

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**  
**CS F213**

**LAB-1 [Introduction to Eclipse IDE]**

**DATE: 20/08/2018**

**TIME: 02 Hours**

- 1. Eclipse overview**
- 2. Getting Started**
  - a. Installation**
  - b. Start Eclipse**
- 3. Eclipse UI Overview**
  - a. Workspace**
  - b. Perspective**
  - c. Views and editors**
- 4. Create your first java program in Eclipse**
  - a. Create project**
  - b. Create a Java class**
  - c. Run your project in Eclipse**
- 5. Java Examples**
  - a. Example-1: Command Line Arguments (From command prompt)**
  - b. Example-2: Command Line Arguments (From Eclipse IDE)**
  - c. Example-3: Bicycle class**
- 6. Exercise (Write your own Java code)**

## **1. Eclipse Overview**

Eclipse is an open source community that builds tools and frameworks for creating general purpose application. The most popular usage of Eclipse is as a Java development environment which will be described in this article.

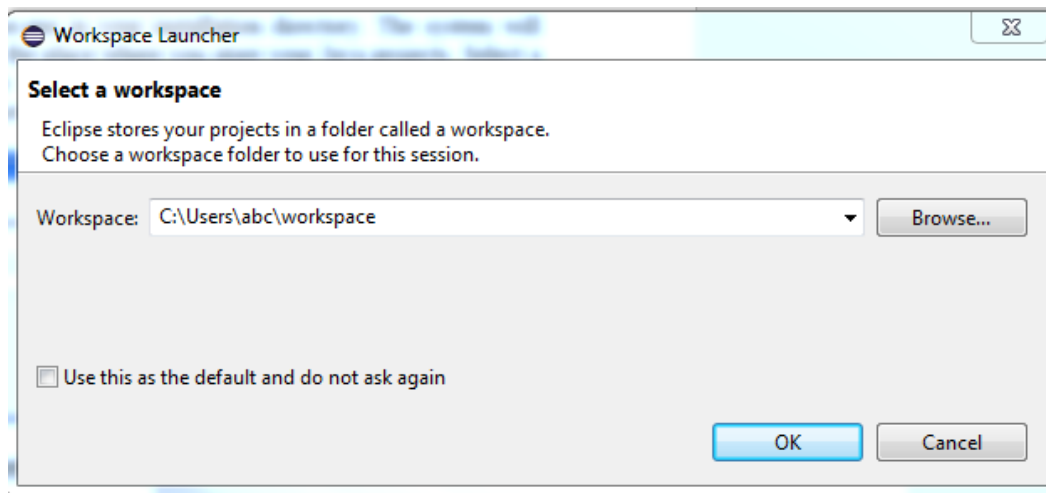
## **2. Getting Started**

### **2.1 Installation**

Download "Eclipse IDE for Java Developers" from its website and unpack it to a directory. This is sufficient for Eclipse to be used; no additional installation procedure is required.

### **2.2 Start Eclipse**

To start Eclipse double-click on the file eclipse.exe in your installation directory. The system will prompt you for a workspace. The workspace is the place where you store your Java projects. Select a suitable (empty) directory and press Ok [See Figure-1 given below]. Don't click and check the checkbox that says "Use this as default and do not ask again".



**Figure-1: Workspace Launcher**

Eclipse will start and show the Welcome screen [see Figure-2 given below].



**Figure-2: Eclipse Welcome Screen**

Close the welcome page by press in little x besides the Welcome screen.

### **3. Eclipse UI Overview**

Eclipse provides perspectives, views and editors. Views and editors are grouped into perspectives. All projects are located in a workspace.

### **3.1. Workspace**

The workspace is the physical location (file path) you are working in. You can choose the workspace during start-up of eclipse or via the menu (File-> Switch Workspace-> Others). All your projects, sources files, images and other artefacts will be stored and saved in your workspace.

### **3.2. Perspective**

A perspective is a visual container for a set of views and editors. You can change the layout within a perspective (close/open views, editors, change the size, change the position, etc.)

For Java development you usually use the "Java Perspective". You can change the layout within a perspective (close/open views, editors, change the size, change the position, etc.). Eclipse allows you to switch to another perspective via the menu (Window->Open Perspective -> Other).

A common problem is that you closed a view and don't know how to re-open this view. You can reset a perspective to its original state via the menu (Window -> Reset Perspective).

### **3.3. Views and Editors**

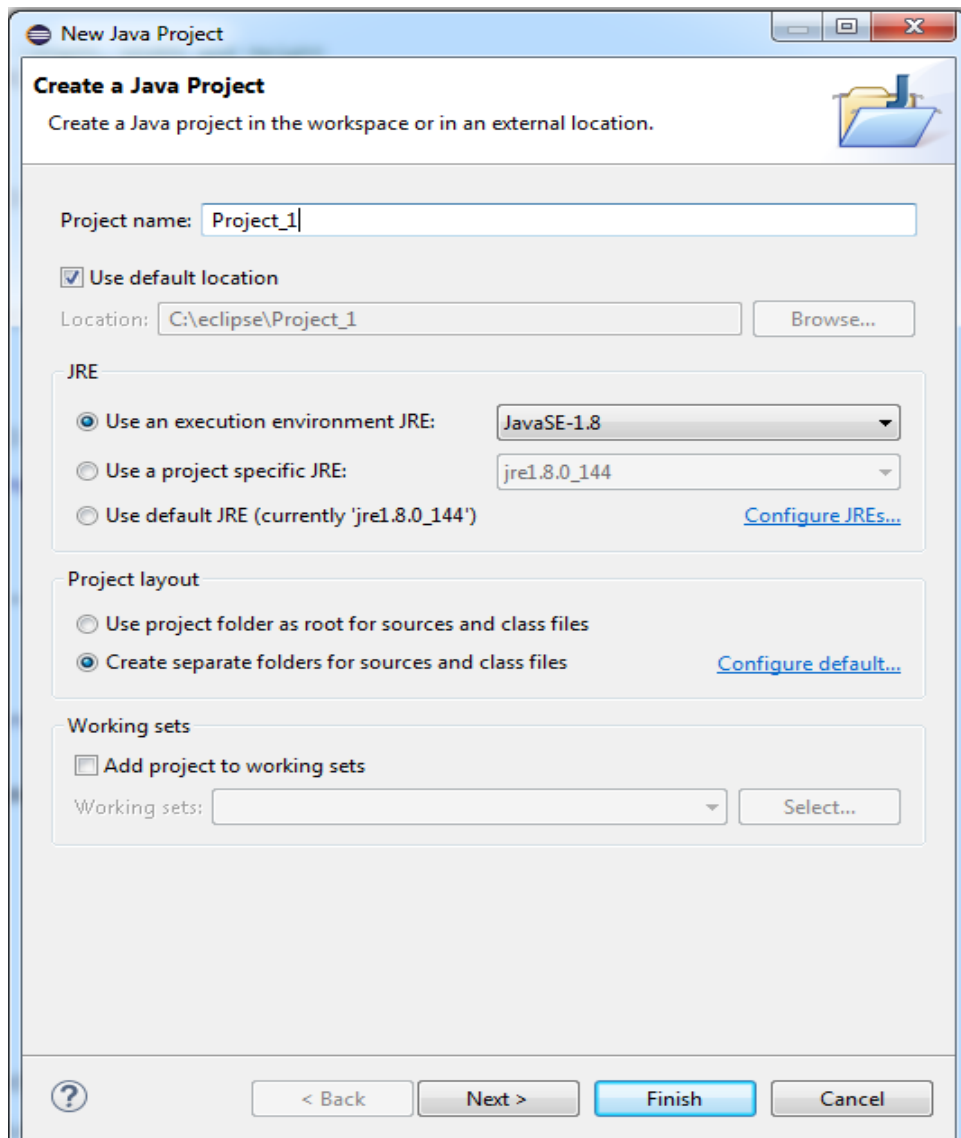
A view is typically used to navigate a hierarchy of information or to open an editor. Changes in a view are directly applied. Editors are used to modify elements. Editors can have code completion, undo/redo, etc. To apply the changes in an editor to the underlying resources, e.g. Java source file, you usually have to save.

## **4. Create your first Java program**

The following will describe how to create a minimal Java program using Eclipse. It will be the classical "Hello World" program. Our program will write "Hello Eclipse!" to the console.

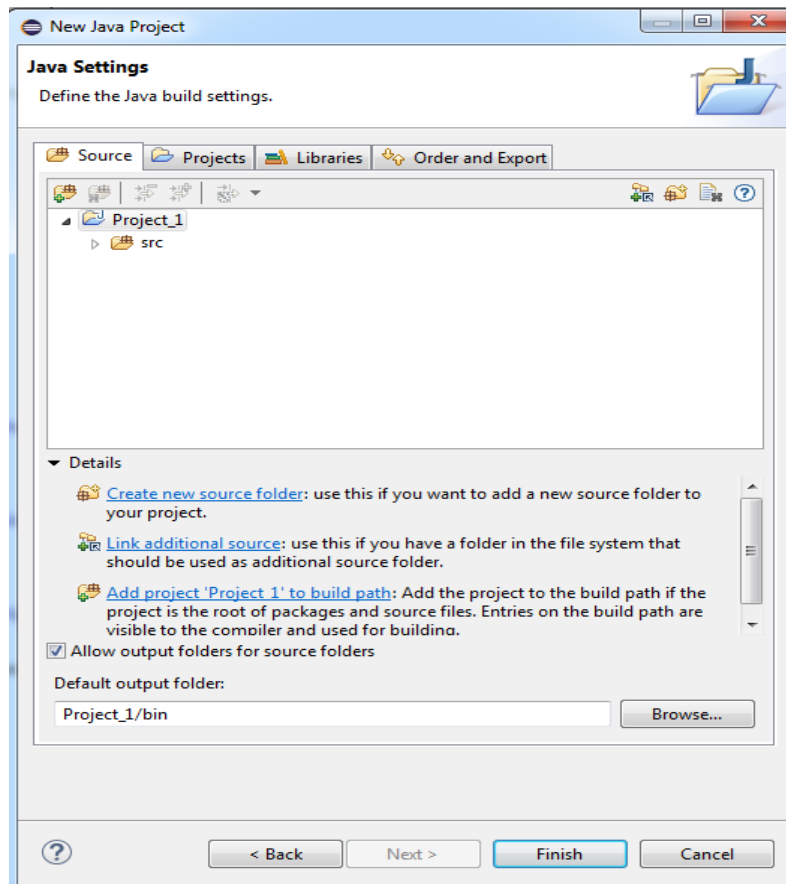
### **4.1. Create project**

Select from the menu File -> New-> Java project. Maintain "oop.eclipse.ide.first" as the project name and press the next button [See Figure-3 below].



**Figure-3: Create New Java Project**

Click on the checkbox that says, "Create separate source and output folders". This generates all your .class files corresponding to the .java files in your workspace into a separate output folder.

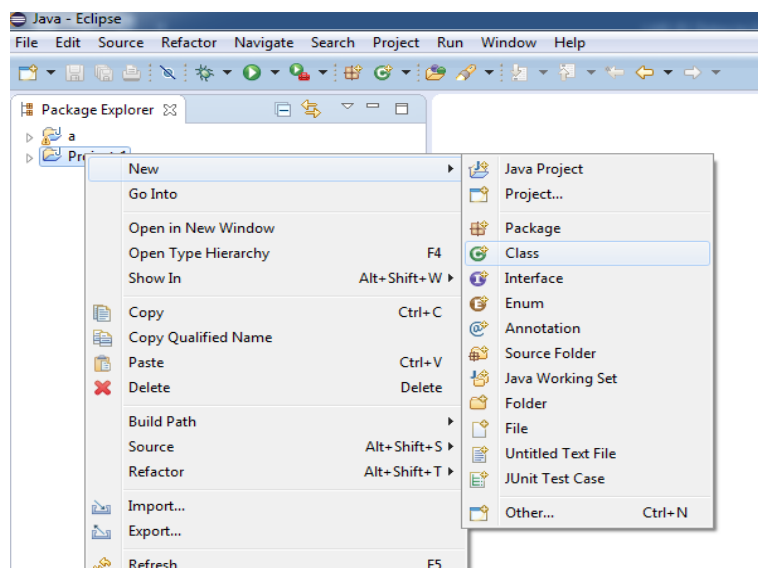


**Figure-4: Allow output folders for source folders**

Press finish to create the project. A new project is created and displayed as a folder. Open the folder "Project\_1".

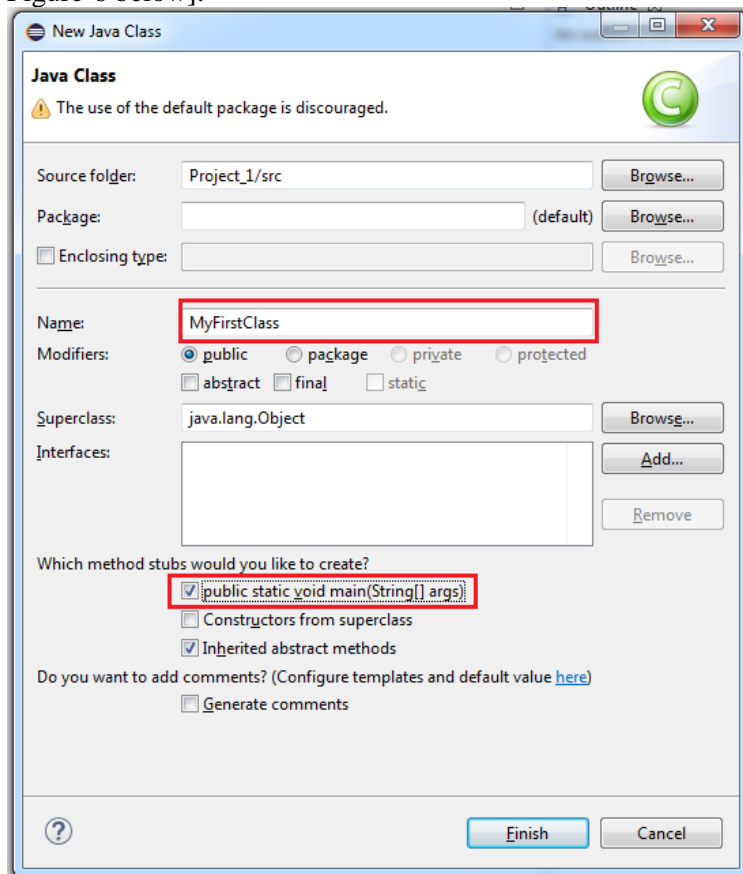
### 4.3. Create Java class

Right click on 'src' and select New -> Class [See Figure-5 below]



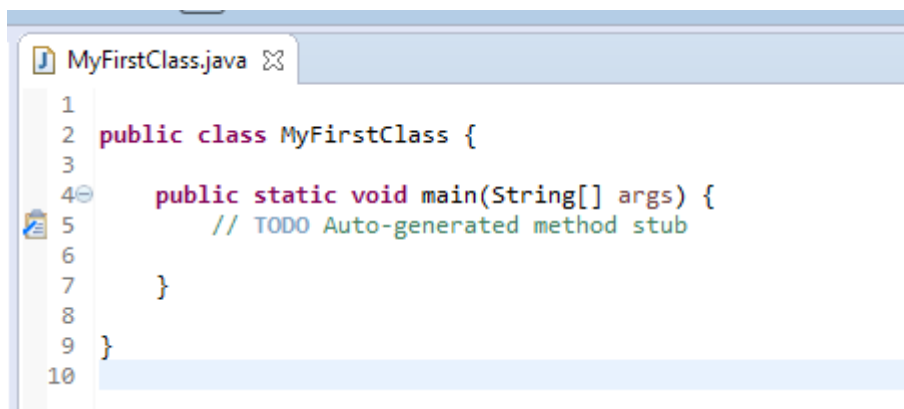
**Figure-5: Create a Java class**

Create MyFirstClass, select the flag "**public static void main (String[] args)**" [See Figure-6 below].



**Figure-6: Specify Class Name**

Maintain the following code.



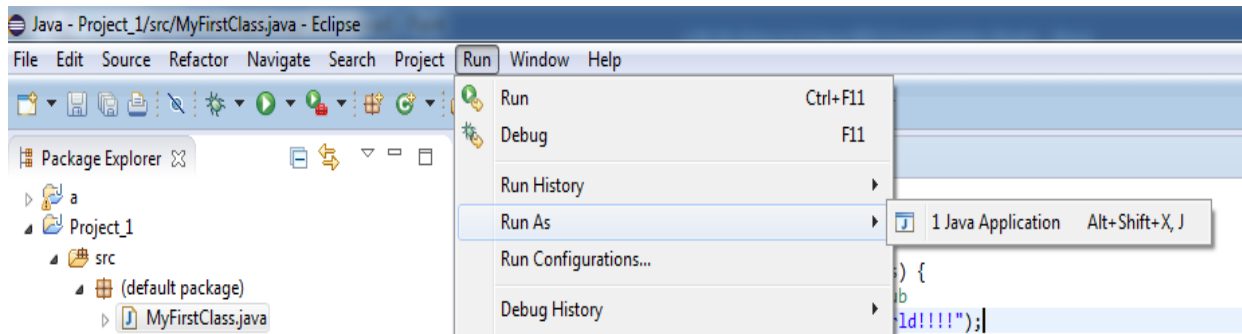
**Figure-7: Your first java program**

Include the following statement in the “*main*” method:

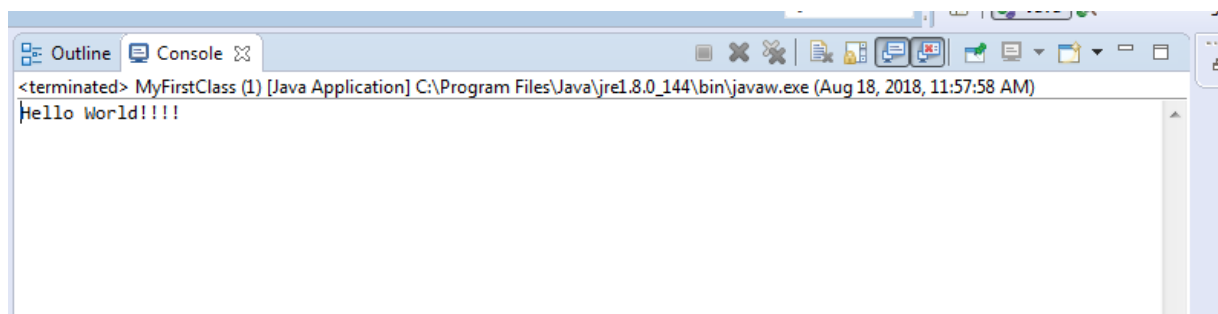
**System.out.println(“Hello Eclipse”);**

#### **4.4. Run your project in Eclipse**

Now run your code. Right click on your Java class and select Run-as-> Java application [See Figure-8 below].



**Figure-8: Run your java program**



**Figure-9: Finished! You should see the output in the console**

## 5. Java Examples

### Example-1(a): Compilation and Execution of java code from command prompt

We know that each object in java is described in terms of its states and behaviours. Also we know that an object is an instance of a class, whereas a class is a blueprint. In this example we are going to represent a real world Bicycle in Java code. We will create a class *Bicycle* which has three states, **speed**, **numberOfGears**, and **cadence** and it has methods to print the values of instance fields of *Bicycle* object.

1. Go to D Drive, create a new Text Document and name it **Bicycle.java**
2. Type the following code in this file and save the file:

```
/* Bicycle class */
class Bicycle {

    int speed=100;
    int noOfGears=5;
    int cadence=40;

    public void printState() {
        System.out.println("Bicycle [cadence=" + cadence +
            ", noOfGears=" + noOfGears + ", speed=" + speed +
            "]");
    }

    Public static void main(String[] args) {
```

```

        /*create instance of Bicycle class */
        Bicycle b1 = new Bicycle();

        /*Invoke method object b1 of type Bicycle*/
        b1.printState();

    }
}

```

3. Click on start, go to run option, type cmd and then hit the enter key
4. Type D: on command prompt
5. Execute the following command  
**D:\>javacBicycle.java** [This command compiles the java code]
6. Execute the following command  
**D:\> java Bicycle** [This command executes your java code]

#### **Example-1(b): Compilation and Execution of java code with Eclipse IDE**

1. Create the Bicycle.java class in Eclipse IDE [follow the same guidelines that are used to create MyFirstClass.java above]. Type the same java code for Bicycle class that is described in Example-1(a).
2. Run your program in Eclipse and see the output in console window.

#### **Example-2: Command Line Arguments (From command prompt) –**

1. Go to D Drive, create a new Text Document and name it **CommandLineArg1.java**
2. Type the following code in this file and save the file:

```

/* This program expects some string as command line arguments,
then it simply outputs the command line arguments to the
console*/

```

```

class CommandLineArg1 {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++){
            System.out.println("args["+ i +"] =" + args[i]);
        }
    }
}

```

3. Execute the following command  
**D:\>javac CommandLineArg1.java** [This command compiles the java code]
4. Execute the following command  
**D:\> java CommandLineArg1 My First Command Line Program in Java**  
 This command executes your java program, the whole string “My First Command Line Program in Java”, goes as command line arguments to your java program.



**Example 3:** Write one more program [as given below], compile and execute it from command prompt.

```
/* This program takes 10 integer arguments as an input from
command line, parse the command line arguments to integers,
finds the sum of these 10 numbers and print the sum */
```

```
class CommandLineArg2 {
    public static void main(String[] args) {
        int sum = 0;
        for(int i=0; i<args.length; i++){
            sum += Integer.parseInt(args[i]);
        }
        System.out.println("Sum = " + sum);
    }
}
```

Compile and execute this program as shown below:

D:\>javac CommandLineArg2.java

D:\> java CommandLineArg2 1 2 3 4 5 6 7 8 9 10

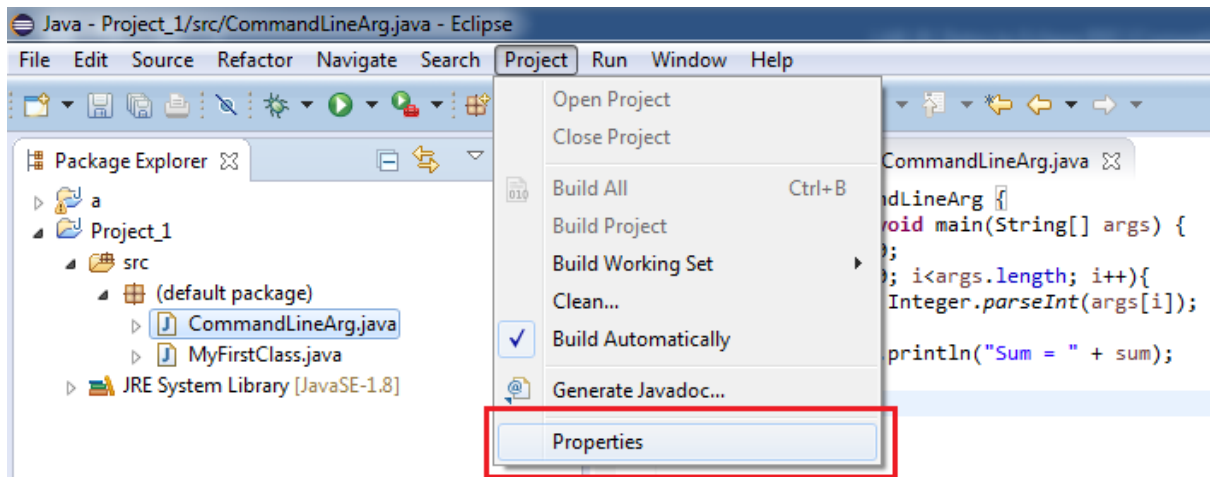
**Example-4: Command Line Arguments from Eclipse IDE–**

1. Create a java class file named CommandLineArguments in your workspace [Code given below].

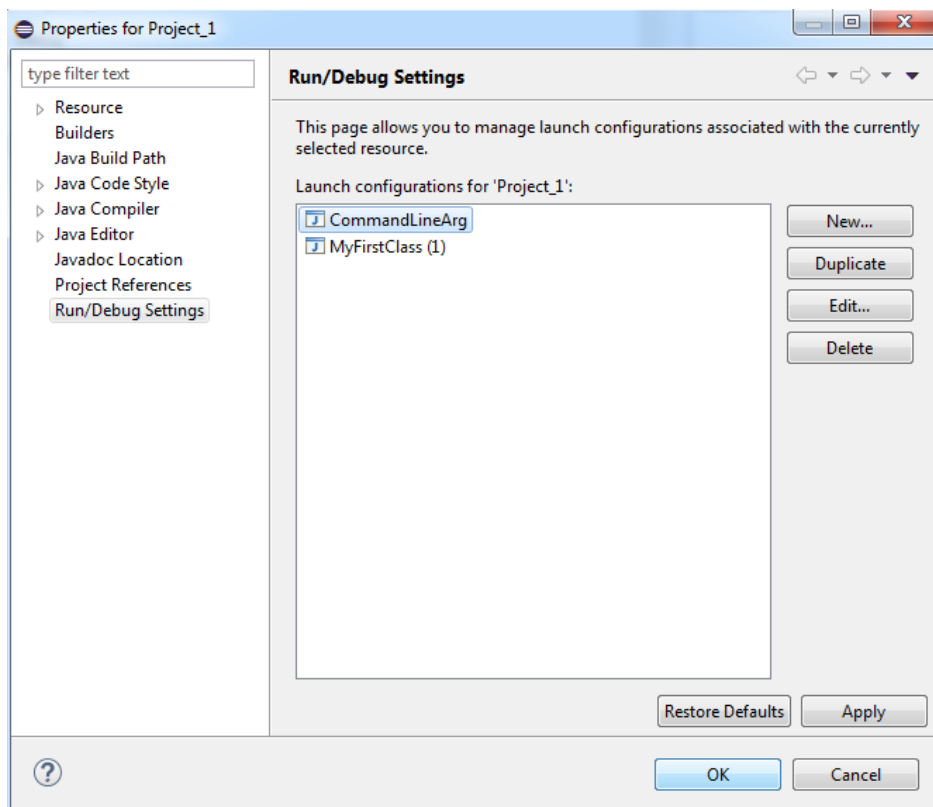
```
/* This program takes 10 integer arguments as an input from
command line, parse the command line arguments to integers,
finds the sum of these 10 numbers and print the sum */
```

```
public class CommandLineArg2 {
    public static void main(String[] args) {
        int sum = 0;
        for(int i=0; i<args.length; i++){
            sum += Integer.parseInt(args[i]);
        }
        System.out.println("Sum = " + sum);
    }
}
```

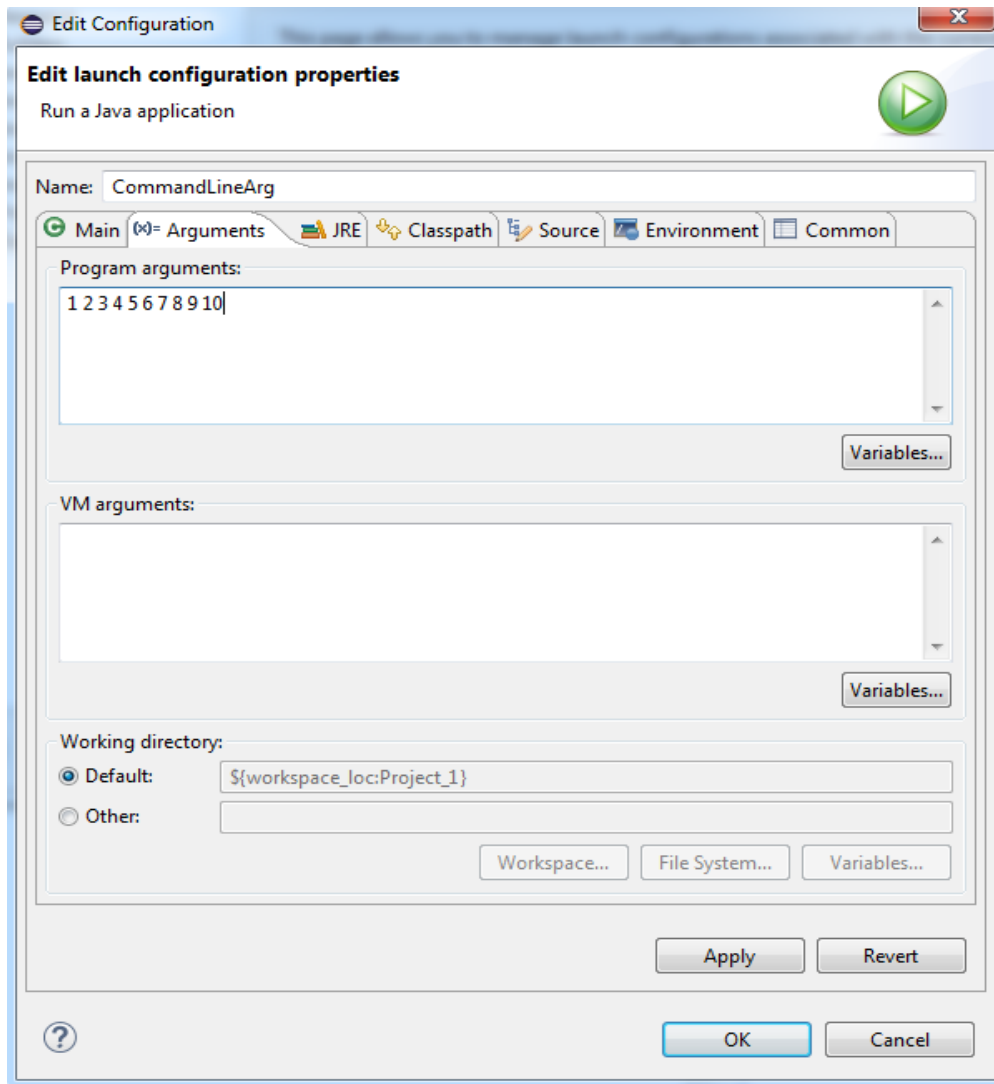
2. We can give command line arguments to our program in eclipse. For passing command line arguments to your program go to (Project->Properties->Run/Debug settings) and then in the launch configuration select your java file to which you want to pass command line arguments to [See Figure below].



**Step-1: Select Project Properties**



**Step-2: In Run/Debug Settings select the file CommandLineArg and press Edit**



**Step-3: Select Arguments tab and enter 10 numbers separated by space then press OK**

Run the program and watch the output in the console.

## EXERCISES

1. Write a program called **Fibonacci** to display the first 20 Fibonacci numbers  $F(n)$ , where  $F(n)=F(n-1) + F(n-2)$  and  $F(1) = F(2) = 1$ . Also compute their average. [ **Note:** the value 20 should be passed as a command line arguments]

**Input:**

20

**Output:**

The first 20 Fibonacci numbers are:

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

The average is 885.5

2. Given an unsorted input array, write a program to find the pair of numbers whose sum is equal to X. Get the input as command line arguments.

**Input:**

Array: 2 1 5 6 8 0

X: 7

**Output:**

2 5

1 6

3. Get a four-digit number as input, write a program that creates all possible unique three digit numbers.

**Input:**

5714

**Output:**

571

574

514

541 ...

4. Write a program called **HarmonicSum** to compute the sum of a harmonic series, as shown below, where  $n=50000$ . The program shall compute the sum from *left-to-right* as well as from the *right-to-left*. Obtain the difference between these two.

$$Harmonic(n) = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

**Hints:**

```
public class HarmonicSum{    // saved as "HarmonicSum.java"
    public static void main (String[] args) {
        int maxDenominator = 50000;
        double sumL2R = 0.0;    // sum from left-to-right
        double sumR2L = 0.0;    // sum from right-to-left

        // for-loop for summing from left-to-right
        for (int denominator = 1;
            denominator <= maxDenominator;
                denominator++) {

            .....
            // Beware that int/int gives int.
        }
        // for-loop for summing from right-to-left
        .....
        // Find the difference and display
        .....
    } //end of main
} //end of HarmonicSum
```

\*\*\*\*\* END OF LAB-1 \*\*\*\*\*