Object Oriented Programming with Java
Practical-7

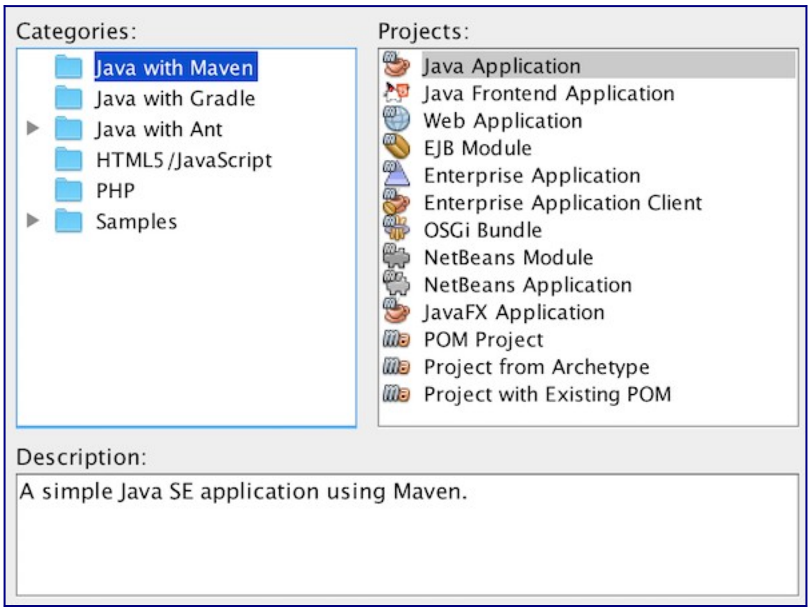
NetBeans IDE and Introduction to Exception Handling

Setting Up the Project in NetBeans

Take the steps below to set up a new Java project.

1. In the IDE, choose File > New Project or click the "New Project" button in the toolbar.

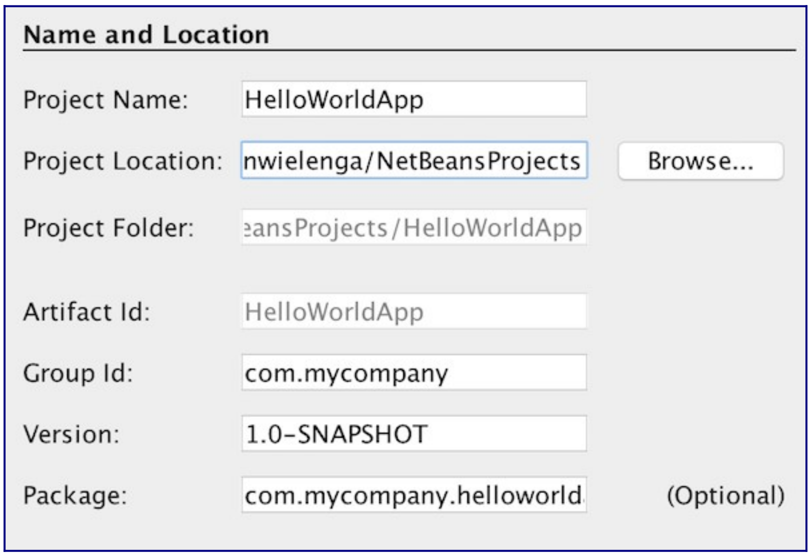
    1. In the New Project wizard, select Java Application, as shown in the figure below. Then click Next.



Description:
A simple Java SE application using Maven.

```
The first time you create a new Java project, you will be prompted to
download and enable support for Java. Follow the prompts and install as
recommended by the wizard.
```

1. In the Name and Location page of the wizard, type `HelloWorldApp` in the Project Name field, (as shown in the figure below):



Name and Location

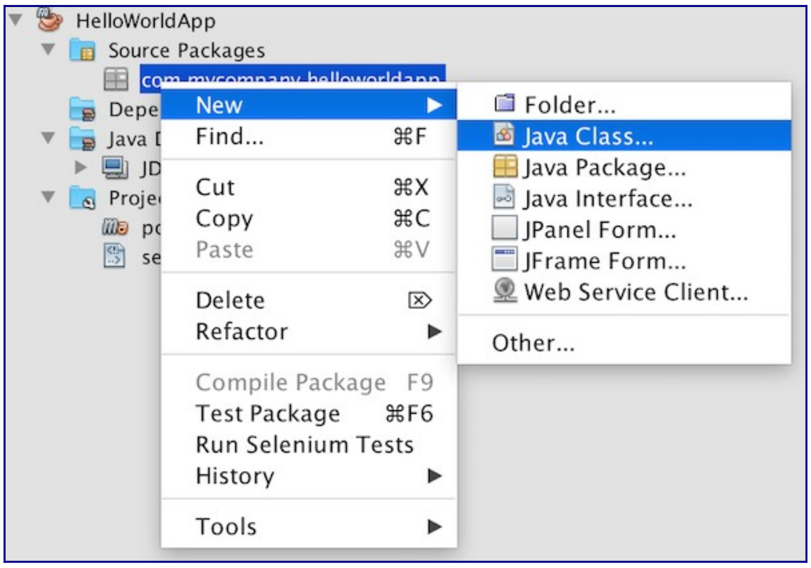
Project Name:	HelloWorldApp	
Project Location:	nwielenga/NetBeansProjects	Browse...
Project Folder:	eansProjects/HelloWorldApp	
Artifact Id:	HelloWorldApp	
Group Id:	com.mycompany	
Version:	1.0-SNAPSHOT	
Package:	com.mycompany.helloworld	(Optional)

Click Finish.

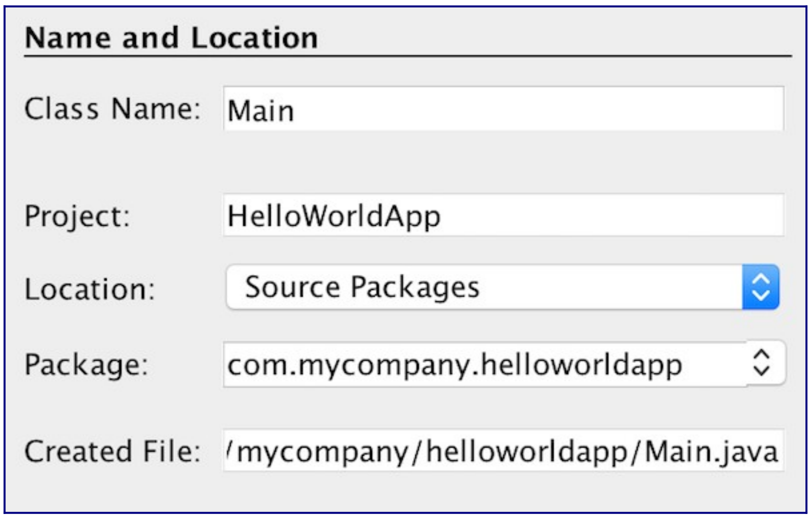
The project is created and opened.

Creating a Java Source File

Right-click the package name and choose New | Java Class, (as shown in the figure below):



In the New Java Class wizard, type `Main` in the Class Name field, (as shown in the figure below):



Name and Location

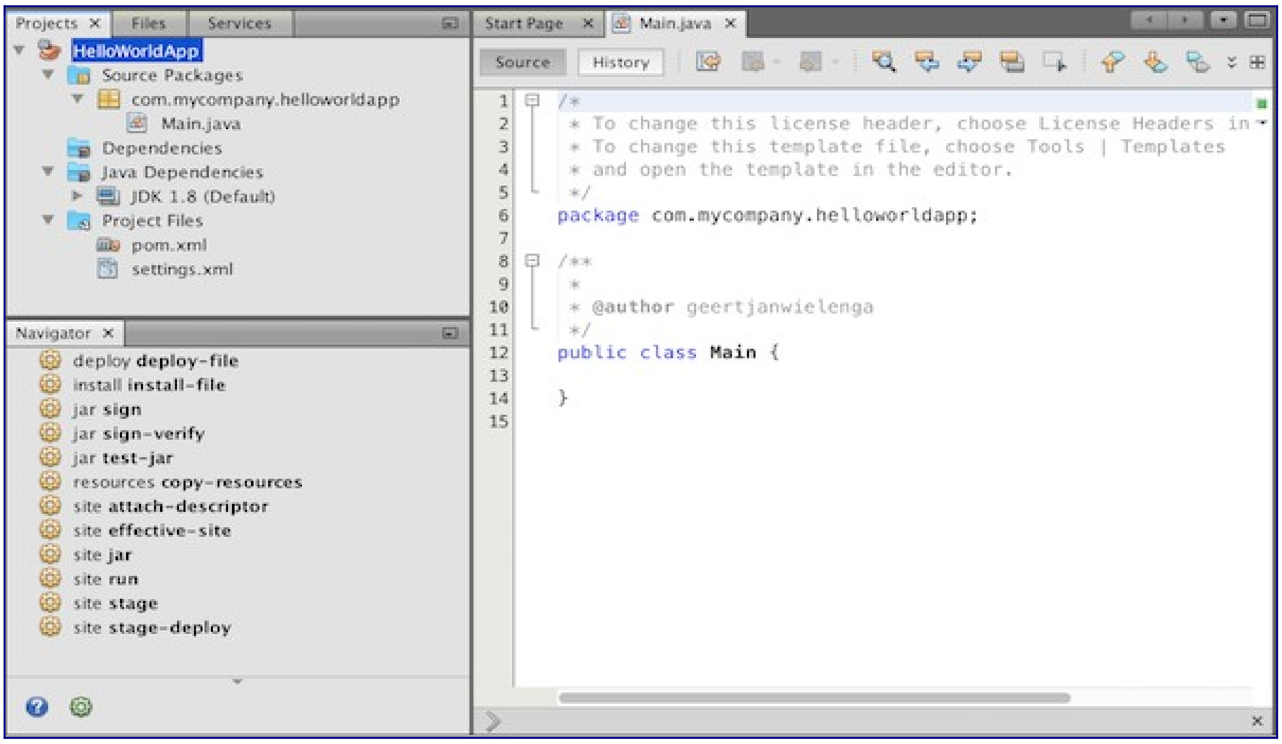
Class Name:	Main
Project:	HelloWorldApp
Location:	Source Packages
Package:	com.mycompany.helloworldapp
Created File:	/mycompany/helloworldapp/Main.java

Click Finish.

The Java source file is created and opened.

You should see the following components, (as shown in the figure below):

- **Projects window:** Top left, contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.
- **Source Editor:** Central area, contains files, where most of your work will be done, currently with a Java source file called `Main` open.
- **Navigator:** Lower left, useful for quickly navigating between elements within the selected class.



```java
/*
 * To change this license header, choose License Headers in
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.mycompany.helloworldapp;

/**
 *
 * @author geertjanwielenga
 */
public class Main {

}
```

Adding Code to the Java Source File

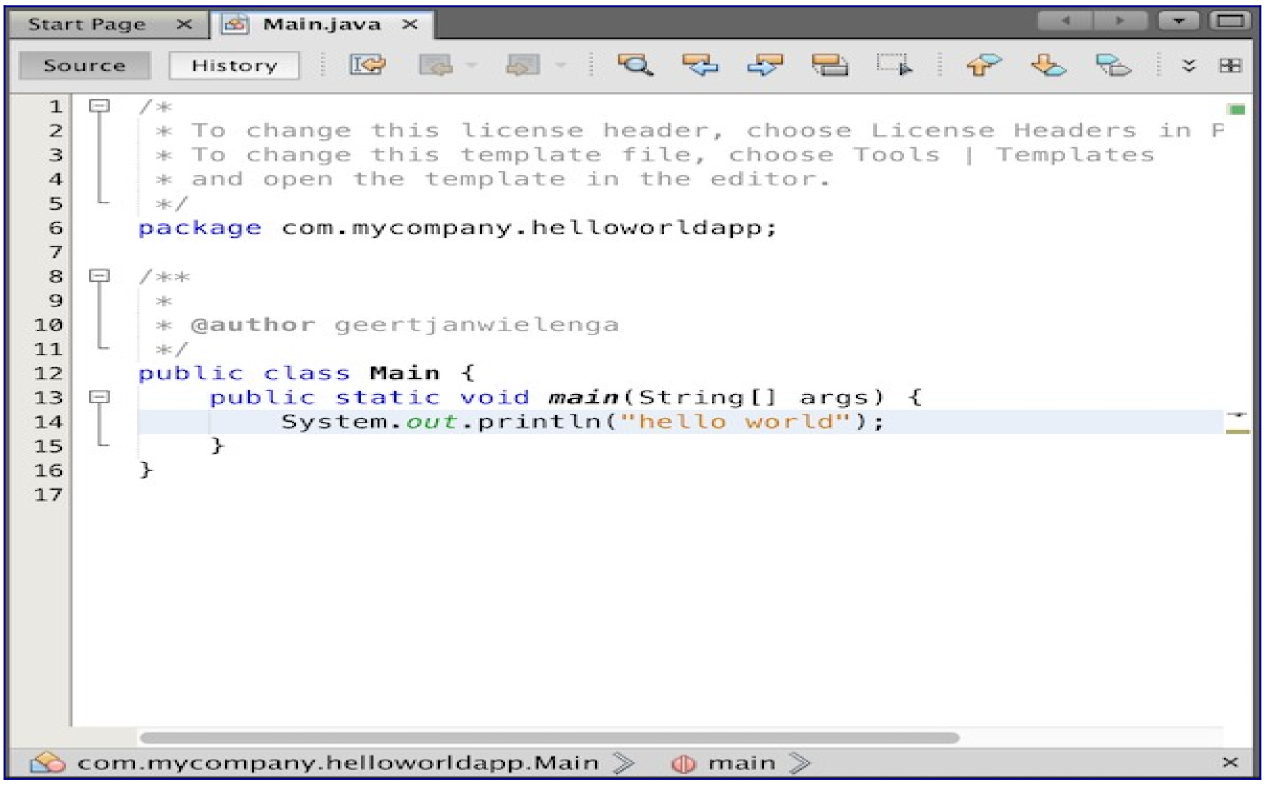
A skeleton main class has been created for you. Let's add some basic content to produce a 'hello world' message.

1. Between the braces, type `psvm` and press `Tab`. You should now see `public static void main` statement.

    1. Within the `public static void main` statement, type `sout` and press `Tab`. You should now see a `System.out.println` statement.

    2. Within the quotation marks, type `hello world`.

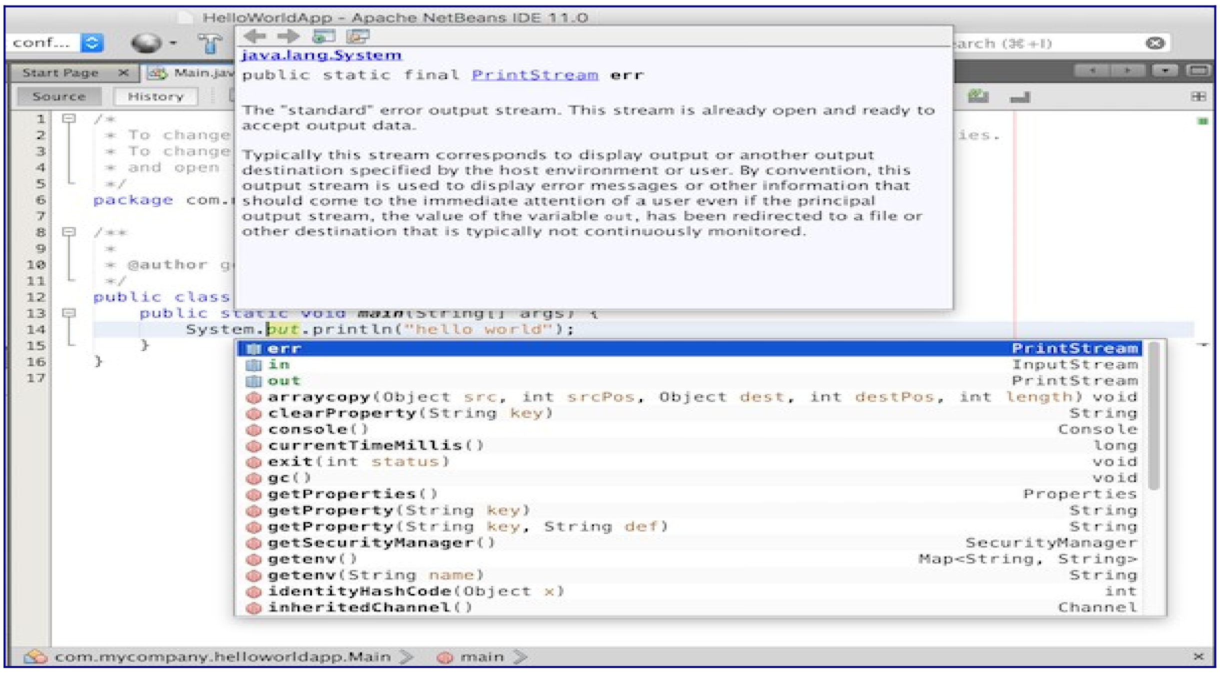
You should now see the following:



```java
/*
 * To change this license header, choose License Headers in P
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.mycompany.helloworldapp;

/**
 *
 * @author geertjanwielenga
 */
public class Main {
    public static void main(String[] args) {
        System.out.println("hello world");
    }
}
```

Notice that when you press `Ctrl+Space`, the editor shows you multiple ways of completing the code at the cursor, as well as related documentation:


HelloWorldApp - Apache NetBeans IDE 11.0
conf...
Start Page
Main.java
Source
History
com.mycompany.helloworldapp.Main
main

java.lang.System

```
public static final PrintStream err
```

The "standard" error output stream. This stream is already open and ready to accept output data.

Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable out, has been redirected to a file or other destination that is typically not continuously monitored.

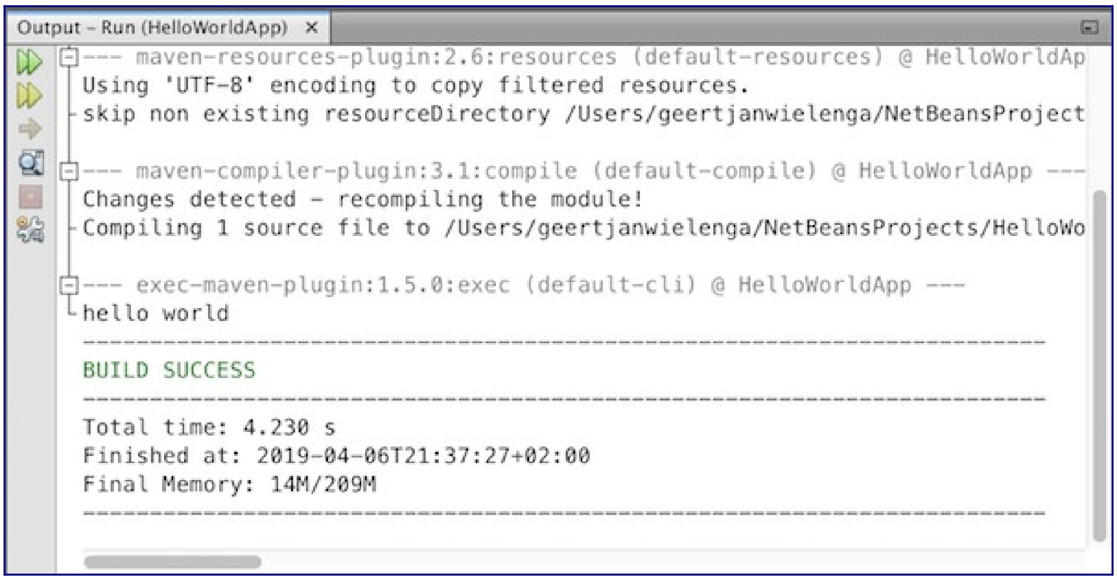
```
System.out.println("hello world");
```

```
err                                                              PrintStream
in                                                               InputStream
out                                                              PrintStream
arraycopy(Object src, int srcPos, Object dest, int destPos, int length) void
clearProperty(String key)                                             String
console()                                                            Console
currentTimeMillis()                                                     long
exit(int status)                                                        void
gc()                                                                    void
getProperties()                                                   Properties
getProperty(String key)                                               String
getProperty(String key, String def)                                   String
getSecurityManager()                                         SecurityManager
getenv()                                                 Map<String, String>
getenv(String name)                                                   String
identityHashCode(Object x)                                               int
inheritedChannel()                                                   Channel
```

Running the Application

Make sure to save the Java source file, right-click the project and choose Run or choose Run Project under the Run menu. Click Select Main Class.

In the Output window (which can be opened from the Window menu), you should see the below.



```
--- maven-resources-plugin:2.6:resources (default-resources) @ HelloWorldAp
Using 'UTF-8' encoding to copy filtered resources.
skip non existing resourceDirectory /Users/geertjanwielenga/NetBeansProject

--- maven-compiler-plugin:3.1:compile (default-compile) @ HelloWorldApp ---
Changes detected - recompiling the module!
Compiling 1 source file to /Users/geertjanwielenga/NetBeansProjects/HelloWo

--- exec-maven-plugin:1.5.0:exec (default-cli) @ HelloWorldApp ---
hello world
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 4.230 s
Finished at: 2019-04-06T21:37:27+02:00
Final Memory: 14M/209M
------------------------------------------------------------------------
```

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

Java try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The `try` and `catch` keywords come in pairs:

Syntax

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Consider the following example:

This will generate an error, because **myNumbers[10]** does not exist.

```
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

The output will be something like this:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
        at Main.main(Main.java:4)
```

If an error occurs, we can use `try...catch` to catch the error and execute some code to handle it:

Example

```
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

The output will be:

```
Something went wrong.
```

Finally

The `finally` statement lets you execute code, after `try...catch`, regardless of the result:

Example

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

The output will be:

```
Something went wrong.
The 'try catch' is finished.
```

The throw keyword

The `throw` statement allows you to create a custom error.

The `throw` statement is used together with an **exception type**. There are many exception types available in Java: `ArithmeticException`, `FileNotFoundException`, `ArrayIndexOutOfBoundsException`, `SecurityException`, etc:

Example

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 
years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

The output will be:

```
Exception in thread "main" java.lang.ArithmeticException: Access 
denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:4)
        at Main.main(Main.java:12)
```

If **age** was 20, you would **not** get an exception:

Example

```
checkAge(20);
```

The output will be:

```
Access granted - You are old enough!
```