

Exercise 0B: Introduction to Programming in Java and Eclipse

Part I:

You have so far used a text editor (e.g. Microsoft's Notepad) for writing code and a command-line interpreter (e.g. Microsoft's Command Prompt) for compiling and running code. To do these tasks more efficiently, many programmers prefer using an integrated programming environment, which facilitates code writing, compiling, and running and MORE. **Eclipse** is a software application that offers such an environment. Eclipse supports several programming languages but was originally designed for Java. In the Appendix, you will find instructions for installing Eclipse to your own computer.

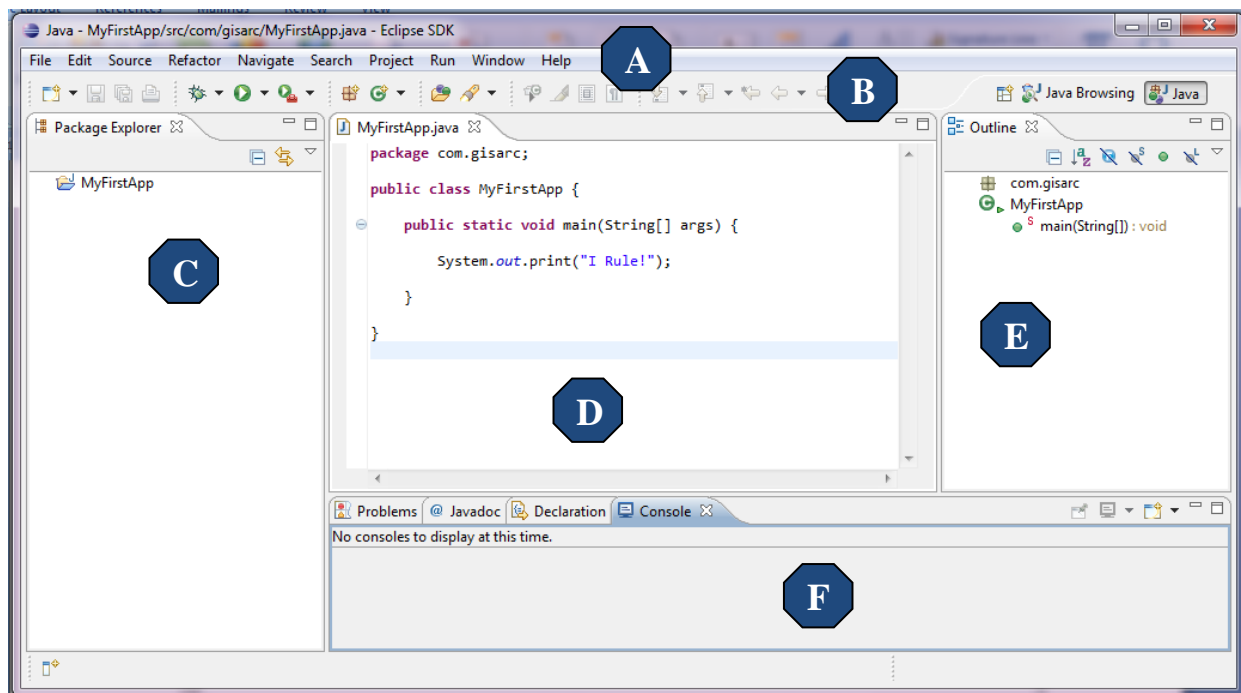
A VERY QUICK GUIDE FOR HOW TO USE ECLIPSE

1. Start Eclipse

Select **Start > Eclipse > eclipse IDE**

When you start Eclipse, it may ask you where your workspace is. Then, choose any (new or existing) folder in the drive assigned to you at KTH (e.g., H:\Eclipse Workspace). You can do this by selecting **File > Switch Workspace > Other...** from the Menu Bar and choosing your workspace. If Eclipse opens up to the Welcome page, close it.

Then you will see a form like this.



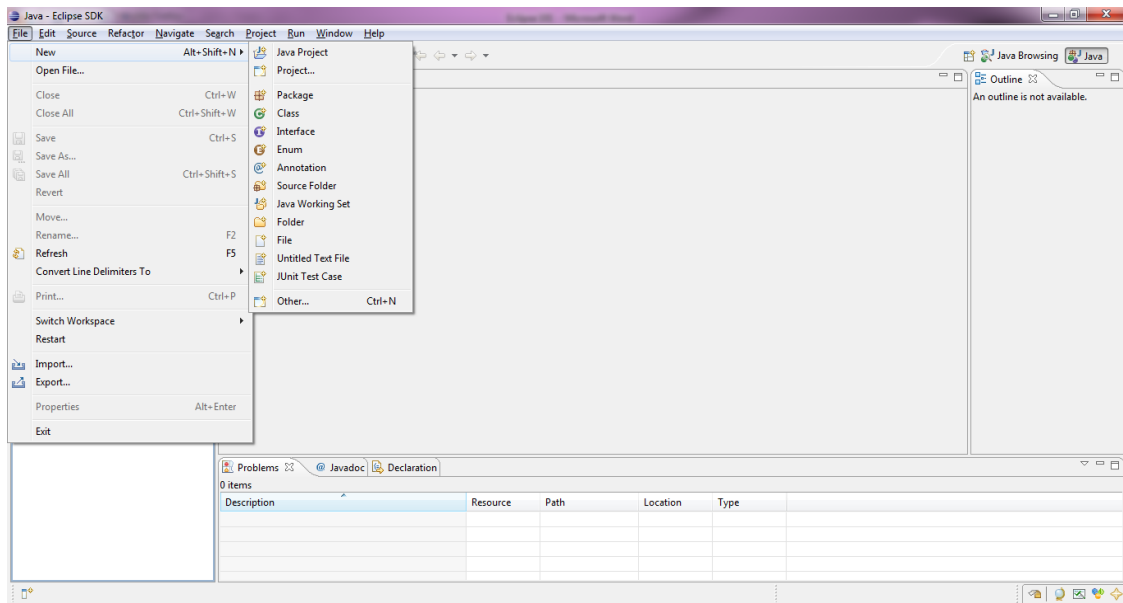
A – Menu Bar B – Tool Bar C – Package Explorer D – Editor Window E – Outline

F – Console

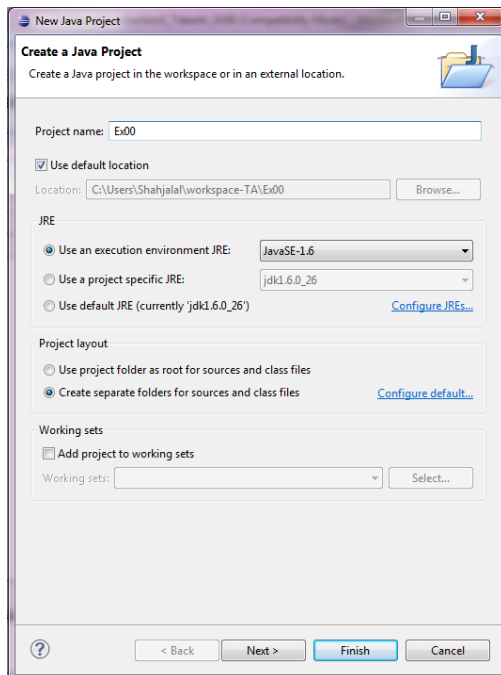
2. Create a Project

All files associated with the current session (e.g. source files, class files, input data, and output data) are preferred to be stored in the same folder (which will be located in the workspace you have previously set). In Eclipse, a **project** is such a folder.

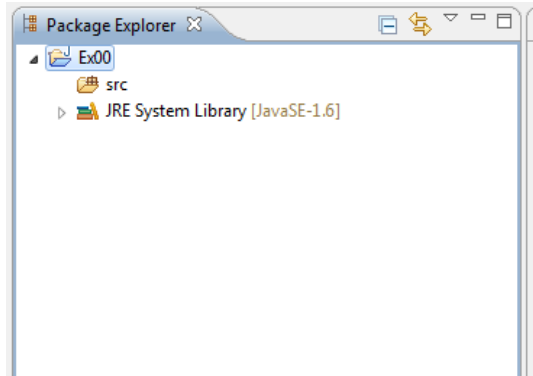
1. To create a new project, select
File > New > Java Project OR **File > New > Project > Java Project**



Then you will see this.



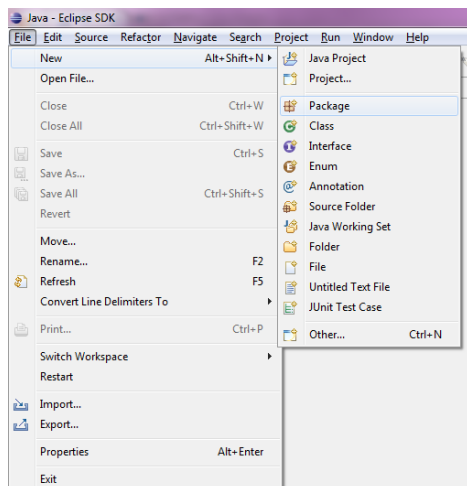
2. Set **Project name** to **Ex00**
Click **Next**
Click **Finish**
3. Check the **Package Explorer**, which has a folder called **src**. This is a folder where you will store all Java source files (files that end with .java) associated with the current project.



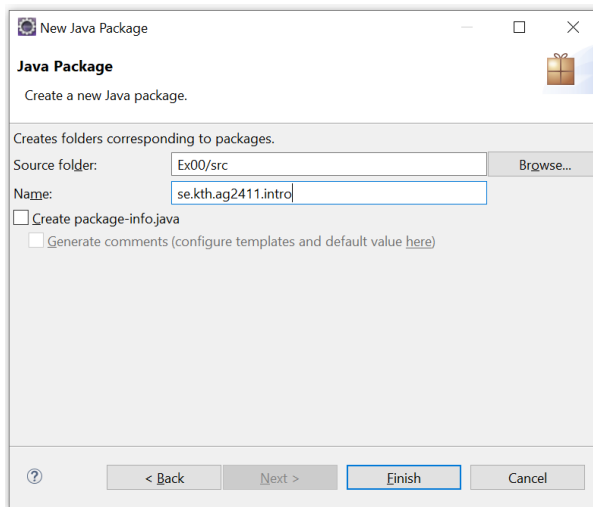
3. Create a Package

A project may contain one or more subfolders called **packages**. If you find some newly created classes related, you will put them in a package. Because a Java program may consist of hundreds or thousands of classes, it is a good practice to group related classes into the same package. In many business contexts, companies use a convention such that a name starts with a reversed Internet domain name, followed by a name that indicates the theme, functionality, etc. of the package. An example is “com.shirabe.gis”.

1. To create a new package, select **File > New > Package**

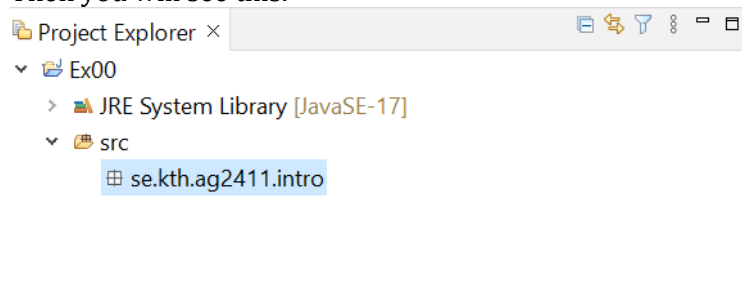


Then you will see this.



2. Set Name to **se.kth.ag2411.intro**
Click **Finish**

Then you will see this.



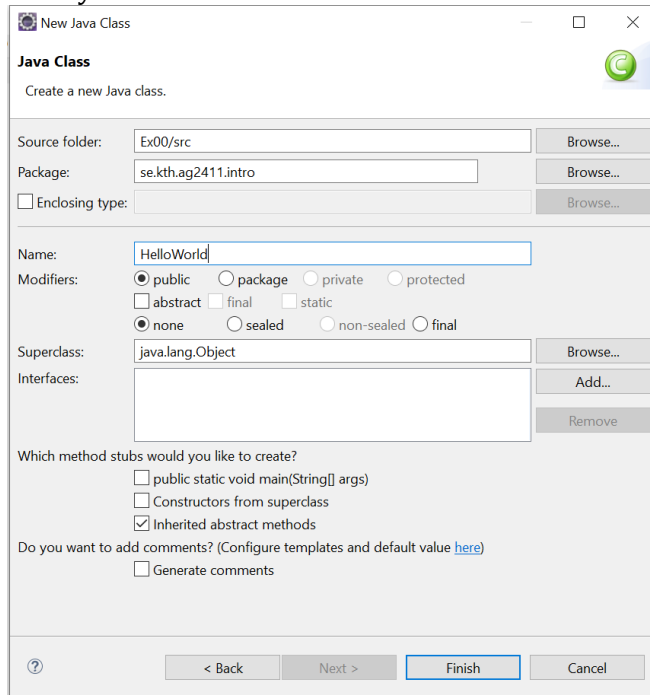
To see what has actually happened, use a file browser (e.g. Microsoft's Windows Explorer) and go to the **src** folder. It has a subfolder called "**se**," in which there is another subfolder called "**kth**," in which there is another subfolder called "**ag2411**," in which another subfolder called "**intro**." This way of managing files and folders gives your project a clear organization and also helps others share your code.

4. Create a Class (and its corresponding source file)

Now we are ready to write Java code.

1. To create a new class, select **File > New > Class**

Then you will see this.



The 'New Java Class' dialog is shown with the following settings:

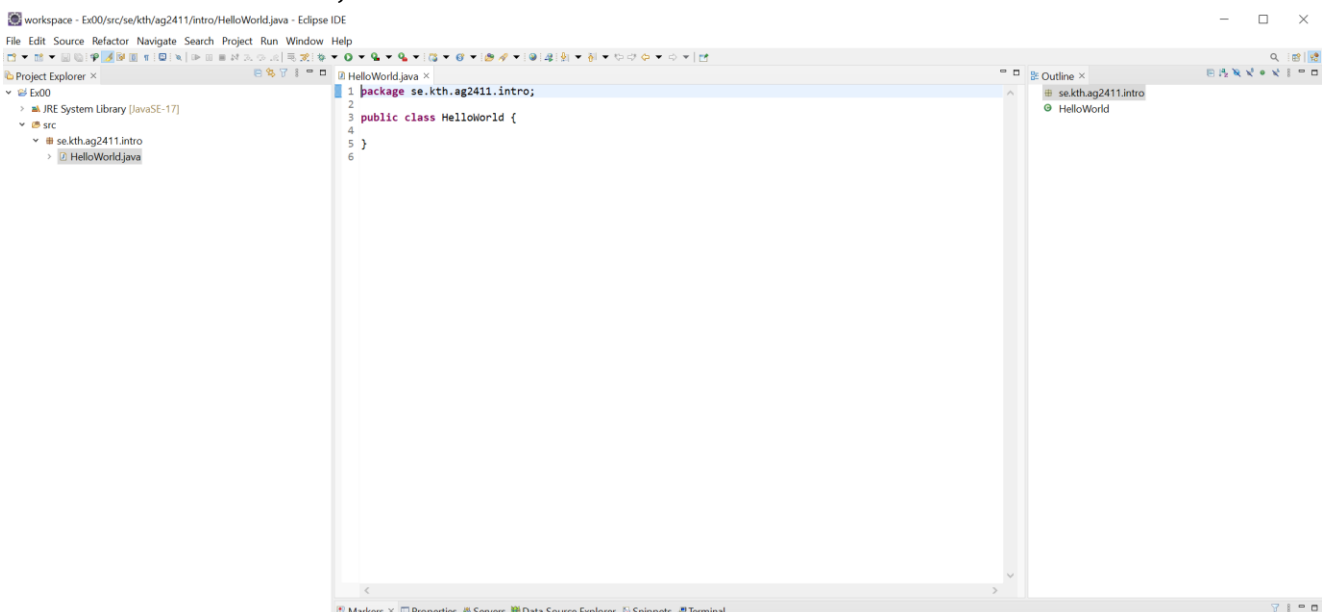
- Source folder:** Ex00/src
- Package:** se.kth.ag2411.intro
- Enclosing type:** (empty)
- Name:** HelloWorld
- Modifiers:** ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static, ☒ none, ☐ sealed, ☐ non-sealed, ☐ final
- Superclass:** java.lang.Object
- Interfaces:** (empty)
- Which method stubs would you like to create?**
 - ☐ public static void main(String[] args)
 - ☐ Constructors from superclass
 - ☒ Inherited abstract methods
- Do you want to add comments?** (Configure templates and default value [here](#))
 - ☐ Generate comments

Buttons at the bottom: < Back, Next >, Finish, Cancel.

2. Set
Source Folder to **Ex00/src**
Package to **se.kth.ag2411.intro**
Name to **HelloWorld**

Click **Finish**

Then HelloWorld.java will be created.

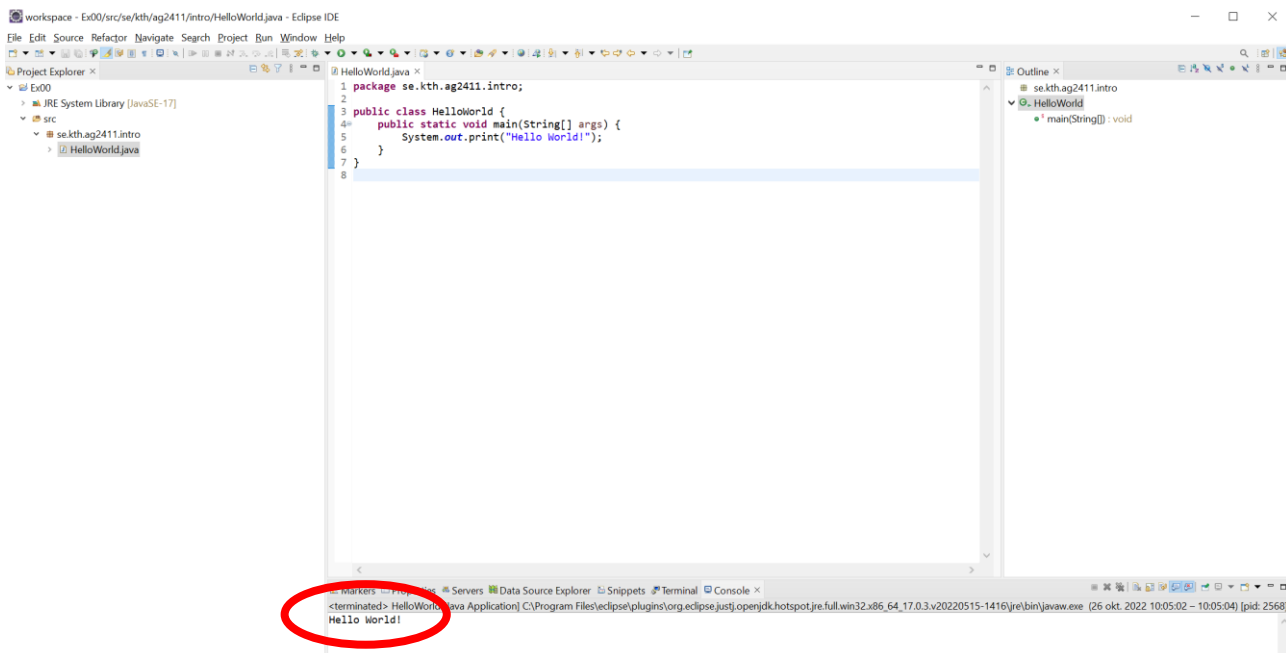


3. Write the following code in Editor Window.

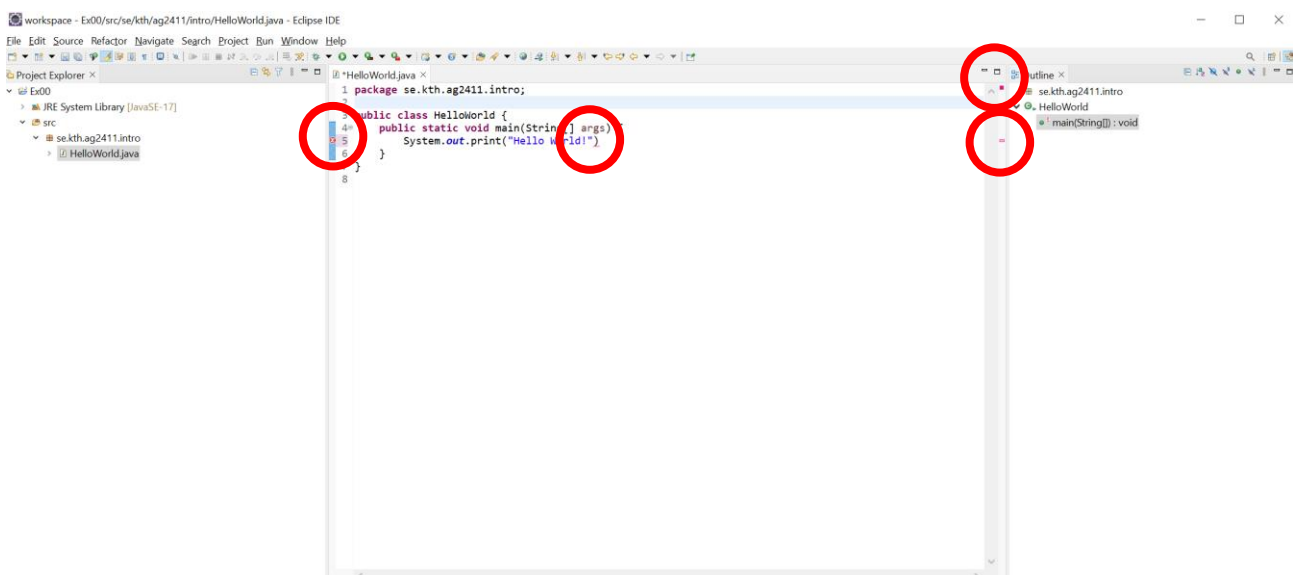
```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.print("Hello World!");  
    }  
}
```

4. To compile and run (at the same time!) the code,
Run > Run As > Java Application

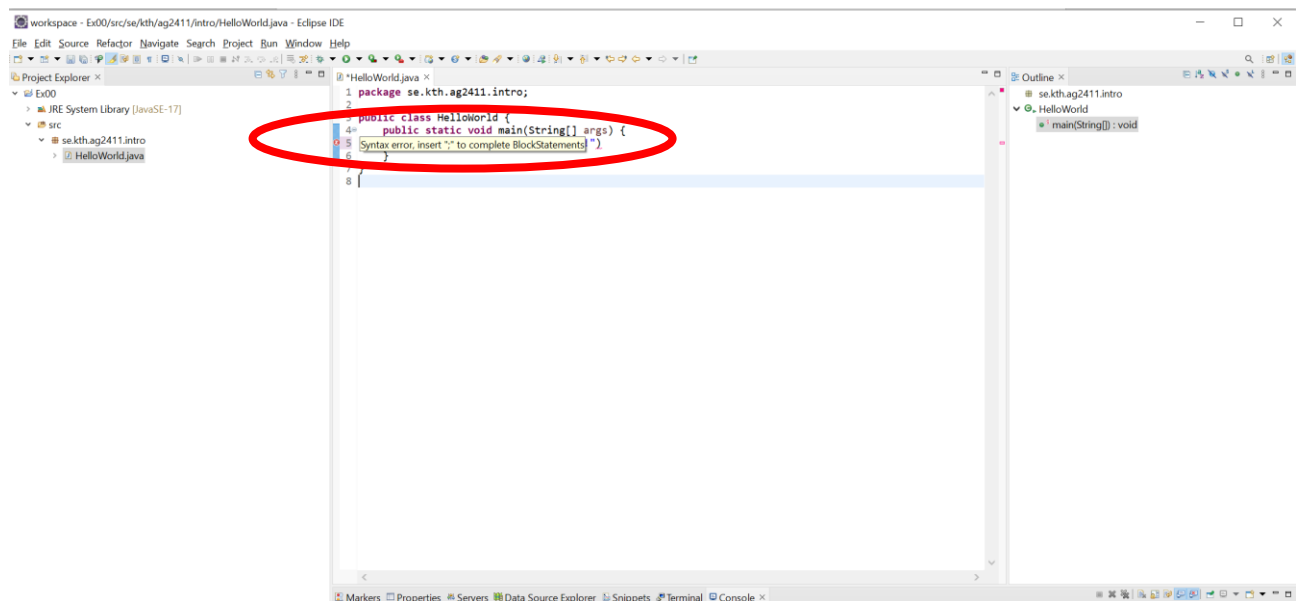
If there are no errors in the code, output will appear in Console Window



If there are errors in the code, Eclipse will tell where they occur.



If you place the cursor on the cross to the left of the line containing an error, you will receive more information about the error.



Part II:

The goal of the second part of this introductory exercise is to learn one of the key concepts of Java programming—**object** and **class**. As briefly explained in the first lecture, an object is like a value, and a class is a (abstract) data type from which an object is created (or **instantiated**). In the following, we will try to understand this concept in a little casual way.

What Is an Object?

Before answering this, see objects in the real world: my desk, your chair, her dog, his bicycle, etc. We can characterize these objects in two ways: what they have and what they do. We may call the former *state* and the latter *behavior*. Dogs have states such as color, breed, and hunger, and behaviors such as barking, fetching, and tail wagging. Bicycles have states such as gear position, cadence, speed, and behaviors such as gear shifting, pedaling, and braking. Even maps have states such as theme, scale, and area of interest, and behaviors such as zooming, panning, and feature selecting.

Characterizing objects in terms of their states and behaviors is very familiar to Java programmers, who sometimes use the terms “**attribute**” and “**method**” for state and behavior, respectively. *In Java, data are often stored in attributes of objects and processed through methods of objects.*

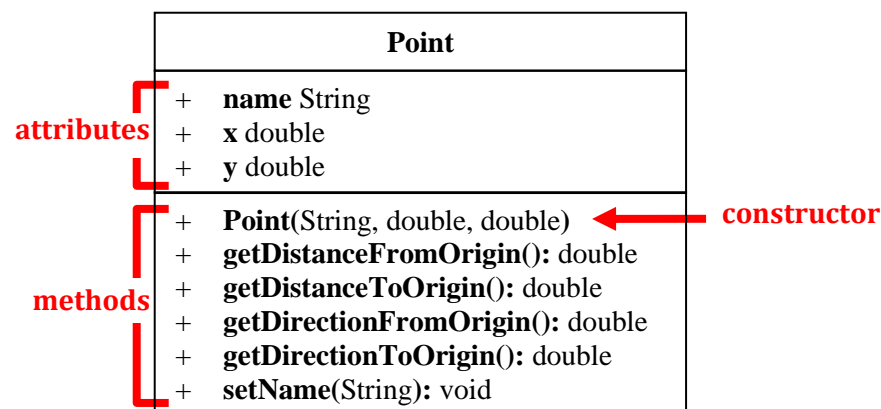
What Is a Class?

In the real world, so many objects look alike. If we go to a parking lot, we will see dozens of cars and may notice that many of them are indeed the same make and model, like Volkswagen Jetta 2009. All VW-Jetta-2009 cars were built from the same blueprint. In the jargon of object orientation, we say that my Jetta, your Jetta, her Jetta, and his Jetta are all instances of the Jetta class. Remember once again that a **class** is an **abstract data type** from which **objects** are created or **instantiated**.

As a blueprint, a class must first specify:

- name and data type of each attribute
- name and data types of input and output of each method

For example, a class Point may be designed as the following diagram (called “class diagram”) shows.



Note that almost all methods follow the same pattern: take *none* or more **parameters** (which are values or objects) as input, and return *none* or one single value or object as output. For instance, the `getDistanceFromOrigin` method takes no parameter and returns a double value; the `setName` method takes a String object as input and returns none.

Note also that every class has at least one method that has a different role and follows a different syntactical rule from all other methods. This special method is to instantiate an object, and thus called a **constructor**. A constructor has the same name as its corresponding class. For example, the Point

class has the Point constructor, which takes a String object and two double values and instantiates a Point object.

Here are more detailed descriptions of the attributes and methods of the Point class.

<Attributes>

- The **name** attribute is the name of the point (instantiated from our Point class).
- The **x** attribute is the x-coordinate of the point.
- The **y** attribute is the y-coordinate of the point.

<Methods>

- The **Point** constructor takes a name (String), x-coordinate (double) and y-coordinate (double) and instantiates an object of Point class (or simply, a Point object).
- The **getDistanceFromOrigin** method calculates the distance from the origin to the point and returns it.
- The **getDistanceToOrigin** method calculates the distance from the point to the origin and returns it.
- The **getDirectionFromOrigin** method calculates the direction from the origin to the point and returns it.
- The **getDirectionToOrigin** method calculates the direction from the point to the origin and returns it.
- The **setName** method takes a name (String) and sets the name attribute to it.

Programming in Java is all about designing classes and manipulating objects. The code of a class is stored in a file with the extension “**java**”. The converse is also true; that is, any file ending with .java stores the code of a class. So, how about HelloWorld.java? Yes, it does store a class. As we will learn later in the course, however, it is a peculiar kind of class.

1. HOW TO USE AN EXISTING CLASS

Point is the most fundamental geometric element and defined by a pair of coordinates (in a two-dimensional surface). It is also a very familiar kind of data to us as GIS users; we have collected points from GPS receivers or created points from addresses. We can handle points in Java. Before doing so, we need to know very basics of Java programming.

- 1) The syntax for declaring a variable as an object (or object variable for short):

Class name object variable name;

For example,

```
String aString;  
Point aPoint;  
Point anotherPoint;
```

- 2) The syntax for instantiating an object:

new Class name(parameters);

For example,

```
new String("Takeshi"); // this is rarely used, though.  
new Point("Home", 125, 331);
```

3) The syntax for assigning an object to an object variable:

object variable name = object;

For example,

```
aString = "Takeshi"; // more common than new String("Takeshi").
aPoint = new aPoint("Home", 125, 331);
anotherPoint = aPoint;
```

4) The syntax for getting access to an attribute of an object is:

object variable name.attribute name;

For example,

```
aPoint.name;
```

5) The syntax for applying to an object one of its methods with a required set of parameters is:

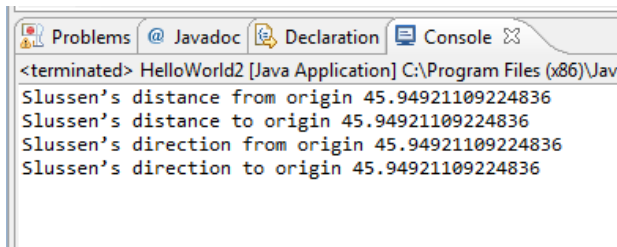
object variable name.method name(parameters)

For example,

```
aString.lenght();
aPoint.setName("Office");
```

Now let's use the Point class to improve the HelloWorld program and call it **HelloWorld2**. This improved version no longer says "Hello World!", but accepts user inputs specifying the name of a point of interest, its x-coordinate, and its y-coordinate, and prints on the console its distance from the origin (0,0), distance to the origin, direction from the origin, and direction to the origin.

The output may look like:



This is sample code for the HelloWorld2 program (HelloWorld2.java). It is also available at Canvas.

```
public class HelloWorld2 {

    public static void main(String[] args) {

        String locName; // name of a location of interest
        double xCrd;    // its x-coordinate
        double yCrd;    // its y-coordinate
        Point aPoint;    // point object that represents the location of interest

        locName = args[0];
        xCrd = Double.parseDouble(args[1]);
        yCrd = Double.parseDouble(args[2]);
        aPoint = new Point(locName, xCrd, yCrd);

        System.out.println(locName + "'s distance from origin " + aPoint.getDistanceFromOrigin());
        System.out.println(locName + "'s distance to origin " + aPoint.getDistanceToOrigin());
        System.out.println(locName + "'s direction from origin " + aPoint.getDirectionFromOrigin());
        System.out.println(locName + "'s direction to origin " + aPoint.getDirectionToOrigin());

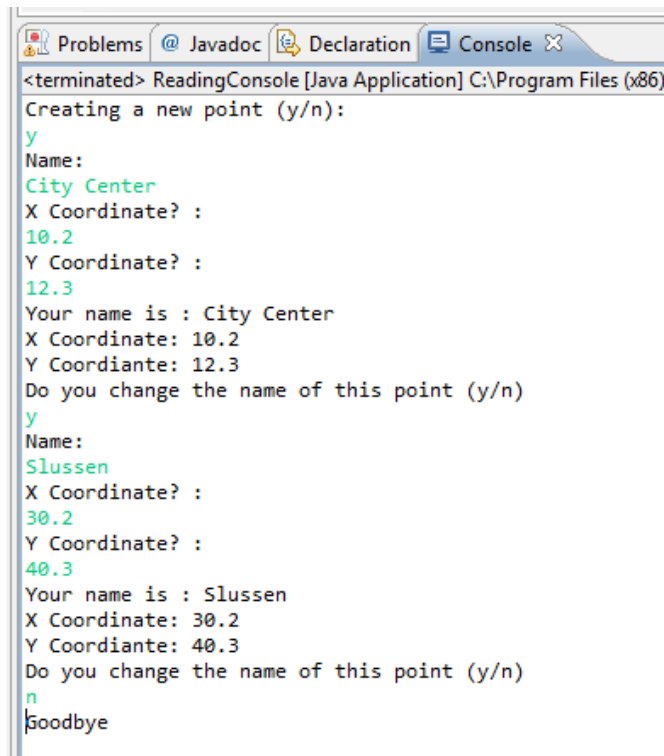
    }

}
```

For this to work, Point.java (available at Canvas) must be copied to the same folder as HelloWorld2.java is stored (i.e., **se.kth.ag2411.intro** package which we have created earlier). Otherwise, Java will have no idea what Point is.

The HelloWorld2 program does a much more meaningful thing than its predecessor. But it may be frustrating that it runs once and terminates. Why don't we give it a more interactive user interface? We actually have a further modified version, **HelloWorld3** (HelloWorld3.java), available at Canvas. This accepts user inputs of typing first the name of a point of interest, then its x-coordinate, and finally its y-coordinate. If the user inputs don't follow this format, the program will terminate. Otherwise, it will return some information on the point. And, it won't terminate there! Instead, the program will give us a chance to change the name of the current point of interest or to create a new point of interest.

The output should look like:



```
<terminated> ReadingConsole [Java Application] C:\Program Files (x86)
Creating a new point (y/n):
y
Name:
City Center
X Coordinate? :
10.2
Y Coordinate? :
12.3
Your name is : City Center
X Coordinate: 10.2
Y Coordinante: 12.3
Do you change the name of this point (y/n)
y
Name:
Slussen
X Coordinate? :
30.2
Y Coordinate? :
40.3
Your name is : Slussen
X Coordinate: 30.2
Y Coordinante: 40.3
Do you change the name of this point (y/n)
n
Goodbye
```

The HelloWorld3 program reads user input from the keyboard (e.g. those words in light green above), which is represented by a **Scanner** object. A Scanner object can be read an input text line by line (see below) or token by token, where a token is a portion of the text separated by white space.

```
Scanner input; // System.in is a stream of bytes coming from the
input = new Scanner(System.in); // keyboard, which is converted to lines of Strings.
String text; // stores each line of Strings temporarily.
text = input.nextLine(); // first line of Strings is read.
System.out.println("this is the first line: " + text);
text = input.nextLine(); // second line of Strings is read.
System.out.println("this is the second line: " + text);
```

2. HOW TO CREATE A NEW CLASS

In the above exercise, we were just users of the `Point` class. Was it one of those built-in classes which Java provides by default, like `String`? No, it was a class that my former teaching assistant made. Here is how he did that.

First he created in his project folder a new file `Point.java` to store the code for the `Point` class (see page 5). You do the same here, and copy and paste to it the contents of `Point.java` available on Canvas.

We will take a look only at selected part of the code. Let's begin with attributes.

```
// Attributes
public String name;
public double x;
public double y;
```

The first line is just a comment (indicated by two slashes `//`) and will be skipped by the Java compiler. The next three lines are short but sufficiently specify the attributes of the `Point` class. The key word `public` says that the corresponding attribute is accessible from other objects. Inaccessible attributes should be followed by the key word `private`. *But let's not worry about this public/private issue for now.* The rest is almost self-explanatory. For example, there is an attribute called `name`, whose data type is `String`.

The next one we look at is the `Point` constructor. As mentioned earlier, every class has a method to instantiate an object, and this special method shares the name with the class.

```
// Constructor
public Point(String name, double x, double y) {
    this.name = name; // name on LHS is attribute, name on RHS is input parameter
    this.x = x; // x on LHS is attribute, x on RHS is input parameter
    this.y = y; // y on LHS is attribute, y on RHS is input parameter
}
```

The constructor takes three parameters, `name`, `x`, and `y`, enclosed by a pair of parentheses, as input. These parameters are `String`, `double`, and `double` data types, respectively. The constructor instantiates a new object and assigns the three parameters to `name`, `x`, and `y` attributes. Note here that each attribute is assigned a value that happens to have the same name. Don't be confused.

Then what is `this`? `this` refers to the point object that is being instantiated. This is useful because we need to get access to the object to get access to its attributes (i.e. `name`, `x`, and `y`). Because of `this`, we know that `name`, `x`, and `y` of the left-hand-side is different from `name`, `x`, and `y` of the right-hand-side.

Lastly, we analyze the `getDistanceFromOrigin` method.

```
public double getDistanceFromOrigin() {
    double deltaX = x - 0;
    double deltaY = y - 0;
    double distance = Math.sqrt(Math.pow(deltaX, 2) + Math.pow(deltaY, 2));
    return distance;
}
```

The word `double` before the method's name implies that the output of this method is a `double` value.

There is nothing between the pair of parentheses after the method's name. This implies that this method does not take any input parameter.

A method of distance calculation is coded between a pairs of waved brackets. `deltaX`, `deltaY`, and `distance` are declared to store the difference between the x-coordinate of the point and that of the origin, the difference between the y-coordinate of the point and that of the origin, and the distance between the point and the origin, respectively. They are called **local variables** because their scope is local, meaning that they can be accessed only from the method in which they are declared. So, no methods but `getDistanceFromOrigin` can use `deltaX`.

Finally, note that the distance is calculated with the help of the `Math` class, one of many useful classes provided by Java. This class has, for example, the `sqrt` method, which performs the square root of any **double** value. If you type `Math.` in Eclipse Editor, you will see all other methods that the `Math` class offers.

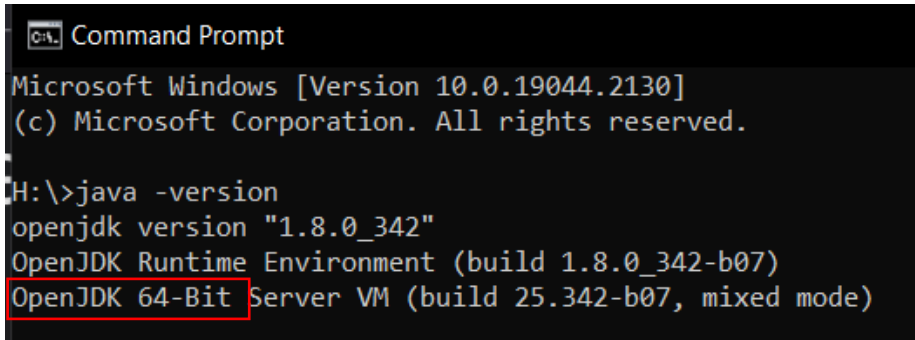
The last line of the code starts with the **return** keyword and the `distance` variable. This means that the `getDistanceFromOrigin` method returns the value assigned to the `distance` variable.

Appendix: Installing Eclipse

1. If your java compiler (JDK) is 64 bit then download 64 bit Eclipse edition.
If your java compiler (JDK) is 32 bit then download 32 bit Eclipse edition.

To find out which compiler you have, type following command to your command prompt:

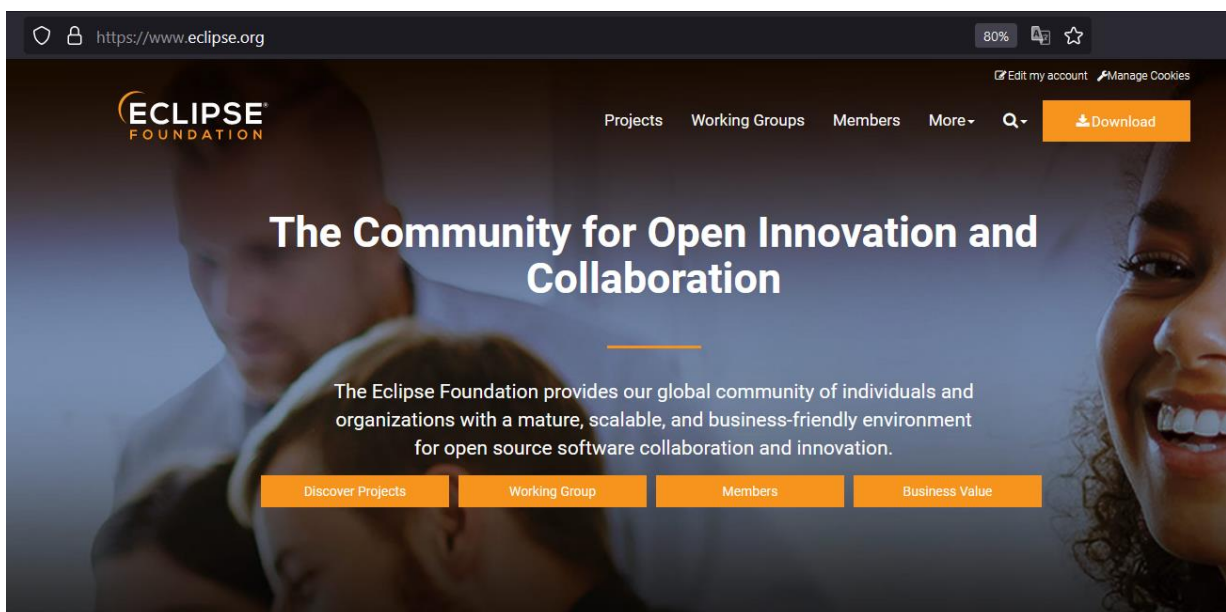
java -version



```
Command Prompt
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

H:\>java -version
openjdk version "1.8.0_342"
OpenJDK Runtime Environment (build 1.8.0_342-b07)
OpenJDK 64-Bit Server VM (build 25.342-b07, mixed mode)
```

2. To download the application for installing Eclipse
Go to <http://www.eclipse.org/>



3. Click on *Download Eclipse* or go directly to <http://www.eclipse.org/downloads/eclipse-packages/>

4. Download - Eclipse IDE for Java EE Developers. It is a zip package.

The Eclipse Installer 2022-09 R now includes a JRE for macOS, Windows and Linux.

Try the Eclipse **Installer** 2022-09 R

The easiest way to install and update your Eclipse Development Environment.

Find out more

1,413,505 Installer Downloads

710,350 Package Downloads and Updates

Download

macOS x86_64 | AArch64

Windows x86_64

Linux x86_64 | AArch64

Get **Eclipse IDE 2022-09**

Install your favorite desktop IDE packages.

Download x86_64

Download Packages | Need Help?

Eclipse IDE 2022-09 R Packages

Eclipse IDE for Java Developers

296 MB | 409,347 DOWNLOADS

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration

Windows x86_64

macOS x86_64 | AArch64

Linux x86_64 | AArch64

Eclipse IDE for Enterprise Java and Web Developers

502 MB | 208,812 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

Windows x86_64

macOS x86_64 | AArch64

Linux x86_64 | AArch64

Click here to open a bug report with the Eclipse Web Tools Platform.

Click here to raise an issue with the Eclipse Platform.

Click here to raise an issue with Maven integration for web projects.

Click here to raise an issue with Eclipse Wild Web Developer (incubating).

Eclipse IDE for C/C++ Developers

345 MB | 42,380 DOWNLOADS

An IDE for C/C++ developers.

Windows x86_64

macOS x86_64 | AArch64

Linux x86_64 | AArch64

RELATED LINKS

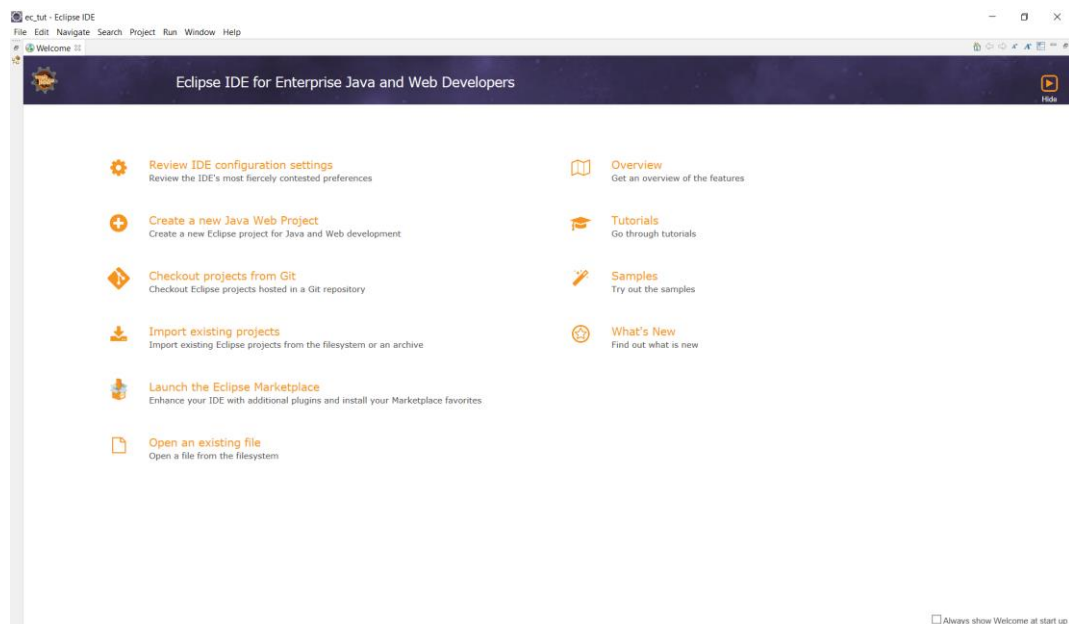
- Compare & Combine Packages
- New and Noteworthy
- Install Guide
- Documentation
- Updating Eclipse
- Forums
- Simultaneous Release

MORE DOWNLOADS

- Other builds
- Eclipse 2022-09 (4.25)
- Eclipse 2022-06 (4.24)
- Eclipse 2022-03 (4.23)
- Eclipse 2021-12 (4.22)
- Eclipse 2021-09 (4.21)
- Eclipse 2021-06 (4.20)
- Eclipse 2021-03 (4.19)
- Eclipse 2020-12 (4.18)

5. Un-zip this package and double click eclipse.exe

If the installation is successful, you are ready to program in java in the Eclipse environment.



Making a shortcut for desktop

You can make a shortcut on Desktop. To do so, right click on eclipse.exe and select **Send To > Desktop (Create Shortcut)**

