Java Programming Basics

Sang Shin
JPassion.com
“Code with Passion!”

Topics

- Dissecting “Helloworld” sample application
- Java comments
- Statements and blocks
- Java identifiers
- Java literals
- Variables
- Primitive types
- Operators

Dissecting “Helloworld” Sample App

Dissecting “Helloworld” Sample App

```
public class Hello {
   /**
    * My first Java program
    */
   public static void main( String[] args ){

         //prints the string Hello world on screen
         System.out.println(“Hello world”);

   }
}
```

Class Declaration

```
public class Hello {
        /**
         * My first Java program
         */
```

- Indicates the name of the class is Hello
- The class uses an access modifier public, which indicates that our class is accessible to other classes

Start of code block

```
public class Hello {
        /**
         * My first Java program
         */
```

- The curly brace { indicates the start of a code block
- In this code, we placed the curly brace at the end of the first line, however, we can also place { in the next line. So, we could actually write our code as:

```
public class Hello
{
```

Comment

```
public class Hello {
        /**
         * My first Java program
         */
```

- 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.

main(..) method

```
public class Hello {
        /**
         * My first Java program
         */
        public static void main( String[ ] args ) {
```

- The *main(..)* method is a special method, which indicates the starting point of a Java program.
- The *main(..)* method always takes command line arguments in the form of String array

Another Comment

```
public class Hello {
        /**
         * My first Java program
         */
        public static void main( String[] args ){

            //prints the string “Hello world” on screen
```

- It is a single line Java comment

System.out.println(..)

```
public class Hello {
        /**
         * My first Java program
         */
        public static void main( String[] args ){

                //prints the string “Hello world” on screen
                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.

Ending Method and Class blocks

```
public class Hello {
        /**
         * My first Java program
         */
        public static void main( String[] args ){

                //prints the string “Hello world” on screen
                System.out.println(“Hello world”);

        }
}
```

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

Coding Requirements: File name & Class name must match

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 For example, if the name of your public class is *Hello*, you should save it in a file called *Hello.java* - otherwise, a compile error will occur

Java Comments

Java Comments

- Comments
  - > These are notes written to a code for documentation purpose
  - > Those texts are not part of the program and does not affect the flow or logic of the program in any way

- 2 Types of comments in Java

```
// Comments  - used for single line comment
/* Comments
   This is used for multi-line
   comment */
```

Statements and Blocks

Java Statements

- Each Java statement is terminated by a semicolon.

```
System.out.println(“Hello world”);
int x = 2;
```

Java Code Blocks

- One or more Java statements are bounded by opening and closing curly braces { … }
- Any amount of white space is allowed

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

Java Identifiers

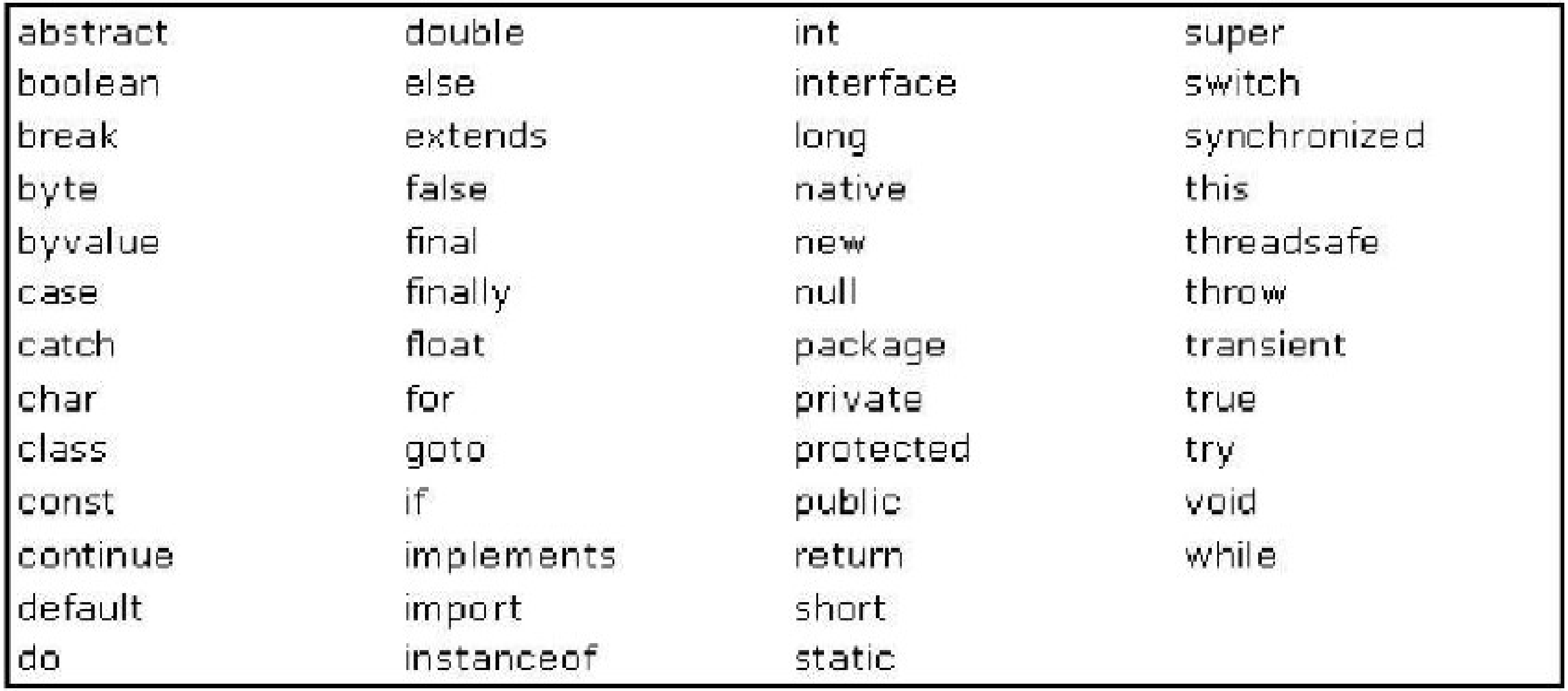
Java Identifiers

- Tokens that represent names of variables, methods, classes, etc.
  - Example identifiers are: Hello, main, System, out.
- Java identifiers are case-sensitive.
  - This means that the identifier **Hello** is not the same as **hello**
- Identifiers must begin with either a letter, an underscore “_”, or a dollar sign “$”. (They cannot begin with numbers.) 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

Java Keywords

- Keywords are predefined identifiers reserved by Java for a specific purpose.
- You cannot use these 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
boolean
break
byte
byvalue
case
catch
char
class
const
continue
default
do
double
else
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
super
switch
synchronized
this
threadsafe
throw
transient
true
try
void
while

Java Literals

Java Literals

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

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)

Java Literals: Boolean

- Boolean literals have only two values, true or false.

Java Literals: Character

- Character Literals represent Unicode characters
- Unicode character set
  - 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 *(Object type)*
  - *double grade = 0.0;   // Primitive type*
  - *Double grade2;   // Reference type (Object type)*
  - *Person x;   // Reference type (Object type)*

Declaring and Initializing Variables: Sample Program

```
public class VariableSamples {
    public static void main( String[] args ){
        // declare a data type with variable name
        // result and boolean data type
        boolean result;

        // declare a data type with variable name
        // option and char data type
        char option;
        option = 'C'; //assign 'C' to option

        // declare a data type with variable name
        // grade, double data type and initialized
        // to 0.0
        double grade = 0.0;
    }
}
```

Outputting Variable Data: Sample Program

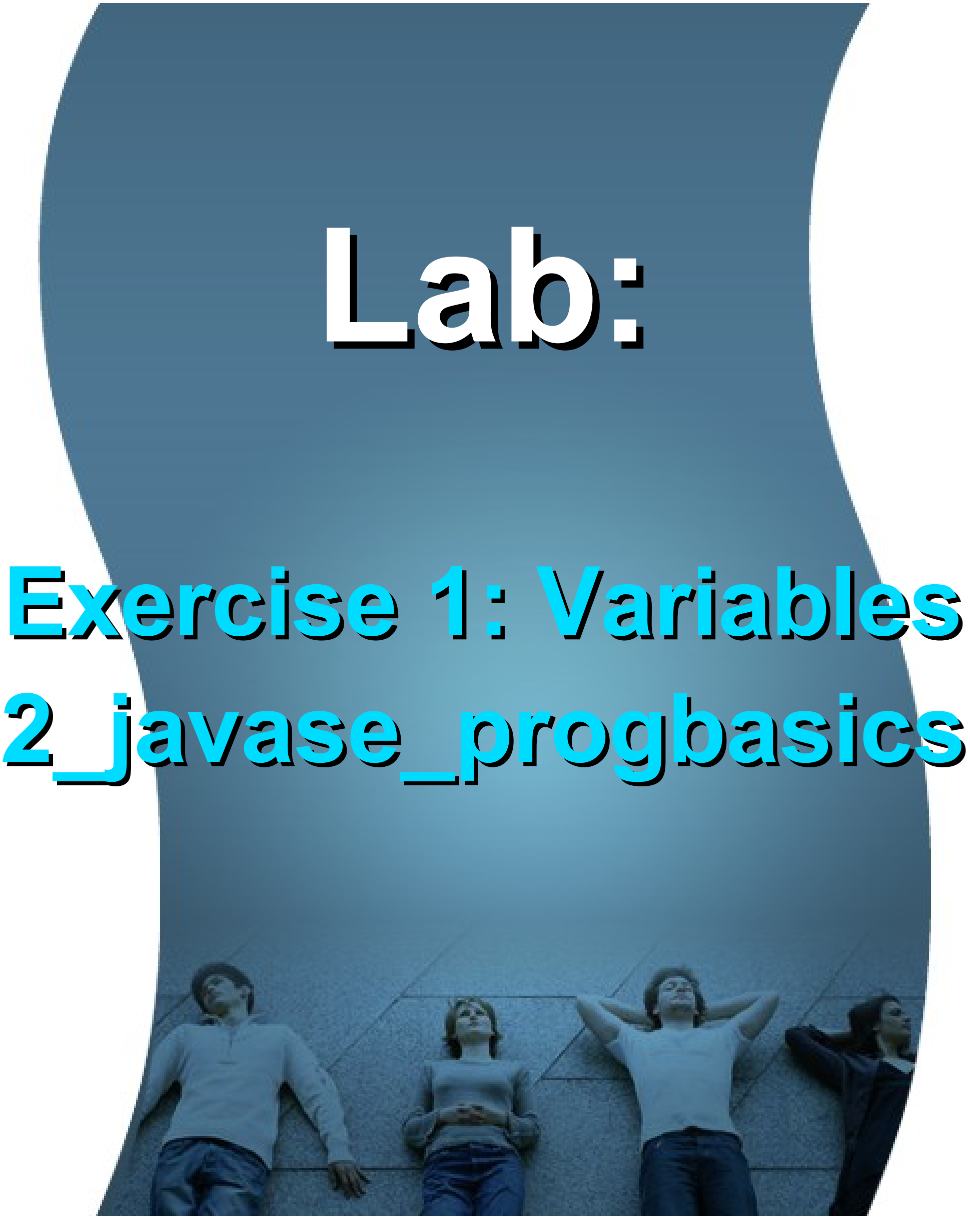
```
public class OutputVariable {
    public static void main( String[] args ){
        int value = 10;
        char x;
        x = ‘A’;

        System.out.println( value );
        System.out.println( “The value of x=“ + x );
    }
}
```

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



Lab:

Exercise 1: Variables
1002_javase_progbasics.zip

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 (non-Reference type values)
- In Java, they are provided for higher performance for compute-intensive applications
  - Other programming languages might not have primitive types

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(’ ’)

```
‘a’   //The letter a
‘\t’ //A tab
```

- To represent special characters like ' (single quotes) or " (double quotes), use the escape character \

```
'\'' //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,

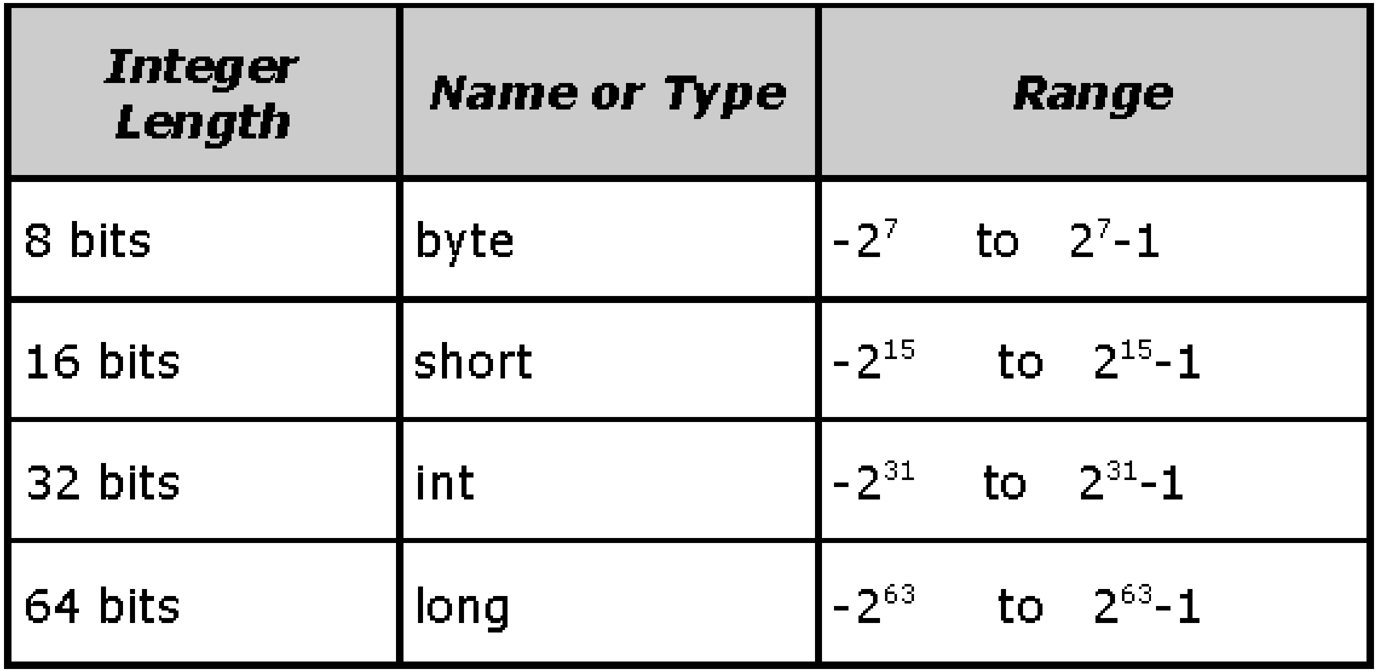
```
2   //The decimal value 2
077 //The leading 0 indicates an octal value
0xBACC //The leading 0x indicates a hex value
```

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

```
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: Floating Point – float and double

- 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)
```

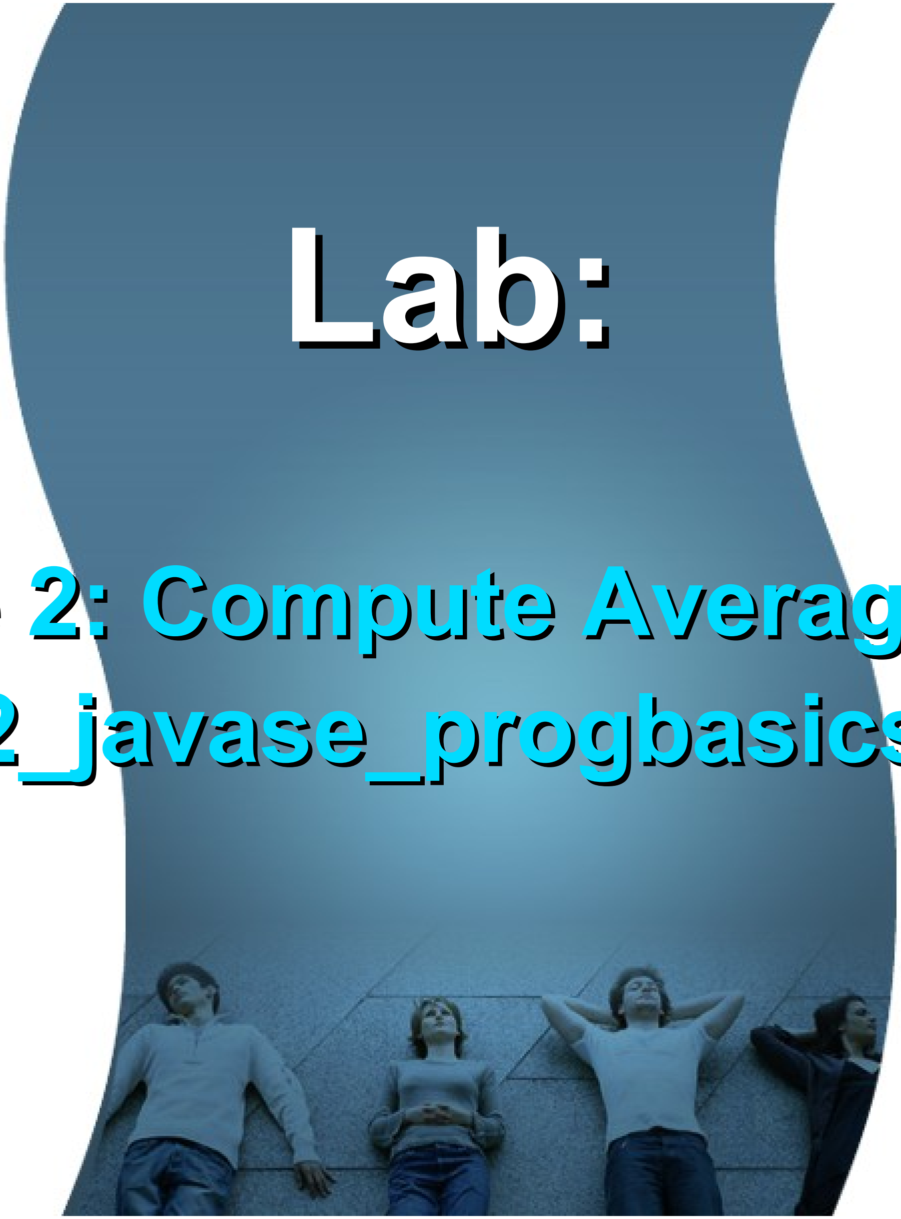
- Examples are,

```
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)



Lab:

Exercise 2: Compute Average & Sum
1002_javase_progbasics.zip

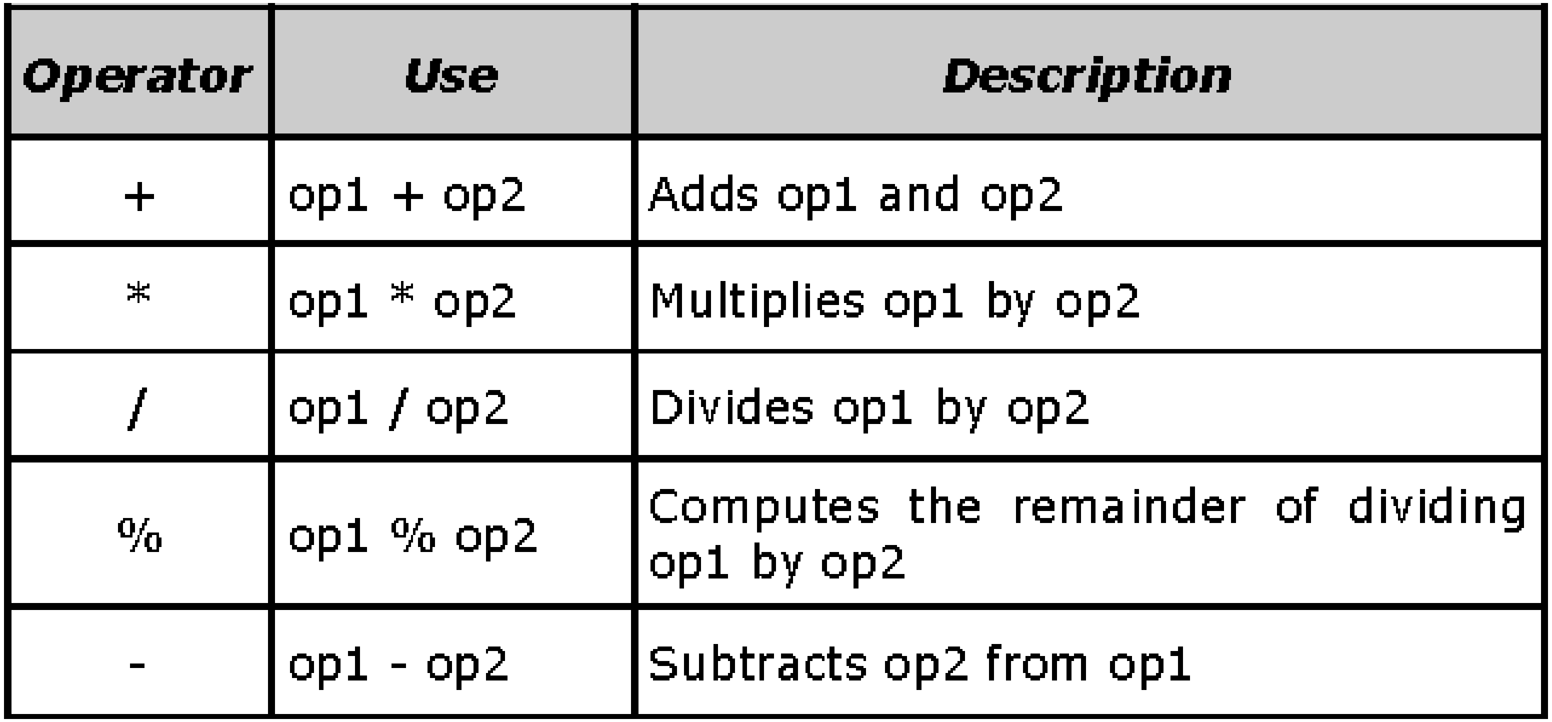
Operators

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 a single statement

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

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++;      // same as above
```

Increment and Decrement Operators

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

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

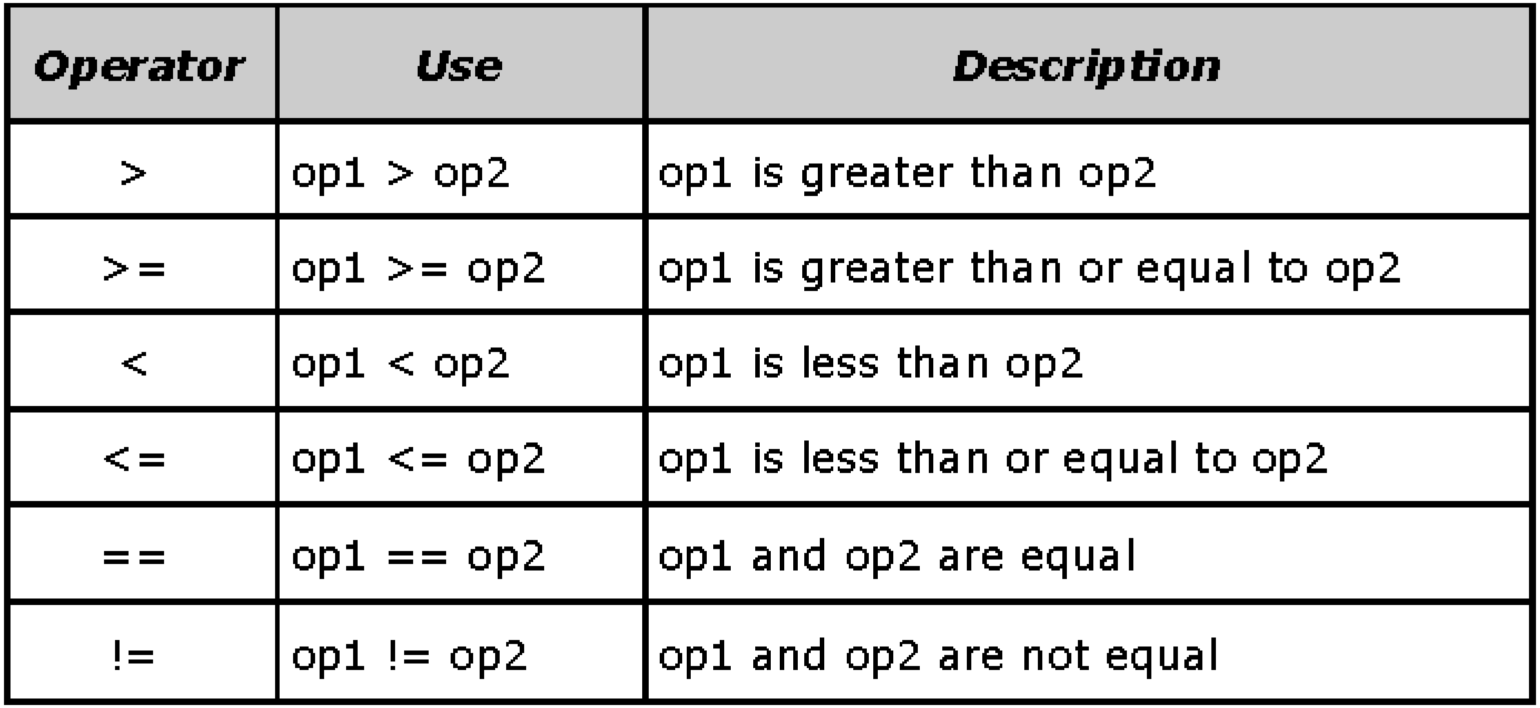
Increment and Decrement Operators

- 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.

```
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.
- Output of evaluation is boolean value: true or false.



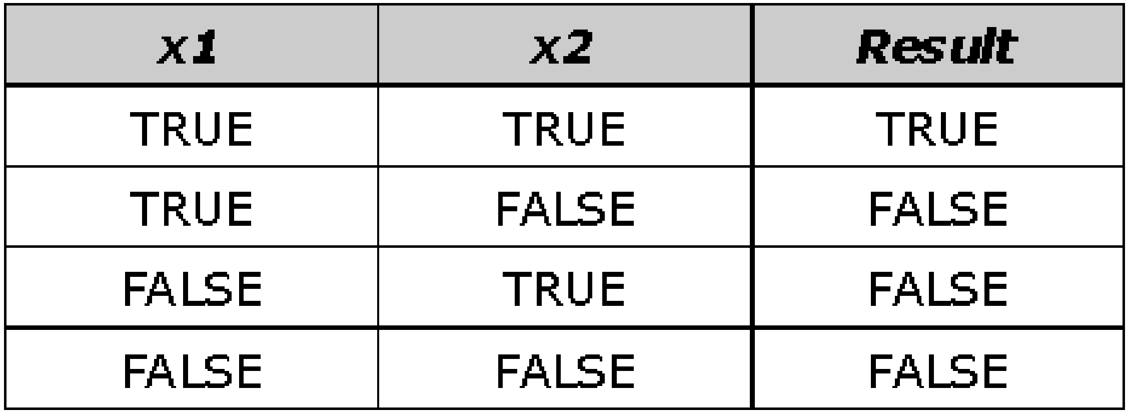
Operator	*Use*	*Description*
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

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: &&

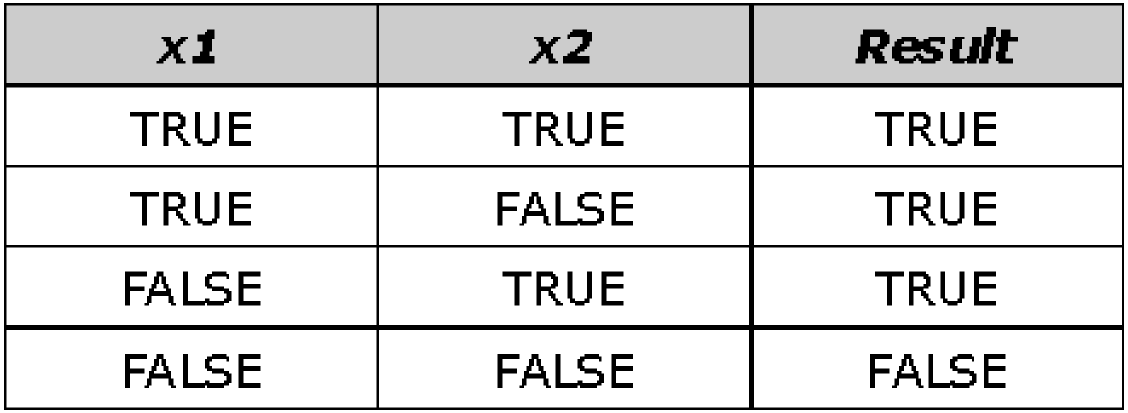
- Here is the truth table for && and &,



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

Logical Operators: ||

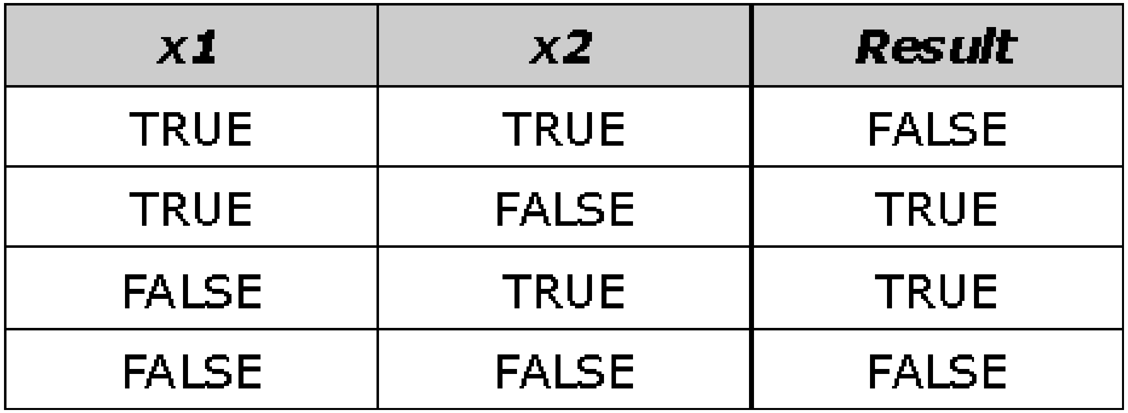
- Here is the truth table for ||



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

Logical Operators: ^ (boolean logical exclusive OR)

- Here is the truth table for ^,

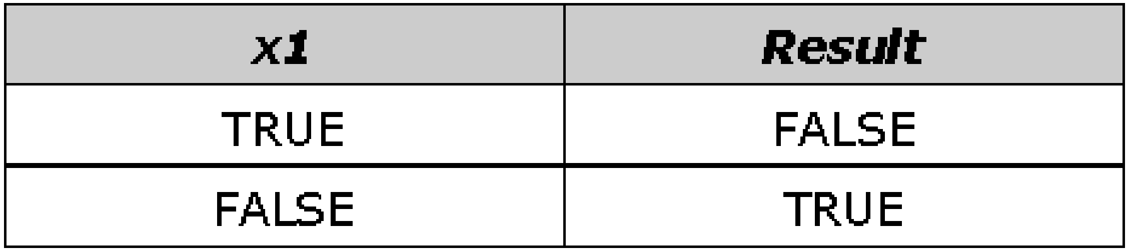


x1	*x2*	*Result*
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- 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.

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: 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:

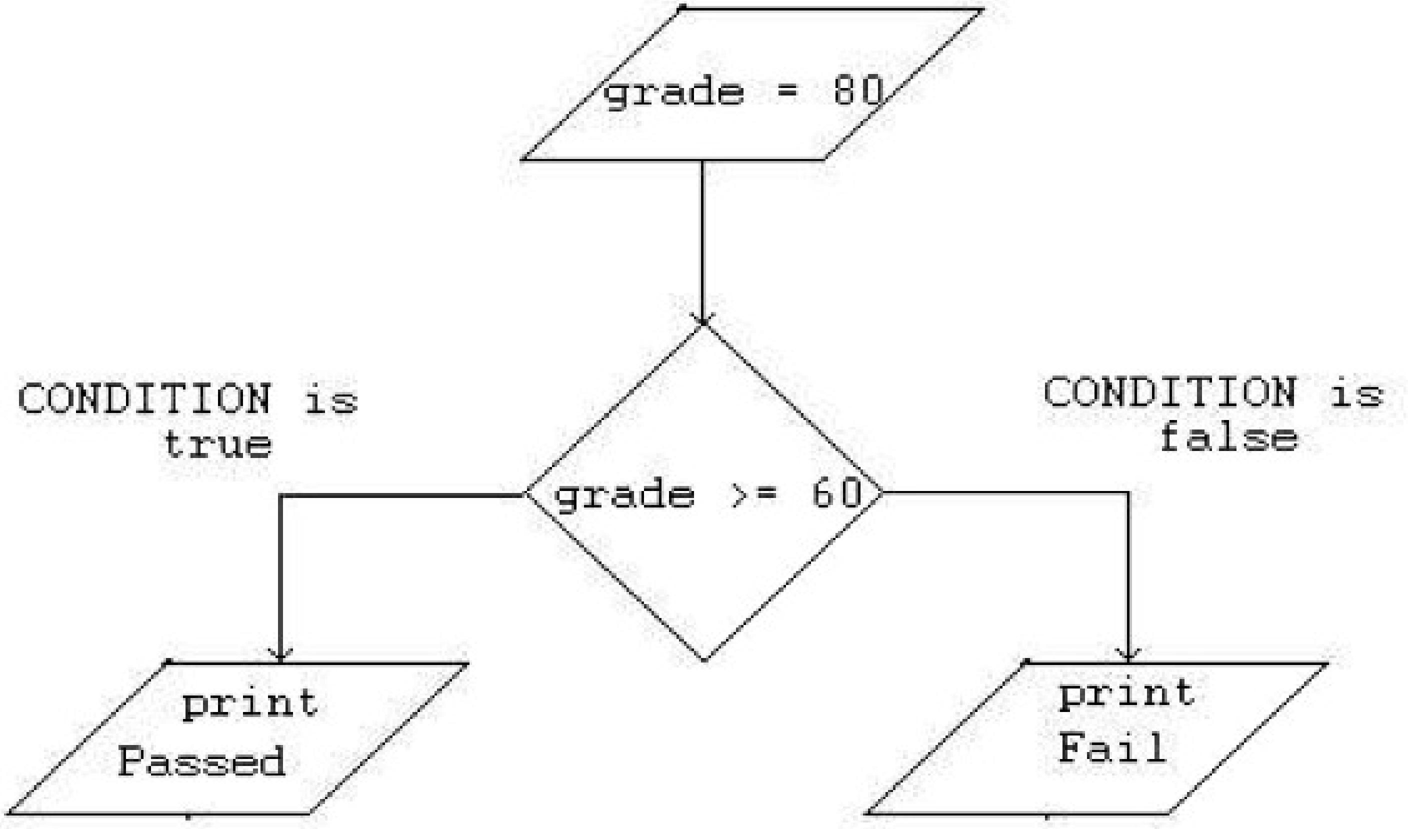
    If exp1 is true, exp2 is the value returned.

    If it is false, then exp3 is returned.

Logical Operators: Conditional Operator (?:)

```
public class ConditionalOperator {
        public static void main( String[] args ){
                String  status = "";
                int grade = 80;
                //get status of the student
                status = (grade >= 60)?"Passed":"Fail";
                //print status
                System.out.println( status );
        }
}
```

Logical Operators: Conditional Operator (?:)



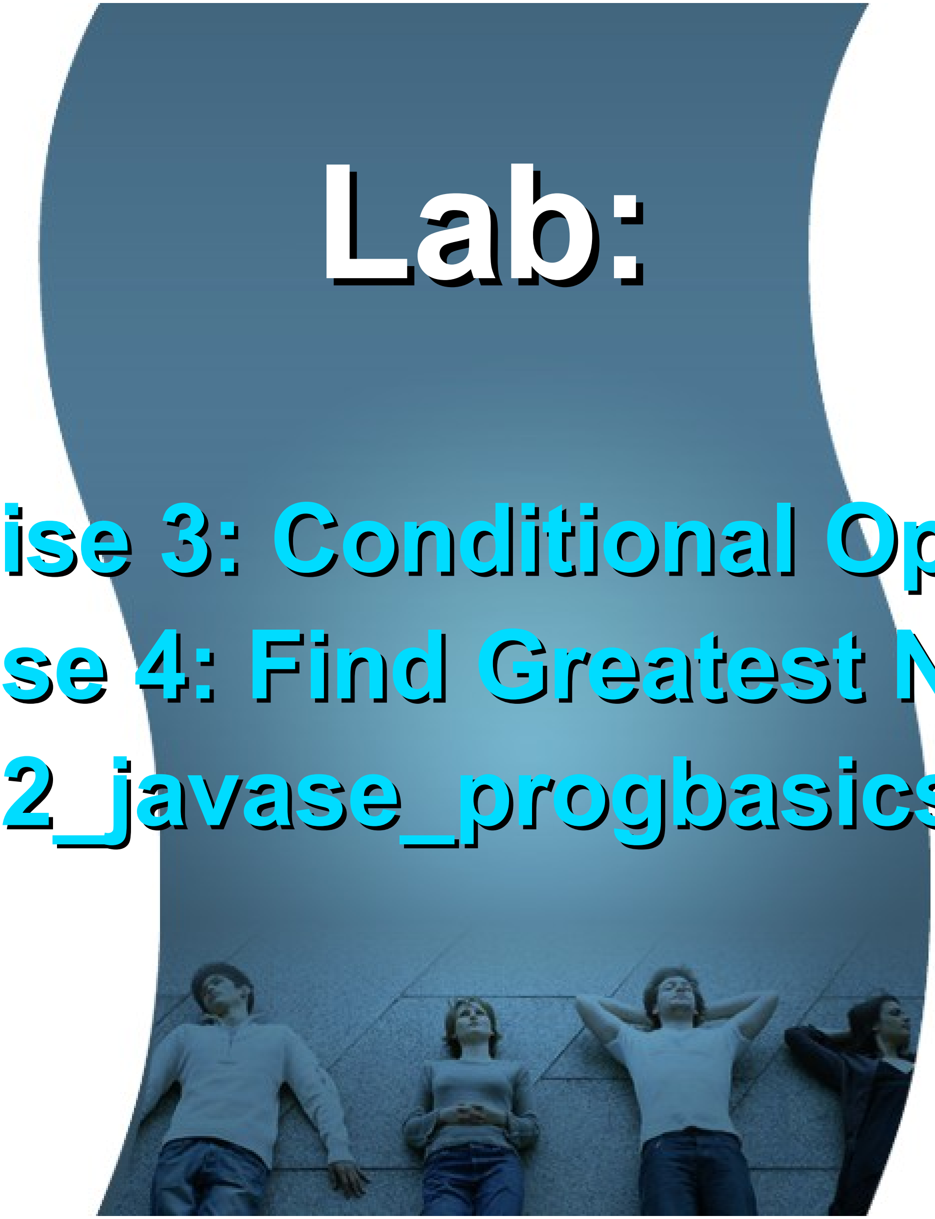
Operator Precedence

- 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;



Lab:

Exercise 3: Conditional Operator

Exercise 4: Find Greatest Number

1002_javase_progbasics.zip

Code with Passion!
JPassion.com