Universidad Carlos III de Madrid

Eclipse Tutorial

Algorithms & Data Structures

Algorithms and Data Structures (ADS) lecturers

Content

1. Defining a Workspace

Eclipse is an open source Java development environment supported by a Graphical User Interface (GUI).

1.1 Workspace

The first time we run Eclipse (and the following ones if we want so), we are asked about the path to the desired workspace. It is the place in our hard drive where Eclipse will store user generated code (.java files) and compiled code (.class files) along with all the information regarding the Java project we are developing.

It is recommended for students to create the workspace directly in a USB memory stick if there is one available. In this way, all the work done by students during a session will go with them once finished. For this purpose, it is only necessary to write or select a path in the pen drive when launching Eclipse.

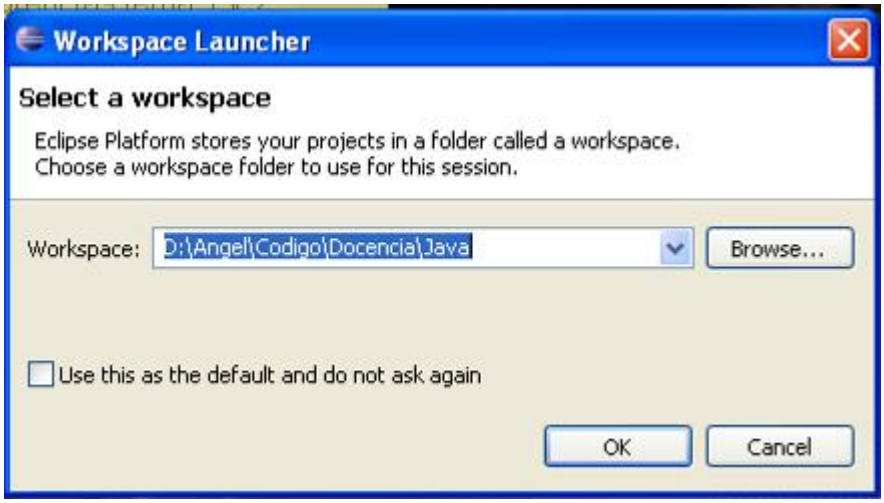

Figure 1: Selecting a workspace

1.2 Java Projects

Eclipse manages Java developments through *projects*.

1.2.1 Creating projects

Before being able to write code in Eclipse it is needed to create a project. To do so, follow these steps:

1. Select option menu *File* >> *New* >> *Java Project*
2. Fill in data for your project. For simple projects, like the ones to be developed along this course, filling in the name of the project is enough. Create a project named 'EDALab'.
3. Once project data have been introduced, press *Finish* button.

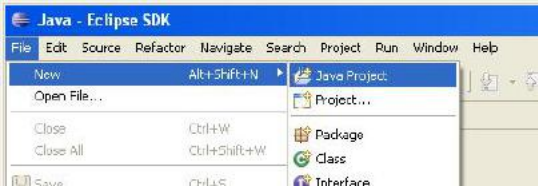

Figure 2: Creating a project.

Eclipse will create a new folder in the workspace for the project with *.classpath* and *.project* files. These files do not contain source code but data about configuration, compilation options and so on, for the given project. Java source code will be stored in a subfolder named 'src'

under the project folder while the compiled code (.class files) will be stored in the subfolder 'bin'. It is possible to change the current workspace by selecting *File >> Switch WorkSpace*.

1.3 Packages

A package is a special folder inside a project including a group of classes that have been created into the project. If no package is given when creating a class, the *default package* is used. Nonetheless, it is recommended to create at least one package per project.

The steps to follow when creating a package are:

1. Select *File>> New >> Package.*
2. Name the package in the dialog window appearing. By convention, all packages in Java are named starting with a non-capital letter. As an example, for the purpose of this lab work, you could create a package called 'maths'.

1.4 Classes

To create a Java class being part of the project you are developing, you must follow the following steps:

1. Select File>> New >> Class.
2. Name the class in the dialog window appearing. You must also say in which package the class must be included. By convention, all Java classes start with a capital letter. As an example, you could create a class called 'Calculator'.
3. It is needed to explicitly mark if a `main` method is desired by selecting the right check box.
4. Select *Finish* button.

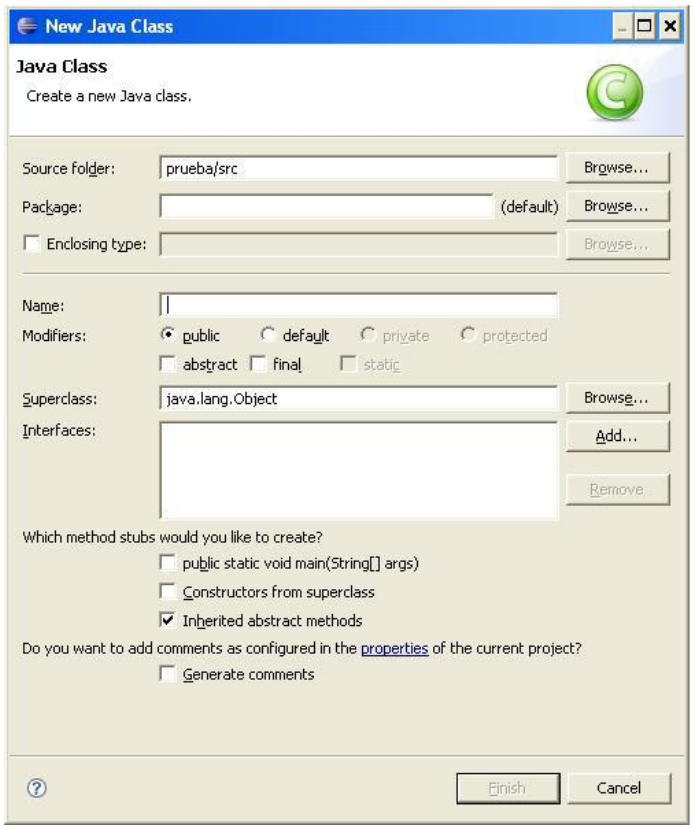

Figure 6: Dialog window to create a class

Eclipse will immediately create a file with the name `<class_name>.java` with the structure for the class (i.e.: the definition of the header for the class according to the elements selected in the

Class creation dialog box). Then, the rest of the code for the class must be provided. For our purposes, you can use the following code:

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

2. Running a project in Eclipse

Once a project has been coded, it can be run by selecting the option *Run >> Run* or by pressing *Control + F11*. When execution is started Eclipse will show the console (at the bottom of the Eclipse window) with the execution result.

2.1.1 Using running parameters

It is possible to include arguments when calling the main method included in the class by selecting *Run >> Open Run Dialog.* In the *Arguments* tab it is possible to include the desired arguments list. Each of these arguments (separated by a blank space or a carriage return) will be stored in adjacent positions of the *args* array. So, the first argument will be stored in *args[0]*, the second one in *args[1]* and so on.

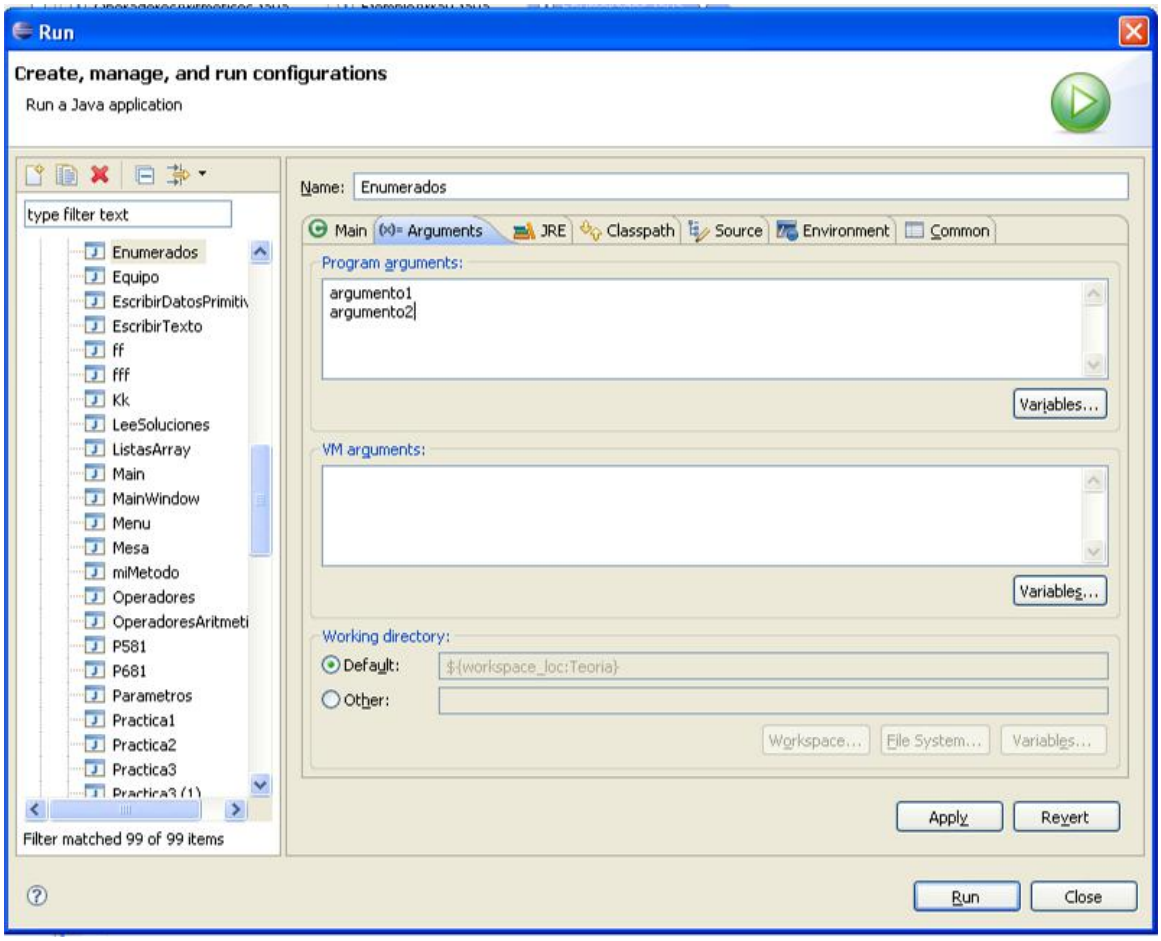

Figure 7: Using arguments when running Java applications

Please, notice that if no `main` method is given in a class implementation there is nothing to run, so an error will appear.

2.1.2 Debugging programs

Eclipse workspace is distributed along the concept of *perspective*. Each of these perspectives shows different functionalities according to a given task. For example, the Java perspective it is useful to include source code and running programs while the Debug perspective it is useful when debugging source code. The active perspective in the Java environment can be changed in *Window >> Open perspective* or by clicking in the button appearing in the right upper corner. As already mentioned, to introduce Java source code the Java perspective must be selected.

Once in the Debug perspective, program execution can be debugged by using breakpoints' i.e., source code lines where the execution process will stop. At these breakpoints it is possible to see the data contained in any of the variables or data structures used in the source code. Menu option *Run >> Toggle BreakPoint* is used to mark a given line and stopping running execution at that line. Once breakpoints have been selected, the debugging process can be launched with the option *Run >> Debug*. Program execution will stop at defined breakpoints.

3. Reusing projects in Eclipse

3.1.1 Exporting projects

Exporting and importing projects it is a core functionality that every Eclipse user should know. Thanks to these functions it is possible for students to work in the same projects at the classroom and at home.

Storing an active project in a .zip file can be done following these steps:

1. Select the desired project to be exported in the *Package Explorer* window – *(Java Perspective)*
2. Select the option menu *File >> New >> Export* (or click on the option *>>Export* appearing in the contextual menu shown by clicking in the right mouse button). See Figure 3.
3. To use a .zip file as a container for the exported project select *General >> Archive File* and press *Next* button.
4. Mark the contents you want to export. For example, it is recommended NOT to export the 'bin' folder because it contains .class files that can be obtained again by compiling the project. In this way you can save disk space. Press Browse button to select the path and .zip file where the project will be exported. Be sure that you have selected to store the project as a .zip file and that '*Create only selected directories'* option is marked. Press *Finish* to perform the export process. (see Figure 4).

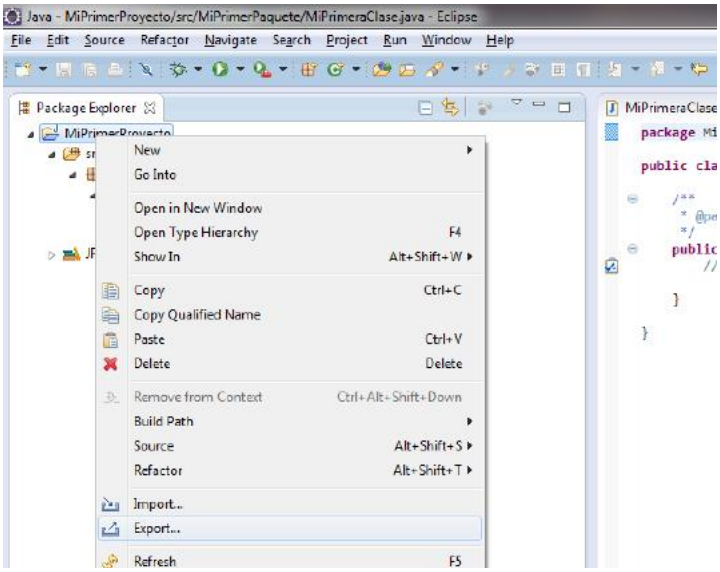

Figure 3: Exporting a project.

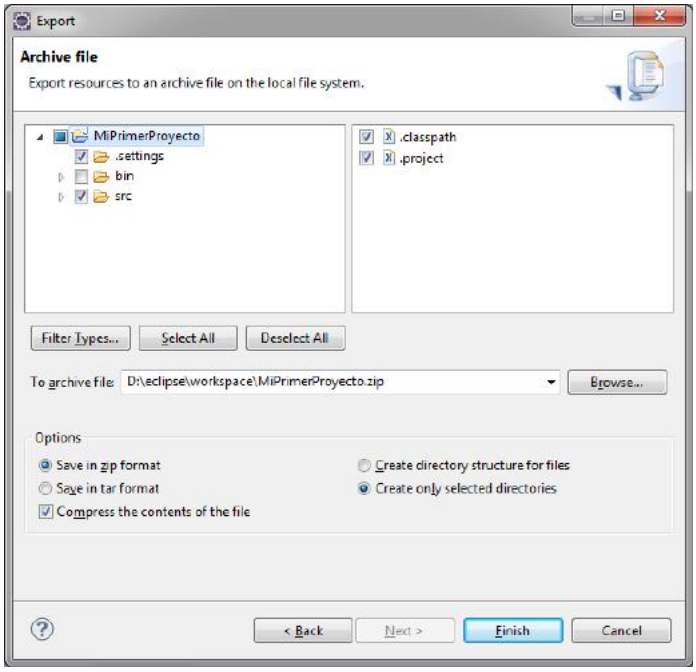

Figure 4: Export to file window

WARNING: It is also possible to export packages, classes, and so on instead of exporting the whole project. To do so you only need to place your pointer (in step 1) in the element you want to export instead of the project folder.

3.1.2 Importing projects

To import a previously exported .zip project (see section 3.1.1) you only have to take the following steps:

1. Create a project
2. Select *File >> New >> Import* (or press right button and select *>>Import* when the mouse is just over the just created project). See Figure 5.
3. Select *General >> Existing Projects into Workspace* and press *Next* button to continue.
4. Select the input .zip file through the option *Select archive File* (located where the project has been previously exported) and press *Finish* button (see Figure 6)

5. Check that all packages, classes and so on have been correctly imported.

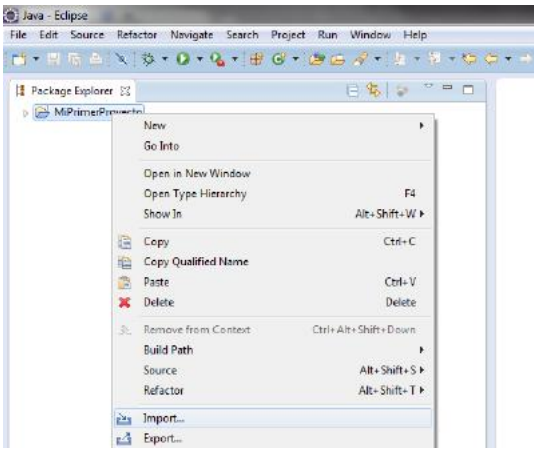

Figure 5: Importing a project.

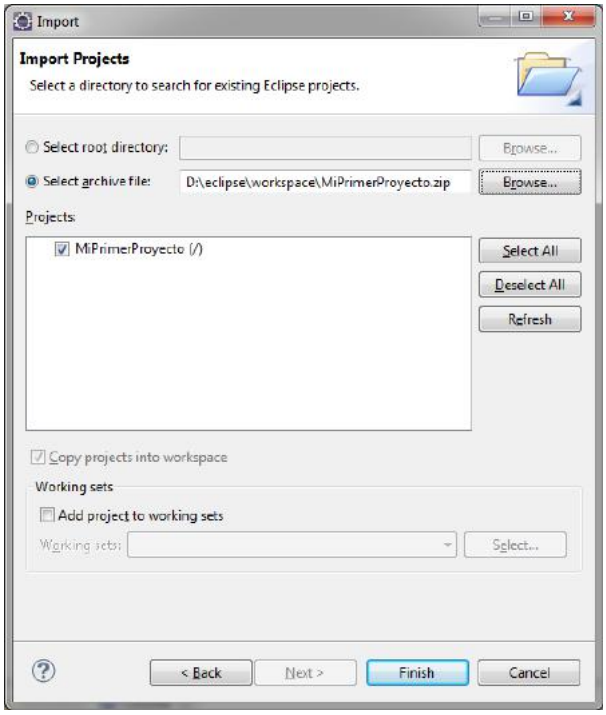

Figure 6: Importing a project from an external .zip file.

WARNING: To successfully import a previously exported project you must be sure that, when the project is deleted after exporting, the files in the hard disk that are part of the project have also been deleted.

3.1.3 Libraries

The most common way of using third party code is by using libraries. A library is a set of compiled code (structured in classes and, probably, including dependencies with other software components) providing a given functionality, developed and deployed by a third party. A library in Java is a .jar file, i.e.: a kind of compressed file containing compiled code (and, sometimes, also source code) ready to be run on any Java environment.

3.1.3.1 Creating a library

Any Java project can be included in a .jar file to be deployed to third-parties. The steps to create a library from a Java project are:

1. Right-click on the name of the project and select the Export option (see Figure 8)
2. Select the option Java >> JAR file. The dialog window shown in Figure 7 will appear.
3. Select the resources you want to export and the .jar file where you want to store the result.

3.1.3.2 Using a third-party library

The use of libraries in Eclipse is managed in the option menu Project >> Properties. A dialog box, shown in Figure 8, appears with several configuration sections related to the project. The library management section can be reached by clicking on 'Java Build Path' and select the 'Libraries' tab. To include an external library in your project follow these steps:

1. Include the JAR file with the library you want to use in a folder in your project (it is recommended to create a 'lib' folder in your project for this purpose).
2. Click on 'Add JARs ...'

Now, you can make reference to classes included in that library by inserting the corresponding 'import' directive in your source code.

So, there are two ways of using external code in our programs: by importing a project (if we have access to the source code) and by using libraries. The preferred way to use third-party code in our programs is by using libraries, even if you have access to the source code. Importing a project is needed when you need to change the source code of the third-party library but it is not the usual situation.

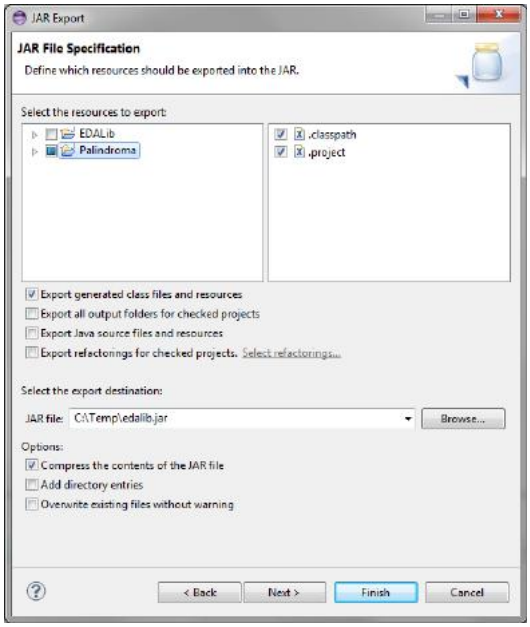

Figure 7. Creating a JAR file for a given project.

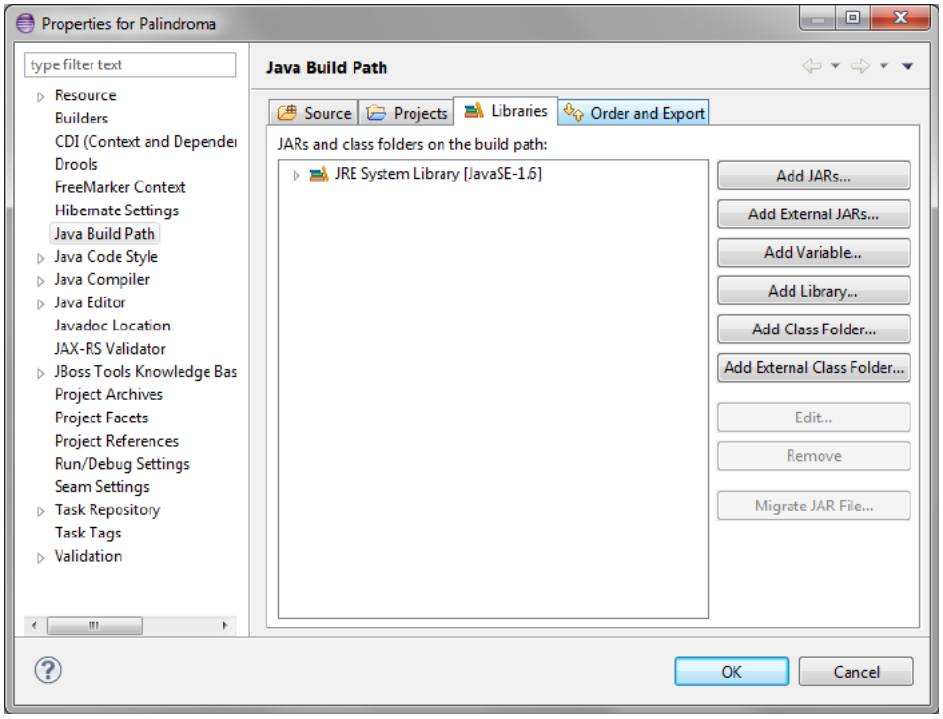

Figure 8. Importing a JAR file to be used in a project.

4. Related additional links

http://eclipsetutorial.sourceforge.net/totalbeginner.html

http://eclipsetutorial.sourceforge.net/Total_Beginner_Companion_Document.pdf

http://eclipsetutorial.sourceforge.net/totalbegginer01/lesson01.html

5. Exercise. Make your work environment ready

This exercise will be devoted to get familiar with the Eclipse IDE:

1. Create a project called *EDALab*. This project could be used to store the different exercises that will be developed during lab sessions.
2. Create a package called ‘maths’.
3. Create a class called Calculator and provide source code for the methods ‘sum’.
4. Create a main method in Calculator class to sum to numbers given as arguments. What is the difference between using a main method and not using it?
5. Export your project to a .zip file
6. Import your project from the .zip file (remember that, before importing, you must delete the project from Eclipse).
7. Export your project as a library (for example you can name it ‘maths’).
8. Create a new project called *OperationsProject*.
9. Implement a class called ‘Addition’ with a method called ‘add’ to obtain the addition of two numbers by using the ‘maths’ library.
10. Remember: you are working in a classroom. Many people use this classroom and the computers available. You will likely not found your source code the next day you come back to class so, SAVE YOUR work frequently and, again, before leaving the class!! DO NOT FORGET TO carry your work with you (export the project in a .zip file and save it in a secure place).

11. It is recommended for you to test the exporting process by importing your project again (remember that, before that, you must delete the project from Eclipse).