Programming Basics

**Sang Shin
Michèle Garoche
www.javapassion.com
"Learn with Passion!"**

Objectives

At the end of the lesson, student should be able to:

- Identify the basic parts of a Java program

- Understand the concepts of Java literals, primitive data types, variable types, identifiers and operators

- Develop a simple valid Java program using the concepts learned in this chapter

JEDI

Dissecting My First Java Program

Dissecting my First Java Program

```
1  public class Hello
2  {
3      /**
4       * My first Java program
5       */
6     public static void main( String[] args ){
7
8            //prints the string Hello world on screen
9            System.out.println("Hello world");
10
11     }
12 }
```

Dissecting my First Java Program

```
1  public class Hello
2  {
3     /**
4      * My first Java program
5      */
```

- Indicates the name of the class which is Hello

- In Java, all code should be placed inside a class declaration

- The class uses an access specifier public, which indicates that our class is accessible to other classes from either the same or other packages (packages are a collection of classes). We will be covering packages and access specifiers later.

JEDI

Dissecting my First Java Program

```
1  public class Hello
2  {
3     /**
4      * My first Java program
5      */
```

- The next line which contains a curly brace { indicates the start of a block.

- In this code, we placed the curly brace at the next line after the class declaration, however, we can also place this next to the first line of our code. So, we could actually write our code as:

  public class Hello {

Dissecting my First Java Program

```
1  public class Hello
2  {
3     /**
4      * My first Java program
5      */
```

- The next three lines indicates a Java comment.

- A comment

  - Something used to document a part of a code.

  - It is not part of the program itself - does not affect the programming logic - and used only for documentation purposes.

  - It is good programming practice to add comments to your code.

Dissecting my First Java Program

```
1  public class Hello
2  {
3      /**
4       * My first Java program
5       */
6      public static void main( String[] args ){
```

- Indicates the name of a method in Hello class - *main (..)* is the name of the method

- The *main(..)* method is the starting point of a Java program.

- All programs except Applets written in Java start with the *main(..)* method.

- Make sure to follow the exact method signature.

JEDI

Dissecting my First Java Program

```
1   public class Hello
2   {
3      /**
4       * My first Java program
5       */
6      public static void main( String[] args ){
7
8           //prints the string "Hello world" on screen
```

- The next line is also a Java comment

Dissecting my First Java Program

```
1  public class Hello
2  {
3     /**
4      * My first Java program
5      */
6    public static void main( String[] args ){
7
8          //prints the string "Hello world" on screen
9          System.out.println("Hello world");
```

- The System.out.println("something to print"); prints the text enclosed by double-quotation on the standard output device - typically a display screen.

Dissecting my First Java Program

```
1  public class Hello
2  {
3     /**
4      * My first Java program
5      */
6     public static void main( String[] args ){
7
8          //prints the string "Hello world" on screen
9          System.out.println("Hello world");
10
11     }
12 }
```

- The last two lines which contain the two curly braces are used to close the main method and class respectively.

Coding Requirements

1. A file that contains Java code must end with the `.java` extension.

2. A file that contains Java class code must match the name of your public class. So for example, if the name of your public class is *Hello*, you should save it in a file called `Hello.java`.

Java Comments

Java Comments

- Comments
  - These are notes written to a code for documentation purposes.
  - Those texts are not part of the program and does not affect the flow of the program.

- 3 Types of comments in Java
  - C++ Style Comments
  - C Style Comments
  - Special Javadoc Comments

Java Comments

- C++-Style Comments

    - C++ Style comments starts with //

    - All the text after // are treated as comments

    - For example:
      ```
      // This is a C++ style or single line comments
      ```

Java Comments

- C-Style Comments

    - C-style comments or also called multiline comments starts with a /* and ends with a */.

    - All text in between the two delimeters are treated as comments.

    - Unlike C++ style comments, it can span multiple lines.

    - For example:

    ```
    /* this is an exmaple of a
       C style or multiline comments */
    ```

Java Comments

- Special Javadoc Comments

  - Special Javadoc comments are used for generating an HTML documentation for your Java programs.

  - Start the line with /** and ending it with */.

  - Like C-style comments, it can also span lines.

  - It can also contain certain tags to add more information to your comments.

  - For example:
    ```
    /**   This is an example of special java doc
          comments used for \n  generating an html
          documentation. It uses tags like:
          @author Florence Balagtas
          @version 1.2
       */
    ```

Statements and Blocks

Java Statements

- Statement

  - One or more lines of code terminated by a semicolon.

  - Example:
    ```
    System.out.println("Hello world");
    ```

Java Blocks

- Block

  - Is one or more statements bounded by an opening and closing curly braces that groups the statements as one unit.

  - Blocks can be nested

  - Any amount of white space is allowed.

  - Example:
    ```
    public static void main( String[] args ){
        System.out.println("Hello");
        System.out.println("world");
    }
    ```

Java Statements and Blocks Coding Guidelines

1. In creating blocks, you can place the opening curly brace in line with the statement. For example:

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

or you can place the curly brace on the next line, like,

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

Java Statements and Blocks Coding Guidelines

2. You should indent the next statements after the start of a block. For example:

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

    System.out.println("Hello");

    System.out.println("world");

}
```

Java Identifiers

Java Identifiers

- Identifiers

  - Are tokens that represent names of variables, methods, classes, etc.

  - Examples of identifiers are: Hello, main, System, out.

- Java identifiers are case-sensitive.

  - This means that the identifier **Hello** is not the same as **hello**.

Java Identifiers

- Identifiers must begin with either a letter, an underscore "_", or a dollar sign "$". Letters may be lower or upper case. Subsequent characters may use numbers 0 to 9.

- Identifiers cannot use Java keywords like class, public, void, etc. We will discuss more about Java keywords later.

Java Identifiers
Coding Recommendations

1. For names of classes, capitalize the first letter of the class name. For example,

   ThisIsAnExampleOfClassName

2. For names of methods and variables, the first letter of the word should start with a small letter. For example,

   thisIsAnExampleOfMethodName

Java Identifiers
Coding Guidelines

3. In case of multi-word identifiers, use capital letters to indicate the start of the word except the first word. For example,

  charArray, fileNumber

4. Avoid using underscores at the start of the identifier such as _read or _write (unless you are writing system level code).

Java Keywords

- Keywords are predefined identifiers reserved by Java for a specific purpose.

- You cannot use keywords as your own identifiers - names for your variables, classes, methods ... etc.

- The next slide contains the list of the Java Keywords.

Java Keywords

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

Java Literals

Java Literals

- Literals are tokens that do not change - they are constant.
- The different types of literals in Java are:
    - Integer Literals
    - Floating-Point Literals
    - Boolean Literals
    - Character Literals
    - String Literals

Java Literals: Integer

- Integer literals come in different formats:

  - decimal (base 10)

  - hexadecimal (base 16)

  - octal (base 8).

Java Literals: Integer

- Special Notations in using integer literals in our programs:

  - Decimal

    - No special notation

    - example: 12

  - Hexadecimal

    - Precede by 0x or 0X

    - example: 0xC

  - Octal

    - Precede by 0

    - example: 014

Java Literals: Floating Point

- Represents decimals with fractional parts
  - Example: 3.1416

- Can be expressed in standard or scientific notation
  - Example: 583.45 (standard), 5.8345e2 (scientific)

JEDI

Java Literals: Boolean

- Boolean literals have only two values, true or false.
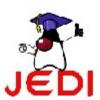
Java Literals: Character

- Character Literals represent single Unicode characters.

- Unicode character
  - a 16-bit character set that replaces the 8-bit ASCII character set.
  - Unicode allows the inclusion of symbols and special characters from other languages.

Java Literals: Character

- To use a character literal, enclose the character in single quote delimiter.

- For example

  - the letter a, is represented as 'a'.

  - special characters such as a newline character, a backslash is used followed by the character code. For example, '\n' for the newline character, '\r' for the carriage return, '\b' for backspace.

Java Literals: String

- String literals represent multiple characters and are enclosed by double quotes.

- An example of a string literal is, "Hello World".

Variables

Variables

- A variable is used to store the state of objects.

- A variable has a:
  - data type
    - The data type indicates the type of value that the variable can hold.
  - name
    - The variable name must follow rules for identifiers.

Declaring and Initializing Variables

- Declare a variable as follows:

    ```
    <data type>  <name> [=initial value];
    ```

- The Java programming language is statically-typed, which means that the <date type> must first be declared before variables can be used

- The <data type> can be either Primitive type or Reference type

    - *double grade = 0.0;   // Primitive type*
    - *Person x;   // Reference type*

Declaring and Initializing Variables: Sample Program

```
1   public class VariableSamples {
2       public static void main( String[] args ){
3           // declare a data type with variable name
4           // result and boolean data type
5           boolean result;
6
7           // declare a data type with variable name
8           // option and char data type
9           char option;
10          option = 'C'; //assign 'C' to option
11
12          // declare a data type with variable name
13          // grade, double data type and initialized
14          // to 0.0
15          double grade = 0.0;
16      }
17  }
```

Declaring and Initializing Variables: Coding Guidelines

1. It always good to initialize your variables as you declare them.

2. Use descriptive names for your variables. Like for example, if you want to have a variable that contains a grade for a student, name it as, grade and not just some random letters you choose.

JEDI

Declaring and Initializing Variables: Coding Guidelines

3. Declare one variable per line of code. For example, the variable declarations,

```
double exam=0;
double quiz=10;
double grade = 0;
```

is preferred over the declaration,

```
double exam=0, quiz=10, grade=0;
```

Outputting Variable Data

- In order to output the value of a certain variable, we can use the following commands:

```
System.out.println()

System.out.print()
```

Outputting Variable Data: Sample Program

```
1  public class OutputVariable {
2        public static void main( String[] args ){
3              int value = 10;
4              char x;
5              x = 'A';
6
7              System.out.println( value );
8              System.out.println( "The value of x=" + x );
9        }
10 }
```

The program will output the following text on screen:

```
10
The value of x=A
```

System.out.println() vs. System.out.print()

- System.out.println()
  - Appends a newline at the end of the data output

- System.out.print()
  - Does not append newline at the end of the data output

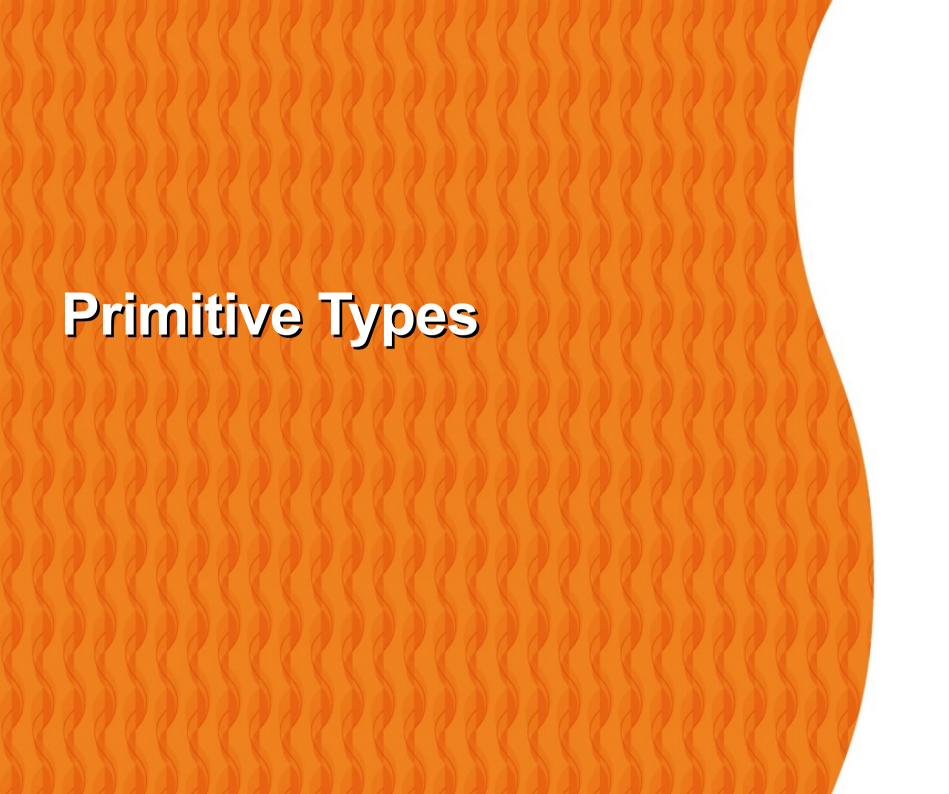# System.out.println() vs. System.out.print() Examples

- Program 1:

```
System.out.print("Hello");
System.out.print("World");
```

Output:
    HelloWorld

- Program 2:

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

Output:
    Hello
    World

Primitive Types

Primitive Data Types

- The Java programming language defines eight primitive data types.

  - boolean (for logical)

  - char (for textual)

  - byte

  - short

  - int

  - long (integral)

  - double

  - float (floating point).

Primitive Data Types

- Used to hold non-Object values

- Provided for higher performance

Primitive Data Types: Logical-boolean

- A *boolean* data type represents two states: *true* and *false*.

- An example is,
  boolean result = true;

- The example shown above, declares a variable named *result* as *boolean* type and assigns it a value of *true*.

Primitive Data Types: Textual-char

- A character data type (char), represents a single Unicode character.

- It must have its literal enclosed in single quotes(' ').

- For example,
  ```
  'a'  //The letter a
  '\t' //A tab
  ```

- To represent special characters like ' (single quotes) or " (double quotes), use the escape character \. For example,
  ```
  '\'' //for single quotes
  '\"' //for double quotes
  ```

Primitive Data Types: Integral – byte, short, int & long

- Integral data types in Java uses three forms – decimal, octal or hexadecimal.

- Examples are,
  ```
  2   //The decimal value 2
  077 //The leading 0 indicates an octal value
  0xBACC //The leading 0x indicates a hex value
  ```

- Integral types has int as default data type.

- You can define its long value by appending the letter l or L.

- For example:
  ```
  10L
  ```

Primitive Data Types: Integral – byte, short, int & long

- Integral data type have the following ranges:

Integer Length	Name or Type	Range
8 bits	byte	-2^7 to 2^7-1
16 bits	short	-2^{15} to $2^{15}-1$
32 bits	int	-2^{31} to $2^{31}-1$
64 bits	long	-2^{63} to $2^{63}-1$

Primitive Data Types: Integral – byte, short, int & long

- Coding Guidelines:

    - In defining a long value, a lowercase L is not recommended because it is hard to distinguish from the digit 1.

Primitive Data Types: Floating Point – float and double

- Floating point types has double as default data type.

- Floating-point literal includes either a decimal point or one of the following,
  ```
  E or e //(add exponential value)
  F or f //(float)
  D or d //(double)
  ```

- Examples are,
  ```
  3.14   //A simple floating-point value (a double)
  6.02E23     //A large floating-point value
  2.718F      //A simple float size value
  123.4E+306D//A large double value with redundant D
  ```

Primitive Data Types: Floating Point – float and double

- Floating-point data types have the following ranges:

Type	Size	Range
float	4 bytes (32 bits)	+/- 3.4 * 10 (power of 38)
double	8 bytes(64 bits)	+/- 1.8 * 10 (power of 308)

Primitive Type vs. Reference Type

Reference type variables vs. Primitive type variables

- Primitive type variables

  - Variables with primitive data types such as int or long.

  - Stores data in the actual memory location of where the variable is

- Reference Variables

  - Variables that stores the memory address of an object

    - Points to another memory location where the actual data is

  - When you declare a variable of a certain class, you are actually declaring a reference variable to the object with that certain class.

Example

- Suppose we have two variables with data types int and String.

```
int num = 10; // primitive type
String name = "Hello"; // reference type
```

Example

- The picture shown below is the actual memory of your computer, wherein you have the address of the memory cells, the variable name and the data they hold.

Memory Address	Variable Name	Data
1001	num	10
:		:
1563	name	Address(2000)
:		:
:		:
2000		"Hello"

Operators

Operators

- Different types of operators:

    - arithmetic operators

    - relational operators

    - logical operators

    - conditional operators

- These operators follow a certain kind of precedence so that the compiler will know which operator to evaluate first in case multiple operators are used in one statement.

JEDI

Arithmetic Operators

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2
-	op1 - op2	Subtracts op2 from op1

Arithmetic Operators: Sample Program

```
1   public class ArithmeticDemo  {
2         public static void main(String[] args){
3               //a few numbers
4               int i = 37;
5               int j = 42;
6               double x = 27.475;
7               double y = 7.22;
8               System.out.println("Variable values...");
9               System.out.println("    i = " + i);
10              System.out.println("    j = " + j);
11              System.out.println("    x = " + x);
12              System.out.println("    y = " + y);
                System.out.println("Adding...");
13              System.out.println("    i + j = " + (i + j));
14              System.out.println("    x + y = " + (x + y));
```

Arithmetic Operators: Sample Program

```
15          //subtracting numbers
16            System.out.println("Subtracting...");
17            System.out.println("    i - j = " + (i - j));
18            System.out.println("    x - y = " + (x - y));
19
20          //multiplying numbers
21            System.out.println("Multiplying...");
22            System.out.println("    i * j = " + (i * j));
23            System.out.println("    x * y = " + (x * y));
24
25           //dividing numbers
26            System.out.println("Dividing...");
27            System.out.println("    i / j = " + (i / j));
28            System.out.println("    x / y = " + (x / y));
```

Arithmetic Operators: Sample Program

```
29      //computing the remainder resulting from dividing
30       // numbers
31      System.out.println("Computing the remainder...");
32      System.out.println("    i % j = " + (i % j));
33      System.out.println("    x % y = " + (x % y));
34
35      //mixing types
36      System.out.println("Mixing types...");
37      System.out.println("    j + y = " + (j + y));
38      System.out.println("    i * x = " + (i * x));
39    }
40}
```

Arithmetic Operators: Sample Program Output

```
Variable values...                 Dividing...
    i = 37                             i / j = 0
    j = 42                             x / y = 3.8054  Computing
    x = 27.475                     the remainder...
    y = 7.22                           i % j = 37
Adding...                              x % y = 5.815
    i + j = 79                     Mixing types...
    x + y = 34.695                     j + y = 49.22
Subtracting...                         i * x = 1016.58
    i - j = -5
    x - y = 20.255
Multiplying...
    i * j = 1554
    x * y = 198.37
```

Arithmetic Operators

- Note:

    - When an integer and a floating-point number are used as operands to a single arithmetic operation, the result is a floating point. The integer is implicitly converted to a floating-point number before the operation takes place.

Increment and Decrement Operators

- unary increment operator (++)

- unary decrement operator (--)

- Increment and decrement operators increase and decrease a value stored in a number variable by 1.

- For example, the expression,

```
count=count + 1;//increment the value of count by 1
```

is equivalent to,

```
count++;
```

Increment and Decrement Operators

- The increment and decrement operators can be placed before or after an operand.

- When used before an operand, it causes the variable to be incremented or decremented by 1 first, and then the new value is used in the expression in which it appears.

- For example,

```
int i = 10;
int j = 3;
int k = 0;
k = ++j + i; // j =4, k = 4+10 = 14
```

Increment and Decrement Operators

- When the increment and decrement operators are placed after the operand, the old value of the variable will be used in the expression where it appears.

- For example,

```
int i = 10;
int j = 3;
int k = 0;
k = j++ + i; // k = 3+10 = 13, j = 4
```

Relational Operators

- Relational operators compare two values and determines the relationship between those values.

- The output of evaluation are the boolean values true or false.

Operator	Use	Description
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

JEDI

Relational Operators: Sample Program

```
1   public class RelationalDemo{
2       public static void main(String[] args){
3           //a few numbers
4           int i = 37;
5           int j = 42;
6           int k = 42;
7           System.out.println("Variable values...");
8           System.out.println("    i = " +i);
9           System.out.println("    j = " +j);
10          System.out.println("    k = " +k);
11           //greater than
12          System.out.println("Greater than...");
13          System.out.println("    i > j = "+(i>j));//false
14          System.out.println("    j > i = "+(j>i));//true
15          System.out.println("    k > j = "+(k>j));//false
```

Relational Operators: Sample Program

```
16        //greater than or equal to
17            System.out.println("Greater than or equal to...");
18            System.out.println("    i >= j = "+(i>=j));//false
19            System.out.println("    j >= i = "+(j>=i));//true
20            System.out.println("    k >= j = "+(k>=j));//true
21         //less than
22            System.out.println("Less than...");
23            System.out.println("    i < j = "+(i<j));//true
24            System.out.println("    j < i = "+(j<i));//false
25            System.out.println("    k < j = "+(k<j));//false
26       //less than or equal to
27            System.out.println("Less than or equal to...");
28            System.out.println("    i <= j = "+(i<=j));//true
29            System.out.println("    j <= i = "+(j<=i));//false
30            System.out.println("    k <= j = "+(k<=j));//true
```

Relational Operators: Sample Program

```
31  //equal to
32     System.out.println("Equal to...");
33     System.out.println("    i == j = " + (i==j));//false
34     System.out.println("    k == j = " + (k==j));//true
35  //not equal to
36     System.out.println("Not equal to...");
37     System.out.println("    i != j = " + (i!=j));//true
38     System.out.println("    k != j = " + (k!=j));//false
39    }
40  }
```

Relational Operators: Sample Program Output

```
Variable values...                 Less than or equal to...
    i = 37                             i <= j = true
    j = 42                             j <= i = false
    k = 42                             k <= j = true
Greater than...                    Equal to...
    i > j = false                      i == j = false
    j > i = true                       k == j = true
    k > j = false                  Not equal to...
Greater than or equal to...            i != j = true
    i >= j = false                     k != j = false
    j >= i = true
    k >= j = true
Less than...
    i < j = true
    j < i = false
    k < j = false
```

Logical Operators

- Logical operators have one or two boolean operands that yield a boolean result.

- There are six logical operators:
    - && (logical AND)
    - & (boolean logical AND)
    - || (logical OR)
    - | (boolean logical inclusive OR)
    - ^ (boolean logical exclusive OR)
    - ! (logical NOT)

Logical Operators

- The basic expression for a logical operation is,

    `x1 op x2`

    where,
    x1, x2 - can be boolean expressions, variables or constants
    op - is either &&, &, ||, | or ^ operator.

- The truth tables that will be shown next, summarize the result of each operation for all possible combinations of x1 and x2.

Logical Operators: &&(logical) and &(boolean logical) AND

- Here is the truth table for && and &,

x1	x2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Logical Operators: &&(logical) and &(boolean logical) AND

- The basic difference between && and & operators :
    - && supports short-circuit evaluations (or partial evaluations), while & doesn't.

- Given an expression:

      exp1 && exp2

    - && will evaluate the expression exp1, and immediately return a false value is exp1 is false.

    - If exp1 is false, the operator never evaluates exp2 because the result of the operator will be false regardless of the value of exp2.

- In contrast, the & operator always evaluates both exp1 and exp2 before returning an answer.

Logical Operators: &&(logical) and &(boolean logical) AND

```
1  public class TestAND {
2      public static void main( String[] args ){
3          int  i  = 0;
4          int  j  = 10;
5          boolean test= false;
6          //demonstrate &&
7          test = (i > 10) && (j++ > 9);
8          System.out.println(i);
9          System.out.println(j);
10         System.out.println(test);
11         //demonstrate &
12         test = (i > 10) & (j++ > 9);
13         System.out.println(i);
14         System.out.println(j);
15         System.out.println(test);
16     }
17 }
```

Logical Operators: &&(logical) and &(boolean logical) AND

- The output of the program is,
  0
  10
  false
  0
  11
  false

- Note, that the j++ on the line containing the && operator is not evaluated since the first expression (i>10) is already equal to false.

Logical Operators: || (logical) and | (boolean logical) inclusive OR

- Here is the truth table for || and |,

x1	x2	Result
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Logical Operators: || (logical) and | (boolean logical) inclusive OR

- The basic difference between || and I operators :

  - || supports short-circuit evaluations (or partial evaluations), while | doesn't.

- Given an expression:

  `exp1 || exp2`

  - || will evaluate the expression exp1, and immediately return a true value is exp1 is true

  - If exp1 is true, the operator never evaluates exp2 because the result of the operator will be true regardless of the value of exp2.

  - In contrast, the | operator always evaluates both exp1 and exp2 before returning an answer.

Logical Operators: || (logical) and | (boolean logical) inclusive OR

```
1  public class TestOR {
2      public static void main( String[] args ){
3          int  i  = 0;
4          int  j  = 10;
5          boolean test= false;
6          //demonstrate ||
7          test = (i < 10) || (j++ > 9);
8          System.out.println(i);
9          System.out.println(j);
10         System.out.println(test);
11         //demonstrate |
12         test = (i < 10) | (j++ > 9);
13         System.out.println(i);
14         System.out.println(j);
15         System.out.println(test);
16      }
17 }
```

Logical Operators: || (logical) and | (boolean logical) inclusive OR

- The output of the program is,

      0
      10
      true
      0
      11
      true

- Note, that the j++ on the line containing the || operator is not evaluated since the first expression (i<10) is already equal to true.

Logical Operators: ^ (boolean logical exclusive OR)

- Here is the truth table for ^,

x1	x2	Result
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- The result of an exclusive OR operation is TRUE, if and only if one operand is true and the other is false.

- Note that both operands must always be evaluated in order to calculate the result of an exclusive OR.

JEDI

Logical Operators: ^ (boolean logical exclusive OR)

```
1  public class TestXOR {
2     public static void main( String[] args ){
3         boolean val1 = true;
4         boolean val2 = true;
5         System.out.println(val1 ^ val2);
6         val1 = false; val2 = true;
7         System.out.println(val1 ^ val2);
8         val1 = false; val2 = false;
9         System.out.println(val1 ^ val2);
10        val1 = true; val2 = false;
11        System.out.println(val1 ^ val2);
12    }
13 }
```

Logical Operators: ^ (boolean logical exclusive OR)

- The output of the program is,
    false
    true
    false
    true

Logical Operators: ! (logical NOT)

- The logical NOT takes in one argument, wherein that argument can be an expression, variable or constant.

- Here is the truth table for !,

x1	Result
TRUE	FALSE
FALSE	TRUE

Logical Operators: ! (logical NOT)

```
1   public class TestNOT {
2      public static void main( String[] args ){
3          boolean val1 = true;
4          boolean val2 = false;
5          System.out.println(!val1);
6          System.out.println(!val2);
7      }
8   }
```

- The output of the program is,
        false
        true

Logical Operators: Conditional Operator (?:)

- The conditional operator ?:

  - is a ternary operator.

    - This means that it takes in three arguments that together form a conditional expression.

  - The structure of an expression using a conditional operator is
    exp1?exp2:exp3

    wherein,

    exp1 - is a boolean expression whose result must either be true or false

  - Result:

    If exp1 is true, exp2 is the value returned.

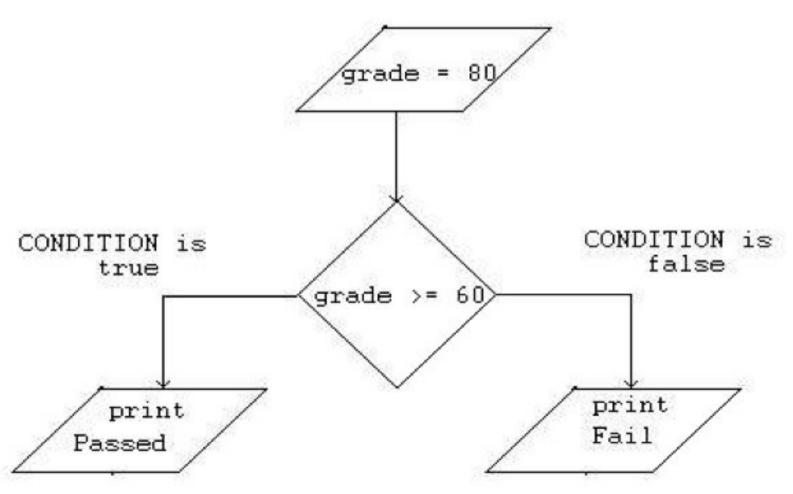   If it is false, then exp3 is returned.

Logical Operators: Conditional Operator (?:)

```
1   public class ConditionalOperator {
2      public static void main( String[] args ){
3         String  status = "";
4         int grade = 80;
5         //get status of the student
6         status = (grade >= 60)?"Passed":"Fail";
7         //print status
8         System.out.println( status );
9      }
10 }
```

- The output of this program will be,
  Passed

Logical Operators: Conditional Operator (?:)

Operator Precedence

The following is the list of Java operators from highest to lowest precedence:

```
        .          [ ]       ( )
      ++           --         !          ~
       *           /          %
       +           -
      <<          >>         >>>        <<<
       <           >          <=         >=
      ==          !=
       &           |
       ^
      &&
      ||
      ?:
       =
```

The highest precedence is on the top row and the lowest is on the bottom row.

JEDI

Operator Precedence

- Given a complicated expression,

  6%2*5+4/2+88-10

  we can re-write the expression and place some parenthesis base on operator precedence,

  ((6%2)*5)+(4/2)+88-10;

Operator Precedence: Coding Guidelines

- To avoid confusion in evaluating mathematical operations, keep your expressions simple and use parentheses.

Summary

Summary

- Java Comments (C++-Style Comments, C-Style Comments, Special Javadoc Comments)

- Java statements, blocks, identifiers, keywords

- Java Literals (integer, floating point, boolean, character, String)

- Primitive data types(boolean, char, byte, short, int, long, float, double)

JEDI

Summary

- Variables (declare, initialize, output)

- System.out.println() vs. System.out.print()

- Reference Variables vs. Primitive Variables

- Operators (Arithmetic operators, Increment and Decrement operators, Relational operators, Logical operators, Conditional Operator (?:), Operator Precedence)

JEDI

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"