University of Windsor
SCHOOL OF COMPUTER SCIENCE

20th

Windsor Regional Secondary School Computer Programming Competition

Hosted by
The School of Computer Science, University of Windsor

SSPC

Workshop

[Accelerated Overview of the Java and Eclipse IDE]

Student Guide

Author:
Dr. Ziad Kobti
School of Computer Science
University of Windsor
Please email: kobti@uwindsor.ca or
call 519-253-3000 ext. 2990 or 3773

SSPC Registration, Transportation, Arrangements Information:

Ms. Margaret Garabon
garabon@uwindsor.ca
519-253-3000 ext. 3714

References:

Online repository of questions with the automated judge:
https://onlinejudge.org/

A good Java Tutorial:
http://docs.oracle.com/javase/tutorial/index.html

Guide available online:
http://kobti.myweb.cs.uwindsor.ca/sspc

Additional resources:
https://leetcode.com/
https://www.hackerrank.com/

Credits:

Team Coaches, Parent volunteers
Student, Facult and Staff Volunteers

Getting Started

In order to setup your Java/Eclipse environment there are two general steps to follow:

1. You can download the Java Software Development Kit from Oracle, and then download Eclipse. This step will allow you to download any other IDE (Integrated Development Environment) such as NetBeans or other editors like notepad++ or CodeBlocks (Also BlueJ is a really good free development IDE that integrates Unit testing). They are not required here.
   Start here: **https://www.oracle.com/java/** you will be redirected to the local site, then look for the Java Developer Kit (JDK) and download the latest SE version for your computer. You can also get the documentation download from here. Once you install the Java development environment proceed to the next step. Note: there is also the Open JDK project where you can also download the JDK for free (https://openjdk.org) (https://www.oracle.com/java/technologies/downloads/)

2. Then you can go to **https://eclipse.org** and download the latest Eclipse. You can download the latest eclipse edition or a bundled edition, for example the one that contains the Java developer packages. There is an extensive set of plug-ins (add-ons) and tutorials for Eclipse. Try looking up specific tutorials here: http://help.eclipse.org/ Note that you can use Eclipse for developing software in other languages with the right setup.

Online Java Tutorials/API

There is no shortage of online Java language tutorials. A good one is from the makers of Java at this link for the latest version or the one you are working with:

https://docs.oracle.com/en/java/

Direct link to API 23: https://docs.oracle.com/en/java/javase/23/docs/api/index.html

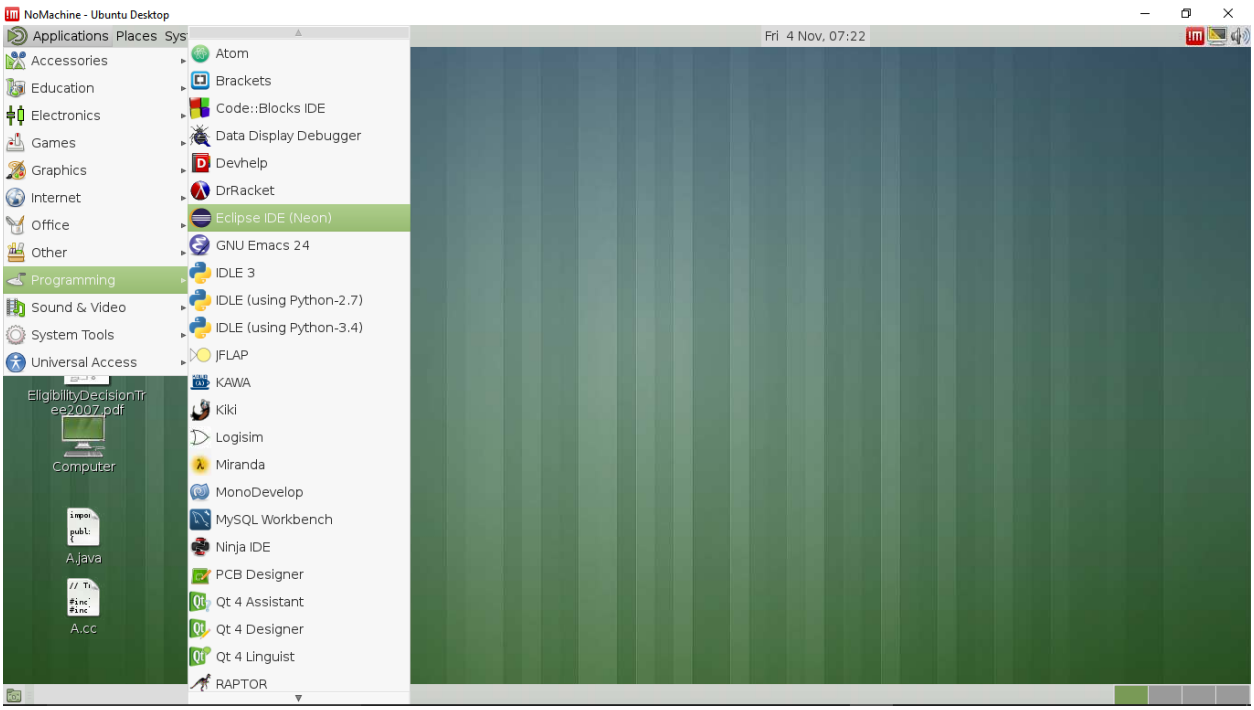
Select the Java SE API Documentation for the complete reference for all packages, classes and methods. Be sure to get the documentation matching your JDK version!

An API is the acronym for an Application Programming Interface.
The Java API is like the dictionary of all the built-in functionality of the language.
It is organized into Packages (libraries), classes, and methods.
You need to "import" the package that contains the class/method that you want to use.
Watch out for the classes that "extend" other classes, this means they also support the methods of the base class (i.e. the class that they extend or inherit).

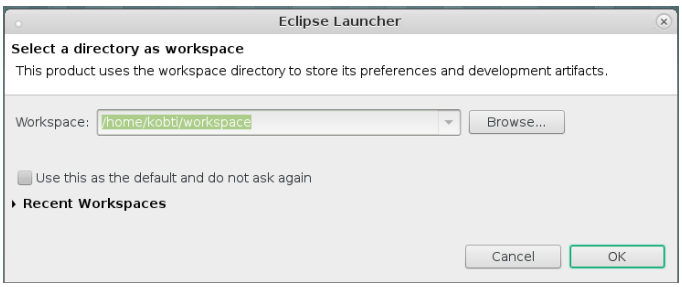
HelloWorld.Java

To start your first Java project you need to follow these simple steps. Remember that no two systems are identical, so please improvise as you see fit and specific for your own environment and version. The demonstrations here are for Eclipse on the CS Ubuntu Linux servers.
NOTE: screenshots are representatives of a typical install and you may see something slightly different.

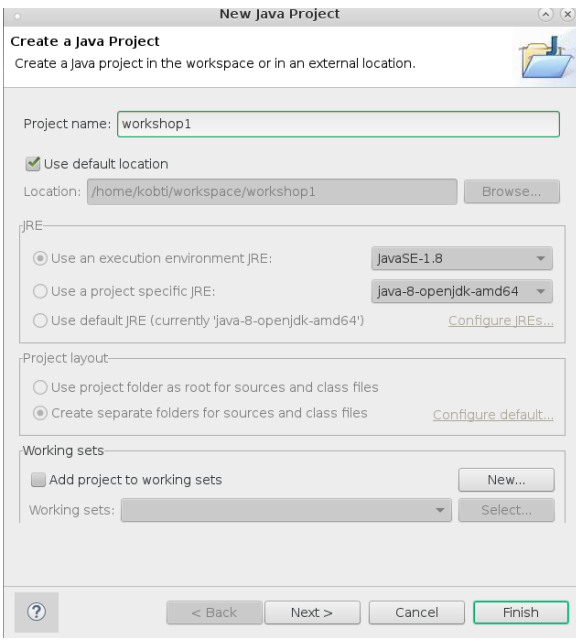
Start Eclipse from the menu: Applications → Programming → Eclipse IDE

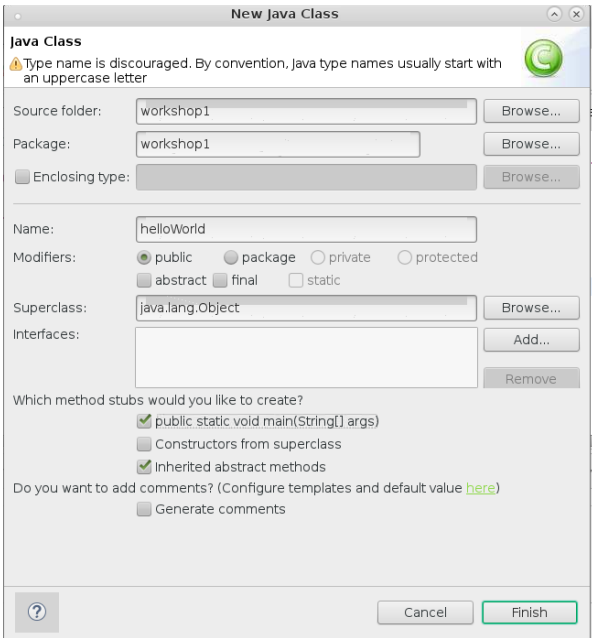


Press "OK" button to accept the default path. This means your files will be saved in a folder called "workspace" inside your home directory. I don't recommend changing it unless you plan to change the location where you want the files to be stored.



Next, from the top left menu, click on FILE → NEW → Java Project





Under "Project name:" type the name of the project you like, I called it "workshop1". Click Finish.

Type your first program:

```
/**
 * @author Your name here
 * @version 2016, date here
 */

// Class name "helloWorld" must be the same as
// the file name "helloWorld.java"
public class helloWorld
{

	/**
	 * This is where my program starts!
	 */
	public static void main(String[] args)
	{
		System.out.println("Hello World!");
	}
}
```

To run the program: from the menu: Run → Run as → Java Application

Your output will show up in the "Console" tab at the bottom of the eclipse window.

Now do the rest of the exercises.

Now let's understand the code we

wrote: Class Declaration:

```
public class helloWorld {
```

Class is the building block in Java, each and every methods & variable exists within the class or object. (instance of program is called object). The *public* word specifies the accessibility of the class. The visibility of the class or function can be public, private, etc. The following code declares a new class "helloWorld" with the public accessibility:

The main Method:

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

...
...
...
..

}
```

The main method is the entry point in the Java program and java program can't run without main method. Java Virtual Machine (JVM) calls the main method of the class. This method is always first thing that is executed in a java program. Here is the main method:

{ is used to start the beginning of main block and } ends the main block. Everything in the main block is executed by the Java Virtual Machine.

The code:

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

prints the "my first java project" on the console. The above line calls the *println* method of *System.out* class.

The keyword static:

The keyword **static** indicates that the method is a *class* method, which can be called without the requirement to instantiate an object of the class. This is used by the Java interpreter to launch the program by invoking the **main** method of the class identified in the command to start the program.

QUICK JAVA TUTORIAL:

Standard I/O:

Reading input from the keyboard and writing output to the display is done by standard streams. The Java platform supports three Standard Streams: *Standard Input*, accessed through System.in; *Standard Output*, accessed through System.out; and *Standard Error*, accessed through System.err.

Ex:
```
// to print an error message to the user on the screen
 (unbuffered! Good for debugging)

System.err.println("unexpected input type");

//to print to the user's screen a message or an output

System.out.println("Hello there");

//to read user input from the keyboard System.in.read(b);
```

Better if you use the Scanner class. Go to the Java API and read all about Scanner. Need to import `java.uil.Scanner;`

Variable Declaration:

Variables store pieces of data for later use. Before using a variable you must declare it. In the declaration you give the variable a name and you specify the type of data the variable will hold. The declaration looks like: **Type name**

The variable's type determines what kind of values it can hold and what operations can be performed on it. You use the name of the variable to refer to the data the variable holds. Variable names can be any legal identifier (no spaces, starting with a letter) as long as they are not Java reserved words. Here are the eight primitive data types in Java:

- int: 32-bit integer.
- long: 64-bit integer.
- short: 16-bit integer.
- byte: 8-bit integer.
- float: single-precision 32-bit floating point.
- double: double-precision 64-bit floating point.
- boolean**:** has only two possible values: true and false.
- char**:** a single character of 16 bits.

Ex: int x; float y; boolean b; int z = 10; boolean c = false;

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 to 16 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Hint: Want a REALLY BIG NUMBER stored? Try this class: Java.Math.BigInteger

BigInteger in Java

The **BigInteger** class in Java is designed for mathematical operations involving very large integers that exceed the limits of primitive data types. This class is particularly useful in competitive programming and scenarios requiring high-precision arithmetic.

Initialization and Basic Operations

You can initialize a BigInteger using various methods:

```
import java.math.BigInteger;

BigInteger A = BigInteger.valueOf(54); // Using valueOf
BigInteger B = new BigInteger("123456789123456789"); // Using a string
BigInteger C = BigInteger.ONE; // Using predefined constants (ONE, ZERO, TEN)
```

Basic arithmetic operations can be performed using methods like `add`, `subtract`, `multiply`, and `divide`:

```
BigInteger sum = A.add(B);
BigInteger difference = A.subtract(B);
BigInteger product = A.multiply(B);
BigInteger quotient = A.divide(B);
```

Conversion and Comparison

You can convert a BigInteger to other types or compare it with other BigIntegers:

```
int intValue = A.intValue(); // Convert to int
long longValue = A.longValue(); // Convert to long
String stringValue = A.toString(); // Convert to String

int comparison = A.compareTo(B); // Compare A and B
boolean isEqual = A.equals(B); // Check equality
```

Advanced Operations

The BigInteger class provides a wide range of methods for advanced mathematical operations:

```
BigInteger gcd = A.gcd(B); // Greatest common divisor
BigInteger mod = A.mod(B); // Modulus
BigInteger pow = A.pow(2); // Power
BigInteger sqrt = A.sqrt(); // Square root
```

```
BigInteger[] divAndRem = A.divideAndRemainder(B); // Division and remainder
```

Example: Calculating Factorial

Here's an example of calculating the factorial of a large number using BigInteger:

```
import java.math.BigInteger;

public class Example {
 static BigInteger factorial(int N) {
 BigInteger f = BigInteger.ONE;
 for (int i = 2; i <= N; i++) {
  f = f.multiply(BigInteger.valueOf(i));
 }
 return f;
}

public static void main(String[] args) {
int N = 20;
System.out.println(factorial(N));
}
}
```

Output:

```
2432902008176640000
```

Considerations

While BigInteger is powerful, it is not as fast as primitive types due to its internal use of arrays for processing. Operations on BigInteger objects take time proportional to their length, affecting the complexity of programs.

Arithmetic Operators and Precedence Rules:

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result. However, we need to know which operators have the highest precedence.

The operators in the following table are listed according to precedence order. The closer to the top of the table an operator appears, the higher its precedence.

- Operators with higher precedence are evaluated before operators with relatively lower precedence.
- Operators on the same line have equal precedence. For operators of equal precedence that appear in the same expression, there is a rule that must be applied to determine which is evaluated first.

All binary operators except for the assignment operators are evaluated from left to right while assignment operators are evaluated right to left.

operator	precedence
Unary	++, --, +, -, ~
Multiplicative	*, /, %
Additive	+, -
Bitshift	>>, >>>, <<
Comparison	>, =, <
Equality	==, !=
Integer Bitwise	&, |
Boolean	&&, ||
Ternary	? :
Assignment	=, +=, *= ...

Conditional Statements:

Generally the code is executed from top to bottom, in the order that they appear. **Control flow statements**, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to **conditionally** execute particular blocks of code. Here we describe the decision-making statements (if-then, if-then-else,), and the looping statements (for, while, do-while) in Java.

The if-then Statement

It tells your program to execute a certain section of code only if a particular test evaluates to true. For example:

```
if ( Moving) {
      System.out.println("the object is moving");
}
```

If this test *Moving* evaluates to true, the statement inside the if statement gets executed and "the object is moving" string is displayed on the screen. If it evaluates to false, control jumps to the end of the if-then statement.

The *if-then-else* Statement

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false. For example:

```
if (Moving)
{
      System.out.println("the object is moving");
}
else
{
      System.out.println("the object is not moving");
}
```

If this test *Moving* evaluates to true, the statement inside the if statement gets executed and "the object is moving" string is displayed on the screen, and control doesn't go into the else part. The control goes into the else part to print "the object is not moving" only if the test *Moving* evaluates to false.

The *while* Statement:

The while statement continually executes a block of statements while a particular condition is true.

```
while (Expression)
{
        Statement(s)
}
```

The while statement evaluates *Expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *Statement*(s) in the while block. The while statement continues testing the Expression and executing its block until the expression evaluates to false.

```
int count = 1;
while (count < 11)
{
       System.out.println("inside while loop");
       count++; //increments count (adding 1)
}
```

Starting with count having a value of 1, this will loop until count has a value of 11. When count=11 the control goes to the end of the while statement. Thus this code will print the string "inside while loop" ten times only.

The *do-while* Statement:

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

```
count = 1;
do
{
       System.out.println("inside while loop");
       count++; //increments count (adding 1)
}while (count < 11);
```

Here the string "inside while loop" will be printed once before testing the value of count. Then count is incremented and tested. When count=11, the execution of the loop stops. So the string gets printed 11 times and not 10 as it was the case with the while loop.

The *for* statement:

The for statement provides a compact way to iterate over a range of values. it repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment) {
        statement(s)
}
```

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.

• The *increment* expression is invoked after each iteration through the loop;it is perfectly acceptable for this expression to increment *or* decrement a value.

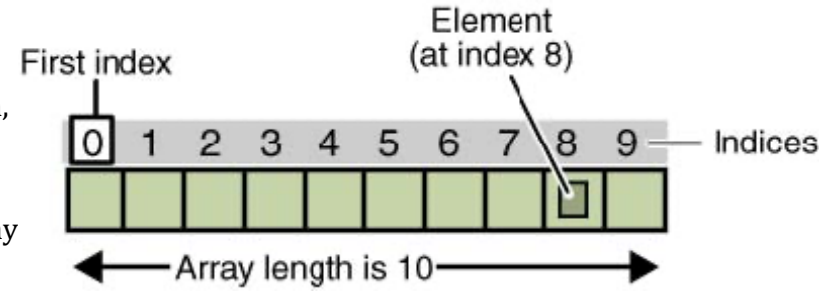
Here is an example:

```
for(int i=1; i<11; i++)
{
      System.out.println("inside for loop");
}
```

This will print "inside for loop" ten times.

Arrays:

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.



This is an array of ten elements.

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the above illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

The following program, MyArray, creates an array of integers, puts some values in it, and prints each value to standard output.

```
class MyArray
{
  public static void main(String[] args)
  {
    int[] anArray; // declares an array of integers
    anArray = new int[10]; //allocates memory for 10 integers
    anArray[0] = 100; // initialize first element
    anArray[1] = 200; // initialize second element
    anArray[2] = 300; // etc.
    anArray[3] = 400;
    anArray[4] = 500;
    anArray[5] = 600;
    anArray[6] = 700;
    anArray[7] = 800;
    anArray[8] = 900;
    anArray[9] = 1000;
    System.out.println("Element at index 0: " + anArray[0]);
    System.out.println("Element at index 1: " + anArray[1]);
    System.out.println("Element at index 2: " + anArray[2]);
    System.out.println("Element at index 3: " + anArray[3]);
    System.out.println("Element at index 4: " + anArray[4]);
    System.out.println("Element at index 5: " + anArray[5]);
    System.out.println("Element at index 6: " + anArray[6]);
    System.out.println("Element at index 7: " + anArray[7]);
    System.out.println("Element at index 8: " + anArray[8]);
    System.out.println("Element at index 9: " + anArray[9]);
  }
}
```

The output from this program is:

```
Element at index 0: 100
Element at index 1: 200
Element at index 2: 300
Element at index 3: 400
Element at index 4: 500
Element at index 5: 600
Element at index 6: 700
Element at index 7: 800
Element at index 8: 900
Element at index 9: 1000
```

PRACTICE:
Problems located at http://sspc.cs.uwindsor.ca/

Additional Problems:

Problem 1: Write an application that asks the user to enter two integers, obtains them from the user and prints their sum, product, difference and quotient (division).

Problem 2: Write an application that reads five integers, determines and prints the largest and smallest integers in the group.

Problem 3: Write an application that reads an integer and determines and prints whether it is odd or even [hint: use the remainder operator. An even number is a multiple of 2. Any multiple of 2 leaves a remainder of 0 when divided by 2]

Problem 4: Write an application that reads two integers, determines whether the first is a multiple of the second and prints the result.

Problem 5: Write an application that inputs one number consisting of five digits from the user, separates the number into its individual digits and prints the digits separated from one another by three spaces each. For example, if the user types in the number 42339, the program should print: 4 2 3 3 9

Problem 6: Write a program that inputs five numbers and determines and prints the number of negative numbers input, the number of positive numbers input and the number of zeros input.

Problem 7: Write an application that reads three nonzero integers and determines and prints whether they could represent the sides of a right triangle.

Problem 8: A palindrome is a sequence of characters that reads the same backwards as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message and allow the user to enter a new value.

Problem 9: Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

Problem 10: Write an application that calculates the product of the odd integers from 1 to 15.

Problem 11: Factorials are used frequently in probability problems. The factorial of a positive integer *n* (written *n!* and pronounced “n factorial”) is equal to the product of the positive integers from 1 to *n*. Write an application that evaluates the factorials of the integers from 1 to 10. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 100?

Problem 12: The greatest common divisor (GCD) of two integers is the largest integer that evenly divides each of the two numbers. Write a method gcd that returns the greatest common divisor of two integers. Incorporate the method into an application that reads two values from the user and displays the result.

Problem 13: Write method *distance* to calculate the distance between two points (x1, y1) and (x2, y2). All numbers and return values should be of type double. Incorporate this method into an application that enables the user to enter the coordinates of the points.