Lab Assignment 1: Writing, Compiling and Executing a Java Program

Course: Advanced Programming (EE423)
Institution: IGEE, Boumerdes University

October 13, 2022

1 Objectives

Within this lab, in the first part, you will learn how to set up your Java environment on an operating system and start coding. You will learn the difference between a JDK[1] and an IDE[2] like Eclipse. You will also learn how to write your first object oriented program using the Java language and see how to compile it and run it by passing parameters. You will also learn how to read the generated compilation as well as run-time error messages that will help you debug and fix your code.

In the second part, you will write basic Java classes and you will see how to use encapsulation to protect variables. You will also learn how to instance a class by creating and manipulating objects.

2 Assignment

2.1 Setting up the environment

2.1.1 Obtaining the JDK

Before you can compile and run Java programs, you must have the Java Development Kit (JDK) installed on your computer. The JDK is available free of charge from Oracle. It can be downloaded from this link[3].

The JDK supplies two primary programs.

javac which is the Java compiler. It translates your code into Bytecode that can be run on the JVM[4].

java which is the standard Java interpreter and is also referred to as the application launcher.

[1] Java Development Kit
[2] Integrated Development Environment
[3] www.oracle.com/technetwork/java/javase/downloads/index.html
[4] Java Virtual Machine

Note: In the newer versions of Java, when you use the command *java*, it will automatically call the command *javac*. In other words, your code will be compiled and ran with one *java* command.

The JDK runs in the command prompt environment and uses command-line tools. It is not a windowed application. It is also not an Integrated Development Environment (IDE). There are several high-quality IDEs available for Java, such as *NetBeans* and *Eclipse*. An IDE can be very helpful when developing and deploying commercial applications.

2.1.2 Write your first Java Program

In Java programs, classes are the basic building blocks. Java classes have two primary elements:

methods often called functions or procedures in other languages,

fields more generally known as variables.

Together these are called the members of the class. Variables hold the state of the program, and methods operate on that state.

Let's start by typing, then compiling and running the short sample program shown here:

```
/* This is a simple Java program */
class Example {
   // a java program begins with a call to main()
   public static void main(String args[]){ // a method
      String message = "Hello world!";   // a variable
      System.out.println(message);
   }
}
```

You can use any text editor like notepad on windows to write the previous code. The name you give to a source file, when saving it, must be the same with the name of the class, including capitalization, because Java is case sensitive. In this case, it must be named *Example.java* and it must have the *.java* filename extension. This convention makes it easier to organize and maintain your programs. In Java, a source file is officially called a compilation unit. It is a text file that contains one or more class definitions.

2.1.3 Compile the program

To compile the *Example.java* program, execute the compiler, *javac*, specifying the name of the source file on the command line, as shown here:

```
javac Example.java
```

To run the previous command, start cmd on Windows. Navigate to the folder where you saved the file Example.java. You can use the commands:

dir show the files contained in a directory

cd .. go up the current directory

cd C: go to another partition(C) in Windows

cd nameFolder go into the directory nameFolder

> Note: If, when you try to compile the program, the computer cannot find javac (and assuming that you have installed the JDK correctly), you may need to specify the path to the command-line tools. In Windows, for example, this means that you will need to add the path to the command-line tools to the paths defined for the PATH environmental variable. For example, if JDK 9 was installed under the Program Files directory, then the path to the command-line tools will be similar to:
>
> `C:\ProgramFiles\Java\jdk-9\bin.`
>
> (Of course, you will need to find the path to Java on your computer, which may differ from the one just shown. Also the specific version of the JDK may differ.)

The javac compiler, if no errors were found, creates a file called *Example.class* that contains the *bytecode* version of the program. Remember, bytecode is not executable code. Bytecode must be executed by a Java Virtual Machine. In this case, when you try to compile the code you get a compilation error.

2.1.4 Debug the program

When you compile the class *Example.java* you get an error. That's the compiler telling you that there is an error in your code. Read the error given to you by the compiler and try to fix the code. From the error message we can see that there is a problem with the variable message. Here is a fix:

```
/* This is a simple Java program */
class Example {
   // a java program begins with a call to main()
   String message = "Hello world!"; // solution 1:
                                    // declare it as a class
                                    // variable
   public static void main(String args[]){
      //static String message = "Hello world!"; //solution 2:
                                                // mark it static
      System.out.println(message);
   }
}
```

Now, when you compile your code you should get a file with a *.class* extension.

2.1.5 Run the program

To actually run the program, you must use the Java interpreter, *java*. To do so, pass the class name *Example* as a command-line argument, as shown here:

```
java Example
```

When the program is run, the following output is displayed:

```
Hello world!
```

When Java source code is compiled, each individual class is put into its own output file named after the class and using the .class extension. This is why it is a good idea to give your Java source files the same name as the class they contain (the name of the source file will match the name of the .class file).

When you execute the Java interpreter as just shown, you are actually specifying the name of the class that you want the interpreter to execute. It will automatically search for a file by that name that has the .class extension. If it finds the file, it will execute the code contained in the specified class.

2.1.6 Passing parameters

```
public class Zoo {
   public static void main(String[] args) {
      System.out.println(args[0]);
      System.out.println(args[1]);
      System.out.println(args[2]);
   }
}
```

Compile the class Zoo and run it as we did previously. This time you will get a run-time error and not a compilation error. The reason is the class Zoo expected parameters as input which we did not pass.

In Java, to run a program by passing parameters, type something like this:

```
$ java Zoo "San Diego" Zoo 2
```

Use double quotes to include spaces in the parameter ”San Diego”. Notice that 2 is treated as a String here and not as an integer. The output:

```
San Diego
Zoo
2
```

The keyword static binds a method to its class so it can be called by just the class name, as in, for example, Zoo.main(). If a main() method is not present in the class we name with the .java executable, the process will throw an error and terminate. Even if a main() method is present, Java will throw an exception if it is not static.

2.1.7 Installing an IDE (optional)

As you may have noticed. It is time consuming to go through the compilation and execution steps each time your write or change a class. Fortunately, there are tools that automate this process like Eclipse, Netbeans, Intellj, or VSC. For the coming lab assignments, it is highly recommended that you pick an IDE of your preference, and get acquainted with it.

2.2 Manipulating classes and objects

2.2.1 Creating classes

Let's create our first class called *Point*. This class represents the x, y coordinates. The goal is to encapsulate these two real values within a class and provide getters and setters to access those two variables. When we want a new point, we can either create an object and initialize those values with 0 or initialize them with the values that we pass as parameters. Let's see how this is done.

First, let's encapsulate two real values within a class called *Point*.

```
class Point {
 double x, y;
}
```

2.2.2 Using access specifiers

To prevent those values from being manipulated outside of the class we have to protect them by adding the keyword *private*.

```
class Point {
 private double x, y;
}
```

The next step is to create another class in which we will be creating our points objects. Let's call it *ManipulatePoints*.

```
class ManipulatePoints{
 public static void main(String[] args){
  Point pt = new Point();
 }
}
```

Here, the object referenced by pt will have x=0.0 and y=0.0 because instance variables are initialized by default to zero. Since we encapsulated the variables with the keyword private, if we try to access them (read or write) directly from a different class, we will get a compilation error. Let's try this code:

```
class ManipulatePoints{
 public static void main(String[] args){
  Point pt = new Point();
  pt.x = 0; // compilation error
  pt.x = pt.y; // compilation error
 }
}
```

2.2.3 Providing getters and setters

To be able to read the values from the class *ManipulatePoints* we will have to add getters for x and y. They must be *public* so that we call them from different classes.

```
class Point {
  private double x, y;
  public double getX(){return x; }
  public double getY(){return y; }
}
```

We have added two getters, one for each variable. A getter is a function that has as a name *get* appended to it the name of the variable, with its first character capitalized (*getX* and *getY* in this case). For example, a getter for a variable called *name* would be *getName()*. The getter function returns the value of the instance variable so its return type must be the same with the type of the variable.

Hint: if you are using a modern IDE, you can always call a special function to generate getters and setters.

Now we can access the variables x and y through their getters.

```
class ManipulatePoints{
 public static void main(String[] args){
  Point pt = new Point();
  System.out.println(pt.getX());
  System.out.println(pt.getY());
 }
}
```

Now that we can read the values, we will add setters so that we can modify them from outside the class.

```
class Point {
 private double x, y;
 public double getX(){
  return x;
 }
 public double getY(){
  return y;
 }
 public void setX(double newX){
  x = newX;
 }
 public void setY(double newY){
  y = newY;
 }
}
```

A setter is a function that takes as a parameter a value that will be given to the variable. This is why the type of the parameter must be the same with the type of the variable. Since we don't get anything back, its return type is *void*. The name of the function follows the same logic with the getter except it's *set* appended to the name of the variable. Let's give the point the values (x=4,y=3)

```
class ManipulatePoints{
  public static void main(String[] args){
    Point pt = new Point();
    pt.setX(4);
    pt.setY(3);
  }
}
```

2.2.4 Adding a constructor

The last requirement is to provide an additional *constructor* that allows us to create objects and initialize its variables x and y with specific values passed when the object is created. The constructor does not have a return type and its name is identical with the name of the class. Here is how it's done.

```
class Point {
 private double x, y;
 public Point(double xv, double yv){
  x = xv;
  y = yv;
 }
 public Point(){
 }
 public double getX(){
  return x;
 }
 public double getY(){
  return y;
 }
 public void setX(double newX){
  x = newX;
 }
 public void setY(double newY){
  y = newY;
 }
}
```

The names of the variables xv and yx can be any valid java identifiers.

Now let's see how we can create points, set and get their variables and print them.

```
class ManipulatePoints{
  public static void main(String[] args){
    Point pt1 = new Point();
    pt1.setX(4);
    pt1.setY(3);
    Point pt2 = new Point(6, 7);
    System.out.println(pt2.getX());
    System.out.println(pt2.getY());
  }
}
```

2.2.5 Exercise 1

Add a method *print* to the class *Point* that prints the coordinates. The format should be (x,y). For example, (4,7). In Java, use + to concatenate two strings. For example to print (,), you can either write:

```
System.out.println("(,)");
//Or
System.out.println("(" + "," + ")");
```

In Java, if we want the Garbage Collector (GC) to get rid of an object we assign a *null* value to its reference.

```
class ManipulatePoints{
  public static void main(String[] args){
    Point pt = new Point();
    pt = null; //it can be collected by GC
  }
}
```

2.2.6 Exercise 2

Create a class called *Rectangle*. It has as attributes *width* and *height*. Add its setters and getters. Define also a *constructor* that can be used to set the values of *width* and *height*. Add a *public* method called *getArea* that calculates the area of the rectangle

2.2.7 Exercise 3

Create a class called *Circle*. It has one attribute called *radius*. Add its setter and getter. Add a *constructor* that can be used to set the value of *radius*.

2.2.8 Exercise 4

Create a class called *ManipulateShapes*. Add the main function to it. Within this class create an object of type *Rectangle* and set *width=4* and *height=5* using the constructor. Print the values of the rectangle. After this make this object eligible for the garbage collector. Within the same main function, create an object of type *Circle* with a default constructor. After the creation of the object set its radius to 10. Print the radius of the circle.

2.2.9 Exercise 5

Create a class called *Student*. It has as attributes *first name*, *last name*, *ID* (12 digits), *date of birth*, and *graduated* (boolean). Make all these attributes *private*. Add a main function to this class and create an object called student. Set the ID of the student to 123456789000, first name to Joe, last name Bob and graduated to false. Finally, print the student's ID, first name, last name, and his/her graduation status.

3 Conclusion

After finishing this assignment, you now know how to write and compile java programs. You should be able to spot compilation and run-time errors and understand them. You may now install your favorite IDE and start coding using a modern object oriented programming language.