

## Lesson 3 - Comic App

### Objectives

- Describe how the AsyncTasks class is used
- Query an API using the AsyncTasks class
- State the limitations of the AsyncTasks class
- Modify the android Manifest to set permissions and fix the orientation
- Download JSON data given a URL of an API call
- Parse JSON data using the JSONObject class
- Download an Image file given a URL
- Display the information in the UI

### Introduction

In this lesson, we are going to download data from the xkcd.com web API and display the comic image in our app.

## The Android/Java that you need to know

### Abstract Classes

A class declared as **abstract** cannot be instantiated.

Typically, **abstract classes** have **abstract methods**, which you will recall is a **method signature** that includes the keyword **abstract**.

Why do we do this?

- **What is to be done** is specified by the abstract methods (and interfaces too)
- **How it is to be done** is specified by their implementation

Because abstract classes can behave as a datatype, it promotes **code reuse**.

Recall the principle of **Program to a supertype**: You may write methods that take in an abstract class as an input, and you are guaranteed that objects passed to it will have the abstract method implemented. (This remark also applies to java interfaces.)

Your abstract class can have constructors specified. One use case is if you would like to force the initialization of any instance variables.

Remember that constructors are not inherited.

Thus, in the constructor of the child classes, you will need to call the superclass constructor using the **super()** keyword.

To use an abstract class, sub-class it and implement any abstract methods.

In the example below, complete the class **Tiger**.

```
public class TestAbstract {

    public static void main(String[] args){
        Feline tora = new Tiger("Tiger","Sumatran Tiger");
        makeSound(tora);
    }

    public static void makeSound(Feline feline){
        feline.sound();
    }
}

abstract class Feline {

    private String name;
    private String breed;

    public Feline(String name, String breed) {
        this.name = name;
        this.breed = breed;
    }

    public String getName() {
        return name;
    }

    public String getBreed() {
        return breed;
    }

    public abstract void sound();
}

class Tiger extends Feline{

    //write the constructor and complete the class
}
```

## Template Method Design Pattern

One application of an abstract class is in the **template method design pattern**.

In this design pattern, there is an algorithm with a fixed structure, but the implementation of some steps are left to the subclasses. The following example is taken from “Heads First Design Patterns - A brain-friendly guide”.

We have a fixed way of brewing caffeine beverages (Coffee, Tea etc), but you’ll agree that how you brew it and what condiments to add depends on the beverage.

In the example below,

- The algorithm to make the caffeine beverage, **prepareRecipe()** is declared **final** to prevent subclasses from altering the algorithm.
- The steps in the algorithm that are common to all beverages are implemented.
- The steps that can vary are declared **abstract**.

```
public abstract class CaffeineBeverage {  
  
    final void prepareRecipe(){  
        boilWater();  
        brew();  
        addCondiments();  
        pourInCup();  
    }  
  
    abstract void brew();  
  
    abstract void addCondiments();  
  
    void boilWater(){  
        System.out.println("Boiling Water");  
    }  
  
    void pourInCup(){  
        System.out.println("Pouring in Cup");  
    }  
}
```

We may then have our sub-classes of **CaffeineBeverage**.

```
class GourmetCoffee extends CaffeineBeverage{
    @Override void brew() {
        System.out.println("Put in Coffee Maker");
    }

    @Override void addCondiments() {
        System.out.println("Adding nothing, because GourmetCoffee");
    }
}
```

We may then brew our coffee:

```
CaffeineBeverage caffeineBeverage = new GourmetCoffee();
caffeineBeverage.prepareRecipe();
```

## Generic Classes & Interfaces

You would have used the `ArrayList` class in the following way::

```
ArrayList<Integer> arrayList = new ArrayList<>();
```

Recall that this design is for **type safety** - an operation cannot be performed on an object unless it is valid for that object.

This means that the code below would cause a compile-time error.

```
arrayList.add("abc");
```

You would also have worked with the `Comparable` interface:

```
public class Octagon implements Comparable<Octagon>{
    //code not shown
    @Override
    public int compareTo(Octagon octagon) {
        //code not shown
    }
}
```

The `ArrayList` class is an example of a **Generic class** where a class takes in an object or objects as a parameter and operates on it.

Similarly, the `Comparable<T>` interface is an example of a **Generic interface**.

## **AsyncTasks Abstract Class allows tasks to run in the background or rather “asynchronously”**

In this app, we are going to query a web API to retrieve data.

The length of time this task will take depends

- on the internet connection,
- file size, etc

While the data is being retrieved, you want the UI to remain responsive and inform the user that the download is still in progress.

This means that the **download should be done on a separate thread from the UI.**

Android provides an abstract class called **AsyncTask** to carry out this task.

**Question:** Do you know any programming language which is asynchronous by default?

## AsyncTasks is an abstract class with three generic types

```
AsyncTask<Params, Progress, Result >
```

**Params** - this is the type of the object sent to the background task to begin execution

**Progress** - this is the type of the object that is reported to the UI while the background task executes

**Result** - this is the type of the object that is returned when the background task completes

**Example.** Suppose you want to download an image given a URL, and not report the download progress to the UI, then you would subclass AsyncTask as follows.

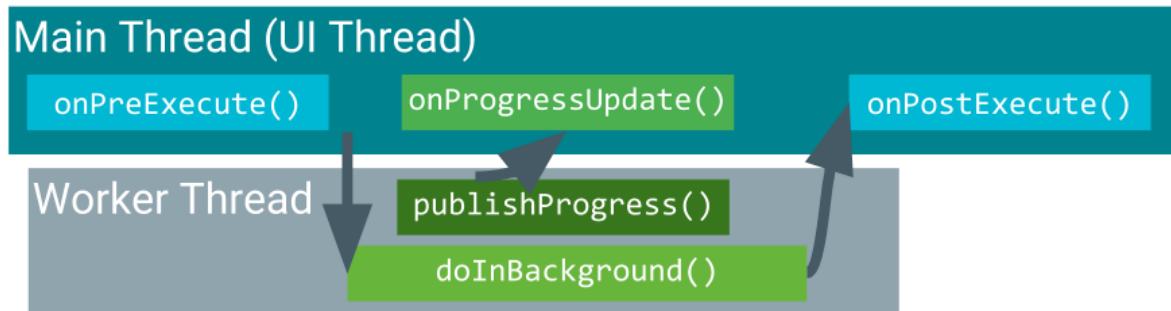
```
class MyBackgroundTask extends AsyncTask<URL, Void, Bitmap>{
}
```

Recall that **URL**, **Void**, **BitMap** are the *type parameters* to the generics. The underlying method design in AsyncTask<> is fixed regardless of what these type parameters are. (However, the individual step/method can be overridden).



## AsyncTasks has four methods you can override

The diagram below is taken from the Android developer fundamentals.



The four methods that you can override are

**onPreExecute()** - seldom-used. Execute any jobs before the background task begins.

**doInBackground()** - always used. Execute the background task on the worker thread.

**onPostExecute()** - sometimes-used. Execute any jobs once the background task ends.

**onProgressUpdate()** - this method puts information on the UI that is provided by **publishProgress()**. This is useful if you have a long download and want to tell your user the progress.

Hence, you override this if you have specified the second generic type, **Progress**.

You call **publishProgress()** within **doInBackground()**

These methods (except **doInBackground()**) are also known as the *callbacks*.



## How to use AsyncTask

### Step 0. Prevent your activity from changing orientation.

- AsyncTask is easy to use but has a major flaw.
- In the Android Activity Lifecycle, recall that the app layout is destroyed and re-created if the screen is rotated.
- If AsyncTask is running in the background during this process, it will not be able to display the result on the re-created activity. (c.f. Phone number changes)
- Hence, you need to constrain your activity so that it is always in your desired orientation.
- This is done in the android manifest.

### Step 1. Write an inner class in MainActivity that extends AsyncTask.

Make the following decisions to help you decide the generic types:

- What information launches the background task?
- What information do I want to give the user as the task proceeds?
- What information does the background task provide?

### Step 2. Decide what jobs are to be run in each of the four methods:

- **doInBackground()** always carries out the background task e.g. downloading the image data from a URL.
- Decide if you need **onPreExecute()** - I have not found a reason to use it so far.
- **onPostExecute()** carries out the job to be done after the background task is complete. Suppose the task is to download an image given a URL, this is where you write code to display the image on the UI.
- If you use **publishProgress()** within **doInBackground()**, decide how you want this information to be displayed on the UI using **onProgressUpdate()**

**Step 3. Subclass AsyncTask with the parameters and implement the methods.**

One possible code stump is shown below. In this code stump, `onPreExecute` is not needed. Look at the method signatures and observe how the generic types are used.

```
class GetComic extends AsyncTask<URL,String, Bitmap>{

    @Override
    protected Bitmap doInBackground(URL... urls) {

        Bitmap bitmap = null;
        return bitmap;
    }

    @Override
    protected void onProgressUpdate(String... values) {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        super.onPostExecute(bitmap);
    }
}
```

**Step 4. Decide how the user is to execute the background task.**

Let's say your user will download the image with a button click. Then you would put the following code in the anonymous class within `setOnClickListener()`

```
GetComic getComic = new GetComic();
getComic.execute(url);
```

If you recall the **template method design pattern**, what we have done is

- Override the steps of the algorithm
- Execute the algorithm using the `execute()` method

## Using the xkcd.com web API

Many websites provide an API for you to access the data. In this lesson, we are going to use the xkcd.com API to access the comics.

This is all the documentation that is provided, in the **About** section:

### **Is there an interface for automated systems to access comics and metadata?**

Yes. You can get comics through the JSON interface, at URLs like <http://xkcd.com/info.0.json> (current comic) and <http://xkcd.com/614/info.0.json> (comic #614).

Hence, this is the easiest web API that I could find and we are going to use it.

## Building a URL

A URL is a URI that refers to a particular website. The parts of a URL are as follows

Component	Example
<b>Scheme</b>	https
<b>Authority</b>	xkcd.com
<b>Path</b>	info.0.json or 614/info.0.json

To prevent parsing errors you may use the `Uri` builder before creating a `URL` object.

Using such a builder helps to eliminate coding errors.

The string constants may be placed in the `strings.xml` file instead.

After you create the `Uri` object, use it to make a `URL` object.

The constructor of the `URL` class throws a `MalformedURLException`.

As this is a **checked exception**, you have to put the code in a **try-catch** block.

```
final String scheme = "https";
final String authority = "xkcd.com";
final String back = "info.0.json";
URL url = null;

Uri.Builder builder = new Uri.Builder();
builder.scheme(scheme)
    .authority(authority)
    .appendPath(back);

Uri uri = builder.build();

try{
    url = new URL(uri.toString());
}catch(MalformedURLException ex) {
    Log.i(TAG, "malformed URL: " + url.toString());
}
```

## Connecting to the internet

With a URL object, you are ready to download data from the internet.

In the starter code, you are provided with a `Utils` class.

There are three static methods for you to use.

You need not worry about how they are implemented, although if you are querying an API for your 1D project, you might find the code useful.

You should see that some of these methods print data to the logcat.

**The following methods take in the comic number, queries the xkcd API and returns the image URL for that comic. Please note the exceptions thrown by this method.**

```
static String getImageURLFromXkcdApi(String comicNo)
```

**The following method takes in a URL for an image and returns the image as a Bitmap object. Please note the exceptions thrown by this method.**

```
static Bitmap getBitmap(URL url)
```

**The following method takes in a Context object and checks if a network connection is available. True is returned if a network connection is available, False if is not. Context is the superclass of AppCompatActivity.**

```
static boolean isNetworkAvailable(Context context)
```

If you use the Android Emulator, you may experience difficulties with checking for an internet connection. Please borrow a phone.

## Android Manifest

In order to

- Constrain the activity orientation
- Access the internet

The android manifest needs some additional information.

They are shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.norman_lee.comicapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="ComicApp"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:screenOrientation="portrait"
            android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

## The JSON format

**JSON** stands for **JavaScript Object Notation** and stores data using key-value pairs.

A sample response from the xkcd API is

```
{ "month": "11", "num": 2068, "link": "", "year": "2018", "news": "",  
  "safe_title": "Election Night", "transcript": "", "alt": "\"Even the  
blind\u00e2\u0080\u0094those who are anxious to hear, but are not able  
to see\u00e2\u0080\u0094will be taken care of. Immense megaphones have  
been constructed and will be in use at The Tribune office and in the  
Coliseum. The one at the Coliseum will be operated by a gentleman who  
draws $60 a week from Barnum & Bailey's circus for the use of his  
voice.\"\"", "img": "https://imgs.xkcd.com/comics/election_night.png",  
  "title": "Election Night", "day": "5" }
```

Before you make use of this data, you would have to make sense of it. Online JSON viewers are available to help you.



## Parsing JSON data using the JSONObject class

Once you are able to make sense of this data, you are ready to parse it. One way is to use the **JSONObject** class.

The **JSONObject** constructor takes in a string variable that contains the JSON data.

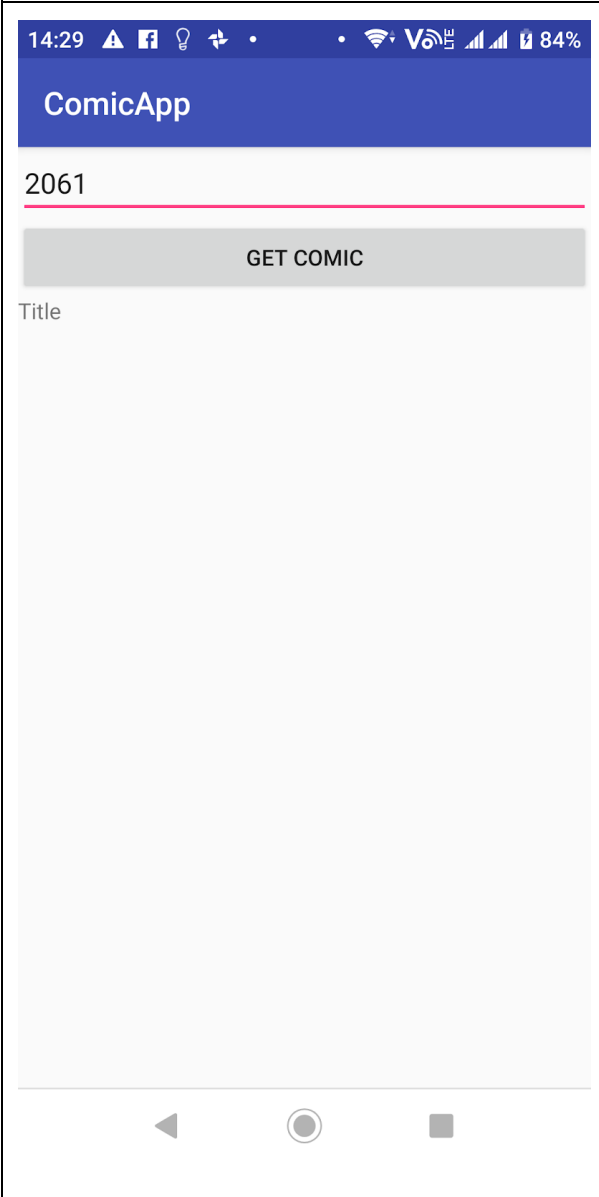
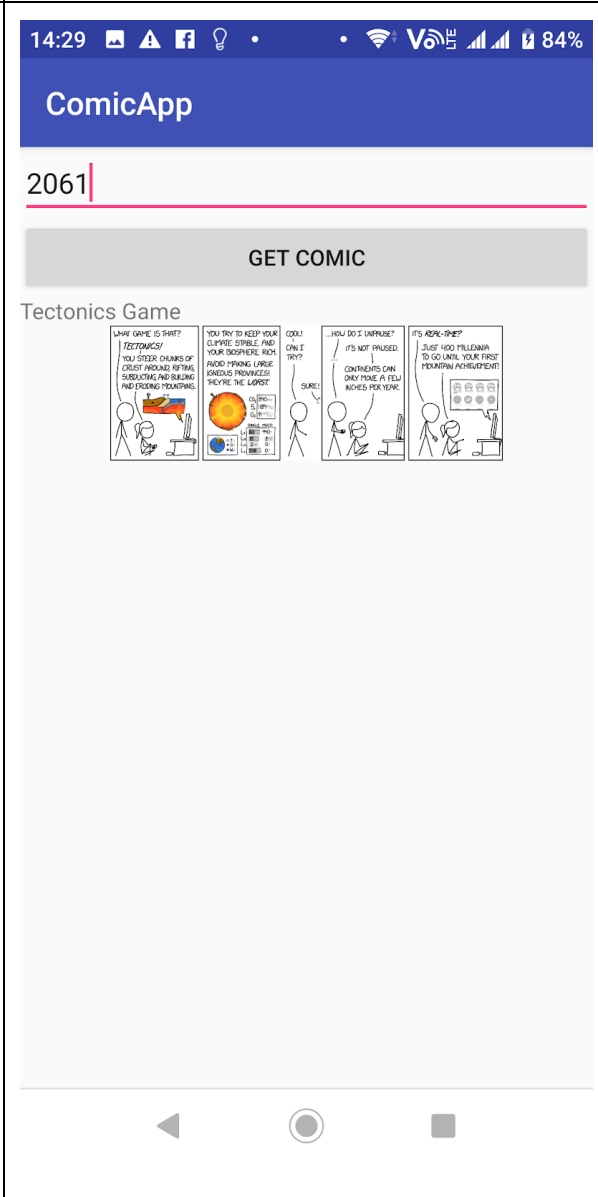
You then retrieve the value using the key and one of the appropriate get methods.

```
JSONObject jsonObject = new JSONObject(json);  
String safe_title = jsonObject.getString("safe_title");
```

Another alternative is the GSON library, which many programmers find useful in parsing JSON data. I will leave you to learn it on your own and you won't be tested on it.

## Completing Your App

### What the app should do

<p><b>Step 1.</b> A user enters a number from 1 to the latest xkcd comic number and clicks <b>Get Comic</b>.</p>	<p><b>Step 2.</b> The comic and its <b>URL</b> is displayed. (sorry image below is not updated)</p>
	

**BEFORE YOU BEGIN**, download the starter code and compile your app. You may need to **Build → Clean Project**.

**TODO 6.0** Study the **Utils** class and see what methods are available for you

**TODO 6.1** Ensure that **Android Manifest** has permissions for internet and has the orientation fixed

**TODO 6.2 - 6.4** When button is clicked, Get the user input

- Get references to widgets
- Set up `setOnClickListener` for the button
- Retrieve the user input from the `EditText`

**TODO 6.5 - 6.9** Write the **GetComic AsyncTasks** class to download the comic

- Make `GetComic` extend `AsyncTask<String, String, Bitmap>` (note the different types and ask yourself why ... )
- (`doInBackground`) Call `Utils.getImageURLFromXkcdApi` to get the image URL from comicNo
- (`onProgressUpdate`, `doInBackground`) Call `publishProgress`, write code to update `textViewTitle` with the image URL
- (`doInBackground`) Call `Utils.getBitmap` using the URL to get the bitmap
- (`onPostExecute`) Assign the `Bitmap` downloaded to `imageView`. The bitmap may be null.

**TODO 6.10** Back in `setOnClickListener`, If network is active, instantiate your `AsyncTask` class and call the `execute` method