Programming I

Section 3

Operators, Branching & Looping

Learning Outcomes

- After this lecture you will be able to:
  - Distinguish between Expressions & Statements
  - Describe the Java Syntax
  - Outline the various Operators
  - Create Conditional Tests
  - Describe and implement Conditional Branching
  - Use various looping constructs

- ***Reading*** and ***studying*** recommended readings is essential to improve your understanding of the above

Expressions

- *"A construct made up of variables, operators and method invocations, which are constructed according to the syntax of the language, that evaluate to a single value". Source: oracle.com*
- Expressions are core components of statements.

```
int  age = 25;

myArray[0] = 100;

int  result = 2 + 5;

if(value1 == value2)
    System.out.print("value1 == value2");
```

Statements

- Roughly equivalent to sentences in natural language
  - forms a complete unit of execution
  - expressions can be made into statements by terminating expression with ;
  - declarations, assignments method calls etc.

```
int  x = 7; // declare & initialise variable

String name = "James";

x = x * 10;

System.out.println("x equals " + x);

double a = Math.random();

// this is a single line comment

/*
 *  this is a multi-line comment
 */
```

Syntax

- Each statement must ends in a semicolon

```
x = x * 10;
```

- Most white space doesn't matter

```
number =   x     +     1;
```

- Variables are declared with a name and a type

```
int  age; //  type: int       name: age
```

- Classes and methods must be surrounded by curly brackets

```
public void move() {
   // put code here
}
```

Assignment, Arithmetic Operators

- Binary operators require two operands
- Assignment operator =
  (= can also be used on objects to assign *object references*)
- Arithmetic Operators (binary)
  - + Additive operator (also used for String concatenation)
  - - Subtraction operator
  - * Multiplication operator
  - / Division operator
  - % Remainder operator

Unary & Ternary Operators

- unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

  + Unary plus operator; indicates positive value (numbers are positive without this, however)
  - Unary minus operator; negates an expression
  ++ Increment operator; increments a value by 1
  -- Decrement operator; decrements a value by 1
  ! Logical complement operator; inverts the value of a boolean

- Ternary Operator

  ? : (shorthand for if-then-else statement) `maxVal = (a > b) ? a : b;`

Conditional Tests

- A *conditional test* is an expression that results in a *boolean value* (True of False)

- Equality Operators
  - ==(equal to)
  - != (not equal to)

- Relational operators include:
  - < (less than)
  - <= (less than or equal to)
  - > (greater than)
  - >= (greater than or equal to)

Comparing Objects

- An if selection statement allows the program to make a decision on the basis of a conditions value.
- With primitive data types, we have only one way to compare them, but with objects (reference data type), we have two ways to compare them.
  - We can test whether two variables point to the same object (use ==), or
  - We can test whether two distinct objects have the same contents.
- *assignment* operator vs *equals* operator

```
String s1 = "Java";
String s2 = new String("Java");

if (s1 == s2) // compares address
if (s1.equals(s2)) // compares contents (inherited from class object)
```

Conditional Branching

```
public class Test {

    public static void main(String[] args) {

        int y = 15; // assign 15 to y

        if (y == 15){ // executes only if condition is true
            System.out.println("y is equal to 15");
        }

        System.out.println("This runs no matter what happens");

    }

}
```

If/else Statement

```
import java.util.Scanner;

public class TestSocre{

  public static void main(String[] args) {

    int examResult; // declare variable

    Scanner input = new Scanner(System.in); // create Scanner object

    System.out.print("Enter your Exam Result: ");
    examResult = input.nextInt();

       if (examResult >= 40){
           System.out.println("Congratulations you have passed");
       }
       else {
           System.out.println("Unfortunately you have failed");
       }

  } // end method

} // end class
```

This statement is executed if the examResult is greater than or equal to 40.

This statement is executed if the examResult is less than 40.

If/else Ladder

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

    int marks = 76;
    String grade;

      if (marks >= 70 && marks <= 100) {
          grade = "First Class Honours";
      } else if (marks >= 60) && marks < 70 {
          grade = "2.1";
      } else if (marks >= 50 && marks < 60) {
          grade = "2.2";
      } else if (marks >= 40 && marks < 50) {
          grade = "Pass";
      } else {
          grade = "Fail";
      }
      System.out.println("Grade = " + grade);

  }
}
```

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

    int marks = 76;
    char grade;

   if (marks >= 70) {
       grade = 'A';
   } else if (marks >= 60) {
       grade = 'B';
   } else if (marks >= 50) {
       grade = 'C';
   } else if (marks >= 40) {
       grade = 'D';
   } else {
       grade = 'F';
   }
     System.out.println("Grade = " + grade);
  }
}
```

Nested-if Statement

- The then and else block of an if statement can contain any valid statements, including other if statements. An if statement containing another if statement is called a nested-if statement.

```
if (testScore >= 40) {

    if (studentAge < 10) {
        System.out.println("You did a great job");
    } else {
        System.out.println("You did pass"); //test score >= 40 and age >= 10
    }

} else { //test score < 40
        System.out.println("You did not pass");
}
```

switch Statement

```
import java.util.Scanner;

public class SwitchExample {

    public static void main(String[] args) {
        int gradeLevel;
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your year (First Year - 1, Second Year - 2,...):" );
        gradeLevel = scanner.nextInt();

        switch (gradeLevel) {

          case 1: System.out.print("Go to IT 125");
           break;
          case 2: System.out.print("Go to the Cairnes Theatre");
            break;
          case 3: System.out.print("Go to the Larmour Theatre");
           break;
          case 4: System.out.print("Go to the Quadrangle");
           break;
          default: System.out.print("Input error: Invalid Data");
           break;
          }
     }
}
```

This statement is executed if the gradeLevel is equal to 1.

This statement is executed if the gradeLevel is equal to 4.

Boolean Operators

- The && and || operators perform *Conditional-AND* and *Conditional-OR* operations on two boolean expressions.
- These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.

```
class ConditionalDemo {

    public static void main(String[] args){
        int value1 = 1;
        int value2 = 2;

        if((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");

    }
}
```

Semantics of Boolean Operators

- Boolean operators and their meanings:

P	Q	P && Q	P || Q	!P
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Operator Precedence Rules

Group	Operator	Precedence	Associativity
Subexpression	( )	10 (If parentheses are nested, then innermost subexpression is evaluated first.)	Left to right
Postfix increment and decrement operators	++ --	9	Right to left
Unary operators	- !	8	Right to left
Multiplicative operators	* / %	7	Left to right
Additive operators	+ -	6	Left to right
Relational operators	< <= > >=	5	Left to right
Equality operators	== !=	4	Left to right
Boolean AND	&&	3	Left to right
Boolean OR	||	2	Left to right
Assignment	=	1	Right to left

Loops

- Repetition statements control a block of code to be executed for a fixed number of times or until a certain condition is met.
- Count-controlled repetitions terminate the execution of the block after it is executed for a fixed number of times.
- Sentinel-controlled repetitions terminate the execution of the block after one of the designated values called a *sentinel* is encountered.
- Repetition statements are called loop statements also.
- Java has three standard looping constructs
  - while, do-while and for
  - Logic - as long as some condition is true, execute the code in the loop block

```
while (something == true){
        keepDoingSomething();
}
```

  - The loop block is bounded by curly brackets { }

Counter-Controlled Repetition

- Counter-controlled repetition requires
  - a **control variable** (or loop counter)
  - the **initial value** of the control variable
  - the **increment** (or **decrement**) by which the control variable is modified each time through the loop (also known as **each iteration of the loop**)
  - the **loop-continuation condition** that determines if looping should continue.

Counter controlled Loop

```
// Counter controlled while loop
public class WhileCounter {

    public static void main(String[] args) {

    // declare & initialise counter
        int counter = 1;

        while (counter <= 10){
            System.out.printf("%d ", counter);
            ++ counter; // counter += 1;
        }
        System.out.println();
    } // end main method
} // end class
```

Boolean Expression

Statement (loop body)

Sentinel Controlled Loop

```java
import java.util.Scanner;

public class WhileSentinel {

    public static void main(String[] args) {

        int number = 0; // declare & initialise variable

        Scanner scanner = new Scanner(System.in);

        while (number != -1){
            System.out.print("Enter mark (type -1 to quit): ");
            number = scanner.nextInt();
        }

        System.out.println("Program Terminated ...");


    } // end main method

} // end class
```

The do-while Statement

```java
int sum = 0, number = 1;

do {

    sum += number;
    number++;

} while ( sum <= 1000000 );
```

These statements are executed as long as sum is less than or equal to 1,000,000.

for Repetition Statement

- With a for statement:
  - when execution begins, the control variable is declared and initialized.
  - Next, the program checks the loop-continuation condition, which is between the two required semicolons.
  - If the condition initially is true, the body statement executes.
  - After executing the loop's body, the program increments the control variable in the increment expression, which appears to the right of the second semicolon.
  - Then the loop-continuation test is performed again to determine whether the program should continue with the next iteration of the loop.
  - A common logic error with counter-controlled repetition is an **off-by-one error**.

Counter controlled loop

```
public class forCounter {
    public static void main(String[] args) {
        for (int counter = 1; counter <= 10; counter ++){
            System.out.printf("%d ",counter);
        } // end for statement
        System.out.println();
    } // end main method
} // end class
```

Initialization

Boolean Expression

Increment

The Nested-for Statement

- Nesting a for statement inside another for statement is commonly used technique in programming.
- Let's generate the following table using nested-for statement.

```
Console  Problems  @ Javadoc  Declaration
<terminated> forTable [Java Application] /System/Library/Java/J
      5   10   15   20   25
 11 1045 2090 3135 4180 5225
 12 1140 2280 3420 4560 5700
 13 1235 2470 3705 4940 6175
 14 1330 2660 3990 5320 6650
 15 1425 2850 4275 5700 7125
 16 1520 3040 4560 6080 7600
 17 1615 3230 4845 6460 8075
 18 1710 3420 5130 6840 8550
 19 1805 3610 5415 7220 9025
 20 1900 3800 5700 7600 9500
```

Formatted Nested Loop

```java
import java.util.Formatter;

public class ForTable {

     public static void main(String[] args) {

          final double PRICE_PER_SQ_FT = 19.00; //€19 per sq. ft.
          double price;
          Formatter formatter = new Formatter(System.out);

          for (int i=5; i<=25; i+=5){
               formatter.format ("%8d ", i);
          }
          System.out.println();

          for (int width = 11; width <=20; width++){
               System.out.print (" " + width);

               for (int length = 5; length <=25; length+=5){
                    price = width * length * PRICE_PER_SQ_FT;
                    formatter.format ("%8.2f ", price);
                    // System.out.format("%8.2f ", price);
               }
               System.out.println(" ");
          }
          formatter.close();

     } //end main()
} //end class
```

println vs print

- Print things out on a new line use:

```
System.out.println("output to screen");
// inserts a new line
```

- Print things out on the same line use:

```
System.out.print("output to screen");
// keeps printing on the same line
```

- Special characters \n \t etc.

```
public class BeerSong {

    public static void main(String[] args) {

        int beerNum = 99;
        String word = "bottles";

        while (beerNum > 0) {

            if (beerNum == 1) {
                word = "bottle";
            }

            System.out.println(beerNum + " " + word + " of beer on the wall");
            System.out.println(beerNum + " " + word + " of beer");
            System.out.println("Take one down.");
            System.out.println("Pass it around.");

            beerNum = beerNum - 1;

            if (beerNum > 0) {
                System.out.println(beerNum +  " " + word + " of beer on the wall");
            }
            else {
                System.out.println("No more bottles of beer on the wall");
            }
        } // end loop

    } // end main method

} // end class
```

Summary

- Statements end in a semicolon ;
- Code block are defined by a pair of curly brackets {}
- Assignment operator is One equals sign =
- equals operator is Two equals signs ==
- A while loop runs everything within its block ({}) once conditional test is ***true***
- If the conditional test is ***false***, the while loop block will not execute and control moves down to the code immediately after the loop block
- Put a Boolean test inside parentheses:
  - `while (y == 12){ }`

Summary

- Expressions & Statements
- Operators
- Conditional Testing
- if, if/else, if else ladder & switch statements
- while, do while & for loops
- Nested Loops
- Formatter Class
- print VS println