> println("Empty tank")
    in 1..39 -> println("Got fish!")
    else -> println("That's a lot of fish!")
}
```

# Loops

while

```
var number = 10

while (number > 0) {
    number--
}
```

```
do {
    val value = doSomethingSmart()
} while (value != null)
```

# Loops

for

```
for (item in collection){
    print(item)
}
```

```
for (i in 1..15) {
    println(i)
}
```

```
for (i in 16 downTo 0 step 2) {
    println(i)
}
```

# Classes

If the primary constructor does not have any annotations or visibility modifiers, the constructor keyword can be omitted:

```
class Person { /*...*/ }

class Empty


class Person constructor(firstName:String){

}

class Person (firstName:String){

}
```

# Data class

## Hold Data

Kotlin

```kotlin
data class Person(var name :String, var age:Int)
```

Java

```java
public class Person {
    private String name;
    private String age;

    public Person(String name, String age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAge() {
        return age;
    }

    public void setAge(String age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return name.equals(person.name) &&
                age.equals(person.age);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
```

# Objects

```
object MyObject {
    // further implementation
    fun printHello() {
        println("Hello World!")
    }
}
```

```
class MyClass {

    // some implementations

    companion object {
        val SOME_STATIC_VARIABLE = "some_static_variable"
        fun someStaticFunction() {
            // static function implementation
        }
    }
}
```

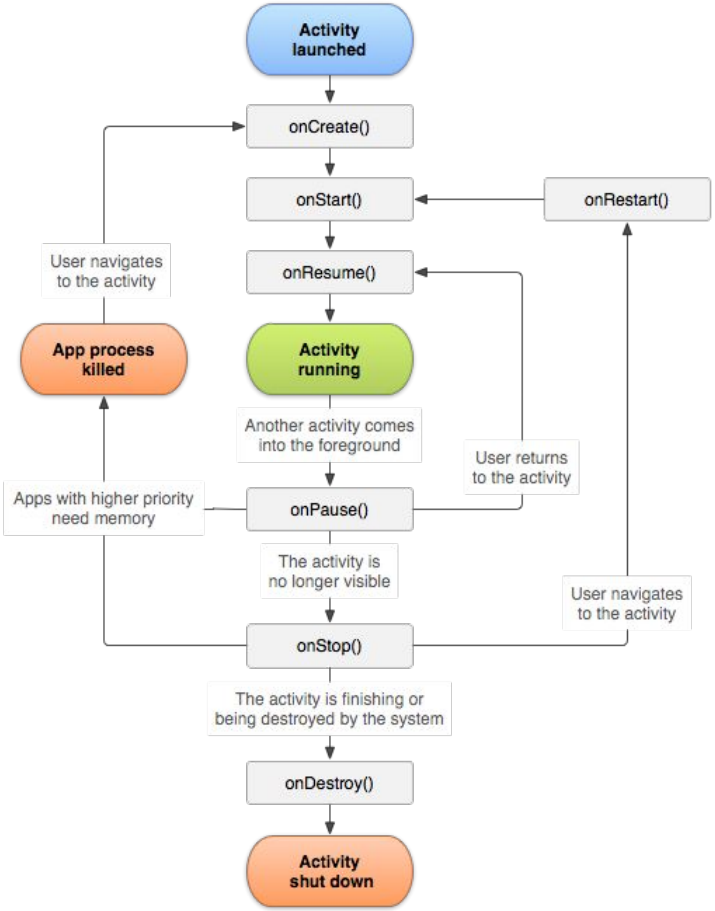
Java

```java
public class Singleton{
    private static Singleton INSTANCE;
    private Singleton(){}
    public static Singleton getInstance(){
        if (INSTANCE == null) {                 // Single Checked
            synchronized (Singleton.class) {
                if (INSTANCE == null) {         // Double checked
                    INSTANCE = new Singleton();
                }
            }
        }
        return INSTANCE;
    }
    private int count = 0;
    public int count(){ return count++; }
}
```

Kotlin

```kotlin
object Singleton {
    private var count: Int = 0

    fun count() {
        count++
    }
}
```

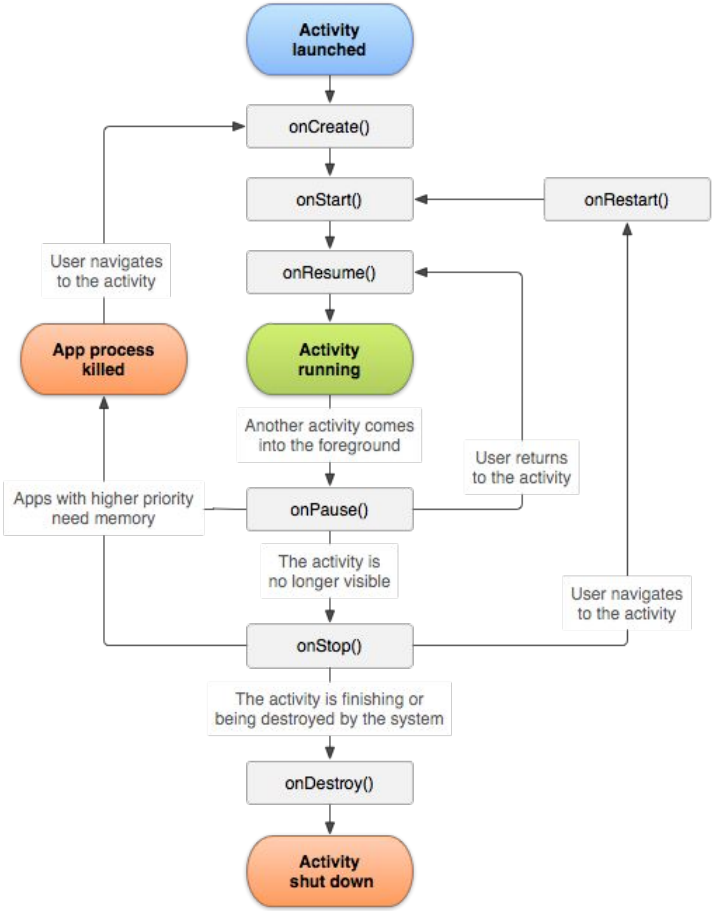
# Activity lifecycle



# Activity lifecycle

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

# Activity lifecycle



# Activity Lifecycle

1. **onCreate():** This is the first method of the activity lifecycle callback which was called when an Activity is created. We render the UI of the activity, initialize the UI components and other resources here.
2. **onStart():** This method gets called when the activity has started after creating. This method is called after **onCreate()** method if the Activity is just created or it can be called after **onRestart()** method if the Activity is restarting again from stopped state.
3. **onResume():** This method gets called when the activity is in the resumed state. This method is called after **onStart()** method if the Activity has started or it can be called when the app is fully visible after paused state means after **onPause()** method. Here the UI of the activity is visible and interactive to the user.

# Activity Lifecycle

1. **onPause():** This method gets called when the **UI** is **partially visible** to the user.The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode)
2. **onStop():** This method gets called when the **UI** is **not visible** to the user. Then the app goes to stopped state.
3. **onDestroy():** This method gets called before the Activity has destroyed.
4. **onRestart():** This method gets called when a stopped **Activity is restarted** from the **stopped state**. So, this method always gets called after the **onStop()** method.

# Intents

- Explicit
- Implicit

# Intents

- **Explicit intents** specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

# Intents

- Explicit

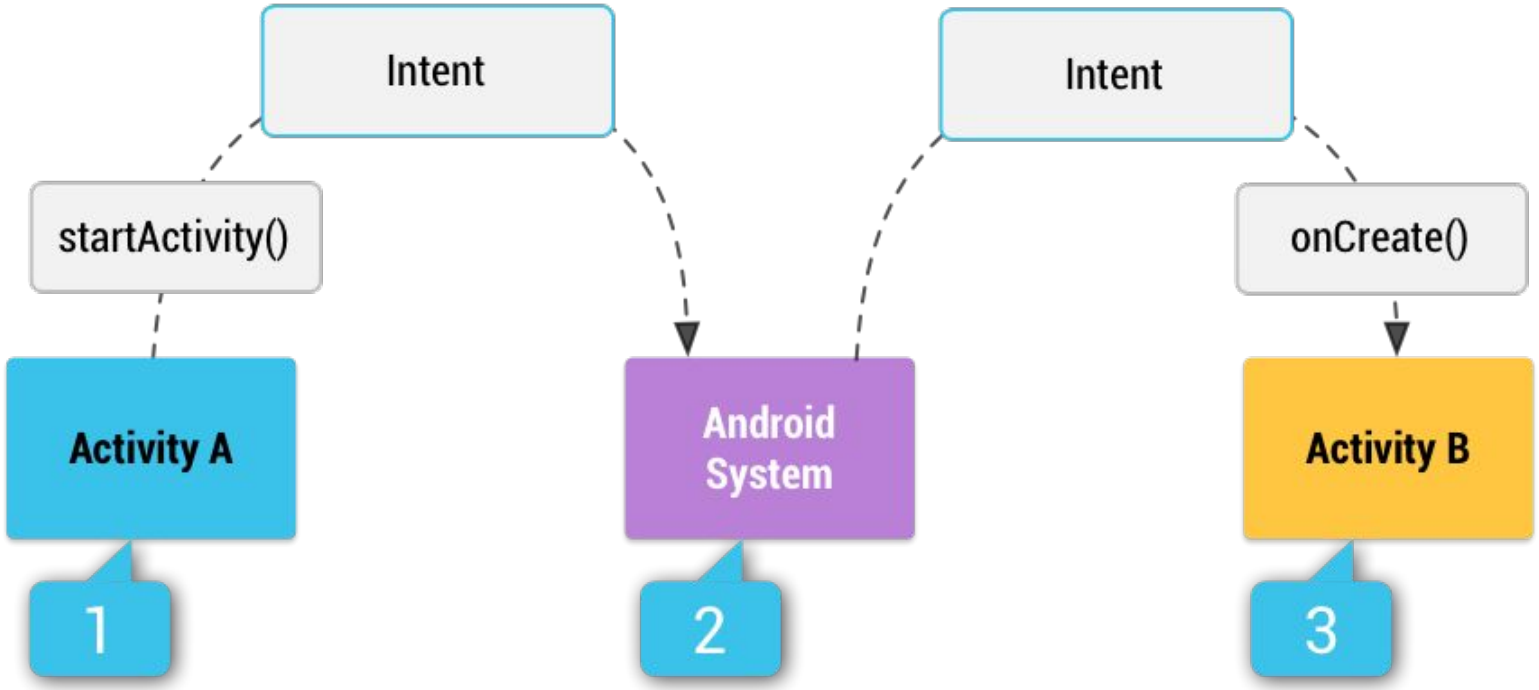
```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}

   downloadIntent.data = Uri.parse(fileUrl)
startService(downloadIntent)
```

# Intents

- Implicit

```
// Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

# Intents


Intent
Intent
startActivity()
onCreate()
Activity A
Android System
Activity B
1
2
3

# Bundle

Android Bundle is used to pass data between activities. The values that are to be passed are mapped to String keys which are later used in the next activity to retrieve the values.

```
val bundle = Bundle()

bundle.putString("key_1", "Hello world")

bundle.putBoolean("key_2", true)
```