# Lab: Asynchronous Programming

This document defines the lab overview for the ["Java Advanced" course @ Software University](https://softuni.bg/trainings/1377/advanced-java-may-2016). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1377/advanced-java-may-2016).

# Introduction

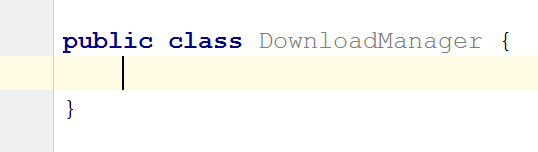
In our final piece of the **BashSoft** we are going to implement functionality for downloading a file from the internet using asynchronous and synchronous methods. First we are going to implement the synchronous one and after that reuse it, to make it asynchronous.

## Start Implementing the Download Method

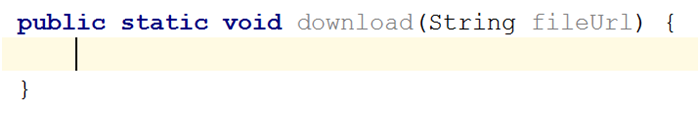
Let’s first start by making a new package called **Network** in our project with a new public class called **DowloadManager**:



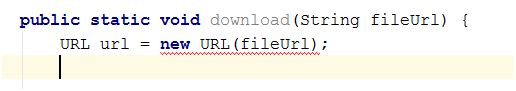
…



All we need as information in order to download a file is the **URL** to the specified file. That’s why we are going to make a public static void **Download** method with one input parameter which is the **fileURL**:

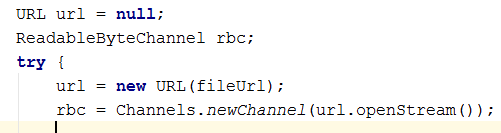


Since we are going to make a communication with the Internet, we are going to use a class that gives us such functionality. Such is the **URL** class which can be found in the **java.net** package. It is not a static class and for that reason we are going to have to make a new **url** in our method and pass the **fileURL** to its constructor.



As you can see the constructor of this class may throw a **MalformedURLException**, so handle it.

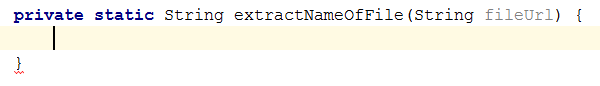
Now we need to create a **ReadableByteChannel** and create a new channel from the stream of our URL:

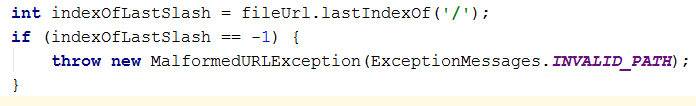


We can easily get the current path on the PC, but first we need to be able to extract the name of the file from the **URL**, so that we can use it to save the file with the same name on our PC. For that reason, we should make a new method called **extractNameOfFile** that, is private static returns a String (the actual file name) and take as a parameter the **fileURL**.

## Extract Name of File from URL

This is how the method should look before the declaration of its implementation:

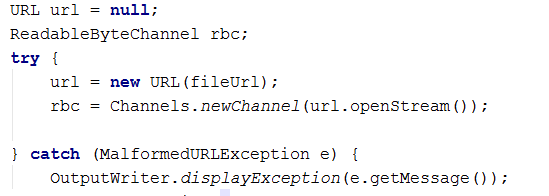


All we need to do it take the index of the last forward slash, because it separates the file name + extension from the folder it is in, on the server. Since the result can be either a non-negative number if the **fileURL** contains such a character, or **-1** if it does not, we want to make sure that we do the job only when the index is correct and otherwise throw a new **MalformedURLException** with the message for invalid path.  
  


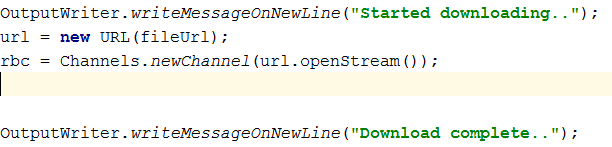
Our method is still underlined, because we haven’t returned a value for every case of our program, so let’s do this and go back to the implementation of the **Download** method. All we need to do is take everything from index found + 1 till the end of the **URL** and use it as a file name.



## Finish Implementing the Download Method

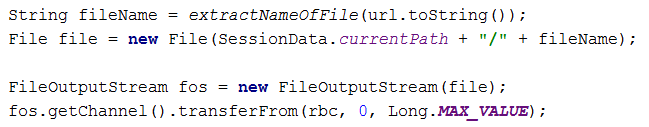
Since we are going to use the **extractNameOfFile** and it can raise an exception, we would like to put all the functionality that we are going to write in a try block and in the catch block we are going to catch a web exception and display its message using the **DisplayException**.  
  


On the first line **in** the **try** **block** we should **indicate** the **user** that we’re **starting** **to** **download** the file and **in** the **last** **line** in the try we should **indicate** the **user** that the **download** is **complete**.

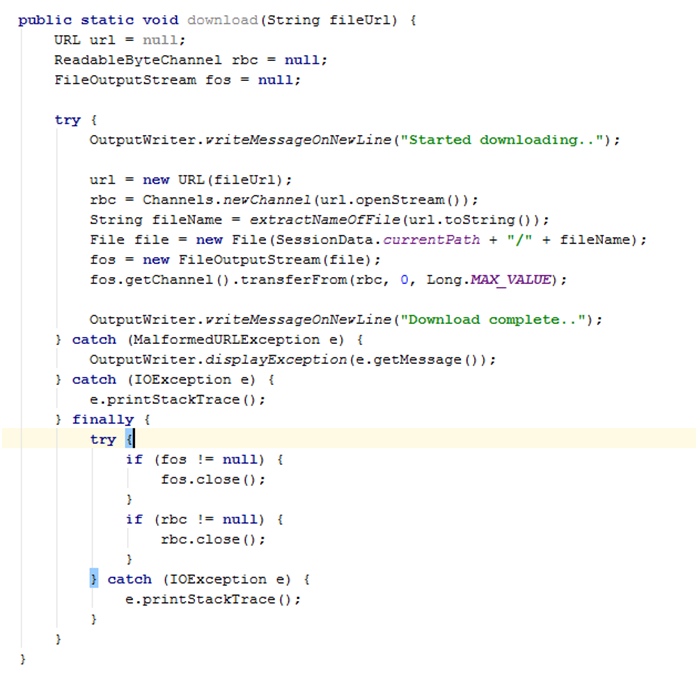


Between the two messages we should save the extracted name of the file, after that generate the local path + the name of the file and create a new **File** object with the **fileURL** and the **local** **path** **+** the **name** **of** the **file**.

Then we must create a **FileOutputStream** which we will use to save the file with. This is how it's done:

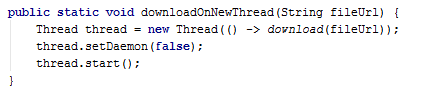


There are other possible exceptions that could be raised, but most of them are **IOExceptions** so we can generalize the catch block or add another one. Don't forget to close your streams in **finally** blocks. In the end your method should look something like this:



## Implement Parallel Downloading

Now that we have implemented the **download** of a file in **Java** we can easily extend it to be asynchronous, by running it on another thread. Since the declaration of the method is pretty close to the previous, here is how it should look:



Notice that we don't **join()** our new thread here because that would cause our **main thread** to wait for the **worker thread** to finish. The **main thread** however has more important stuff to do like processing our input and displaying output.

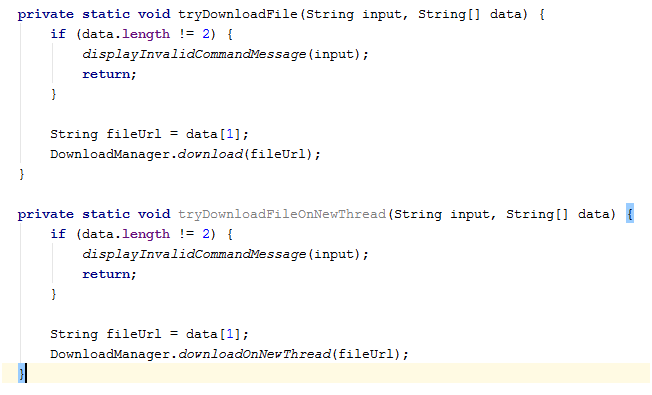
We still need to **join all non-daemon** threads except the **main** one before terminating the main thread. We can do this in the **readInput()** method of the **InputReader** class with the following code:



Notice how we use an if statement to not join the **main thread** or **any daemon thread**.

## Implement Calls from the Command Interpreter

The implementation of the functions that call the **DownloadManager** is pretty common to all others in the current class. We check **if** the **number** **of** **parameters** is **2** and if so, **pass** the **second** **parameter** to the corresponding method.

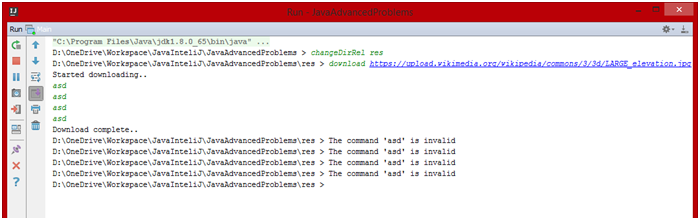


After that all we should do is **call** **them** **in** the **appropriate** **place** **in** the **interpredCommand** method.

Now let’s test the functionality:

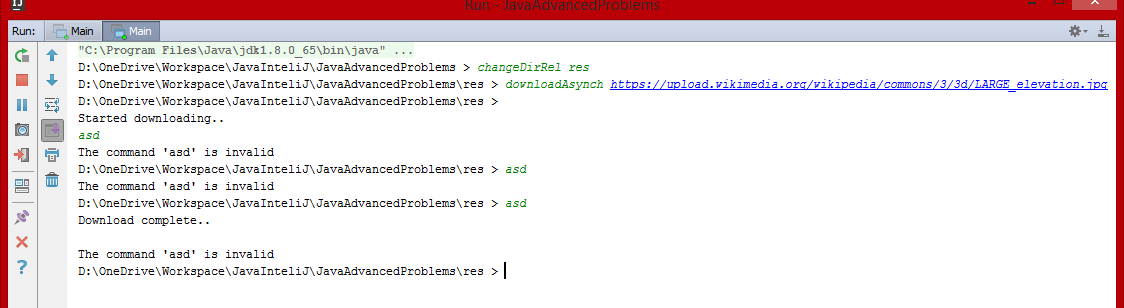
You can use this link: https://upload.wikimedia.org/wikipedia/commons/3/3d/LARGE\_elevation.jpg

**Synchronously:**



Notice how our invalid commands aren't processed while we are using the **main thread** for downloading. We don't even get our prompt... I mean what the hell main thread!? Get your stuff together.

**Asynchronously:**

  
We can do much more stuff while downloading the file.

## Fix the IO

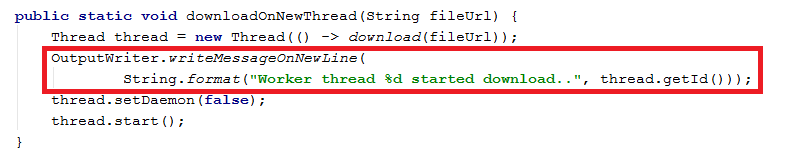
Do you notice how our input and output are kind of unsynchronized? This is because the main thread (or UI thread) should be the only one that processes input and output. In our case a **worker thread** is given the task to write on the console and that messes up our pretty little BashSoft terminal. In frameworks like **JavaFX** there are easy ways to do this, but in our case we will have to fix it ourselves.

The easiest way to do this is by wrapping the "**Started downloading..."** and "**Download complete..."** messages in an **if** statement so that they are only executed by the normal **download** command:



Do this for the other message too.

But now when we use the **downloadAsynch** command we won't get any message. So add the following code to the **downloadOnNewThread** method:



Think about how you can **notify** the main thread so that it you can get the **Download complete** message too.

This is everything from the functionality that we are going to implement in the current course. Since many of the things that we did are quite smelly as a code, it will be our duty to fix those pieces of code and extend the functionality even more.

Congratulations! You’ve successfully completed the lab exercise for Asynchronous Programming.