CS–313
Object Oriented Analysis and Design (OOA&D)
Lab # 01

Objectives:
- How to install and configure JDK (Java Development Kit)
- How to install and use Eclipse ID for Java
- How to program in Java (Hello World)
- How to declare/define variables and use along with operators

1. Java

Java Slogan
"Write once, run anywhere"

Why Java?
The computer world currently has many platforms. This has its pros and cons. On the one hand it gives more choices to people; on the other hand it becomes more and more difficult to produce software that runs on all platforms. With its Java Virtual Machine and API, the Java Platform provides an ideal solution to this as shown in Figure 2: Java independent of platform. The Java Platform is designed for running highly interactive, dynamic, and secure applets and applications on networked computer systems.

For the end users, the platform provides live, interactive content on the World Wide Web, with just-in-time software access. Applications are readily available on all operating systems at once. Users do not have to choose operating systems based on the applications; they can run the applications on their favorite machines.

Developers can develop applications on one platform to deliver to that same platform -- the Java Platform, which is available on a wide variety of OS and hardware platforms. This reduces the developing cost.

Basic Execution Cycle of Java:
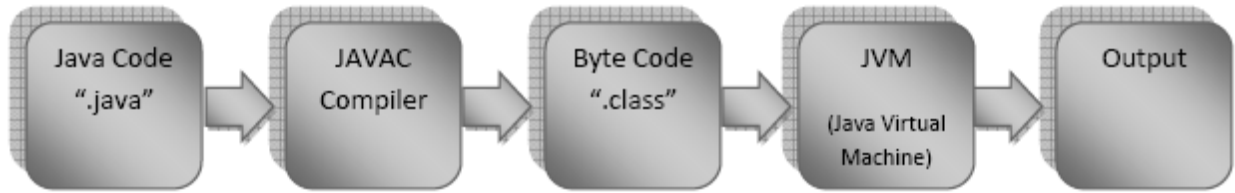Cycle of Execution of a java Program is shown in the figure below.
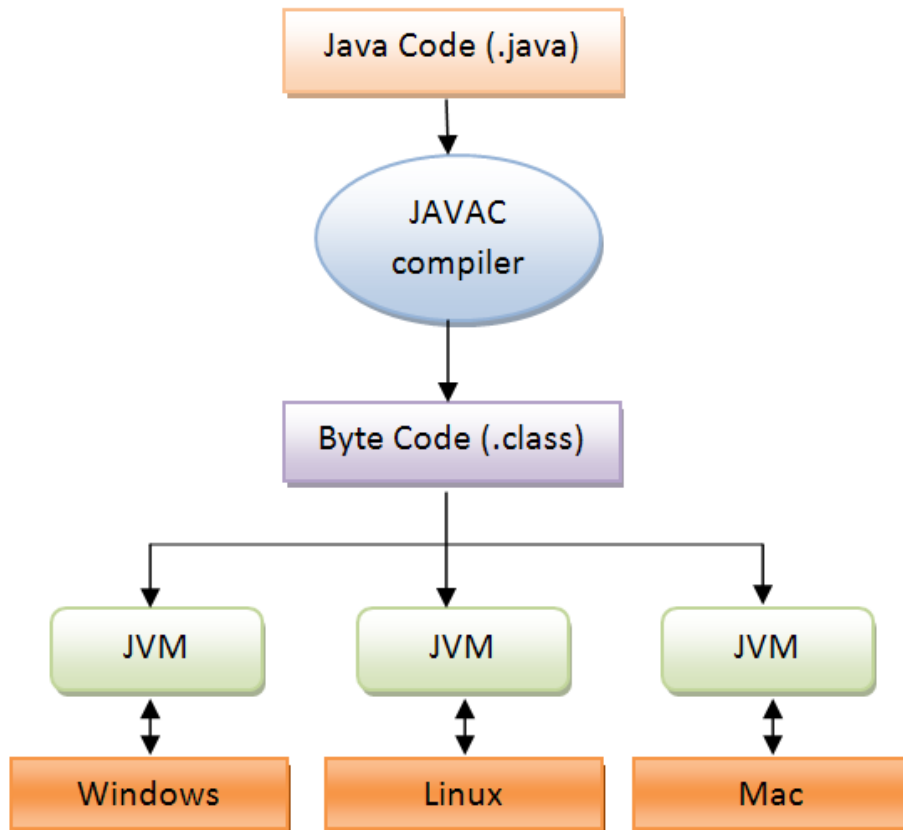
Figure 1: Java Execution Cycle

Figure 2: Java Independence of Platforms

1.1. Installation of Java

Java might already be installed on your machine. You can test this by opening a console (if you are using Windows: Win+R, enter cmd and press Enter) and by typing in the following command:

```
java -version
```

If Java is correctly installed, you should see some information about your Java installation. If the command line returns the information that the program could not be found, you have to install Java.

1.2. Downloading and Installing J2SE Software on Windows Platform

To download J2SE for development visit http://www.java.sun.com/j2se and download J2SE on your machine. The java 2 Platform or (JDK) can be downloaded from the sun. Formerly known as the java Development kit, or JDK, Downloading java is really about downloading the java 2 platforms that come in three editions, J2ME, J2SE and J2EE. If you are learning java then you should start by downloading J2EE.

Once you have downloaded the j2se on your system, you are ready to install. In the following section we will learn how to install jdk development environment on your machine. Here are the step to install JDK on your windows machine.
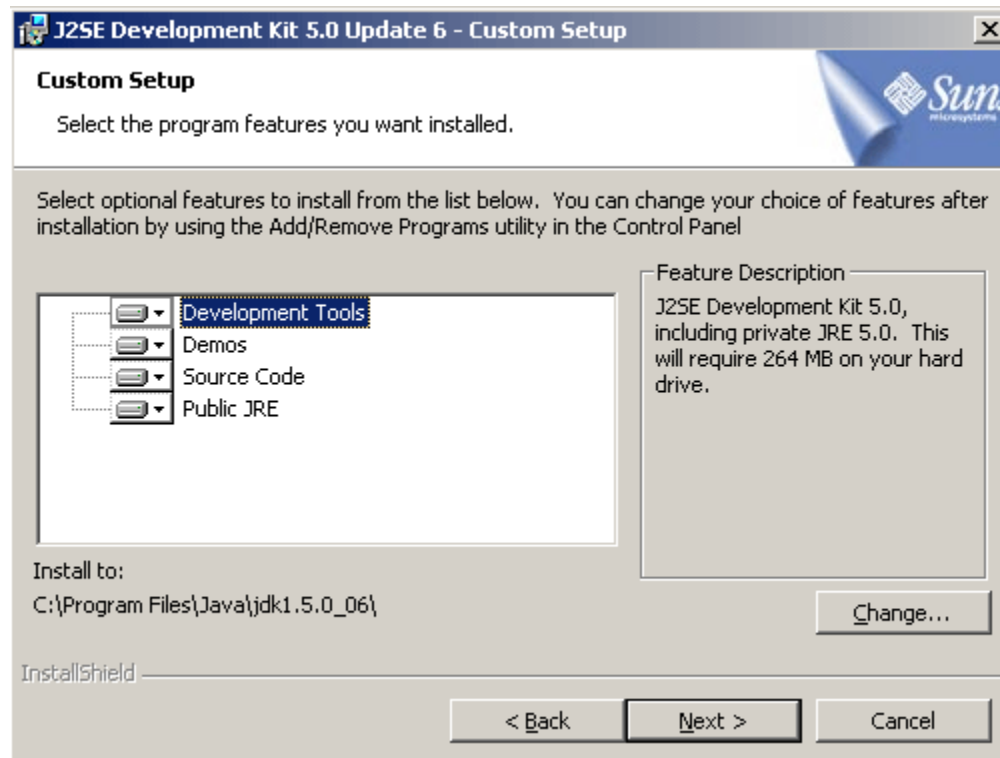
Step 1

Double click the JDK downloaded file, the executable extracts the required contents to the temporary directory and then License agreement screen appears. On the license agreement page read and accept the license and the click the next button .
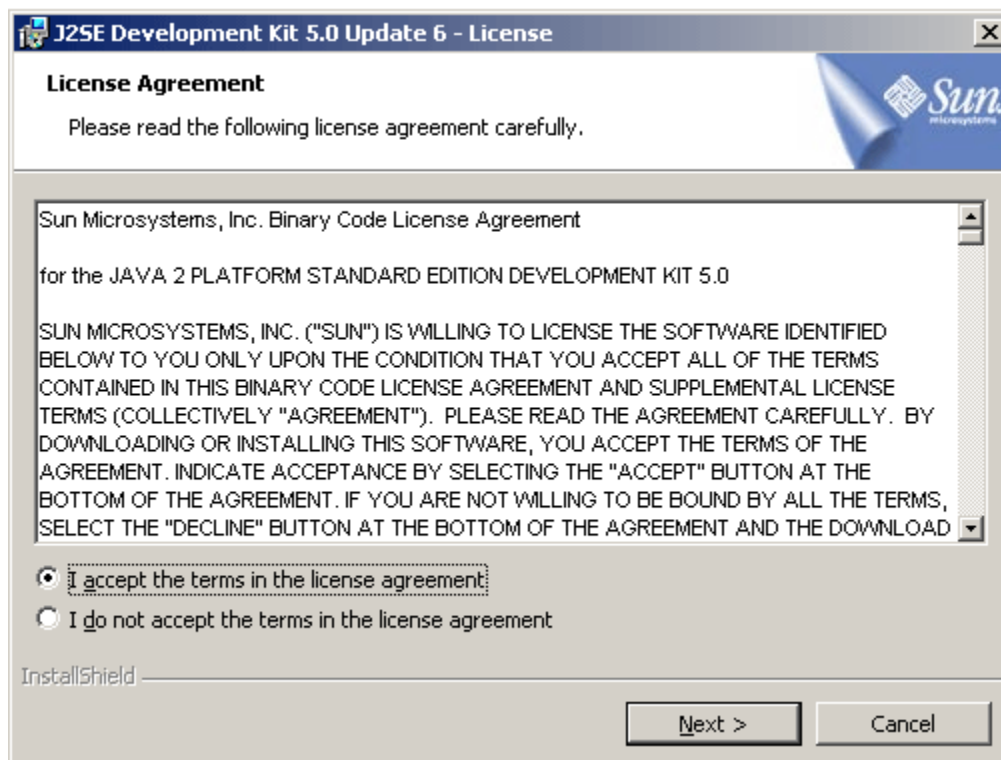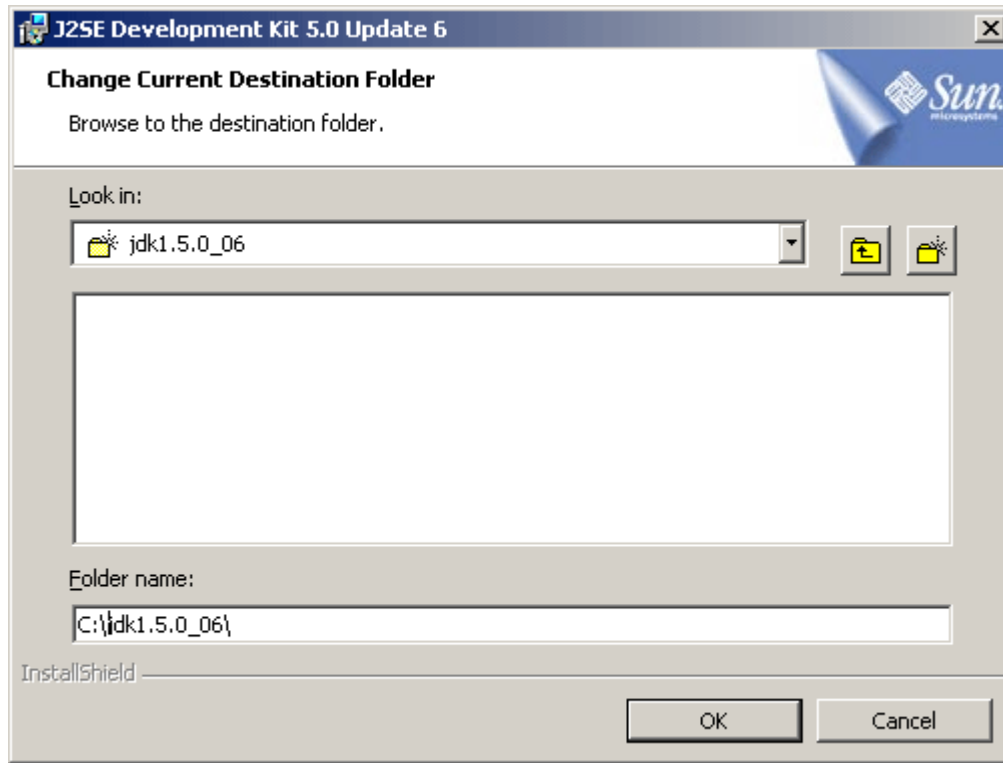
Step 2

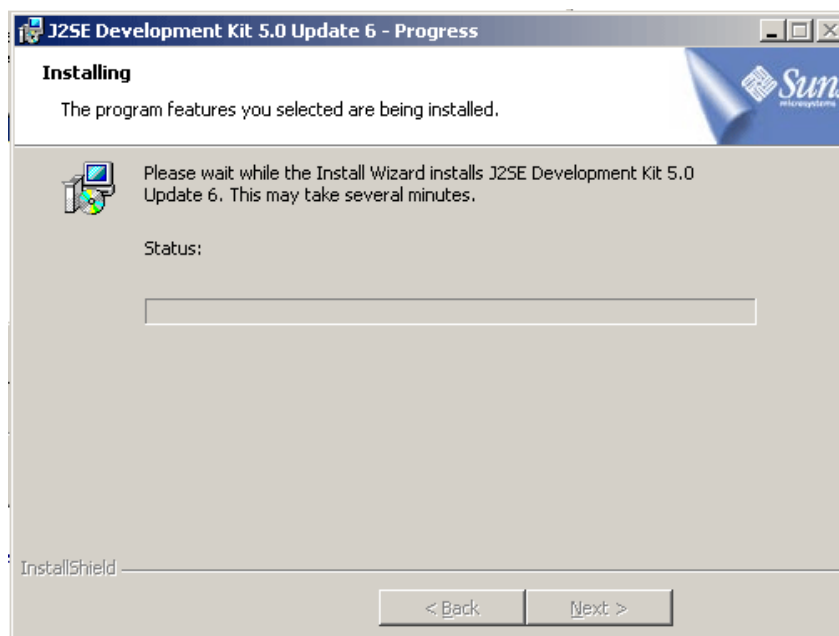The custom setup screen appears as follows.

Step 3

Click on the change button to change the installation directory to "**c:\jdk1.5.0_06**" as shown in the following screen.
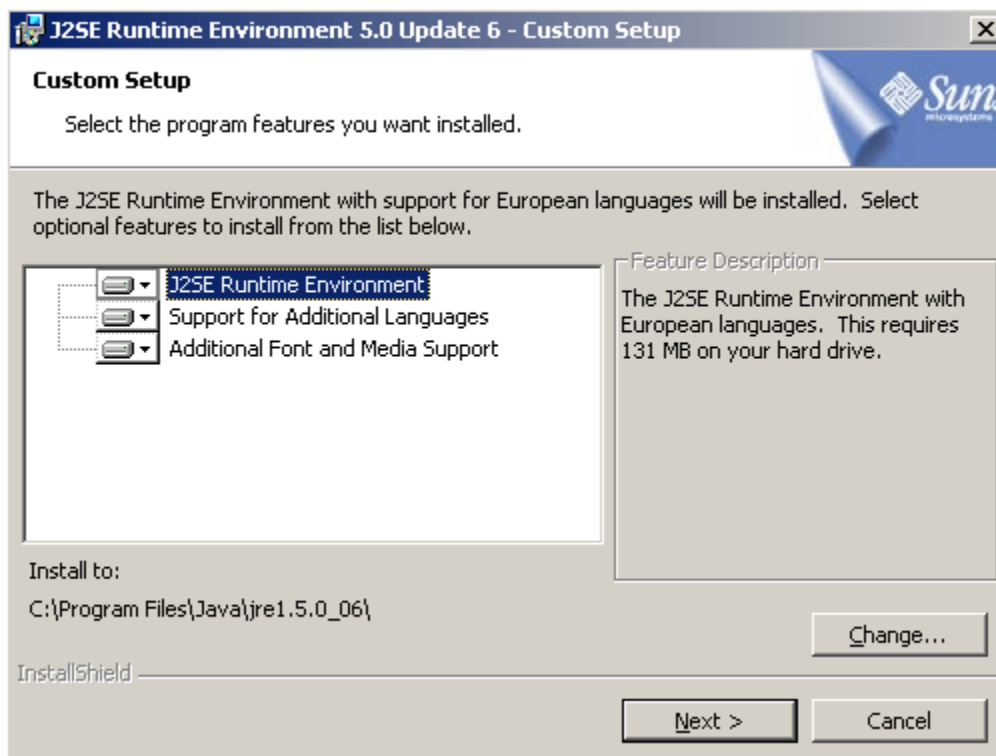

and click on the "OK" button. After clicking on the "OK" button installation begins:
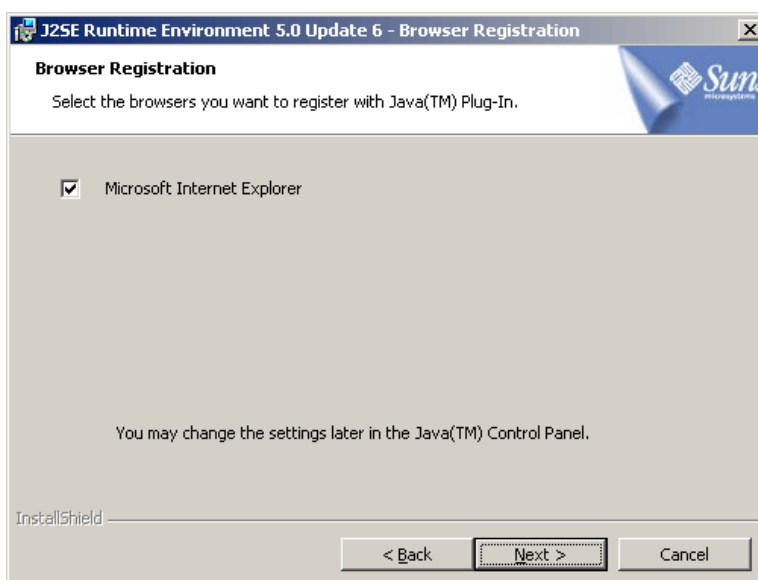
Step 4

In the next window installer asks for the installing the runtime as shown in the following screen:

Step 5

Click on next button install the J2SE runtime on your machine. Next screen shows the browser selection:

Click on the "Next" button.

Step 6

Once the installation is finished it shows you the final screen indications the success. Now you have successfully installed J2SE on your machine. Installer shows the following final confirmation window as shown below:
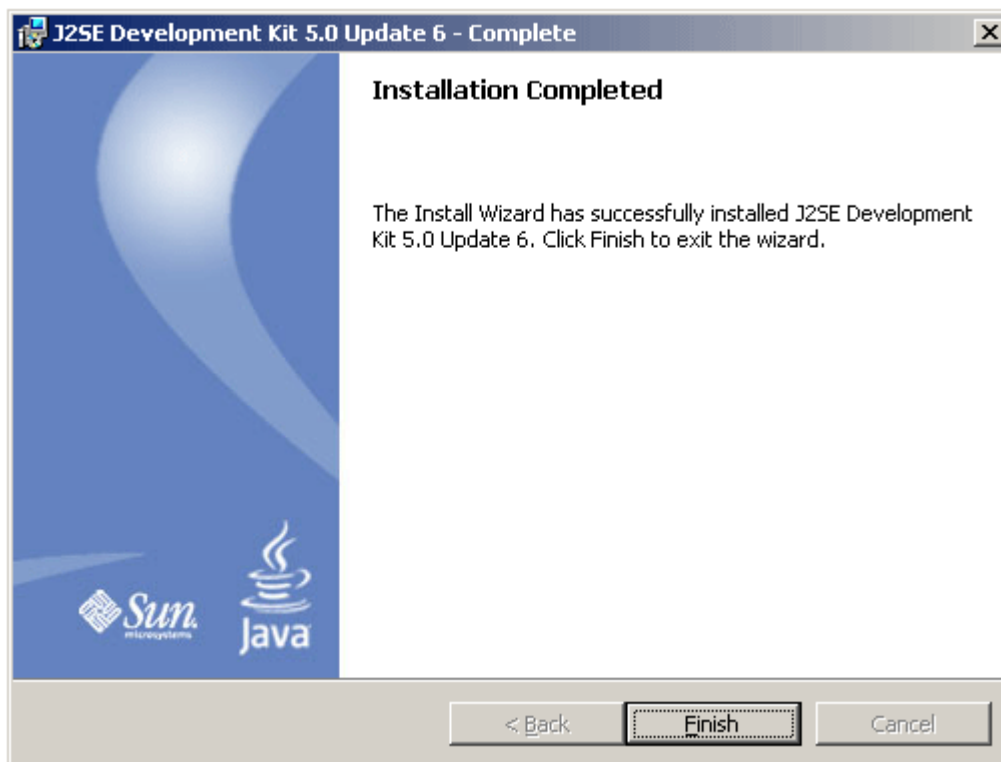
Click on the "Finish" button to exit from the installer.

Configuring the installation on windows machine:

In this section we will add some settings to the windows environment so that the java compiler and runtime becomes available for compiling and running the java application.

Go to the control panel and double click on "System Properties" and to the **advanced** tab.

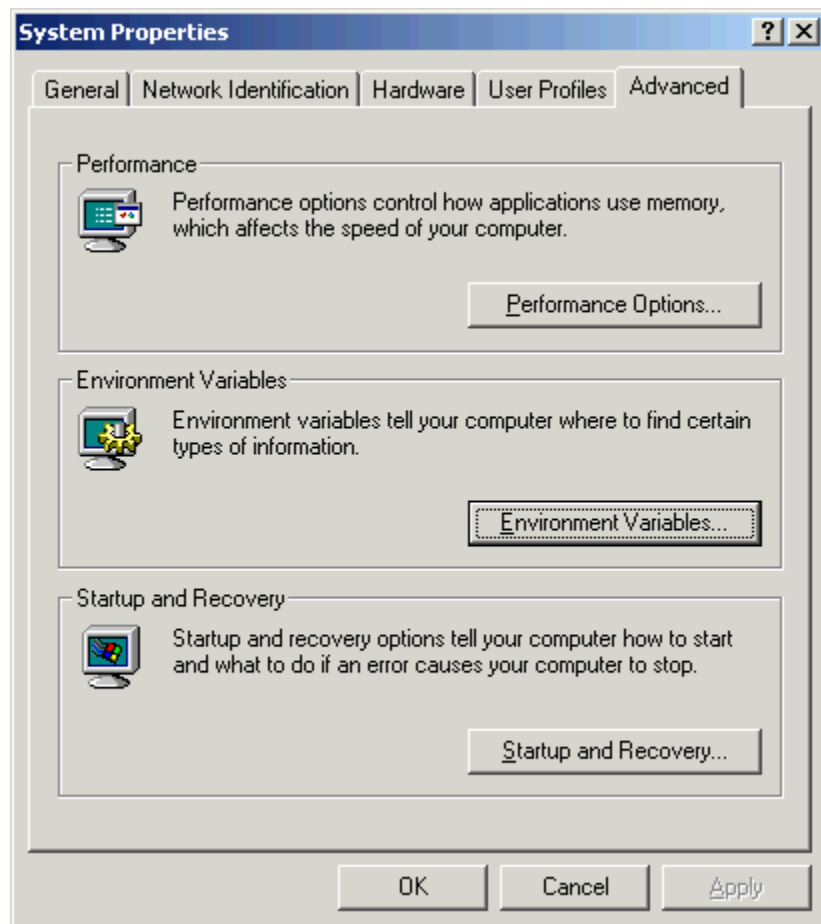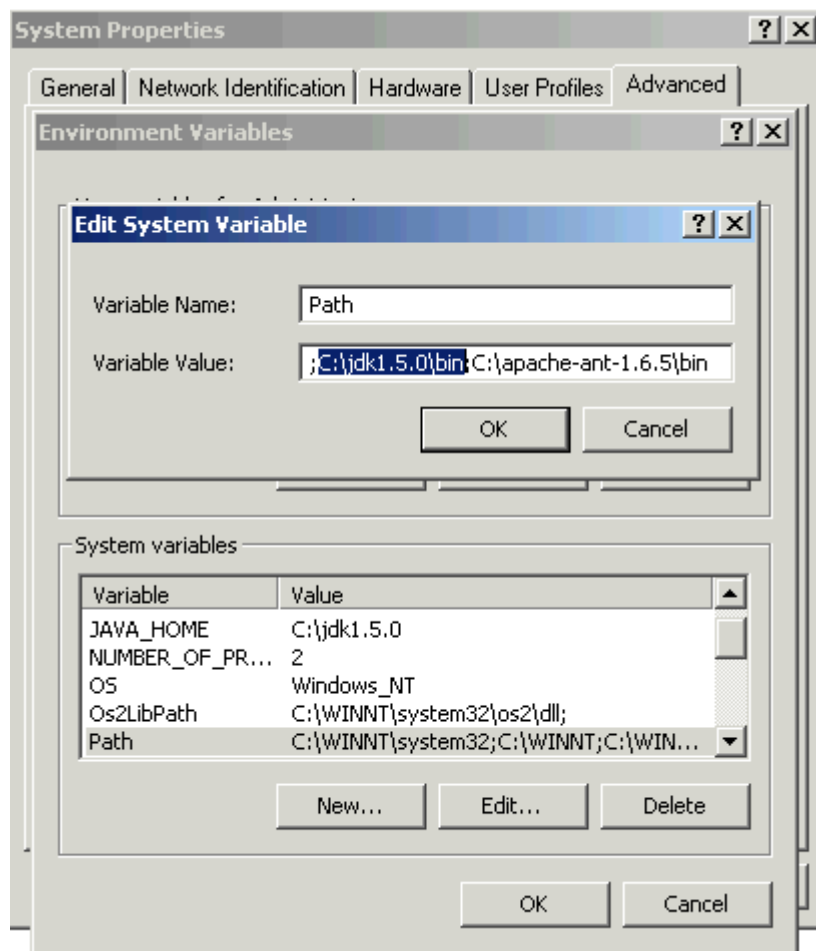and add "**c:\jdk1.5.0_06**" to path variable:

and click on ok button. To save the setting click on "OK" button.

2. What is Eclipse?

The idea of its Eclipse Project is to create an "Apache for developer tools" — an open source framework that provides many of the underlying services software developers need. This would be a "toolkit for designing toolkits." Not just a set of APIs, the framework will consist of real code designed to do real work.

The Eclipse Platform is the foundation for constructing and running integrated end-to-end software development tools. The platform consists of open source software components that tool vendors use to construct solutions that plug in to integrated software workbenches. The Eclipse Platform incorporates technology expressed through a well-defined design and implementation framework.

2.1. Java Requirements of Eclipse

Eclipse requires an installed Java Runtime. Eclipse 4.2 requires at least Java 5 to run.

For this tutorial you should use Java in version 6 or higher.

The Eclipse IDE contains its own Java compiler. The *Java Development Tools* are required if you compile Java source code outside Eclipse and for advanced development scenarios, e.g. if you use automatic builds or if you develop web development.

2.2. Download Eclipse

The Eclipse.org website provides pre-packaged Eclipse distributions.

Download the *Eclipse IDE for Java Developers* package from the following URL:

```
http://www.eclipse.org/downloads
```

The following screenshot shows the Eclipse download website for a Linux system, press on the link beside the package, e.g. Linux 64 Bit to start the download.
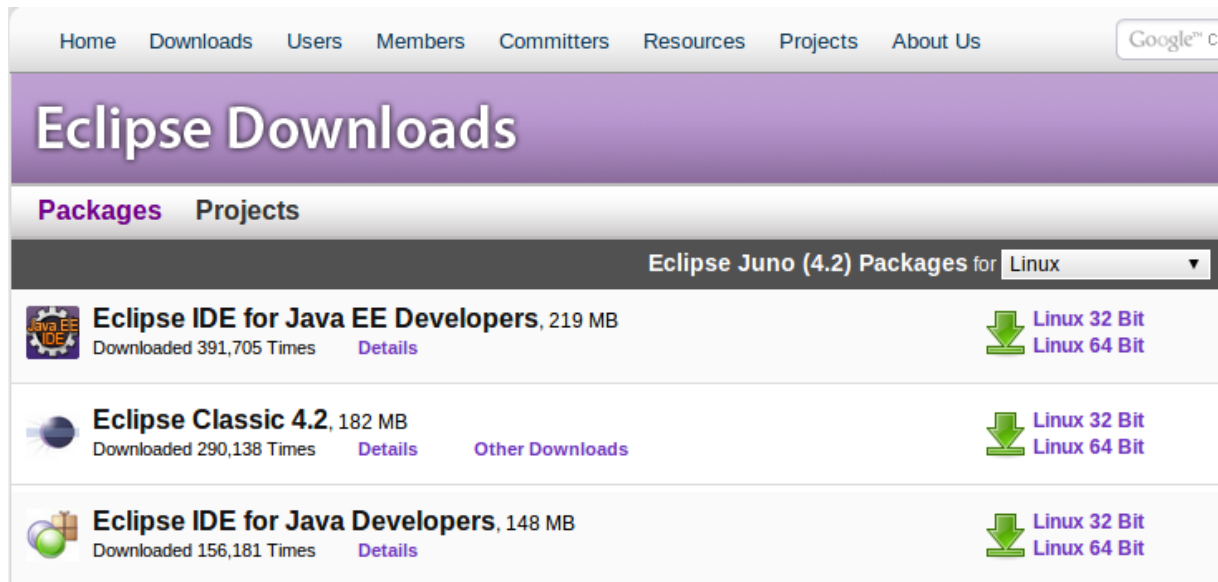
The download is a `.zip` file.

2.3. Install Eclipse

After you downloaded the `.zip` file which contains the Eclipse distribution you unpack it to a local directory.

Most operating system can extract zip files in their file browser, e.g. *Windows7* via right mouse click on the file and selecting "Extract all...". Use a directory path which does not contain spaces in its name, as Eclipse sometimes has problems with that.

After unpacking the downloaded zip file, Eclipse is ready to be used; no additional installation procedure is required.

3.   Getting started

3.1.   Starting Eclipse

To start Eclipse double-click on the file `eclipse.exe` (Microsoft Windows) or `eclipse` (Linux / Mac) in the directory where you unpacked Eclipse.

The system will prompt you for a *workspace*. The *workspace* is the place in which you work. Select an empty directory and press the OK button.
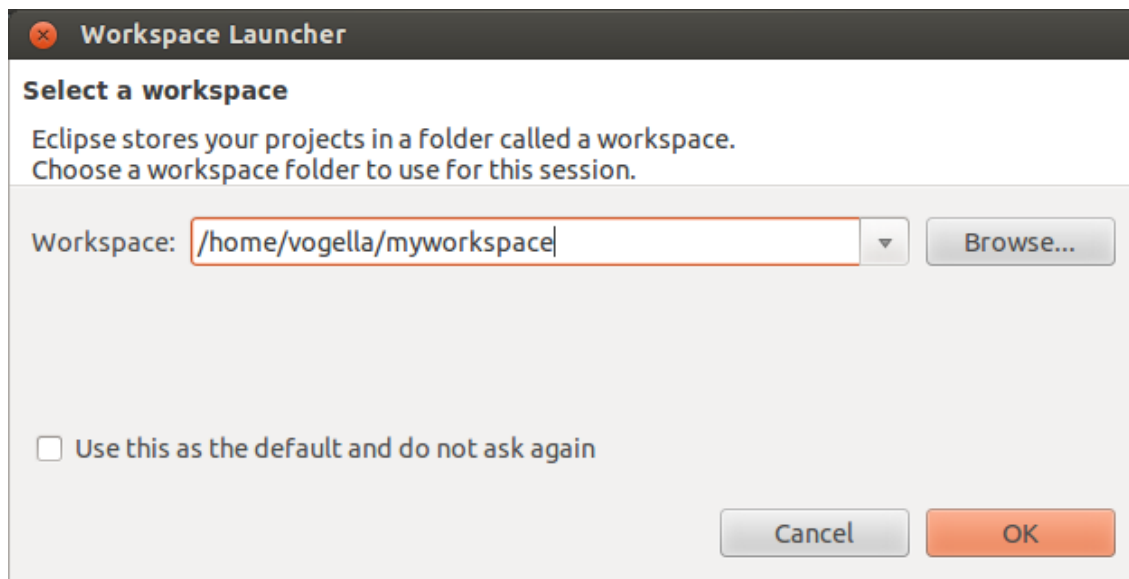
Eclipse will start and show the Welcome page. Close the welcome page by pressing the X beside Welcome.
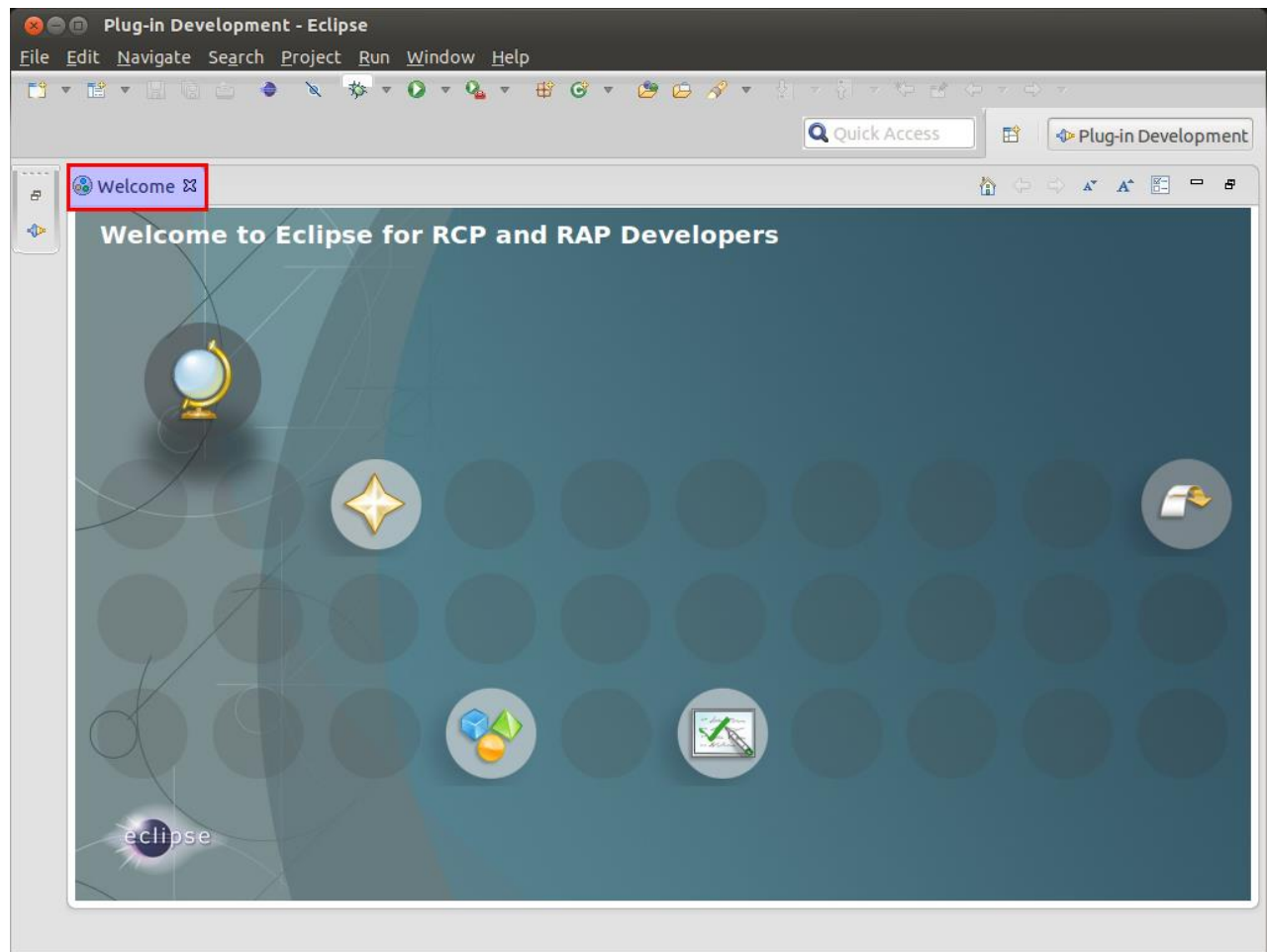
After you closed the welcome screen you should see a screen similar to the following.
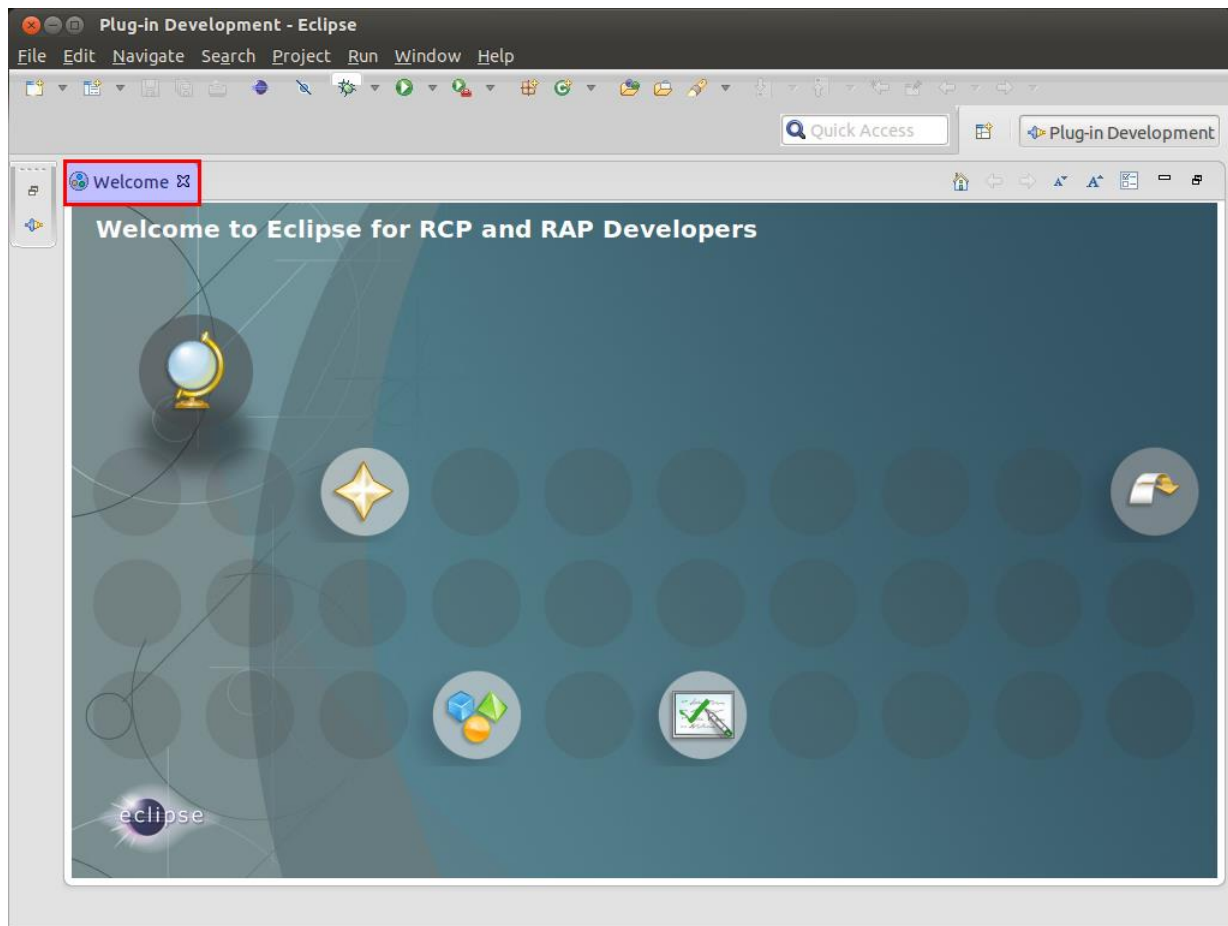
3.2. Appearance

The appearance of Eclipse can be changed. By default Eclipse ships with a few themes but you can also extend Eclipse with new themes.

To change the appearance, select from the menu Window → Preferences → General → Appearance

The Theme selection allows you to change the appearance of your Eclipse IDE. Please note that you need to restart Eclipse to apply a new styling correctly.
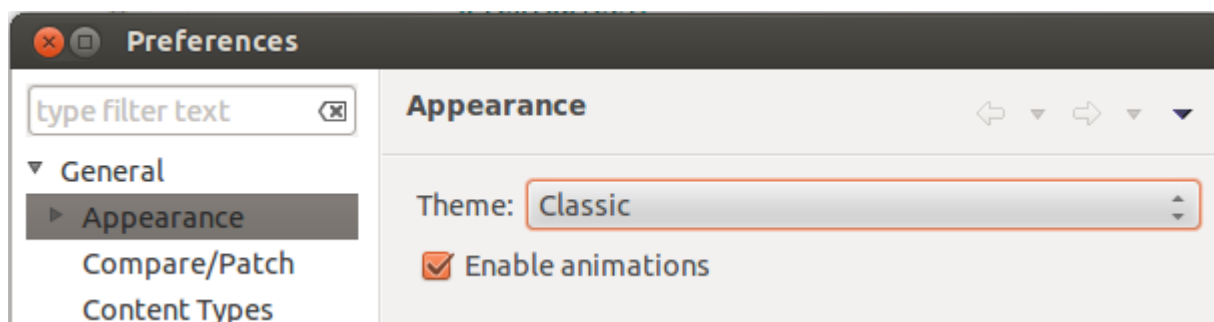
4. Eclipse user interface overview

Eclipse provides *Perspectives*, *Views,* and *Editors*. *Views* and *Editors* are grouped into P*erspectives*.

4.1. Workspace

The *workspace* is the physical location (file path) you are working in. Your projects, source files, images, and other artifacts can be stored and saved in your workspace but you can also refer to external resources, e.g. projects, in your *workspace*.

You can choose the workspace during startup of Eclipse or via the menu (File → Switch Workspace → Others).

4.2. Parts

Parts are user interface components which allow you to navigate and modify data. *Parts* are typically divided into *Views* and *Editors*.
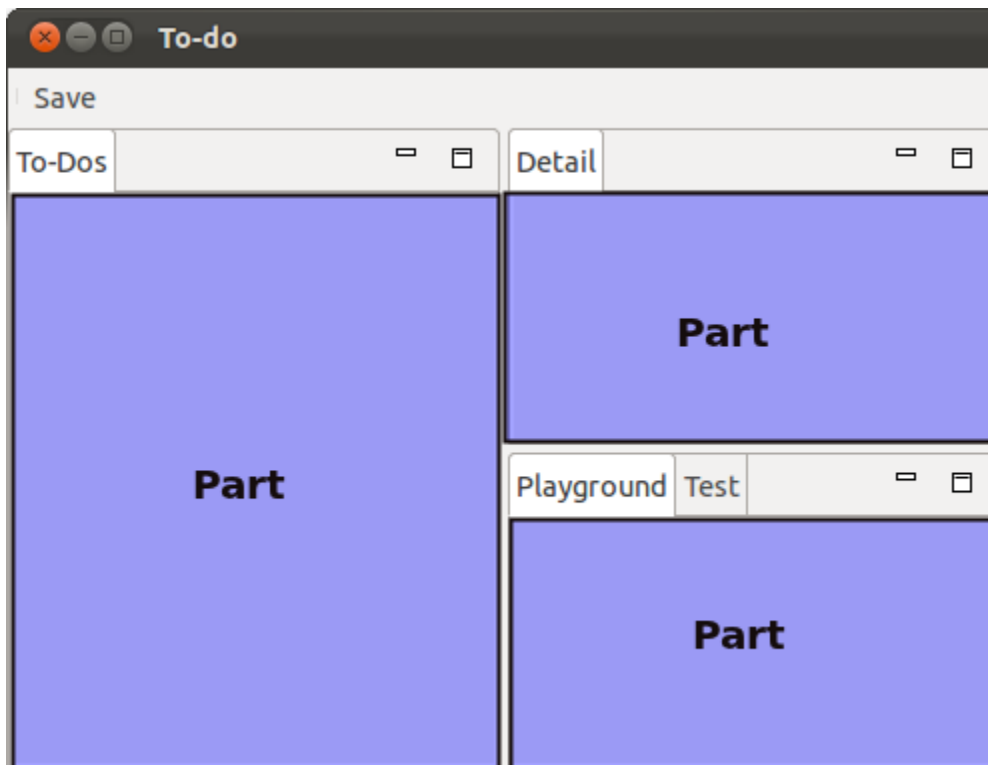
The distinction into *Views* and *Editors* is primarily not based on technical differences, but on a different concept of using and arranging these *Parts*.

A *View* is typically used to work on a set of data, which might be a hierarchical structure. If data is changed via the *View*, this change is typically directly applied to the underlying data structure. A *View* sometimes allows us to open an *Editor* for a selected set of the data.

An example for a *View* is the *Java Package Explorer*, which allows you to browse the files of Eclipse Projects. If you choose to change data in the Package Explorer, e.g. if you rename a file, the file name is directly changed on the file system.

Editors are typically used to modify a single data element, e.g. a file or a data object. To apply the changes made in an editor to the data structure, the user has to explicitly save the editor content.

For example the Java Editor is used to modify Java source files. Changes to the source file are applied once the user selects the *Save* command.

4.3. Perspective

A *Perspective* is a visual container for a set of *Parts*. The Eclipse IDE uses *Perspectives* to arrange *Parts* for different development tasks.

You can change the layout and content within a *Perspective* by opening or closing *Parts* and by re-arranging them.

5. Create your first Java program

The following describes how to create a minimal Java program using Eclipse. It is a tradition in the programming world to create a small program which writes "Hello World" to the console. We will adapt this tradition and will write "Hello Eclipse!" to the console.

5.1. Create project

Select from the menu File → New → Java project. Enter OOAD_Prog1 as the project name. Select the Create separate folders for sources and class files flag.
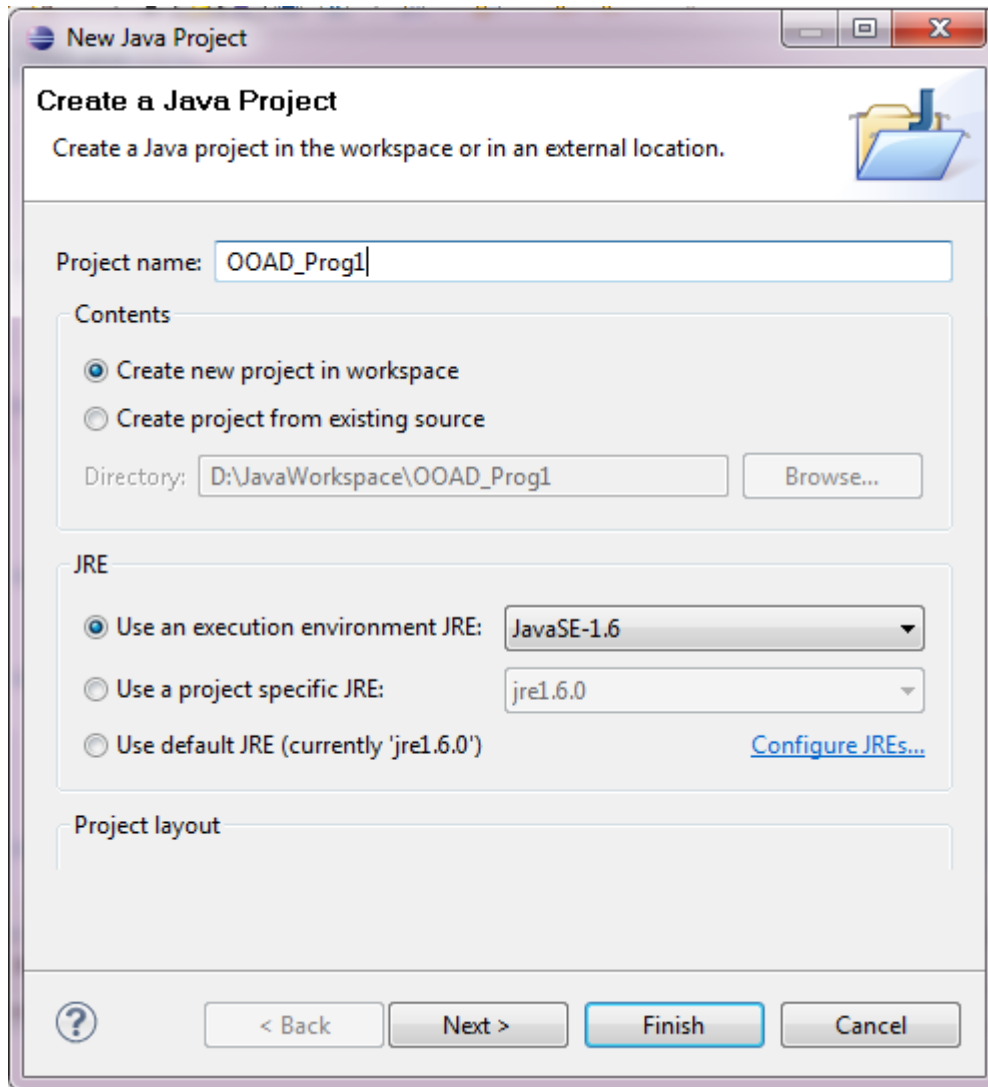
Press the Finish button to create the project. A new project is created and displayed as a folder. Open the `OOAD_Prog1` folder and explore the content of this folder.

5.2. Create package

In the following step you will create a new `package`. A good convention is to use the same name for the top level package and the project.

To create the `OOAD_Prog1` package, select the folder `src`, right click on it and select New → Package.
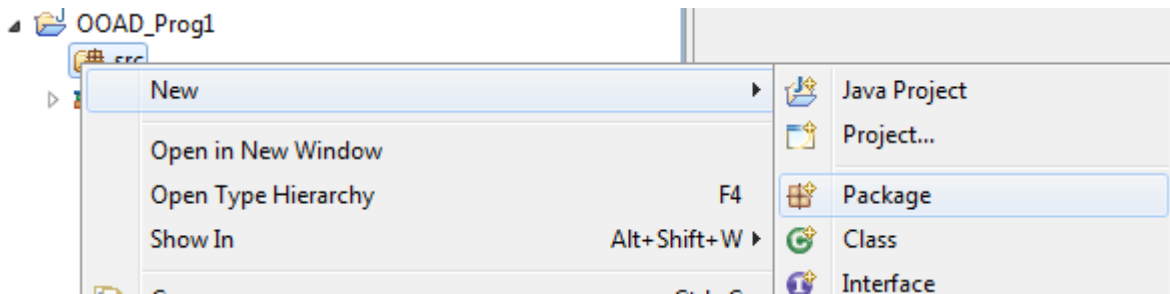
Enter the name of your new package in the dialog and press the Finish button.
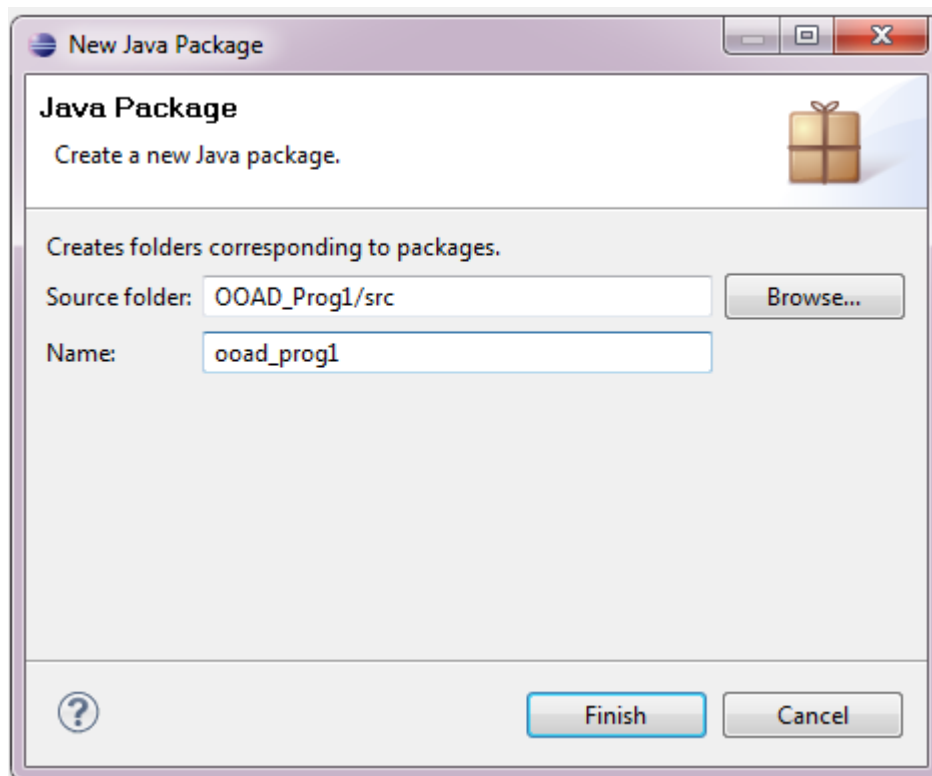
5.3. Create Java class

Create a Java class. Right click on your package and select New → Class.
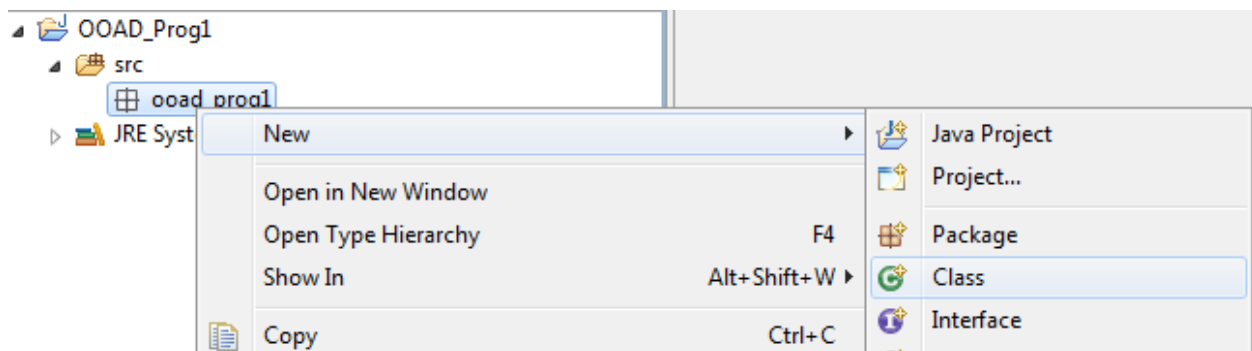
Enter `MyFirstClass` as the class name and select the public static void main (String[] args) flag.
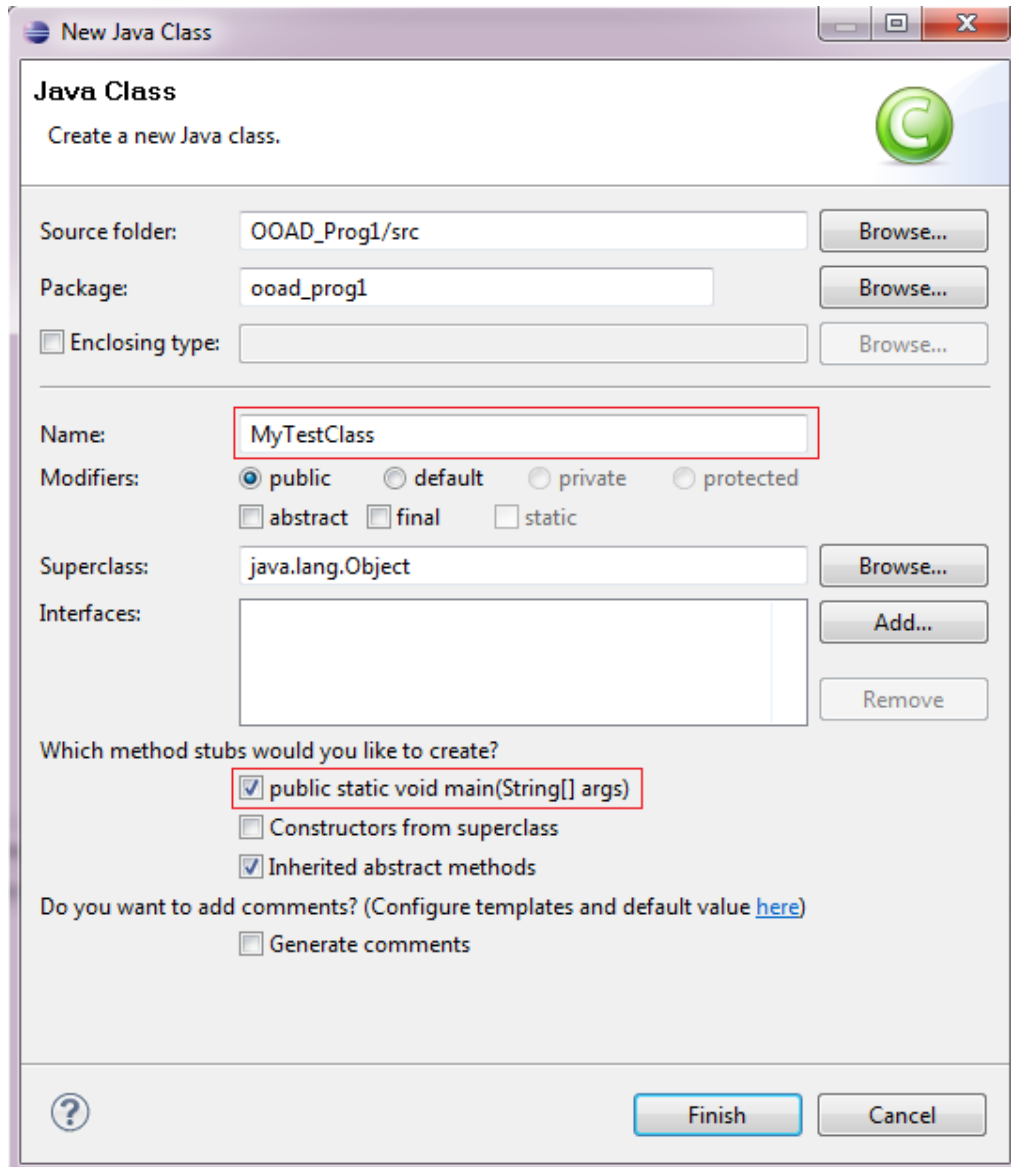
Press the Finish button.

This creates a new file and opens the *Editor* for Java source files. Change the class to the following example.

```java
package ooad_prog1;

public class MyTestClass {

    public static void main(String[] args) {

        System.out.println("Hello Eclipse!");
```

```
        }

}
```

5.4. Run your project in Eclipse

Now run your code. Right click on your Java class and select Run-as → Java application.
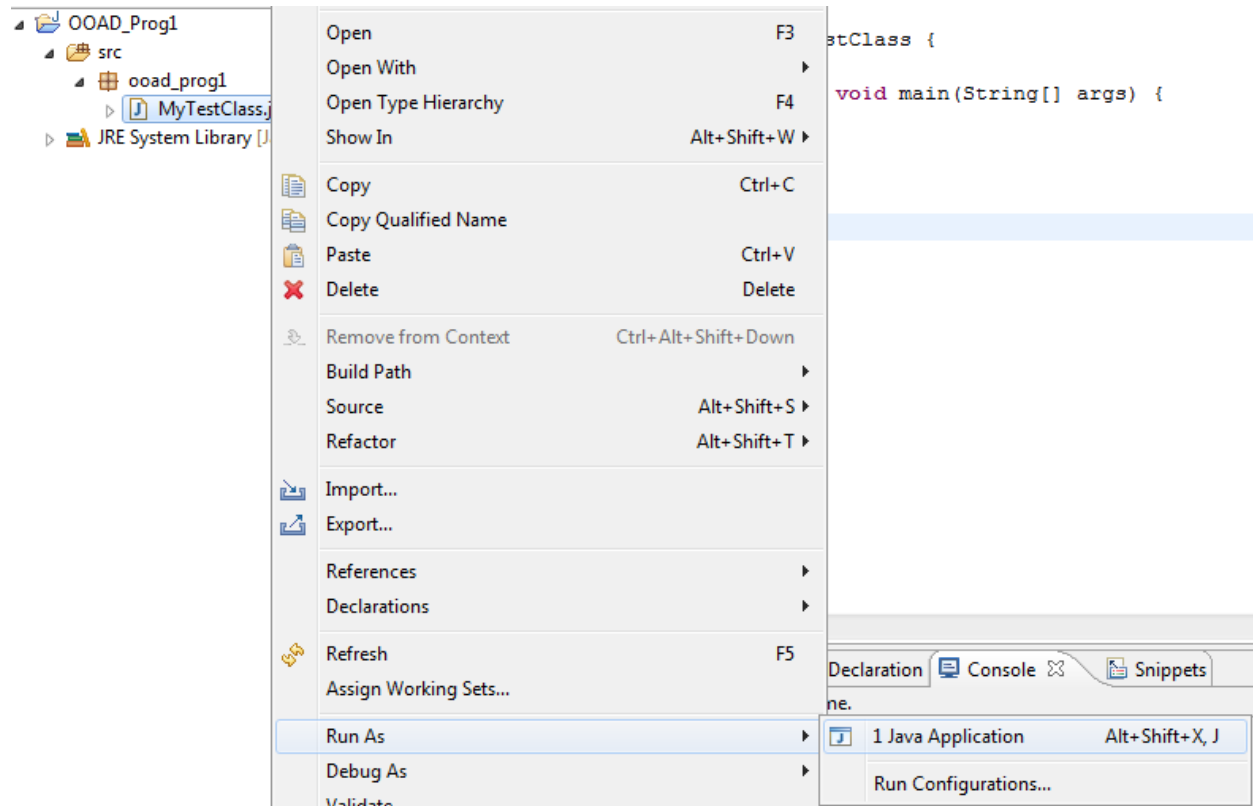
Eclipse will run your Java program. You should see the output in the Console *View*.
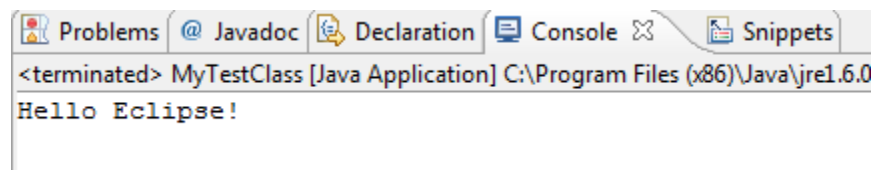
Congratulations! You created your first Java project, a package, a Java class and you ran this program inside Eclipse.

6.   Content Assist, Quick Fix and Class Navigation

6.1.   Content assist

The content assistant allows you to get input help in an editor. It can be invoked by pressing **Ctrl**+**Space**

For example type `syso` in the editor of a Java source file and then press **Ctrl**+**Space**. This will replace `syso` with `System.out.println("")`.

If you have a reference to an object, for example the object `person` of the type `Person` and need to see it's methods, type `person.` and press **Ctrl**+**Space**.
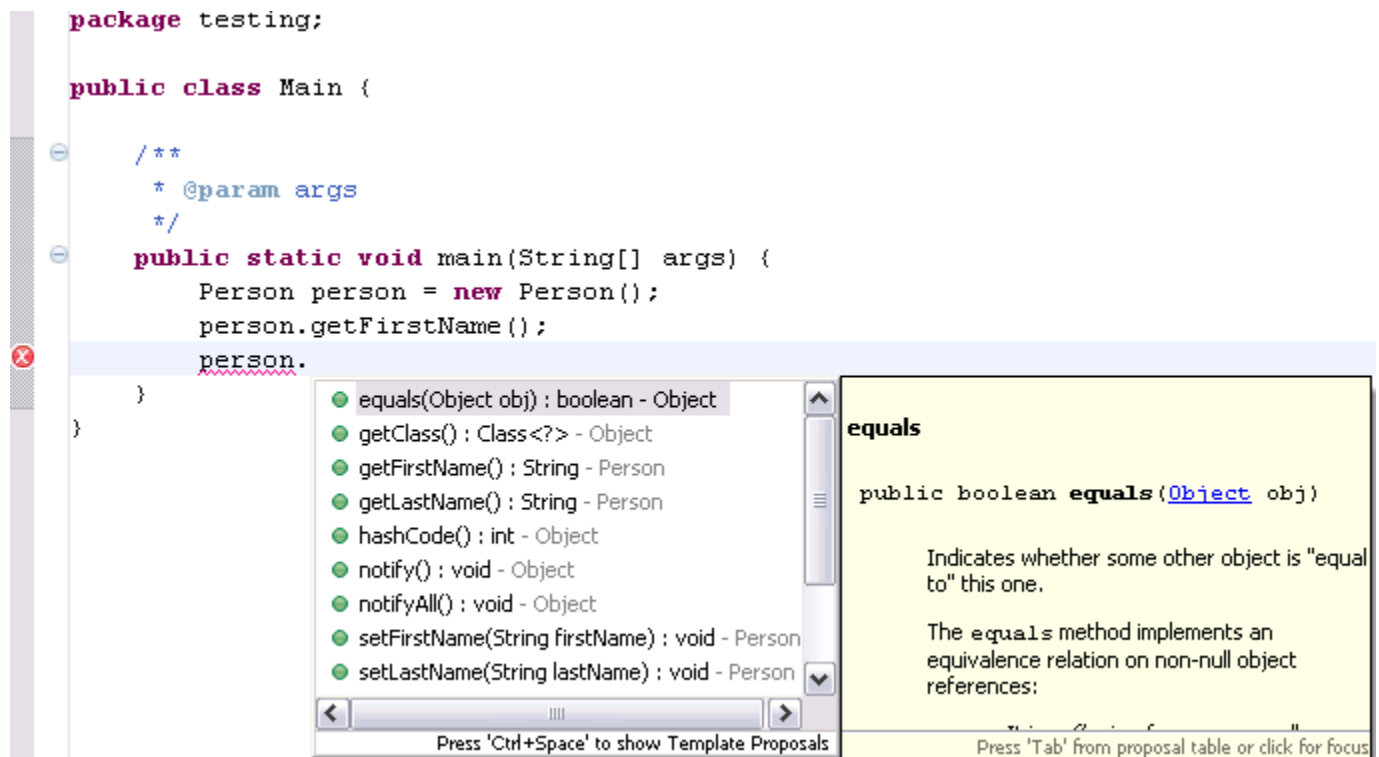
6.2.   Quick Fix

Whenever Eclipse detects a problem, it will underline the problematic text in the editor. Select the underlined text and press **Ctrl**+**1** to see proposals how to solve this problem.

For example type `myBoolean = true;` If myBoolean is not yet defined, Eclipse will highlight it as an error. Select the variable and press **Ctrl**+**1**, Eclipse will suggest creating a field or local variable.

```java
package ooad_prog1;

public class MyTestClass {

    public static void main(String[] args) {

        System.out.println("Hello Eclipse!");

        message = 4;
    }

}
```

message cannot be resolved

4 quick fixes available:

- Create local variable 'message'
- Create field 'message'
- Create parameter 'message'
- Remove assignment

Press 'F2' for focus
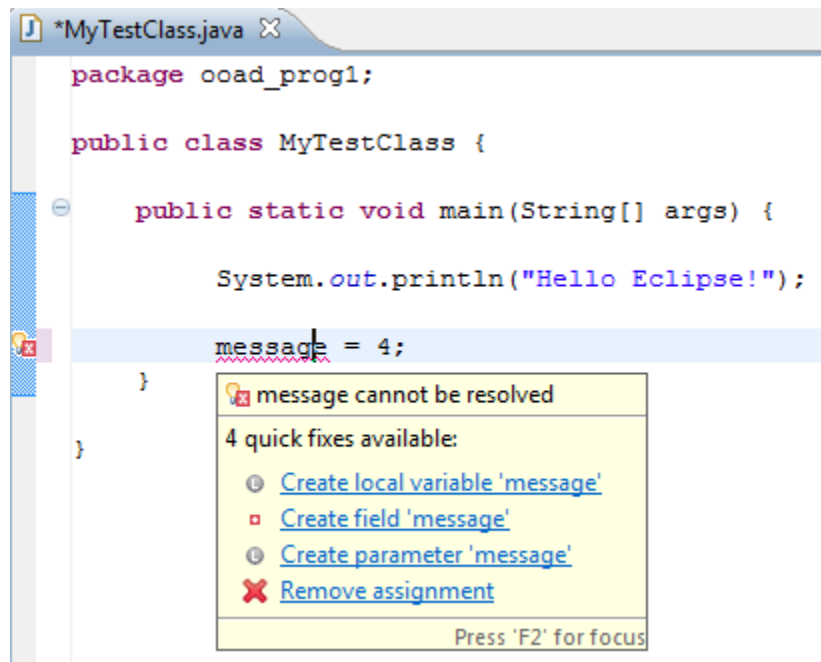
Quick Fix is extremely powerful. It allows you to create new local variables and fields as well as new methods and new classes. I can put try-catch statements around your exceptions. It can assign a statement to a variable and much more.

7.   Opening a class

You can navigate between the classes in your project via the "Package Explorer" *View*.

In addition you can open any class via positioning the cursor on the class in an editor and pressing **F3**. Alternatively, you can press **Ctrl**+**Shift**+**T**. This will show a dialog in which you can enter the class name to open it.
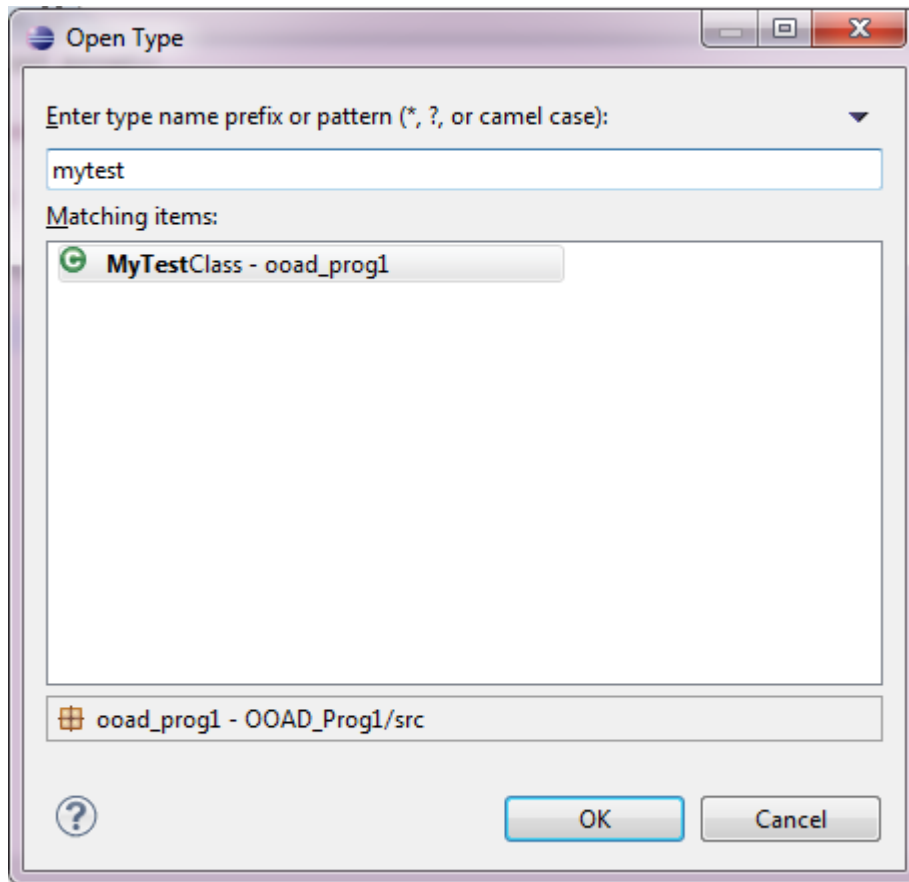
8.   Variables and Data Types

Variables are reserved areas in computer memory, holding unique name and expecting some data format to store called data type. Data types in Java fall into two categories: primitive data types and user-defined data types (classes).

Primitive data types are Java built-in data types representing most common data types like numbers and characters. Primitive data types are declared using special keywords. Table 1 shows primitive data types in Java.

Table 1: Java Data Types

Data Type	Size	Default Value (for fields)
Byte	8 – Bit	0
Short	16 – Bit	0
Int	32 – bit	0
Long	64 – Bit	0L
float	32 – bit	0.0f
double	64 – Bit	0.0d
char	16 – Bit (Unicode)	'\u0000'
String (or any object)		null
boolean	0/1 True/False	false

To declare a variable, use a statement with the following format: datatype variableName;

Data type is one of primitive data types shown in table 1, variable name is a valid Java variable name. The name is valid if it:
- Is not a reserved word.
- Is not the name of a previously defined variable, even of different type (unique).
- Starts with a letter ('A' to 'Z' or 'a' to 'z'), a dollar sign '$' or an underscore '_'.
- Contains only letters, dollar signs '$', underscores '_' and numbers.
- Has no spaces.

In addition to previous conditions, we have some optional conventions but they are highly recommended to follow them in order to stay up with the global standards of Java. One of these conventions is to start class name (and each fragment in the name if it has more than one) with an upper-case letter like ClassName and MyFirstJavaClass. We will be talking about conventions as they appear. Here are some examples of declaring data types:

```
int anInt;
double d;
float average;
boolean done;
long x1, x2, x3;
```

As you can see, the first part is a primitive data type from table 1, and the second type is a valid name. You can notice from the last statement that we can define several variables of the same type by using commas ',' between names on the following format:

```
datatype var_1, var_2, ..., var_n;
```

A notable naming conventions in Java is the one we've used in the first statement, a variable name should start with a small letter, and each following fragment with a capital letter like

```
variableName and averageOfMarks.
```

So whats next? We have to assign some values in the variables we have just declared. Value and variable must be of the same type, for example, you cannot store 7.34 in a variable of type int (unless you convert). Table 2 shows examples of values and their appropriate data types.

Value	Data Type
178	int
8864L	long
37.266	double
37.266D	double
87.363F	float
26.77e3	double
'c'	char
true	boolean
false	boolean

As you can see, numerical values with decimal point are treated as double by default, if you want to assign them to a float variable, you have to *explicitly* define it as float using F after the value. The value 26.77e3 is equal to $26.77 * 10^3$.

To assign a value to a declared variable, we use *assignment* operation denoted by equality sign '=' in the following format variable = value; for example, to assign a value 10 to some integer x

```
int x;
x = 10;
//Assignment can be done in the same statement in which we declare the variable
float average = 81.5f;
boolean b = false;
System.out.println(average);//prints "81.5"
System.out.println(b);//prints "false"
```

8.1.   Operations on Variables

As we have seen, most primitive data types are numerical, so we can perform binary operations on them (addition, subtraction, multiplication and division), those arithmetic operations are called *binary operations* because they are used with exactly two operands.
Table 3 shows arithmetic operators.

Operation	Operator	Usage
Assignment	=	var1 = var2
Addition	+	var1 + var2
Subtraction	-	var1 – var2
Multiplication	*	var1 * var2
Division	/	var1 / var2
Modulus (division remainder)	%	var1 % var

We have seen assignment before, and we know that is stores the value of right operand on the variable on the left operand. Addition, subtraction, multiplication, and division operations generate *expressions*, an expression is a series of calculations that produces a single value.
Here are some examples of expressions:

```
int a, b, c, d, e = 3;        //variables definition
a = b = 5;                    //multiple assignment
c = 6;
d = c % a;                    //d = 1
e = e + (a + b * c – d);      //expression evaluation (e = 37)
System.out.println("The value of e is " + e);
```

Comments on the previous code:
- It is valid to define multiple variables in the same statements and assign values to them (statement 1).
- You can assign single value to multiple variables in the same statement (statement 2).
- We can use brackets '(' and ')' to specify which expressions must be evaluated first (precedence overriding).
- By default, multiplication has the highest precedence, followed by division then addition and subtraction (in statement 5, b * c is calculated first).
- Mod operator calculates the remainder when dividing first operator on the second.
- We can *concatenate* a string with an integer by using "+" operator (last statement).
- We have some shortcuts with binary operations, for example, expression a = a + b, could be written in the form a += b, this way is valid for other operations as table 4.4 illustrates.

Expression	Shortcut
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b
a = a + 1	a++, ++a
a = a - 1	a--, --a

The difference between a++ and ++a is the value of the expression itself; at the end, both of them increment the value of a by one, but the value of a++ when evaluated is the original value of a, and the value of ++a is the incremented value. The following example shows the difference.

```
int a, b, c;
a = b = c = 1;
a = ++b + c;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
b = a++ - 1;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
c = a + a++ + --b + b;
System.out.println("a = " + a + ", b = " + b + ", c = " + c);
```

If you compile and run the above program, you should see the following output:

```
a = 3, b = 2, c = 1
a = 4, b = 2, c = 1
a = 5, b = 1, c = 10
```

To clearly understand how it works, let's monitor the value of b before, during and after the execution of the statement a = ++b + c;. Before the execution, it is clear that b = 1, once the statement ++b executed, the value of b is incremented by 1, then the rest of expression is evaluated using the new value of b. This happened because we used pre-increment. If we use post-increment, the value of b will also be incremented, but when b++ itself is evaluated, it gives the old value of b before increment, so post increment does not affect the expression in which it is, but the following expressions.

Boolean variables have different set of unary and binary operations according to the special data type they store, which is logical. Table 5 shows primary boolean operations.

Operation	Operator	Usage
Logical AND	`&&`	`op1 && op2`
Logical AND	`&`	`op1 & op2`
Logical OR	`||`	`op1 || op2`
Logical OR	`|`	`op1	op2`
Logical NOT	`!`	`!op1`

Discussing the difference between && and &, and | and ||, when we use && or ||, op2 is not evaluated when the value of the whole expression could be determined by the value of op1 and the operation. For example, true || false expression does not evaluate the second operand, because the logical OR between true and any other boolean expression is always true, similarly, false && true does not evaluate the second operand because logical AND between false and any other boolean expression is always false.

In addition to these operations, there are also comparison operations which we apply to numerical variables to get boolean results, table 4.6 shows comparison operations.

Operation	Operator	Usage
Equals	`==`	`op1 == op2`
Greater than	`>`	`op1 > op2`
Greater or equals	`>=`	`op1 >= op2`
Less than	`<`	`op1 < op2`
Less or equals	`<=`	`op1 <= op2`
Does not equal	`!=`	`op1 != op2`

The following example evaluates some boolean expressions and prints their results.
boolean b1, b2, b3;

```
b1 = true || false;
b2 = b1 && (5 > 3);
b3 = b2 && (13 == 6);
System.out.println("b1=" + b1 + ", b2=" + b2 + ", b3=" + b3);
```

If you compile and run the above code you should get the following output:

```
b1=true, b2=true, b3=false
```

Once again, we have used concatenation to convert the whole output to string, the next section focuses more on conversion.

8.2. Operator Precedence

Sr. No.	Operators	Precedence
1.	postfix	`expr++ expr--`
2.	unary	`++expr --expr +expr -expr ~ !`
3.	multiplicative	`* / %`
4.	additive	`+ -`
5.	shift	`<< >> >>>`
6.	relational	`< > <= >= instanceof`
7.	equality	`== !=`
8.	bitwise AND	`&`
9.	bitwise exclusive OR	`^`
10.	bitwise inclusive OR	`|`
11.	logical AND	`&&`
12.	logical OR	`||`
13.	ternary	`? :`
14.	assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

8.3. Conversion between Data Types

We can convert the value of a variable to the format of another variable of different type and make an assignment between them, conversion is different from another operation called *casting*, the following discussion shows the difference between them.

```
int i; float f; double d;
i = 1;
f = 4.4f;
d = 5.5;
d = i;
i = (int)f;
f = (float)d;
System.out.println("i = " + i + " f = " + f + " d = " + d);
```

The above code does the following:
- Converts the value of i to double and stores it in d. This conversion is done automatically by the compiler, because double data type is normally wider than int, there is absolutely no risk storing int in double.
- In the following two statements, notice that we put a large value into a smaller data type, in this case, a possible loss of data occurs in which we have to be aware of. Because that,

compiler (by default) refuses to store float value in int, or double in float. Because of that, we use *casting* to tell to compiler that we know what we are doing!

Another way to convert between variables is using some defined methods that convert between variables. One of them is a well known method that converts a String to integer which is Integer.parseInt() method, the following example shows how to use it.

```
String s = "115";
int x = Integer.parseInt(s);
x++;
System.out.println(x); //prints 116
```

Similarly, we can also use Double.parseDouble(), Float.parseFloat() and Long.parseLong().
A common method among all objects in Java is toString() method which converts the object to a string, toString() method will be covered in more details later.

Types of conversion mentioned above are called *explicit* conversion, because we request that conversion explicitly. Another type of conversion is called *implicit* conversion (promotion), because it is done automatically without requiring the programmer to request that conversion, for example, concatenating an integer or a character with a String converts them automatically to a string.

```
int x = 10;
String s = "x equals " + x;
```

In the above code, the value of x which is 10 is converted from integer to string and concatenated with the previous string "x equals ".
Promotion also occurs when an expression of values from different types is evaluated. In general, promotion converts all operands to the largest data type among them. The following code shows an example:

```
int x = 10;
double d = 12.5;
float f = 1.5f;
double result = (d/f) + x;
System.out.println("The result is " + result);
```

If you run the above code, you will see the output "The result is 18.33333333". The expression was evaluated as the following:
1. The value of f is converted to double and d/f is evaluated. Clearly, the result is of type double.
2. The value of x is converted to double and added to the previous expression.
3. The whole expression is evaluated to type double and stored in result.

8.4. Literal Values and Expressions

It is important to realize that a value of any type can be expressed in different ways, and all of them are equivalent. The following example shows how to express integer value of 8 in different ways.

```
int x = 8; //literal value
int y = 5 + 3; //Constant expression
int z = Integer.parseInt("8"); //Method return type
int w = (x + y) / 2; //Expression
```

From the above example we can notice different expression types:
Literal value: by defining the value we wish to store in the variable literally. For example, assigning 8 directly to an integer, 'a' directly to a character or "Hello" directly to a string.

Expression: expressions are evaluated in runtime to generate a single value, for example, the expression 1 + 3 − 5 evaluate at runtime and generate a final value of -1.

Methods: Some methods returns a value of certain type, one of them is the previously discussed parseInteger() method which returns an integer generated from some string.

8.5. Java Reserved Keywords

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	final	finally	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
package	private	protected	public	return
short	static	strictfp	super	switch
synchronized	this	throw	throws	transient
try	void	violate	while	

8.6.   Final Variables

Some variables are declared using final keyword, final means that the value remains constant at the time of execution and it is assigned during compile time. The value of a final variable cannot be modified once it is assigned. The final keyword is used before specifying data type in variable declaration.

final int MAX_VALUE = 100;

It is important to realize that you can assign a value to a final variable only once. You can only assign the value in the same statement in which you define the variable, so the following statements are invalid.

final int MAX_VALUE;
MAX_VALUE = 100; //Error

As you can see, the naming convention of final variables is to use upper letters and underscores "_".

Class Tasks

1. Start Eclipse. Create a new project "MyInfo" in a package called fastNU. Create a class StudentFAST in this project. Display your name, registration no, semester, and cgpa separated by tab stops.
2. Write a program that calculates a + b + c + d and prints the result on the screen where:
   - a, b, c and d are integers.
   - a is assigned a literal value of 3.
   - b is assigned a value of 6 using an arithmetic expression of 3 literal values.
   - c is assigned a value of 2 generated from an expression of a and b.
   - d is assigned a value of 4 converted from a string using parseInt() method.
   - The result is printed directly without defining a new variable to store the result.

Exercises

1. Assuming that a = 3, b = 5, c = 1, x = 2, y = -2, and z = 6 and following precedence rules. Evaluate each expression of the following independently and mention any changes in the variable values after expression is evaluated (all variables are integers):
   - (a / c) + z / a
   - c++ + b / x − y
   - x + ++x * x--
   - --a * ++a − (z++ * z++)
   - (true && (5 > 7)) || ((3 <= 10) && !false))
   - (3 < 6) || (c++ == x)

2. Write a program that evaluates all the expressions in exercise 1 and prints the result on the screen.