Classes

Introduction to Programming I

Lecture 1

Introduction and Overview of Module

Classes as the Foundation of All Computation

Agenda

- Introduction to your Instructors
- Introduction to the Module
- Week 1 Lecture Coverage
  - Concept of Objects
  - Introducing Classes
  - About Java
  - Introduction to BlueJ
  - Java Character Set
  - Tokens
  - Identifiers

Agenda

- Week 1 Lecture Coverage (Contd..)
  - Data Types
  - Variables
  - Constants
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Assignment Operator
  - Expressions
  - Java Statements

Module Leader

- Abhinav Dahal (M.Sc., Linnaeus, Sweden)
  - Senior Lecturer, Islington College, Nepal
  - Previous Appointments
    - PHP developer, WorldLink Technologies, NP
    - Chief developer for www.sharesansar.com
  - Specialization
    - Java (Core & Advance)
    - Java Android
    - PHP

Introduction to the Module

- Overview of Module
  - What can you expect?
- Learning Objectives
  - How will you benefit?
- Learning Strategy
  - How you will learn?
- Assessment Overview
  - How you will be tested?

Overview of Module

- This is a beginner’s module aimed at introducing Object Oriented Programming concepts to students who have had no prior experience of programming.
- The module will teach students to write, analyze and correct program codes using graphical interface (BlueJ).
- At the end of the course, students will be able to write small programs giving them knowledge and confidence for computing major.

Learning Objectives

Servlets & JSP

Android

Advance Java Programming

Core Java Programming

Learning Strategy

- Taught over 1 semester
- Total 12 weeks of class (12 weeks each semester)
- Each week consists of 1 Lecture (2 Hours) and 1 lab (2 Hours)
  - Lecture: Learn how to acquire programming skills.
  - Lab: Review and practice programming skills through in-class practical exercises to actually acquire them.

Learning Strategy

Attendance for all classes is important!

- <u>Programming cannot be learnt by memorizing</u>. It is learnt through constant practice.
- We want to provide you the environment to practice and improve effectively.

Assessment Details

Semester		
Test 1 (20%)	**Test 2 (30%)**	**Coursework (50%)**
Classes	Decision Making Statements	Individual
Constructors	Iterations	Week 14
Methods	Using Library Classes	
Classes as User Defined Types	Encapsulation	
Week 5	Arrays	
	Week 11	

Assessment Details

- **Module Grading Standards in the UK**

Range of Marks	Grade	Remarks
70 - 100	**A**	Excellent: outstanding performance with only minor errors
60 - 69	**B**	Very Good: above the average standard but with some errors
50 - 59	**C**	Good: generally sound work with a number of notable errors
43 - 49	**D**	Satisfactory: fair but with significant shortcomings
40 - 42	**E**	Sufficient: performance meets the minimum criteria
0 - 39	**F**	Fail: performance does not meet the minimum criteria and considerable further work is required

Assessment Details

More Information about the assessments will be provided in subsequent weeks.

Any Questions?

Let’s get started with Lecture 1

An Introduction to **Classes**

Concept Of Objects

- The world we live in is composed of objects.
- Everything that we see is an example of object.
- We, ourselves, are examples of objects.
- Chair, car, pets that you have, house you live in..........

Introducing Classes

- Classes are blueprints of objects.
- An object is a “real” or “realistic” entity, a class is representation of only an abstraction.
- An abstraction is a named collection of attributes(properties) and behaviours(methods) relevant to modelling a given entity for some particular purpose.

Introducing Classes

- Abstraction is always relative to the purpose or user.
- Example, if we talk of a student, we can talk of anything that belongs to her in the real world, like her name, brothers, sisters, parents' profession, locality she lives in, marks obtained by her, her roll number in the class, her medical history, her hobbies, her awards etc.

Introducing Classes

- But when we talk of a student result tracking system, the abstraction for it would be her roll number, name, marks obtained etc.
- For extra-curricular activities, the abstraction would be her roll number, talents and awards.

Introducing Classes

- *A class is a named software representation for an abstraction.*
- *An object is a distinct instance of a given class that is structurally identical to all other instances of that class.*
- Software code in Object Oriented Programming is written to define classes, instantiate objects and manipulate these objects.

About Java

- History of Java
- Byte code
- Java virtual machine
- Characteristics of Java
  - Write once run anywhere (platform independent)
  - Object oriented language
  - Open product
  - ....

About Java

- Demonstrating a simple java program.

```
/* program HelloWorld*/
class HelloWorld{
        public static void main(String []args){
            System.out.println(“Hello World!!”);
        }
}
```

Types of Java Programs

1. Internet Applets
2. Stand-alone Applications

BlueJ - A Quick Introduction - Demo

- What is BlueJ?
- Starting BlueJ
- Writing programs on BlueJ Environment
  - Creating a BlueJ Project
  - Adding a New Class to the Project
  - Editing a class code

BlueJ - A Quick Introduction - Demo

- Writing programs on BlueJ Environment (Contd..)
  - Compiling the source code in BlueJ
  - Creating objects in BlueJ
  - Executing an object's methods
  - Saving a class
- The main() method

Any Questions?

Java Character Set

- Character set is a set of valid characters that a language can recognize.
- A character represents any letter, digit or any other sign.
- Java uses the **Unicode** character set.
- Unicode is a two-byte character code set that has characters representing almost all characters in almost all human alphabets and writing systems around the world including English, Arabic, Chinese and many more.

Tokens

- Individual words and punctuation marks
- Every unit that makes a sentence.

Keywords

- Special words that convey a special meaning to the language compiler.
- Reserved words for special purpose and must not be used as normal identifier names.

- Examples of keywords:

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	try

……… *See the complete list of Java keywords*

Identifiers

- The name given to variables, objects, classes, functions, arrays etc.
- Identifier forming rules of Java:
  - Identifiers can have alphabets, digits and underscore and dollar sign.
  - They must not be a keyword or boolean literal or null literal
  - They must not begin with a digit.
  - They can be of any length.
  - Java is case sensitive.

Identifiers

- Some valid identifiers:

  myFile   first_name   address1_4   ...

- Some invalid identifiers:

  first-name try   address.12

Identifier Naming Conventions

1. The names of public methods and instance variables should begin with a *lower* case letter. E.g. value, name etc.
2. For names having multiple words, second and subsequent words beginning character is made capital. E.g. firstName, dateOfBirth etc.

Identifier Naming Conventions

3. Private and local variables should use lower case letters. E.g. height, width etc.
4. Constants should be named using all capital letters and underscores. E.g. AVG_VALUE, PI etc.
5. Class names and interface names begin with an uppercase letter. E.g. Employee, Vehicle etc.

Data Types

- Data can be of many types, e.g. character, integers, real, string etc.
- Any thing enclosed in single quotes represents character data.
- Numbers without fractions represent integer data.

Data Types

- Numbers with fractions represent real data.
- Anything enclosed in double quotes represents a string.
- Variables that take in either true or false value represent a boolean data.

Primitive Data Types

- The “basic“ data types offered by Java.
- Numeric integral types
  - byte
  - short
  - int
  - long

Primitive Data Types

- Fractional numeric types (floating-point datatypes)
  - float
  - double
- Character type (char)
- Boolean type (boolean)

Variables

- Named storage locations which holds a data value of a particular data type whose value can be manipulated during program run.
- The following statement declares a variable *x* of the data type *int*

```
int  x ;
```

- Some more examples:

```
double salary, wage;
int month, day, year;
long distance, area;
```

Initialization of Variables

- The first value specified in the declaration/definition of a variable. E.g.

```
int value = 20;
double price=210.5, discount=0.05;
float result=34.12;
long val=20L;
boolean result = true;
```

Constants

- Often in a program you want to give a name to a constant value.
- For example, you might have a fixed *tax rate* of 0.05 *for goods* and a *tax rate* of 0.03 for services.
- These are constants because their value is not going to change when the program is executed.
- It is convenient to give these constants a name.

```
final double TAXRATE = 0.03;
```

- The keyword **final** makes a variable as constant.

ANY QUESTIONS
?
www.rekruitin.com
ReKruiTIn.com

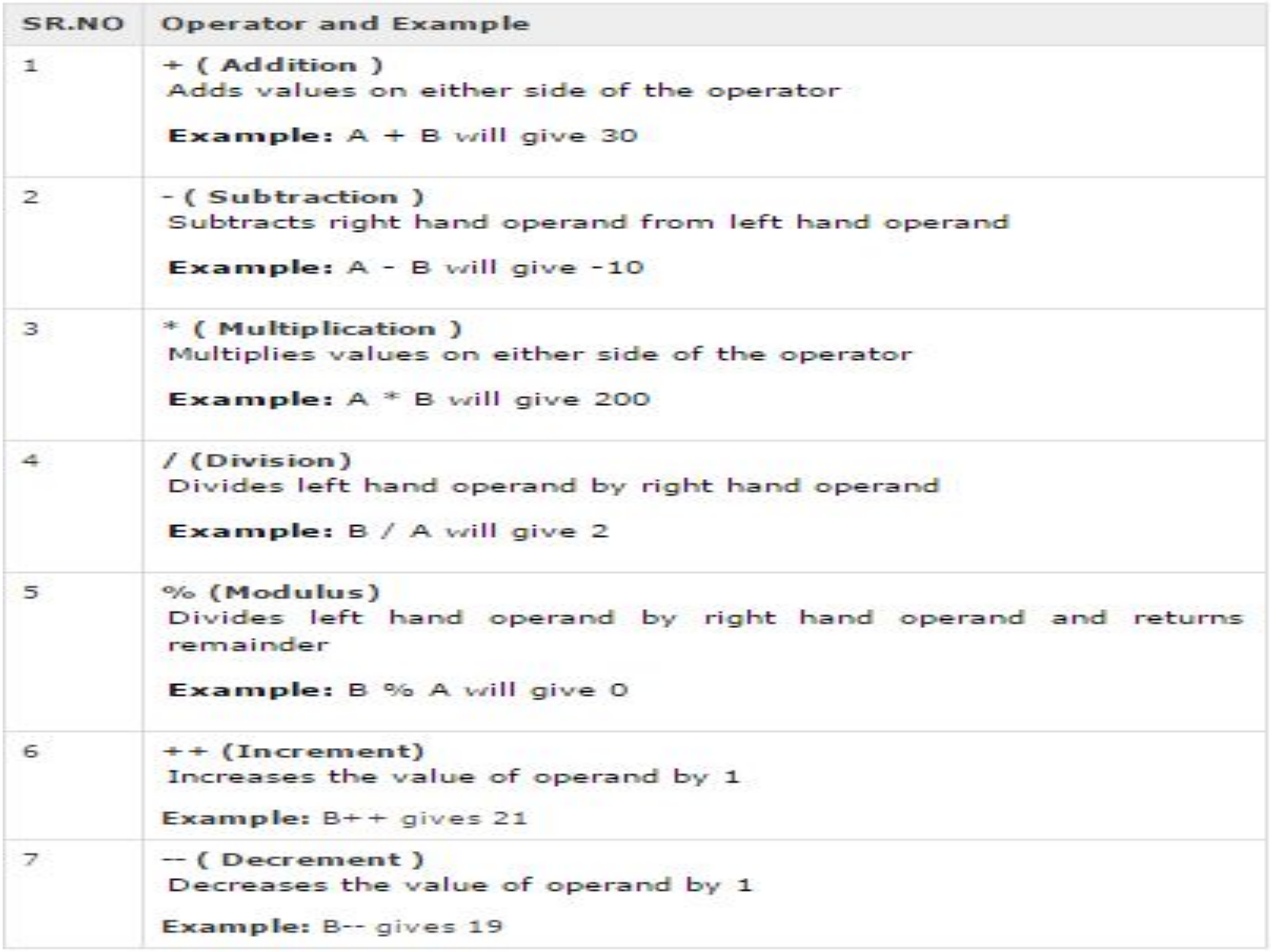
Operators in Java

- Java’s rich set of operators comprises of arithmetic, relational, logical, bitwise , assignment and certain other types of operators.

Arithmetic Operators

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

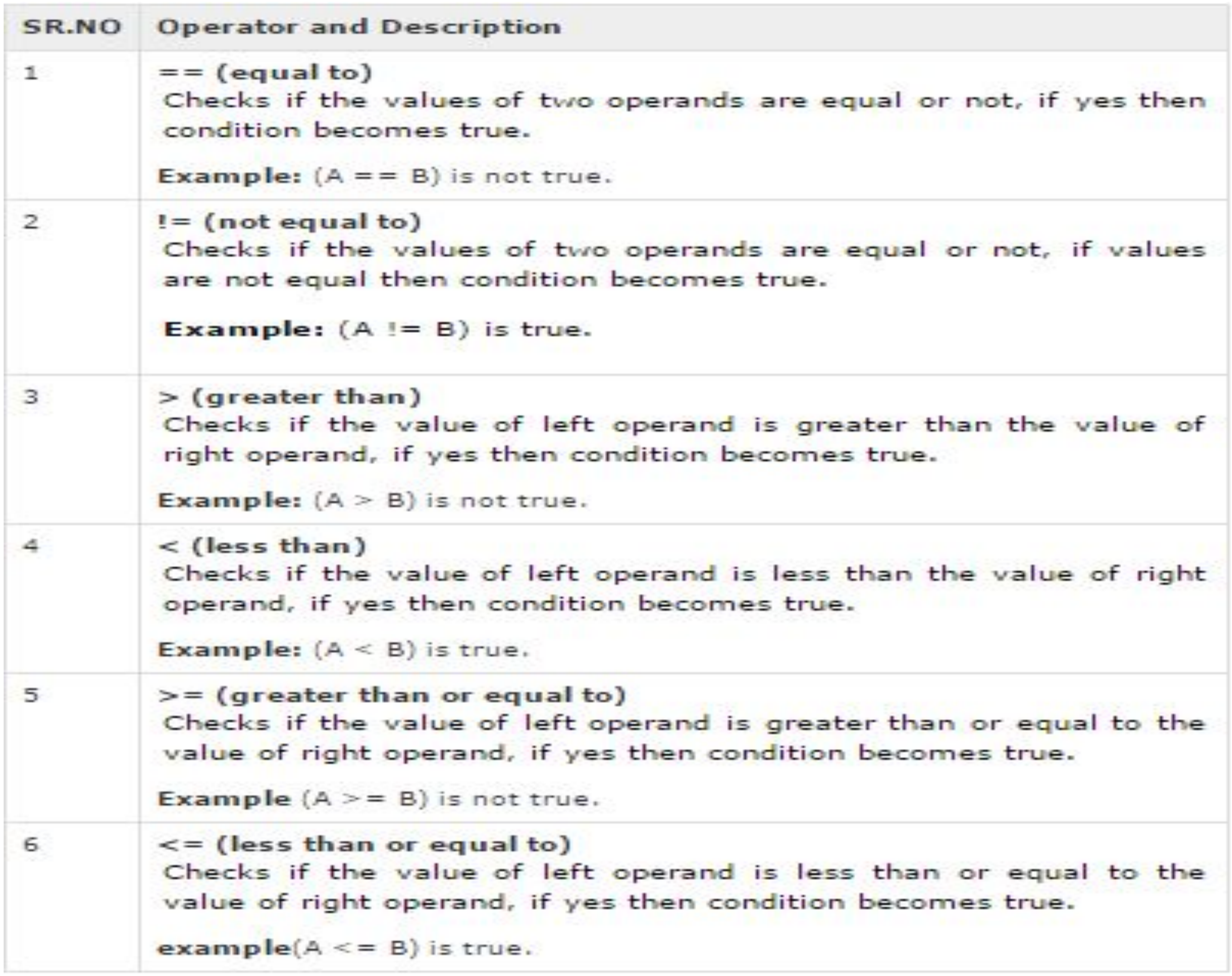
Arithmetic Operators



SR.NO	Operator and Example
1	**+ (Addition)** Adds values on either side of the operator **Example:** A + B will give 30
2	**- (Subtraction)** Subtracts right hand operand from left hand operand **Example:** A - B will give -10
3	*** (Multiplication)** Multiplies values on either side of the operator **Example:** A * B will give 200
4	**/ (Division)** Divides left hand operand by right hand operand **Example:** B / A will give 2
5	**% (Modulus)** Divides left hand operand by right hand operand and returns remainder **Example:** B % A will give 0
6	**++ (Increment)** Increases the value of operand by 1 **Example:** B++ gives 21
7	**-- (Decrement)** Decreases the value of operand by 1 **Example:** B-- gives 19

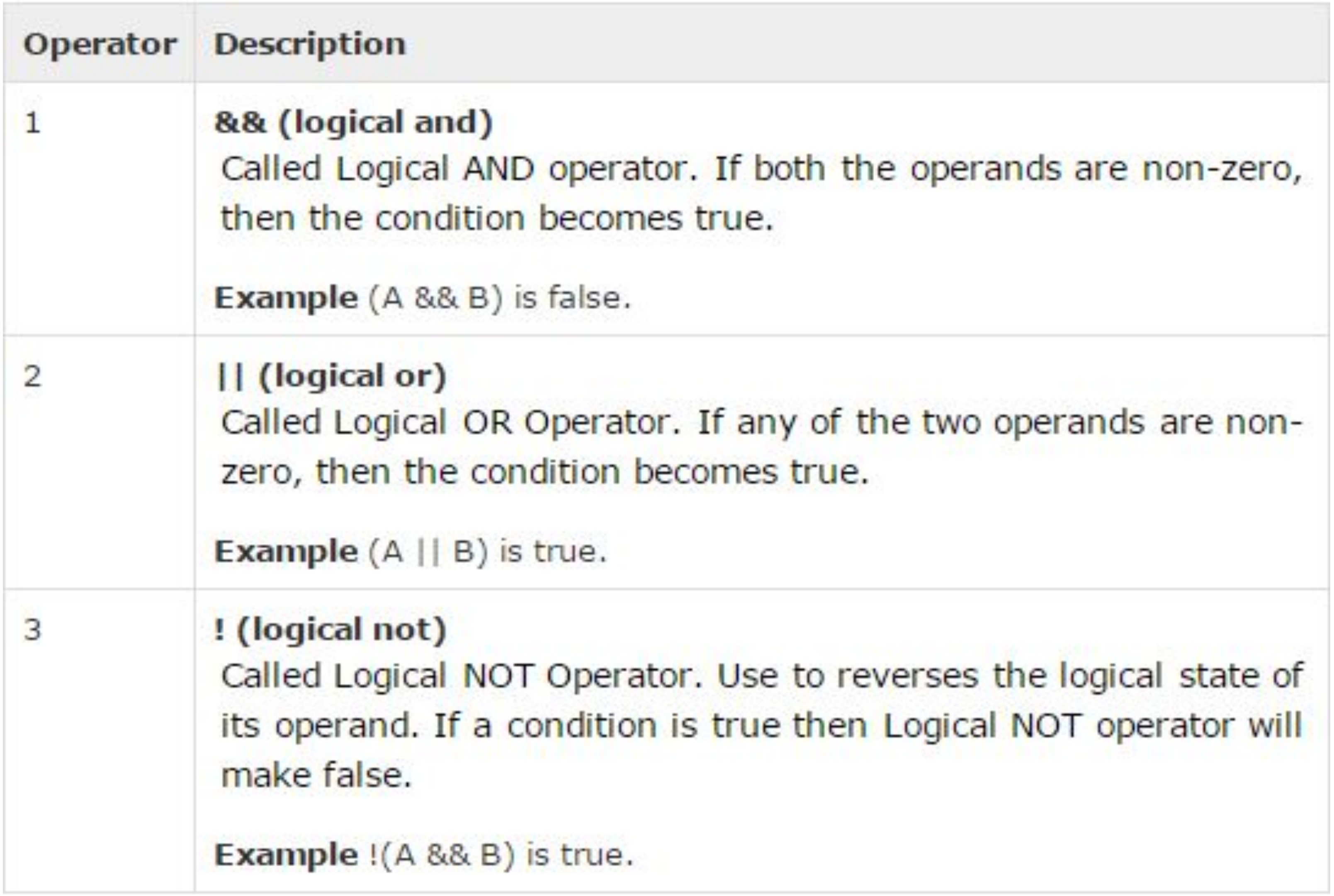
Relational Operators

- Assume variable A holds 10 and variable B holds 20, then:



SR.NO	Operator and Description
1	**== (equal to)** Checks if the values of two operands are equal or not, if yes then condition becomes true. **Example:** (A == B) is not true.
2	**!= (not equal to)** Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. **Example:** (A != B) is true.
3	**> (greater than)** Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. **Example:** (A > B) is not true.
4	**< (less than)** Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. **Example:** (A < B) is true.
5	**>= (greater than or equal to)** Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. **Example** (A >= B) is not true.
6	**<= (less than or equal to)** Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. **example**(A <= B) is true.

The Logical Operator



Operator	Description
1	**&& (logical and)** Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. **Example** (A && B) is false.
2	**|| (logical or)** Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. **Example** (A || B) is true.
3	**! (logical not)** Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. **Example** !(A && B) is true.

Assignment Operator

- Like other programming languages, Java offers an assignment operator =, to assign one value to another. E.g.

```
byte x, y, z;
x = 9;
y = 7;
z = x + y;
z = z * 2;
```

Expressions

- An *expression* is composed of one or more *operations.*
- The objects of the operation(s) are referred to as *operands.*
- The operations are represented by *operators.*
- Therefore, operators, constants, and variables are the constituents of expressions.

Expressions

- The expressions in Java can be of any type:
  - *Arithmetic expressions*
  - *Relational /logical expressions*
  - *Compound expressions*

Java Statements

- Statements are roughly equivalent to sentences in natural languages.
- A *statement* forms a complete unit of execution.
- The following type of expressions can be made into a statement by terminating the expression with a semicolon(;):
  - Assignment expressions
  - Any use of ++ or --
  - Method calls
  - Object creation expressions

Significance of Classes

- The basic unit of object-orientation in Java is **the class**.
- The class is often described as a *blueprint for an object*.
- It allows a programmer to define all of the properties and methods that internally define an object, all of the API (Application Programming Interface) methods that externally define an object, and all of the syntax necessary for handling *encapsulation*, *inheritance* and *polymorphism*.

Significance of Classes

- In Java, the class forms the basis of all computation.
- Anything that has to exist as a part of Java Program has to exist as a part of a class, whether that is a variable or a function or any other code-fragment.
- All Java programs consist of **objects**(data and behavior) that "interact" with each other by calling **methods**.
- All data is stored within objects which are instance of a class.

Objects As Instances of Class

- A class defines only a blueprint and its concrete version comes into effect only through **objects** that implement the functionality as defined by class.
- Example, define a class called **City**. The objects created from this class will have two variables: *name* and *population;* and they will be able to represent cities. The object of **City** class will also have a method namely **display()**.

Objects As Instances of Class

- An object of a class is typically named by a variable of the class type. For example, the program **CitypTrail** declares the two variables **metro1** and **metro2** to be of type **City** as follows:

```
City metro1, metro2;
```

Objects As Instances of Class

- This gives us variables of the class **City**, but so far there are no objects of the class.
- Objects are class values that are named by the variables.
- To obtain an object you must use the *new* operator to create a "new" object.

Objects As Instances of Class

- For example, the following creates an object of the class **City** and names it with the variable **metro1:**

```
metro1 = new City();
```

- This kind of statements is called *constructor* and we will discuss this kind of statement in more detail later on.

```
class_variable = new Class_Name() ;
```

is how a new object of the specified class is created and associated with the class variables.

Any Questions?

Summary: Week 1 Lecture

- Concept of Objects and Classes
- Introduction to BlueJ
- Concept of character set, tokens, identifiers
- Data types
- Variables
- Constants
- Operators
- Expressions
- Statements

What to Expect: Week 1 Lab

- Discussion about the topics covered in the lecture.
- Provide practical questions based on inheritance so it is vital that you go through the lecture slide!!.

Thank you