Third Year Diploma Courses in Computer Science & Engineering, Computer Engineering, Computer Technology & Information Technology Branch.

Java Programming

As per MSBTE 'I' Scheme Syllabus
JPR-22412

Unit- I
Basic Syntactical constructs in java

Total Marks- 10

Prof. Gunwant V. Mankar
B.E(IT), M.Tech(CSE), AMIE, MIAEng, MSCI
Director
(ConnectSoft Infotech, Warora, IN)
e-mail:- info@gunwantmankar.com
website- www.gunwantmankar.com

Unit-I Basic Syntactical constructs in java

Introduction to java

- James Gosling of sun micro-system created java in 1991.
- The original name of java is OAK. OAK is a name of tree, but later the language was change in the JAVA.
- Belongs to Sun- stand ford university network
- In 2010 jan 27 oracle corporation acquired sun microsystem.

Java Application/packages(edition)(API)(in 1995,completed java with three edition)

- By using java we can develop the desktop application, enterprise edition application, & device application.
- To develop desktop application JSE is used(java platform standard edition)
- To develop enterprise application JEE is used(java platform enterprise edition)
- To develop device application JME is used.(java platform micro-edition).
- Old version are J2SE, J2ME, J2EE &
- new version are JSE, JME, JEE
- Core java belongs to JSE.

1.1. Java Features and Java Programming Environment

1.1.1. Java Features

There are many features of JAVA. They are also known as JAVA buzzwords.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

1.Simple

- JAVA is Easy to write and more readable and eye catching
- JAVA has a concise, cohesive set of features that makes it easy to learn and use.
- Most of the concepts are drawn from C++ thus making JAVA learning simpler

2.Object-Oriented

- JAVA programming is object-oriented programming language
- Like C++, JAVA provides most of the object oriented features
- JAVA is pure OOPS Language. (while C++ is semi object oriented).

3.Platform independent

- A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. JAVA provides software based platform.
- The JAVA platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.

It has two components.

- Runtime Environment
- API(Application Programming Interface)

JAVA code can be run on multiple platforms E.G.–Windows,Linux,Sun Solaris,Mac/OS etc. JAVA code is compiled by the compiler and converted into bytecode.

This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

4.Secured

JAVA is secured because

- No explicit pointer
- Programs run inside virtual machine sandbox.
- ***Class loader*** – adds security by separating the package for the classes of the local file system from those that are imported from network sources
- ***Byte code Verifier*** – checks the code fragments for illegal code that can violate access right to objects
- ***Security Manager*** – determines what resources a class can access such as reading and writing to the local disk These security are provided by JAVA language. Some security can also be provided by application developer through SSL, JAAS, cryptography etc.

5. Robust

- JAVA makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which JAVA improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

6. Architecture neutral

• JAVA is not tied to a specific machine or operating system architecture

• Machine Independent i.e JAVA is independent of hardware

• Compiler generates byte codes, which have nothing to do with a particular computer architecture, hence a JAVA program is easy to interpret on any machine

7. Portable

• JAVA programs can execute in any environment for which there is a JAVA runtime system(JVM)

• JAVA programs can be run on any platform (Linux,Window,Mac)

• JAVA programs can be transferred over world wide web (e.g applets)

8. Dynamic

• JAVA programs carry with them substantial amounts of run-time type information that is used to verify and resolve accesses to objects at run time

9 Interpreted

• JAVA supports cross-platform code through the use of JAVA bytecode

• Byte code can be interpreted on any platform by JVM

10. High Performance

JAVA is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, JAVA enables high performance with the use of just-in-time compiler.

11. Multithreaded

A thread is like a separate program, executing concurrently. One can write JAVA programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

12. Distributed

• JAVA was designed with the distributed environment

• JAVA can be transmit, run over internet

1.1.2. Java Programming Environment

- JAVA development environment includes a number of development tools, classes and methods.
- It is a part of the system known as JAVA Development kit (JDK).
- The classes and methods are part of the JAVA standard library known as JSL, and it is known as the application Programming Interface(API).
- The java runtime environment is supported by Java Virtual Machine(JVM).

1. JAVA Development Kit (JDK)

The JDK kit is a collection of tools which are used for developing designing, debugging , executing and running JAVA programs. JDK kit includes

• **appletviewer (for viewing JAVA applets) –**

It enables to run JAVA applets (without actually using a JAVA-compatible browser)

• **javac (JAVA compiler)**–

JAVA compiler is used to compile JAVA files. JAVA compiler components of JDK is accessed using "javac" command.

E.G. – C:\>JAVAc filename.JAVA

• **java (java interpreter)**

JAVA interpreter is used to interpret the JAVA files that are compiled by JAVA compiler. JAVA interpreter components of JDK is accessed using "JAVA" command.

E.G. – C:\>JAVA filename

• **javap (JAVA disassemble) –**

JAVA disassembler, which enables to convert bytecode files into a program description.

• **javah (for C header files)** – produces header files for use with native methods

•**Javadoc (for creating HTML documents)** – creates html-format documentation from JAVA source code files

• **jdb (JAVA debugger)** – JAVA debugger, which helps us to find errors in our program

The following Fig. describes the tools used in JAVA environment.

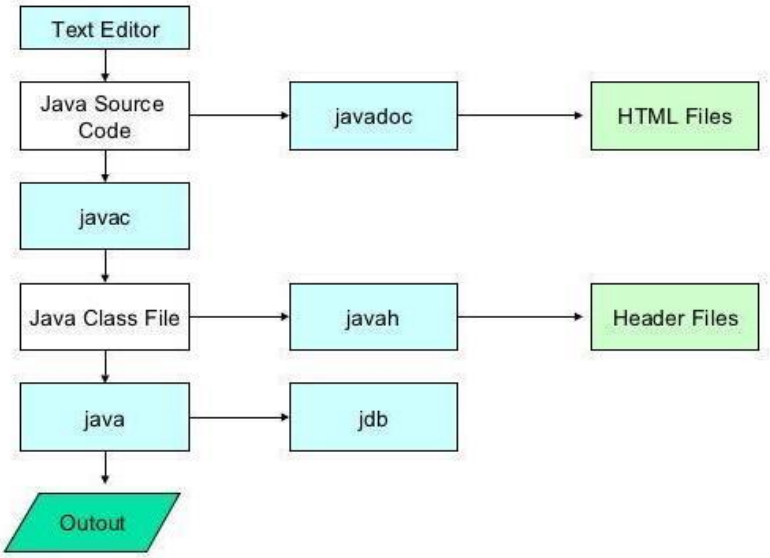


Fig. Execution process of java application program

2. JAVA Virtual Machine(JVM) & Byte Code

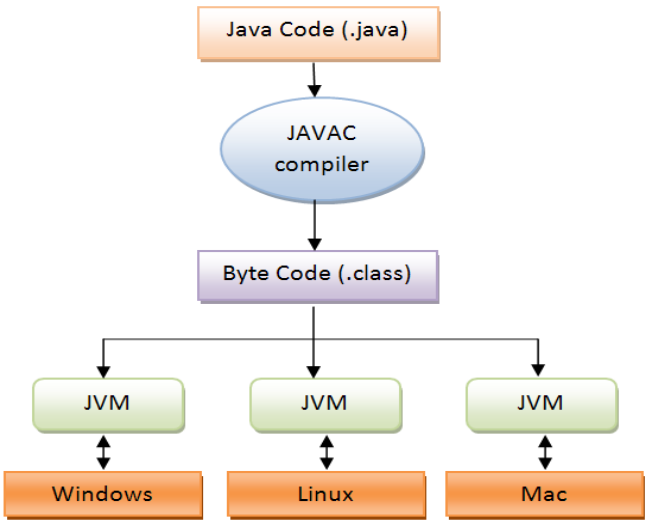
- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

ByteCode

Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.



When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so. After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on. Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.

1.1.3. Simple Java Programming

How to Write and compile the Simple JAVA Program?

For executing any JAVA program, one need to

- Install the JDK.
- Set path of the jdk/bin directory
- Compile and run the JAVA program

After installing JDK, set the path using following command,

Right click on My computer -> properties -> environmental variable -> user variable

New- variable name- 'Path', variable value – "C:\Program Files\JAVA\jdk1.6.0\bin"

To create a simple JAVA program, one need to create a class that contains main method. Open notepad Write this program.

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

Save this file as Simple.java

To compile – javac Filename.java // create Filename.class file

To execute – java Classname // Display output

Open Command Prompt to Compile & Execute JAVA program

C:\Program Files\JAVA\jdk1.6.0\bin> javac Simple.JAVA // Simple.class Created

C:\Program Files\JAVA\jdk1.6.0\bin> java Simple

Hello JAVA

Understanding first JAVA program

Meaning of class, public, static, void, main, String[ ], System.out.println().

- **class** keyword is used to declare a class in JAVA
- **public** keyword is an access modifier which represents visibility, it means it is visible to all
- **static** is a keyword, if one declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory
- **void** is the return type of the method, it means it doesn't return any value
- **main** represents startup of the program
- **String[] args** is used for command line argument. This is explained later
- **System.out.println()** is used print statement. The internal working of System.out.println statement is explained later

Different codes to write the main method

- public static void main(String[ ] args)
- public static void main(String [ ]args)
- public static void main(String args[ ])
- public static void main(String... args)

1.2. Defining a class, creating object and accessing class members

1.2.1. Class

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in JAVA can contain

- data member
- method
- constructor
- block
- class and interface

Syntax to declare a class

class <class_name>

{

```
    data member;
    method;
  }
```

A simple class example

```
class student
  {
    String name;
    int rollno;
    int age;
  }
```

When a reference is made to a particular student with its property then it becomes an object, physical existence of Student class.

student std=new student( );

After the above statement std is instance/object of Student class. Here the new keyword creates an actual physical copy of the object and assign it to the std variable.

1.2.2. Object

An entity that has state and behavior is known as an object E.G. chair, bike, marker, pen, table, car etc. It can be physical or logical.

Simple Example of Object and Class

In this example, a Student class is creater that have two data members viz. id and name. The object of the Student class is created by new keyword and printing the objects value.

```
class  Student
  {
    int id;                              //data member (also instance variable)
    String name;                         //data member(also instance variable)
    public static void main(String args[])
      {
        Student s1=new Student1();       //creating an object of Student
        System.out.println(s1.id);
        System.out.println(s1.name);
      }
  }
```

This object will produce the following result

0 null

1.2.3. new keyword

The new keyword is used to allocate memory at runtime.

Object and class that maintains the records of students

Example-1

In this example, the two objects of Student class are created and initializing the value to these objects by invoking the insertRecord method on it. Here, the state (data) of the objects are displayed by invoking the display Information method.

```
class Student
  {
    int rollno;
    String name;
    void insertRecord(int r, String n)  //method
       {
          rollno=r;
          name=n;
       }
    void displayInformation( )    //method
       {
          System.out.println(rollno+" "+name);
       }
    public static void main(String args[])
       {
          Student   s1=new Student2();
          Student   s2=new Student2();
          s1.insertRecord(111,"Riya");
          s2.insertRecord(222,"Amol");
          s1.displayInformation();
          s2.displayInformation();
       }
  }
```

This will produce the following result

111 Riya

222 Amol

Example-2 of Object and Class

There is given another example that maintains the records of Rectangle class. Its explanation is same as in the above Student class example.

```
class Rectangle
  {
    int length;
    int width;
    void insert(int l, int w)
      {
        length=l;
        width=w;
      }
    void calculateArea()
      {
        System.out.println(length*width);
      }
    public static void main(String args[])
      {
        Rectangle r1=new Rectangle();
        Rectangle r2=new Rectangle();
        r1.insert(11,5);
        r2.insert(3,15);
        r1.calculateArea();
        r2.calculateArea();
      }
  }
```

This will produce the following result

55

45

1.2.4. Method in JAVA

Method describe behavior of an object. A method is a collection of statements that are group together to perform an operation.

Syntax of method is

```
return-type methodName(parameter-list)
  {
    //body of method
  }
```

Example of a Method

```
public String getName(String st)
      {
        String name="CoreJava";
        name=name+st;
        return name;
      }
```

1.3. JAVA Tokens

Tokens are the various JAVA program elements which are identified by the compiler. A token is the smallest element of a program that is meaningful to the compiler. Tokens supported in JAVA include keywords, variables, constants, special characters, operations etc.

Tokens are the smallest unit of Program. There is Five Types of Tokens

- Reserve Word or Keywords
- Identifier
- Literals
- Operators
- Separators

1.3.1. JAVA Key Words

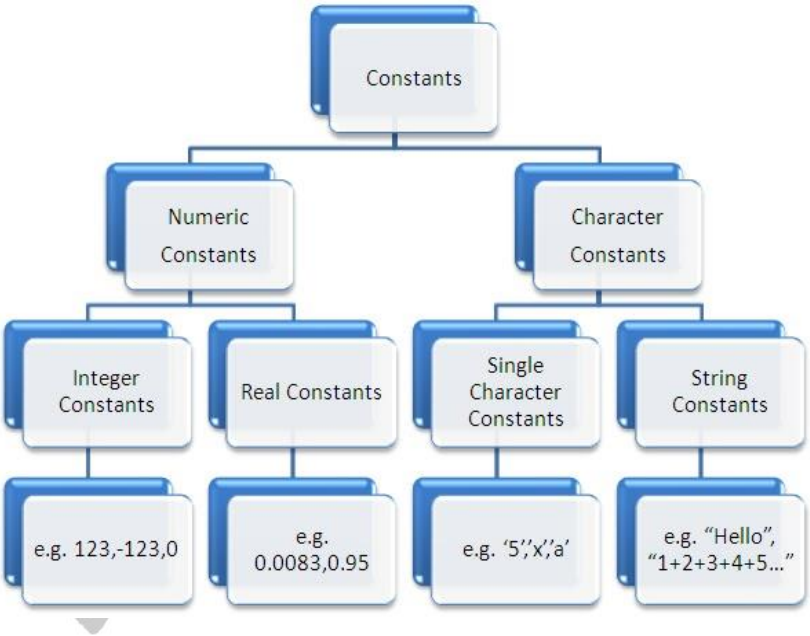
In the JAVA programming language, a keyword is one of 50 reserved words. that have a predefined meaning in the language; because of this, programmers cannot use keywords as names for variables, methods, classes, or as any other identifier.

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

Even though "goto' and "const" are no longer used in the JAVA programming language, they still cannot be used.

1.3.2. Constants

Constants in JAVA are fixed values those are not changed during the Execution of program JAVA supports several types of Constants those are



Integer Constants

Integer Constants refers to a Sequence of digits which Includes only negative or positive Values and many other things those are as follows

- An Integer Constant must have at Least one Digit
- It must not have a Decimal value
- It could be either positive or Negative
- If no sign is Specified then it should be treated as Positive
- No Spaces and Commas are allowed in Name

Real Constants

- A Real Constant must have at Least one Digit
- It must have a Decimal value
- It could be either positive or Negative
- If no sign is Specified then it should be treated as Positive
- No Spaces and Commas are allowed in Name
- Like 251, 234.890 etc are Real Constants

In The Exponential Form of Representation the Real Constant is Represented in the two Parts The part before appearing e is called mantissa whereas the part following e is called Exponent.

- In Real Constant The Mantissa and Exponent Part should be Separated by letter "e"
- The Mantissa Part have may have either positive or Negative Sign
- Default Sign is Positive

Single Character Constants

A Character is Single Alphabet a single digit or a Single Symbol that is enclosed within Single inverted commas.
Like 'S' ,'1' etc are Single Character Constants

String Constants

String is a Sequence of Characters Enclosed between double Quotes These Characters may be digits , Alphabets Like "Hello" , "1234" etc.

Backslash Character Constants

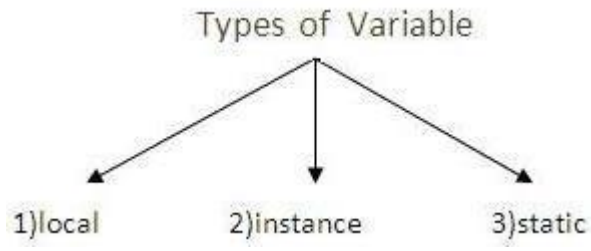
JAVA Also Supports Backslash Constants. These are used in output methods. E.G. – \n is used for new line Character These are also Called as escape Sequence or backslash character Constants.

E.G.
\t For Tab (Five Spaces in one Time)
\b Back Spac
\f form feed
\n line feed
\r carriage reurn
\" double quote
\' single quote
\\ blackslash

1.3.3. Variable

Variable is name of reserved area allocated in memory.

E.G. – int data = 50; // Here data is variable



There are three types of variables in JAVA

- **Local variable** – a variable that is declared inside the method is called local variable
- **Instance variable** – a variable that is declared inside the class but outside the method is called instance variable . It is not declared as static
- **Static variable** – a variable that is declared as static is called static variable. It cannot be local

Variable declarations example

```
int   maxAge;
int   x, y, selectedIndex;
char a, b;
boolean  flag;
double    maxVal, massInKilos;
```

Variable initialization in declaration example

```
int   timeInSeconds = 245;
char a = 'K', b = '$';
boolean  flag = true;
double  maxVal = 35.875;
```

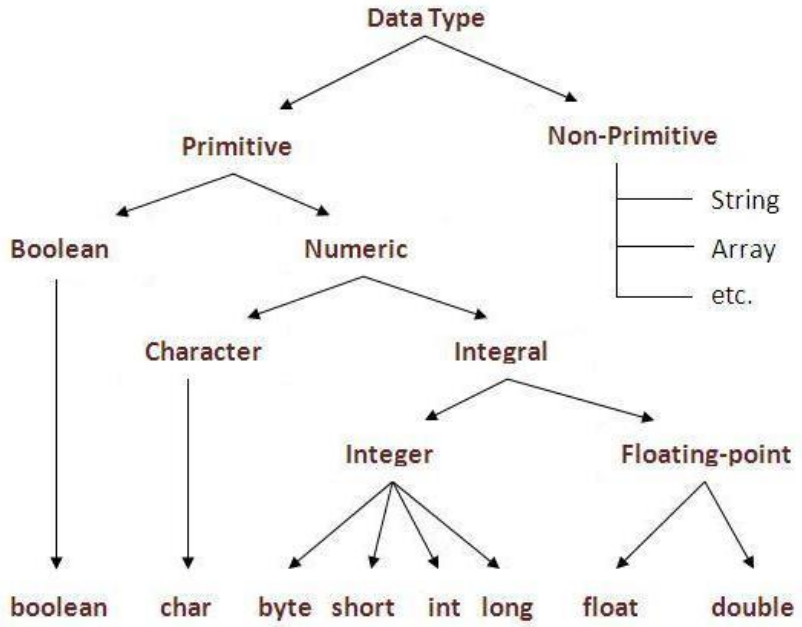
Types of variables example

```
class A
   {
      int data=50;              //instance variable
      static int m=100;         //static variable
      void method()
          {
              int n=90;         //local variable
          }
   }                            //end of class
```

1.3.4. Data Types in JAVA

In JAVA, there are two types of data types

- Primitive data types
- Non-primitive data types



Data Type	Default Value	Default size
boolean	FALSE	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

String Objects

A string in JAVA is not a primitive data type. It is an object from the system defined class String.

String constants example

"watermelon", "fig", "$%&*^%!!", "354", " " (space), "" (null string)

The string declaration example

String item = "apple";

This declaration is short for the object declaration.

String item = new String("apple");

Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.

1.4. Operator and Expression

Operators are special symbols used for mathematical functions, some types of assignment statements, and logical comparisons.

An expression is a statement that can convey a value. Some of the most common expressions are mathematical E.G.

```
int x = 3;
int y = x;
int z = x * y
```

JAVA provides a rich set of operators to manipulate variables. One can divide all the JAVA operators into the following groups

1. Arithmetic Operators
2. Increment & Decrement Operators
3. Relational Operators
4. Bitwise Operators
5. Logical Operators
6. Assignment Operators

1.4.1. Arithmetic Operators

Arithmetic operators are used in mathematical expressions.

Considered two variables A & B

Operator	Meaning	Syntax
+	Addition	A + B
-	Subtraction -	A - B
*	Multiplication -	A * B
/	Division -	A / B
%	Modulus	A% B
++	Increment - Increases the value of operand by 1	A++ ,
--	Decrement - Decreases the value of operand by 1	A --

The following Program Demonstrate the use of Basic arithmetic operators

```
public class ArithmDemo
   {
      public static void main(String[] args)
         {
            int add = 2 + 4;
            System.out.println( "2 + 4 = " + add);
            int subtract = 10 - 3;
            System.out.println("10 - 3 = " + subtract);
            int divide = 6/2;
            System.out.println("6 / 2 = " + divide);
            int multiply = 5 * 5;
            System.out.println("5 * 5 = " + multiply);
            int modulus = 7 % 2;
            System.out.println("7 % 2 = " + modulus);
         }
   }
```

The output from above program will be

```
2 + 4 = 6
10 - 3 = 7
6 / 2 = 3
5 * 5 = 25
7 % 2 = 1
```

1.4.2. Increment and Decrement Operators

The increment operator increases its operand by one. The decrement operator decreases its operand by one.

E.G.

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

The above statement can be rewritten by using the increment operator

```
x++;
```

Similarly, the statement

```
x = x – 1
```

is equivalent to

```
x--;
```

The following program demonstrates the increment operator.

```
class IncDec
   {
```

```
    public static void main(String args[])
        {
            int a = 1;
            int b = 2;
            int c;
            int d;
            c = ++b;
            d = a++;
            c++;
            System.out.println("a = " + a);
            System.out.println("b = " + b);
            System.out.println("c = " + c);
            System.out.println("d = " + d);
        }
}
```

1.4.3. Relational Operators

The relational operators determine the relationship that one operand has to the other.

Operator	Meaning	Syntax
==	Equal to	(A == B)
!=	Not equal to	(A != B)
>	Greater than	(A > B)
<	Less than	(A < B)
>=	Greater than or equal to	(A >= B)
<=	Less than or equal to	(A <= B)

Following program demonstrate the use of Relational Operator ==, !=, >, <, >= and <=

```
class RelOptrDemo
    {
        public static void main(String[] args)
            {
                int a = 10, b = 15, c = 15;
                System.out.println("Relational Operators and returned values");
                System.out.println(" a > b = " + (a > b));
```

```
            System.out.println(" a < b = " + (a < b));
            System.out.println(" b >= a = " + (b >= a));
            System.out.println(" b <= a = " + (b <= a));
            System.out.println(" b == c = " + (b == c));
            System.out.println(" b != c = " + (b != c));
        }
    }
```

Output from above program will be

```
a > b = false
a < b = true
b >= a = true
b <= a = false
b == c = true
b != c = false
```

1.4.4. Bitwise Operators

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format, value of a & b will be

```
a = 0011 1100
b = 0000 1101
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
```

Operator	Description	Syntax
&	Bitwise AND	(A & B)
|	Bitwise OR	(A | B)
^	Bitwise exclusive OR	(A ^ B)
~	Bitwise unary NOT	(~A).
<<	Shift left	A << 2
>>	Shift right	A >> 2
>>>	Shift right zero fill	A >>>2 will give 15 which is 0000 1111

The following is the Demonstrating example of Bitwise Operators

```
public class Test
   {
      public static void main(String args[])
         {
            int a = 60;     /* 60 = 0011 1100 */
            int b = 13;     /* 13 = 0000 1101 */
            int c = 0;
            c = a & b;       /* 12 = 0000 1100 */
            System.out.println("a & b = " + c );
            c = a | b;       /* 61 = 0011 1101 */
            System.out.println("a | b = " + c );
            c = a ^ b;       /* 49 = 0011 0001 */
            System.out.println("a ^ b = " + c );
            c = ~a;          /*-61 = 1100 0011 */
            System.out.println("~a = " + c );
            c = a << 2;     /* 240 = 1111 0000 */
            System.out.println("a << 2 = " + c );
            c = a >> 2;     /* 215 = 1111 */
            System.out.println("a >> 2  = " + c );
            c = a >>> 2;  /* 215 = 0000 1111 */
            System.out.println("a >>> 2 = " + c );
         }
   }
```

Output from above program will be

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 15
a >>> 15
```

1.4.5. Logical Operators

Assume Boolean variables A holds true and variable B holds false then the following table lists the logical operators

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

The following program Demonstrates the use of logical operators.

```
public class Test
   {
      public static void main(String args[])
         {
            boolean a = true;
            boolean b = false;
            System.out.println("a && b = " + (a&&b));
            System.out.println("a || b = " + (a||b) );
            System.out.println("!(a && b) = " + !(a && b));
         }
   }
```

Output from above program will be

```
a && b = false
a || b = true
!(a && b) = true
```

1.4.5. Assignment Operators

Following assignment operators are supported by JAVA

Operator	Description	Syntax/Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

The following program demonstrates the assignment operators

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

```
        int a = 10;
        int b = 20;
        int c = 0;
        c = a + b;
        System.out.println("c = a + b = " + c );
        c += a ;
        System.out.println("c += a  = " + c );
        c -= a ;
        System.out.println("c -= a = " + c );
        c *= a ;
        System.out.println("c *= a = " + c );
        a = 10;
        c = 15;
        c /= a ;
        System.out.println("c /= a = " + c );
    }
}
```

Output from above program will be

```
c = a + b = 30
c += a  = 40
c -= a = 30
c *= a = 300
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 10
c |= a   = 10
```

Special Operators

There are few other operators supported by JAVA Language.

1.4.6 Conditional Operator (? :)

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.

The operator is written as follows

variable x = (expression) ? value if true : value if false

The following program demonstrates the conditional operator

```
public class Test
   {
      public static void main(String args[])
         {
            int a , b;
            a = 10;
            b = (a == 1) ? 20 : 30;
            System.out.println( "Value of b is : " +  b );
            b = (a == 10) ? 20 : 30;
            System.out.println( "Value of b is : " + b );
         }
   }
```

Output from above program will be

```
Value of b is : 30
Value of b is : 20
```

1.4.7. instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type).

instanceof operator is wriiten as

(Object reference variable) instanceof (class/interface type)

The following program demonstrates the instanceof operator

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

```
        {
            String name = "James"; // following will return true since name is type of String
            boolean result = name instanceof String;
            System.out.println( result );
        }
}
```

Output from above program will be

true

1.4.8. Precedence of JAVA Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others;

E.G. – the multiplication operator has higher precedence than the addition operator

E.G. – x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Category	Operator	Associativity
Postfix	( ) [ ] . (dot operator)	Left toright
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

1.4.9. Mathematical functions

- Java provides a support for performing mathematical operations by means of mathematical functions.
- The basic mathematical functions are defined in the **math** class defined in **java.lang** package.
- Various mathematical functions are

sin(double angle)	Returns the sine of angle in radians
cos(double angle)	Returns the cosine of angle in radians
tan(double angle)	Returns the tan of angle in radians
asin(double val)	Returns the angle whose sine is val
acos(double val)	Returns the angle whose cosine is val
atan(double val)	Returns the angle whose tan is val
pow(double a, double b)	It computes a^b
exp(double a)	It computes e^a
sqrt(double a)	It computes square root of a
max(a,b)	It computes the maximum of a & b
min(a,b)	It computes the minimum of a & b
log(val)	It computes the logarithm values of val
abs(val)	It computes the absolute value of val
ceil(val)	It returns the smallest whole no. which is greater than or equal to val
floor(val)	It returns the largest whole no. which is lesser than or equal to val

Program for demonstrating mathematical functions

```
class MathDemo
  {
    public static void main(String args[])
      {
        double val=25;
        System.out.println("The square root of " +val+ "is" +Math.sqrt(val));
        int a=10, b=20;
        System.out.println("The maximum of " +a+  " and"  +b+ "is"+Math.max(a,b));
        System.out.println("The minimum of " +a+  " and"  +b+ "is"+Math.min(a,b));
```

```
            System.out.println("The sine of " +val+ "is" +Math.sin(val));
            System.out.println("The cosine of " +val+ "is" +Math.cos(val));
        }
    }
```

Output from above program will be

The square root of 25.0 is 5.0

The maximum of 100 and 20 is 20

The minimum of 10 and 20 is 10

The sin of 25.0 is -0.13235175009777303

The cos of 25.0 is 0.9912028118634736

1.5. Decision making and looping

JAVA language supports two types of selections statements

- if statements
- switch statements

1.5.1. Decision Making with If Statement

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

- Simple if statement
- if....else statement
- Nested if....else statement
- else if ladder

1.5.1.1. Simple If Statement

The syntax of the "if statement" is shown below

```
if(expression)
    statements;
```

Where a statement may consist of a single statement, a compound statement, or nothing. The expression must be enclosed in parentheses. If the expression evaluates to true, the statement is executed, otherwise ignored.

The following program demonstrates the if statement

```
class Test
    {
```

```
    public static void main(String args[])
      {
        int x = 10;
        if( x < 20 )
          {
            System.out.print("This is if statement");
          }
      }
  }
```

This would produce the following result

This is if statement

1.5.1.2.The If....Else Statement

This form of if allows either-or condition by providing an else clause.

The syntax of the if-else statement is the following

```
if(expression)
        statemen-1;
else
        statement-2;
```

If the expression evaluates to true i.e., a non-zero value, the statement-1 is executed, otherwise statement-2 is executed. The statement-1 and statement-2 can be a single statement, or a compound statement, or a null statement.

The following program demonstrates the if…else statement

```
public class Test
  {
    public static void main(String args[])
      {
        int x = 30;
        if( x < 20 )
          {
            System.out.print("This is if statement");
          }
        else
```

```
            {
                System.out.print("This is else statement");
            }
        }
    }
```

This would produce the following result

This is else statement

1.5.1.3.Nested if....else Statement

A Nested if is an if that has another if in it's 'if's' body or in it's else's body. The nested if can have one of the following three forms

Example 1

```
If (expression1)
    {
        If (expression2)
            statement-1;
        else
            statement-2;
    }
else
    body of else;
```

Example 2

```
If (expression1)
    body-of-if;
else
    {
        If (expression2)
            statement-1;
        else
            statement-2;
    }
```

Example 3

```
If (expression1)
```

```
        {
            If (expression2)
                statement-1;
            else
                statement-2;
        }
    else
        {
            If (expression3)
                statement-3;
            else
                statement-4;
        }
```

The following program demonstrates the nested if…else statement

```
public class Test
  {
    public static void main(String args[])
      {
        int x = 30;
        int y = 10;
        if ( x == 30 )
          {
            If ( y == 10 )
              {
                System.out.print("X = 30 and Y = 10");
              }
          }
      }
  }
```

This would produce the following result

```
    X = 30 and Y = 10
```

1.5.1.4.The else-if Ladder

A common programming construct in the JAVA is the if-else-if ladder , which is often also called the if-else-if staircase because of it's appearance.

It takes the following general form

```
if (expression1)
    statement1;
else
    if (expression2)
        statement2;
    else
        if (expression3)
            statement3;
        else
            statement n;
```

The following program demonstrates the else-if ladder

```
public class Test
  {
    public static void main(String args[])
      {
        int x = 30;
        if ( x == 10 )
            {
              System.out.print("Value of X is 10");
            }
        else
            If ( x == 20 )
                {
                  System.out.print("Value of X is 20");
                }
            else
                if ( x == 30 )
                    {
```

```
                System.out.print("Value of X is 30");
            }
        else
            {
                System.out.print("This is else statement");
            }
    }
}
```

This would produce the following result

Value of X is 30

1.5.2. Switch Statement

JAVA provides a multiple branch selection statement known as switch.

The syntax of enhanced for loop is

```
Switch (expression)
    {
        case value :
            //Statements
        break;      //optional
        case value :
            //Statements
        break;      //optional
            //One can have any number of case statements.
        default :   //Optional
            //Statements
    }
```

The following program demonstrates the switch statement

```
class SwitchDemo
{
    public static void main(String[] args)
        {
            int month = 8;
            switch (month)
```

```
        {
            case 1 :  System.out.println("January");
            break;
            case 2 :  System.out.println("February");
            break;
            case 3 :  System.out.println("March");
            break;
            case 4 :  System.out.println("April");
            break;
            case 5 :  System.out.println("May");
            break;
            case 6 :  System.out.println("June");
            break;
            case 7 :  System.out.println("July");
            break;
            case 8 :  System.out.println("August");
            break;
            case 9 :  System.out.println("September");
            break;
            case 10 : System.out.println("October");
            break;
            case 11 : System.out.println("November");
            break;
            case 12 : System.out.println("December");
            break;
            default : System.out.println("Invalid month.");
            break;
        }
    }
}
```

In this case, "August" is printed to standard output.

1.5.2.1.Nested Switch

One can use switch as part of the statement sequence of an outer switch. This is called a nested switch. Since a switch statement defines in its own block, no conflicts arise between the case statements in the inner switch and those in the outer switch.

```
switch(count)
  {
    case 1 :
      switch(target)
        {
          case 0 :
          System.out.println("target is zero");
          break;
          case 1 :
          System.out.println("target is one");
          break;
        }
      break;
    case2 : //. . .
    }
```

Here the case 1 – statement is inner switch does not conflict with the case 1 : statement in the outer switch. The count variable is only compared with the list of cases at the outer level. if the count is 1, then target is compared with the inner list cases.

No two case constants in the same switch can have individual values. A switch statement enclosed by an outer switch can have case constants in common.

1.5.3. The while Statement

The most simple and general looping structure available in JAVA is the while statement. The syntax of while loop is

```
while(condition)
  {
    // loop-body
  }
```

The following program demonstrates the while statement

```
public class Test
   {
      public static void main(String args[])
         {
            int x = 10;
            while ( x < 20 )
               {
                  System.out.print("value of x : " + x );
                  x++;
                  System.out.print("\n");
               }
         }
   }
```

This would produce the following result

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

1.5.4. The do while Statement

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

The syntax of the do-while loop is

```
do
   {
```

```
        loop-body;
    }
  While (condition);
```

The following program demonstrates the do-while statement

```
public class Test
  {
    public static void main(String args[])
      {
        int x = 10;
        do
          {
            System.out.print("value of x : " + x );
            x++;
            System.out.print("\n");
          }
        While ( x < 20 );
      }
  }
```

This would produce the following result

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

1.5.5. The for Statement

The **for** loop is the easiest to understand of the JAVA loops. All its loop-control elements are gathered in one place (on the top of the loop), while in the other loop construction of C++ , they(top-control elements) are scattered about the program.

The Syntax of the for loop statement is

```
for( initialization expression(s); test condition; update expression)
    {
        loop-body;
    }
```

//program of for loop

```
class forTest
  {
    public static void main(String args[])
      {
        int i;
        for ( i=1; i<=10; i++)
          System.out.println(i);
      }
  }
```

This would produce the following result

```
1
2
..
.
.
10
```

1.5.6. The break Keyword

The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement. The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

The syntax of a break is a single statement inside any loop

```
            break;
```

The following program demonstrates the break keyword

```
public class Test
   {
      public static void main(String args[])
         {
            int [ ] numbers = {10, 20, 30, 40, 50};
            for (int x : numbers )
               {
                  If ( x == 30 )
                     {
                        break;
                     }
                  System.out.print( x );
                  System.out.print("\n");
               }
         }
   }
```

This would produce the following result

```
      10
      20
```

1.5.7. The continue Keyword

The continue keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression

The syntax of a continue is a single statement inside any loop

```
      continue;
```

The following program demonstrates the continue keyword

```
public class Test
   {
      public static void main(String args[])
         {
            int [ ] numbers = {10, 20, 30, 40, 50};
            for (int x : numbers )
               {
                  If ( x == 30 )
                     {
                        continue;
                     }
                  System.out.print( x );
                  System.out.print("\n");
               }
         }
   }
```

This would produce the following result

```
10
20
40
50
```

1.6. Programs

1) Write a program in java to print your name?
2) Program for creating class students and printing the data for two objects?
3) Write a program in java to demonstrate the arithmetic operators?
4) Write a program in java, to calculate the area and circumference of circle?
5) Write a program in java to calculate the area and perimeter of square?
6) Write a program in java calculate the area and perimeter of rectangle?
7) Write a program in java to demonstrate the math functions?
8) Write a program in java to find the large/small no among the two nos?

9) Write a program in java to check the no odd or even?
10) Write a program in java to find the large/small no among the three nos?
11) Write a program in java to check ovel or consonants?
12) Write a program in java to reverse the given nos?
13) Write a program in java to print the multiplication table of 10?
14) Write a program in java to check the no is divisible by 7?
15) Write a program in java to print your name five times?
16) Write a program in java to find the sum of digit of given no?
17) Write a program in java to find the no of and sum of all integers greater than 100 and less than 200 that are divisible by7.
18) Write a program in java to find all the odd nos between 120 to 210?
19) Write a program in java to generates the fibbonacci series of given no?
20) Write a program in java to find the factorial no of given no?
21) Write a program in java to check the prime no?
22) Write a program in java to print the following outputs?

a)
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

b)
```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

c)
```
*
* *
* * *
* * * * *
* * * * * *
```

d)
```
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5
```

e)
```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

End of Unit-I