Per Scholas

Java Programming - 01

Intro to Java Programming

- History of Java
  - Developed by Sun Microsystems and first released in 1995. Now owned by Oracle.
  - Fast, secure, portable and reliable
  - Applications - Java was designed to allow programs/applications to be built on any platform without the need to rewrite the code to run on another platform
- Java Virtual Machine (JVM)
  - Java compiler compiles the source code and generates bytecode.
  - The bytecode is saved in the form of a .class file which the JVM reads and outputs machine language to the central processing unit (CPU)
- Java Runtime Environment (JRE) - https://java.com/en/
- Learning Java - https://docs.oracle.com/javase/tutorial/
- Run your first Java project:
  - Open Eclipse and make sure you're in the correct workspace for your class projects (The default workspace should be a folder named eclipse-workspace in the Student user folder. You can create as many workspace folders as you like, name them whatever you prefer and place them in any directory/folder, but you should only need one workspace for this course and the default workspace should suffice.)
  - Choose File -> New -> JavaProject (alternately, you may click on the New icon in the upper left corner). If "Java Project" does not appear in the pull-down menu, select "Other…" and search for "Java Project".

- Give your project a name (e.g., MyFirstProject) and leave everything else as default
- If the “Create module-info.java” window appears, click “Don’t Create”
- Click “Finish” (if you’re asked to open the Java perspective choose “Open Perspective”)
- Under the “MyFirstProject” folder, right click on the “src” folder and select New -> Package
- Name the package: com.perscholas.my_first_project
- Click “Finish”
- Under the “src” folder, right-click on the package you just created and select New -> Class
- Name the class (e.g., JavaBasics, FirstClass, MyFirstClass, HelloClass).
- In the “Modifiers” section, make sure “public” is selected and select the checkbox for the “public static void main (String[] args)” method stub. Leave everything else as is.
- Click “Finish” - the window for the class file should open automatically
- If you forgot to check the box for “public static void main(String[] args)” then you must write this in yourself
  - inside the class you just created write the main method:

```
1  public class JavaBasics {
2    public static void main(String[] args) {
3
4    }
5  }
```

- You may now create some variables and/or run methods in the main( ) method or follow the instructions given by your instructor for the remainder of the project
  - A simple project to begin with is to print out “Hello World!”

- Type the following command inside the main ( ) method:
  System.out.println(“Hello World!”;
- Save the file and click the Run icon in the toolbar or right click the file in the Package Explorer and select Run As -> Java Application
- The console should print “Hello World!”

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      System.out.println("Hello World!");
6    }
7  }
```

- Alternate methods to print “Hello World!”
  - System.out.printf(“%s %s!”, “Hello”, “World”);
  - System.out.format(“%s %s!”, “Hello”, “World”);
  - The two preceding methods are equivalent to each other
  - The %s symbols are format specifiers which serve as placeholders for String objects which in this case are “Hello” and “World”. Notice the exclamation mark is not part of “World”, but placed within the string format. There are other types of placeholders such as %d for integers and %f for floating point numbers. You can read more about this at the following links:
    - https://docs.oracle.com/javase/tutorial/essential/io/formatting.html
    - https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

      Here are some more examples:

```
public class JavaBasics {
  public static void main(String[] args) {
    System.out.printf("An integer %d: ", 5);
    System.out.format("\nA Double with 2 decimal places: %.2f", 3.14159);
    System.out.format("\nA String: %s, and an integer: %d", "Hello", 5);
  }
}
```

Output:
An integer: 5
A double with 2 decimal places: 3.14
A String: StringExample, and an integer: 5

- Variables
  - Primitive Data Types
    - byte: 8 bits. Range from -128 to +127.
    - short: 2 bytes. Range from -32,768 to +32,767.
    - int: 4 bytes. Range from -2,147,483,648 to +2,147,483,647.
    - long: 8 bytes. Range from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.
    - float: 4 bytes. Range is approximately ±3.40282347E+38F;
    - double: 8 bytes. Range is approximately ±1.79769313486231570E+308.
    - boolean: 1 byte. Values are true or false.
    - char: 2 bytes. Range from 0 to 65,536 (unsigned).
  - Arithmetic operators can be used with all primitive data types, except for boolean
  - Java is a strongly typed language - data types are declared when creating a variable. Java checks if the value being assigned is compatible with the variable's data type.

- Casting is assigning a value of one type to a variable of another type such as casting a double to an integer
  - One way this can be done is through the use of the Cast Operator (datatype)
    - Example: int num = (int) myDouble;

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      double x = 3.14;
6      int y = (int)x;
7      System.out.println(y);
8    }
9  }
```

    Output: 3
- Named constants with the final keyword
  - Constants are declared in Java by using the final keyword
  - Example: final double TAX_RATE = 0.065;
  - Use constants whenever you need a variable that will remain consistent through an application
- Read more about primitive data types at the following link: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

- Strings
  - A sequence of characters. In Java, strings are objects and hence they come with pre-defined methods such as:
    - myString.charAt(index);
    - myString.length( );
    - myString.equals(myOtherString);
    - myString.concat(“World!”);

- Concatenation using the + operator

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      String a = "The first part ";
6      String b = "plus the second part.";
7      String c = a + b;
8      System.out.println(c);
9    }
10 }
```

Output:
The first part plus the second part.

- Read more about String objects at the following link: https://docs.oracle.com/javase/tutorial/java/data/strings.html (be sure to check out more about Java strings on the subsequent pages)

- https://docs.oracle.com/javase/tutorial/java/data/comparestrings.html

- The Scanner class allows the user to input from the keyboard

```
3  import java.util.Scanner;
4
5  public class JavaBasics {
6    public static void main(String[] args) {
7      String name;
8      Scanner s = new Scanner(System.in);
9      System.out.print("Please enter your name: ");
10     name = s.nextLine();
11     System.out.println("You have entered your name as: " + name);
12   }
13 }
```

Output:
Please enter your name: John
You have entered your name as: John

- Read about the Scanner Class at the following link: https://www.geeksforgeeks.org/scanner-class-in-java/

- String examples:

```java
import java.util.Arrays;
public class JavaBasics {
  public static void main(String[] args) {
    String lower = "hello world!";
    String upper = "HELLO WORLD!";

    System.out.println(lower.toUpperCase());
    System.out.println(upper.toLowerCase());
    System.out.println(lower.contains("lo wor")); // true
    System.out.println(lower.contains("LO WOR")); // false
    System.out.println(lower.toUpperCase().contains("LO WOR")); // true
    System.out.println(lower.equals(upper)); // false
    System.out.println(lower.equalsIgnoreCase(upper)); // true
  }
}
```

Output:
HELLO WORLD!
hello world!
true
false
true
false
true

- Java Operators
  - Operator precedence - refer to charts found on web resources
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html
  - Arithmetic operators perform calculations on numeric data types (including char date type)

```
public class JavaBasics {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    int z = x + y;
    System.out.println(z);
  }
}
```

Example using char variables:

```
public class JavaBasics {
  public static void main(String[] args) {
    char a = 'a'; // ascii value for a = 97
    char b = 'b'; // ascii value for b = 98
    int c = a + b; // c = 97 + 98
    System.out.println(c); // 195
  }
}
```

- Logical operators (&&, | |, !) connect two or more relational expressions and return a boolean result depending on the logic of the relational expressions (i.e., if all, some, or none of the expressions are true/false)
  - In the following example both expressions resolve to true so the if-statement resolves to true

```
public class JavaBasics {
  public static void main(String[] args) {
    if (3 < 5 && 'a' < 'b') {
      System.out.println("This if-statement resolves to true");
    }
  }
}
```

  - This example uses the | | (or) operator

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      String currentDay = "Saturday";
6      if (currentDay.equals("Saturday") || currentDay.equals("Sunday")) {
7        System.out.println("It is the weekend!");
8      }
9    }
10 }
```

- This example uses the ! (not) operator to reverse the equality test for Saturday and Sunday

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      String currentDay = "Monday";
6      if (!currentDay.equals("Saturday") && !currentDay.equals("Sunday")) {
7        System.out.println("Need to go to work!");
8      }
9    }
10 }
```

- Relational operators (<, <=, >, >=, !=, ==) determine the relationship between the values of two operands (e.g., x > y) and return true or false depending on the relationship (i.e., if x is greater than y)

```java
public class JavaBasics {
  public static void main(String[] args) {
    int x = 5, y = 7;
    /*The value of x is not equal to y so the
     * expression resolves to true
     */
    if (x != y) {
      System.out.println("x is not equal to y.");
    }
    /*The value of x is less than y so the expression
     * resolves to true.
     */
    if (x < y) {
      System.out.println("x is less than y.");
    }
    /* The less-than-or-equal and the increment
     * operators are demonstrated in this for-loop
     */
    for (int i = 0; i <= 5; i++) {
      System.out.println(i);
    }
  }
}
```

- Bitwise Operators
  - '&' and '|'
    - https://www.geeksforgeeks.org/bitwise-operators-in-java/
    - Bitwise And & Or | Operator in Java - https://www.youtube.com/watch?v=rzfiE0UuZnQ
  - Shift Operators
    - https://www.geeksforgeeks.org/bitwise-operators-in-java/
    - Left and Right Shift Operator in Java - https://www.youtube.com/watch?v=pv1C0_6k78A
- Control Flow Statements

- If statements - run a certain section of code only if a particular test evaluates to true
- If-else statements - provide a secondary path when an “if” clause evaluates to false.
- If-else-if statements
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      // if block
6      if (3 < 5) {
7        System.out.println("Three really is less than five!");
8      }
9      // if-else block
10     String location = "Per Scholas";
11     if (location.equals("Per Scholas")) {
12       System.out.println("We're at the right place.");
13     } else {
14       System.out.println("We must have made a wrong turn.");
15     }
16     // if-else-if block
17     int score = 80;
18     if (score >= 90) {
19       System.out.println("You made an A");
20     } else if (score >= 80) {
21       System.out.println("You made a B");
22     } else {
23       System.out.println("You made a C or worse");
24     }
25   }
26 }
```

Output:
Three really is less than five!
We're in the right place.
You made a B

- The switch statement
  - Runs one statement from multiple conditions. Similar to a series of if-else-if statements.
  - When the variable being switched on is equal to a case, the statements following that case will run until a break statement is reached. If so, the switch statement ends.
  - Not every case needs to contain a break. If no break appears, then the flow of control will “fall through” to the subsequent cases until a break is reached (if any).
  - A default case can be added to perform a task when none of the preceding cases are true/matched.
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

```java
import java.util.Scanner;

public class JavaBasics {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.print("Please enter 1,2 or 3: ");
    int day = s.nextInt();
    s.close();
    switch(day) {
    case 1:
      System.out.println("Sunday");
      break;
    case 2:
      System.out.println("Monday");
      break;
    case 3:
      System.out.println("Tuesday");
      break;
    default:
      System.out.println("Invalid entry");
    }
  }
}
```

Output:
Please enter 1,2 or 3: 2
Monday

- Loops
  - A statement or a group of statements that are repeated a fixed number of times or while a given condition is true.
  - Types of loops:
    - for loop - loops through a block of code for a set amount of times
      - for (int i = 0; i < 10; i++) { }

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      for (int i = 0; i < 10; i++) {
6        System.out.print(i + " ");
7      }
8    }
9  }
```

Output:
0 1 2 3 4 5 6 7 8 9

- enhanced for loop
  - for (String s : myListOfStrings) { }
  - Arrays and Collections will be discussed in another section, but here is an example of how these would be used in an enhanced for-loop

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      String[] colors = {"red","green","blue"};
6      for (String s : colors) {
7        System.out.println(s);
8      }
9    }
10 }
```

Output:
red
green
blue

- while loop - loops through block of code while a condition is true
  - while (continue == true) { // continue looping through this block of code}
  - can also be written: while (continue) { }

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      int counter = 0;
6      while (counter < 10) {
7        System.out.print(counter);
8        counter++;
9      }
10   }
11 }
```

Output
0123456789

- do-while loop - runs a block of code at least once and evaluates the condition after each loop.
  - do { // run this block of code and then evaluate the while condition. Must plan for eventual failure of condition or else loop will run infinitely } while (continue);

```
3  public class JavaBasics {
4    public static void main(String[] args) {
5      int counter = 10;
6      do {
7        System.out.print(counter);
8        counter++;
9      } while (counter < 10);
10   }
11 }
```

Output:
10

- The continue and break keywords
  - The continue statement - in a loop, the current iteration ends and the flow continues to the next iteration.

```java
public class JavaBasics {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4 || i == 7) {
        continue;
      }
      System.out.println(i);
    }
  }
}
```

Output:
01235689

- The break statement - ends the loop and continues to the next block of code

```java
public class JavaBasics {
  public static void main(String[] args) {
    for (int i = 0; i < 10; i++) {
      if (i == 4) {
        break;
      }
      System.out.println(i);
    }
  }
}
```

Output:
0123

- For-loops can be nested such as in the following code:

```java
public class JavaBasics {
  public static void main(String[] args) {
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
        System.out.println("Outer loop: " + i + ", Inner loop: " + j);
      }
    }
  }
}
```

Output:
Outer loop: 0, Inner loop: 0
Outer loop: 0, Inner loop: 1
Outer loop: 0, Inner loop: 2
Outer loop: 1, Inner loop: 0
Outer loop: 1, Inner loop: 1
Outer loop: 1, Inner loop: 2
Outer loop: 2, Inner loop: 0
Outer loop: 2, Inner loop: 1
Outer loop: 2, Inner loop: 2

- Scope
  - The part of the program that has access to a variable
    - A variable is visible only to statements inside its scope
      - Local variable's scope begins at the declaration and ends at the end of the method
      - Variables are within scope inside the blocks they were declared in and in any nested/inner blocks

```
3  public class JavaBasics {
4      public static void main(String[] args) {
5          for (int i = 0; i < 3; i++) {
6              System.out.println(i);
7          }
8          System.out.println(i);
9      }
10 }
```

      - Cannot have two local variables with the same name within the same scope
  - https://www.geeksforgeeks.org/variable-scope-in-java/
  - Scope examples:

Notice the error in line 8. This is due to the second print statement attempting to access the variable i which is declared inside the for-loop block and therefore out of scope.

In the following example, there is no error because the variable i is declared outside the for-loop block and inside the main method block and therefore available to the second print statement.