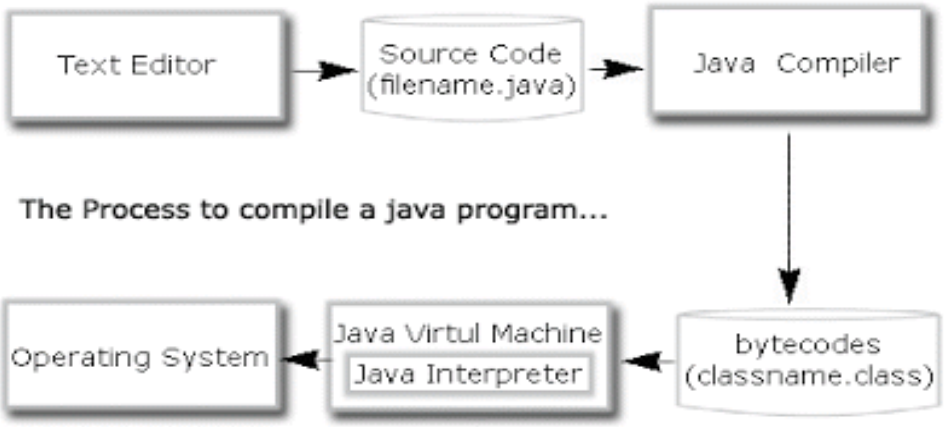
Ch-1 Basic Syntactical Construct in JAVA Marks: 10

Features of Java

1. **Compile & Interpreted**: Java is a two staged system. It combines both approaches. First java compiler translates source code into byte code instruction. Byte codes are not machine instructions. In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compile and interpreted language.



2. **Platform independent and portable**: Java programs are portable i.e. it can be easily moved from one computer system to another. Changes in OS, Processor, system resources won't force any change in java programs. Java compiler generates byte code instructions that can be implemented on any machine as well as the size of primitive data type is machine independent.

3. **Object Oriented**: Almost everything in java is in the form of object. All program codes and data reside within objects and classes. Similar to other OOP languages java also has basic OOP properties such as encapsulation, polymorphism, data abstraction, inheritance etc. Java comes with an extensive set of classes (default) in packages.

4. **Robust & Secure**: Java is a robust in the sense that it provides many safeguards to ensure reliable codes. Java incorporates concept of exception handling which captures errors and eliminates any risk of crashing the system. Java system not only verifies all memory access but also ensure that no viruses are communicated with an applet. It does not use pointers by which you can gain access to memory locations without proper authorization.

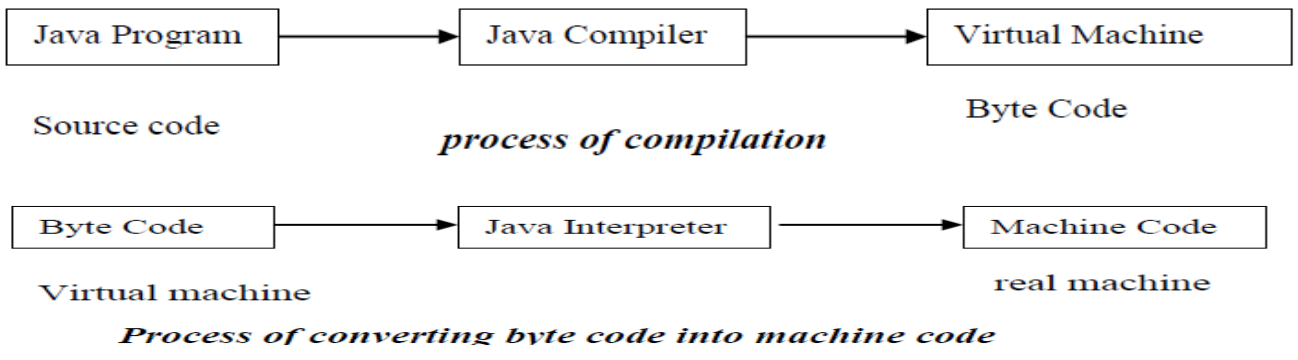
5. **Distributed:** It is designed as a distributed language for creating applications on network. It has ability to share both data and program. Java application can open and access remote object on internet as easily as they can do in local system.

6. **Multithreaded:** It can handle multiple tasks simultaneously. Java makes this possible with the feature of multithreading. This means that we need not wait for the application to finish one task before beginning other.

7. **Dynamic and Extensible**: Java is capable of dynamically linking new class library's method and object. Java program supports function written in other languages such as C, C++ which are called as native methods. Native methods are linked dynamically at run time.

Q. What is Byte-code? What is JVM? Explain any two tools available in JDK.

- **Byte Code:** When java program is compiled, then java compiler produces an intermediate code for a machine that does not exist instead of machine code. This intermediate form of code is known as **byte code.** By Java interpreter we can interpret this Byte code on any machine and run on any machine. The file which contains this code has **.class** extension; e.g. if we compile a file main1.java then after successful compilation main1.class containing byte code will be created. It is not machine specific. Once a Java program has been converted to bytecode, it can be transferred across a network and executed by Java Virtual Machine (JVM).
- **JVM (Java Virtual Machine:** All language compilers translate source code into *machine code* for a specific computer. But, Java compiler produces an intermediate code known as ***byte code*** for a machine that does not exist. This machine is called the ***Java Virtual Machine (JVM)*** and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine. The interpreter is different for different machines.



Process of converting byte code into machine code

JDK (Java Development Kit) Tools:

Component of JDK.

1. **Java Compiler** – Java Compiler is used to compile java files. It translate java source file to byte code. Java Compiler component of JDK (Java Development Kit) is accessed using ―javac command.
2. **Java Interpreter** – Java Interpreter is used to interpret the java files that are compiled by Java Compiler. It is used to run java program. Java Interpreter component of JDK (Java Development Kit) is accessed using ―java command.
3. **Java Disassembler**- Java Disassembler is used to convert a bytecode files into a java program. Java Dissembler component of JDK (Java Development Kit) is accessed using "javap" command.
4. **Java Header File Generator** - Java Header File Generator is used to generate C language header files and source files to implement the native methods. Java Header File Generator component of JDK (Java Development Kit) is accessed using ―javah‖ command.
5. **Java Documentation** – It is used to create HTML- format documentation from java source code file. Javadoc command is used.
6. **Java Debugger** – Java Debugger is used to debug the java files. Java Debugger component of JDK (Java Development Kit) is accessed using ―jdb command.
7. **Java Applet Viewer**

   Java Applet Viewer is used to view the Java Applets. Java Applet Viewer component of JDK (Java Development Kit) is accessed using ―appletviewer command.

Q. What is Command Line Arguments in Java
Q. Write a program to accept two numbers as command line arguments and print the addition of those numbers.

When we provided the input at the time of execution, it is known as command line arguments. Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution. Args is declared as an array of strings. Any arguments provided in the command line are passed to the array args as its elements.

```
class testarg
{
public static void main(String args[])
{
int a = Integer.parseInt(args[0]); //take argument as command line
int b = Integer.parseInt(args[0]); //take argument as command line
int c = a + b;
System.out.println("Addition of number is : "+c);
}
}
```

Q. Give all primitive data types available in Java with their storage sizes in bytes.

Primitive Data Type	Default Value	Default size
boolean	False/true	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

Example:

```
class oct_num
{
public static void main(String args[])
{
int i=010;
int j=07;
System.out.println("i="+i);

System.out.println("j="+j);
}
}
```

```
C:\Program Files\Java\jdk1.8.0_131\bin>java oct_num
i=8
j=7
```

Byte data type cyclic in nature.(rangeis : -128 to 127)

For example:

```
class byte_demo1
{
public static void main(String args[])
{
byte b=127;
b++;
b++;
b++;
System.out.println("b="+b);
}
}
```

Output:

```
C:\Program Files\Java\jdk1.8.0_131\bin>java byte_demo1
b=-126
```

Q. Why java is not 100% object oriented language

According to this assumption, there are several reasons why Java is not a 100% Object Oriented Programming Language:

1) **Primitive data types** (int, char, floats, etc.) are not an object and couldn't be defined as an object, neither a class. It is clear enough, because primitive data don't have any ability/characteristic such as inheritance or polymorphism. So, we can say that Java can be defined as not fully OOP supported language, since it's allowed a non-object-oriented thing to exist. int, float, etc. primitive data types are not objects. But java implemented Wrapper classes for these primitive types to make them in to object types.
2) **Multiple-inheritance through classes is not possible:** As same as the operator overloading, multiple inheritance is also an (ad-hoc) abilities. It does support multiple inheritances directly, instead of that we can use interfaces, and with the help of interface implicitly we can use multiple inheritances.
3) **Availability of static methods and variables:** Since this ability exists in Java, which allows developers to invoke a method without instantiating any object like breaking rules of encapsulation.
4) **Operator overloading is not possible in java:** Except "+" operator which can be used in two different operations (addition operation and concatenation of string), there is no possibilities in Java to do an operator-overloading. Since, this is a kind of polymorphism ability (specific for operators) we also could say that Java isn't 100% OOP Language.

Java Tokens

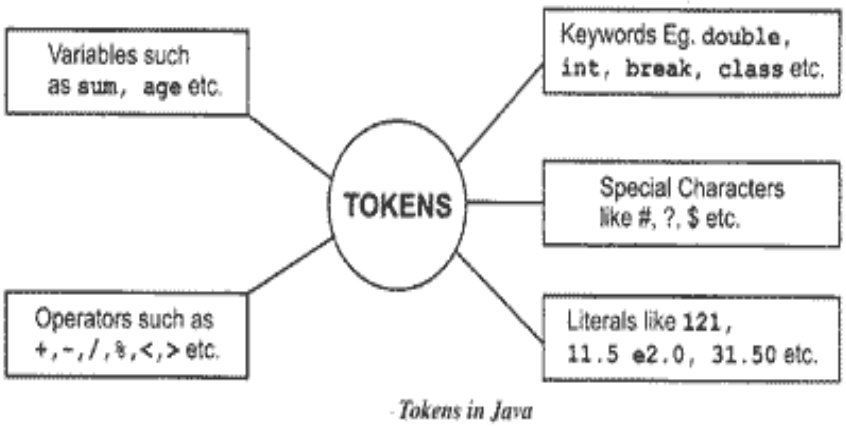
- Java program is made up of classes and methods and in the methods there are containers of various statements and a statement is made up of variables, constants, operators etc.
- Tokens are the various Java program elements which are identified by the compiler.
- A token is the smallest element of a program that a meaningful to the compiler.
- Token supported in Java include keywords, variables, constants, special keyword, operators etc.
- When you compile the program, the compiler scan the text in you source code and extract individual tokens.
- While tokenizing the source file, the compiler recognizes and subsequently removes white spaces and text enclosed within comments.

- E.g.

```
class world
{
public static void main (String arg[])
(
System.out.println("Hello World"),
}
}
```

The source code contains token such as
Class, world, {, public, static, void, main, (, String, arg, [,],), (, System, ., out, ., println, (, “,
Hello, world, “,), ;, }, }

- The resulting tokens are compiled into java bytes codes that are capable of being executed from interpreted Java environments.
- Tokens are useful for compiles to detect error.
- When java tokes are not arranges in particular sequence, the compiler generates an error message.
- Token is the smallest unit of program and types of token are
    1. Keywords
    2. Identifier
    3. Literals
    4. Operators
    5. Separators.

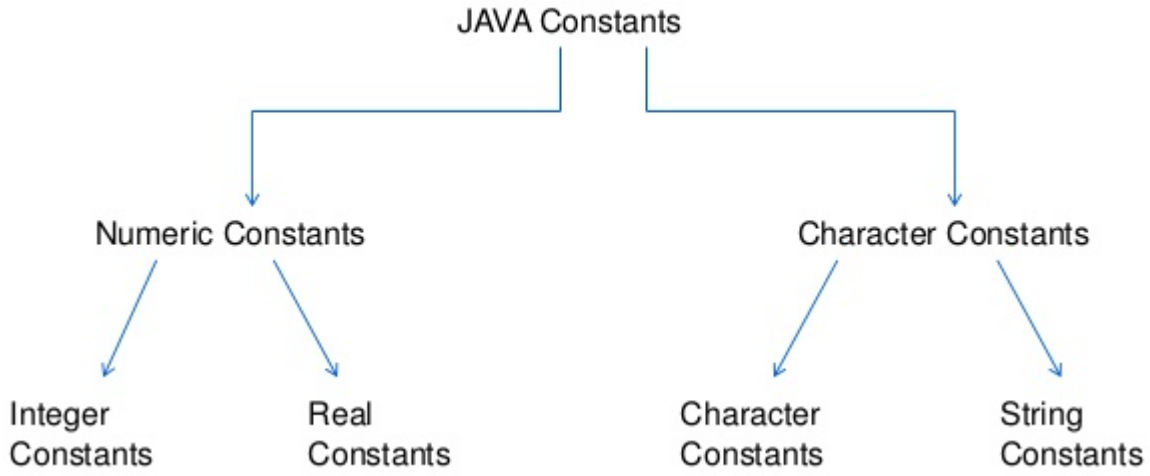


Tokens in Java

Constants

- Constant in java refer to the fixed value that do not changed during execution of program.
- Constants in java can be declared by using keyword final.

  final int max = 100;



Integer constants can be declared in three types:-

1. Decimal integer constants(235, 346)
2. Octal integer constants (O346, O132)
3. Hexadecimal integer constant(5FAD, 2B35)

Symbolic Constants

- We often used certain unique constant in program. These constants may appear repeatedly in a no. of places in the program. E.g. Suppose constant is 3.14 representing a value of mathematical constant “Pi”
- We face 2 problems in the subsequent use of such program.

- Problem in modification of program.
- Problem in understanding the program.

Variables

- A variable is an identifier that denotes location used to store a data value.
- A variable name can be chosen by programmer in a meaningful way e.g. average, grass salary, etc.
- Variable name may consist of alphabets, digits, underscore (_), & dollar character subject to the following condition.
    1. They must not begin with a digit.
    2. Uppercase and lowercase are distinct. That means variable Total is not same as total/TOTAL.
    3. It should not be a keyword.
    4. Whitespace is not allowed.
    5. Variable name can be of any length.

Dynamic Initialization

- To use a local variable, you have to either initialize or assign it before the variable is first used.
- But for class members the compulsion is not so strict.
- If you do not initialize them then compiler takes care of the initialization process and set class member to default value.
- Initializing variable at run time is known as Dynamic initialization.

```
e.g.
public class simple
{
public static void main (String args[])
{
double a=3.0d, b=4.0d;
double c=Maths.sqrt(a*a+b*b*) // c is dynamically initialized
System.out.println("result=" +c);
}
}
```

Scope of variables

- The area of program where variable is accessible is called scope.
- Java variables are classified in to 3 types
    1. **Local Variable**

       Variables are declared and used inside methods are called as local variables.

       They are called so because they are not available for use outside the methods.
    2. **Instance variable**

       Instance variable are created when the objects are instantiated and they are associated with objects. Instance variables are declares private, or public or protected.
    3. **Class variables**

       Class variables are global to the class and belong to the entire set of object that class creates. Visibility of static field will depend upon access specifies.

```
e.g.
class scope
{
public static void main(String args[])
{
int x;
x=10;
if(x==10)
{
int y=20;
System.out.println("x&y"+x+""+y);
}
}
```

Q. What do mean by typecasting? When it is needed?

- The process of converting one data type to another is called casting or type casting.
- If the two types are compatible, then java will perform the conversion automatically.
- It is possible to assign an int value to long variable.
- However, if the two types of variables are not compatible, the type conversions are not implicitly allowed, hence the need for type casting.

- For some type, it is possible to assign a value of 1 type to a variable of different type without a cast.
- Example:

  int m = 50;

  byte n = (byte) m;

  long count = (long) m;
- Type casting is of two types: widening, narrowing.

Widening (Implicit)

- The process of assigning a smaller type to a larger one is known as widening or implicit.

Byte ⟶ short ⟶ int ⟶ long ⟶ float ⟶ double

For e.g.

```
class widening
{
public static void main(String arg[])
{
int i=100;
long l=I;
float f=l;
System.out.println("Int value is"+i);
System.out.println("Long value is"+l);
System.out.println("Float value is"+f);
}
}
```

Narrowing (Explicit)

- The process of assigning a larger type into a smaller one is called narrowing.
- Casting into a smaller type may result in loss of data.
- double ⟶ long ⟶ int ⟶ short ⟶ byte

For e.g.

```
class narrowing
{
```

```
Public static void main(String[])
{
Double d=100.04;
Long l=(long) d;
Int i=(int) l;
System.out.println(“Int value is”+i);
System.out.println(“Long value is”+l);
System.out.println(“Float value is”
}
}
```

Q. Describe arithmetic operators with suitable example.

Arithmetic operators are used to construct mathematical expressions

Operator	Description	Example	Explanation
+	Addition	x = y + z	Adds the value of y and z and stores the result in x.
-	Subtraction	x = y - z	Subtracts z from y and store the result in x.
*	Multiplication	x = y * z	Multiplies the values y and z and stores result in x.
/	Division	x = y / z	Divides y by z and stores the result in x.
%	Modulo	x = y % z	Divides y by z and stores the remainder in x.

```
class ArithmeticOperatorsExample
{
public static void main(String args[])
{
int a = 10,
b = 5, c;
c = a + b; /*Integer arithmetic*/
System.out.println("The sum of integer numbers a and b is: "+c);
c = a - b;
System.out.println("The difference of integers a and b is: "+c);
c = a * b;
System.out.println("The product of integers a and b is: "+c);
c = a / b;
System.out.println("The quotient of integers a and b is: "+c);
c = a % b;
System.out.println("The remainder of integers a and b is: "+c);
}
}
```

Q. Describe increment & decrement operators with suitable example.

Operator	Description	Example	Explanation
++	increment	x = x++ (x= x+1)	Increment value by 1.
--	decrement	x = x-- (x= x-1)	Decrement value by 1.

Q. Describe Assignment operators with suitable example.

It is used to assign the value of an expression to a variable.

Syntax: variable op = expression;

Operator	Example	Explanation
=	x = y	Assign value of x to y.
+ =	x+ = y (x= x+y)	Add the operand and assig result to left operand.
- =	x- = y (x= x-y)	Subtract the right operand from the left operand & store the result in left operand.
*=	x* = y (x= x*y)	Multiplies left operand by right operand & store the result in left operand.
/=	x / = y (x= x/y)	Divide left operand by right operand & store result in left operand.
%=	x% = y (x= x%y)	Divide left operand by right operand & store reminder in left operand.

Q. Describe pre increment and post increment operators in Java

The x++ & ++x mean the same when they form statement independently.

They are different when they are used in expression on the right hand side of assignment statement.

Initial value of x	Expression	Initial value of y	Final value of y
3	y = x++	Y=3	Y=4
3	y = ++x	Y=4	Y=4
3	y = x--	Y=3	Y=2
3	y = --x	Y=2	Y=2

Q. Describe any two relational and any two logical operators in java with simple example

Relational Operators: When comparison of two quantities is performed depending on their relation, certain decisions are made. Java supports six relational operators as shown in table.

Operator	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Program demonstrating Relational Operators

```
class relational
{
public static void main(String args[])
{
 int i=37;
int j=42;
System.out.println("i>j"+(i>j)):        //false
System.out.println("I<j"+(i<j));       //true
System.out.println("i>=j"+(i>=j));   //false
System.out.println("i<=j"+(I<=j));   // true
System.out.println("i==j"+(i==j));   // false
System.out.println("i!=j"+(i!=j));    //true
}
}
```

Logical Operators: Logical operators are used when we want to form compound conditions by combining two or more relations. Java has three logical operators as shown in table:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Program demonstrating 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:
a && b = false
a || b = true
!(a && b) = true

Q. Explain any two bitwise operators with example

Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level. These operators are used for testing the bits or shifting bits right or left in the operations.

1) Bitwise NOT (~): called bitwise complement, the unary NOT operator, inverts all of the bits of its operand.

```
Example: ~ 0111 (decimal 7)
         = 1000 (decimal 8)
```

2) Bitwise AND (&): the AND operator, &, produce a 1 bit if both operands are also 1, A zero is produced in all the cases.

```
e.g  0101 (decimal 5)
 &   0011 (decimal 3)
 =   0001 (decimal 1)
```

3) Bitwise OR(|) : the OR operator, | , combines bits such that if either of the bits in the operand is a 1, then the resultant bit is a 1

```
e.g  0101 (decimal 5)
 |   0011 (decimal 3)
 =   0111 (decimal 7)
```

4) Bitwise XOR(^) : the XOR operator , ^ , combines bits such that if exactly one operand is 1,then the result is 1. Otherwise result is zero.

```
e.g  0101 (decimal 5)
^    0011 (decimal 3)
=    0110 (decimal 6)
```

5) The Left Shift (<<): the left shift operator, <<, shifts all of the bits in a _value‘ to the left a specified number of times specified by _num‘

General form : value <<num

```
e.g. x << 2 (x=12)
 0000 1100 << 2
= 0011 0000 (decimal 48)
```

6) The Right Shift (>>): the right shift operator, >>, shifts all of the bits in a _value‘ to the right a specified number of times specified by _num‘

General form: value >>num.

```
e.g. x>> 2 (x=32)
     0010 0000 >> 2
  = 0000 1000 (decimal 8)
```

Q. Describe ?: (Ternary operator) in java with suitable example

The ternary operator (?:) is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. If it helps you can think of the operator as shortened way of writing an if-else statement. It is often used as a way to assign variables based on the result of an comparison. When used correctly it can help increase the readability and reduce the amount of lines in your code

Syntax: expression1? expression2: expression3

- Expression1 can be any expression that evaluates to a Boolean value.
- If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.
- The result of the? Operation is that of the expression evaluated.
- Both expression2 and expression3 are required to return the same type, which can't be void.

Example:

```
public class test
{
public static void main(String args[])
{
int minval;
int a=10;
int b=20;
minval=(a<b)?a:b;
System.out.println("Minimum Value="+minval);
}
}
```

Output:

Minimum value=10

Instance of operator.

It is used only for object reference variable.

Syntax:

X instance of Y

(x in object and y is name of class).

Dot (.) operator.

The dot operator is used to access the instance variable and method of class object.

For e.g. person.age //reference to the variable age.

Person.salary() //reference to the method salary.

Operator precedence and Associativity.

It determine grouping of term in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than other.

For e.g. a+b*c

In above expression the multiplication (*) operator has higher precedence than plus (+) operator.

Associativity:

When two operators with the same precedence , the expression is evaluated according to its associativity.

For e.g.

1. X=Y=Z=17 is treated as ,
   X=(Y= (Z=17))
   Leaving all 3 variables with the value 17 since the '=' operator has right to left associativity.
2. 70/2/3 is treated as (70/20)/3
   Since the operator has left to right associativity.

Q. Write any four mathematical functions used in Java.

Abs(): abs method is used to find absolute value of variable.

Syntax: Math.abs(variable);

Here variable can be double, float, int and long datatype.

Example: Math.abs(-2.3) = 2.3

Min(): min method is used to find minimum value between variable1 and variable2.

syntax: Math.min(variable1,variable2)

Here variable can be double,float,int and long datatype

Example: Math.min(2,8) = 2

Max(): max method is used to find maximum value between variable1 and variable2.

syntax: Math.max(variable1,variable2)

Here variable can be double,float,int and long datatype

Example: Math.max(2,8) = 8

Pow(): Power of the number. Here variable1 is base value and variable2 is power value

Syntax: Math.pow(variable1,variable2)

Here variable is double datatype

Example: Math.pow(2,3) = 8

Sqrt(): sqrt method is used for finding square root of a number.

Syntax: Math.sqrt(variable)

Here variable is double datatype

Example: Math.sqrt(9) = 3

Exp(): Euler's number e raised to the power of a variable.

Syntax: Math.exp(variable)

Here variable is double datatype

Example: Math.exp(2) = 7.38905609893065

Sin(): To calculate the sine of a value.

Syntax: Math.sin(double variable)

Example: Math.sin(82.55)= 0.763419622322519

Cos(): To calculate the cosine of an angle

Syntax: Math.cos(variable)

Example: Math.cos(82)= 0.949677697882543

Tan(): To calculate the tangent of a number.

Syntax: Math.tan(variable)

Example: Math.tan(80)= 9.00365494560708

Ceil(): In arithmetic, the ceiling of a number is the closest integer that is greater or higher than the

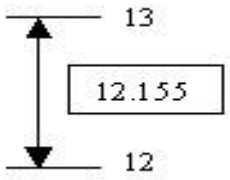
number considered. ceil method is used round up the variable.

Syntax: Math.ceil(variable)

Example: Math.ceil(12.15)=13

Math.ceil(-24.06)=-24

Consider a floating-point number such as 12.155. This number is between integer 12 and integer 13. In the same way, consider a number such as –24.06. As this number is negative, it is between –24 and –25, with –24 being greater.



Floor(): This is the floor() function which returns you the largest value but less than the argument. floor method is used round down the variable.

Syntax: Math.floor(variable)

Here variable is double datatype

Example: Math.floor(8.4) = 8

Math.floor(-24.06)=-25

copySign(): copySign method is used to copy sign of variable2 and assign that sign to variable1.

Syntax: Math.copySign(variable1, variable2)

Here variable is float and double datatype

Example: Math.copySign(2,-8.3) = -2.0

Math.copySign(-2,8.3) = 2.0

Round(): Rounds to the nearest integer.

Syntax: Math.round(variable)

Example: Math.round(1.02)=1

Math.round(-2.1)=-2

Random(): This is the random() function which returns you the random number. It is absolutely system generated. This method can be used to produce a random number between 0 and 100

Log(): Returns the natural logarithm of the argument.

Syntax: Math.log(variable)

Example: Math.log(2)= 0.6931471805599453

Q. Write general syntax of any two decision making statements and also give its examples

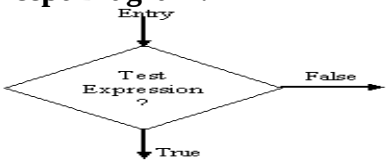
The if Statement:

If the condition is true then the statements in the block are executed otherwise no statements in the block are executed.

Syntax:

```
if (test condition is true)
{
Execute Statements;
}
```

Concept Diagram:



Example of if statement

```
class ifelse
{  Public static void main(String args[])
{         int a=10;
          int b=20;
          if (a<b)
          System.out.println("a is small");
}    }
```

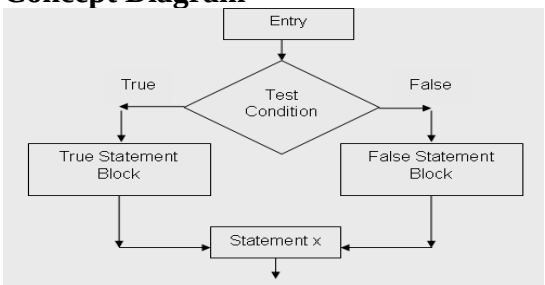
The if…..else statement

If condition is true then the True-block statement(s) is executed otherwise False-block statement(s) is executed.

Syntax:

```
    if(test condition)
    { True statement block; }
    else
    { False statement block; }
```

Concept Diagram



Example of if—else statement

```
class ifelse
{  Public static void main(String args[])
{         int a=10;
          int b=20;
          if (a<b)
          System.out.println("a is small");
          else
          System.out.println("b is small");
}
}
```

Nested-If Statement:

When a series of decisions are involved, we may have to use more than one if-else statement in nested form.

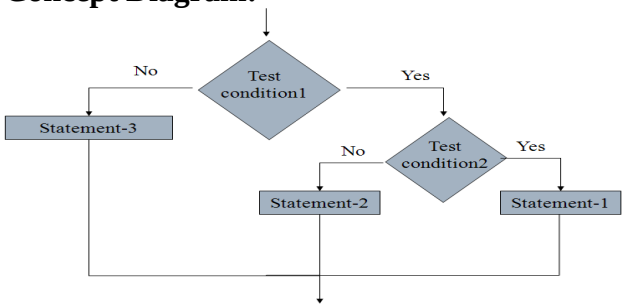
Syntax:

```
If (condition1)
{
     If (condition2)
    {      statement-1;     }
    else
    {      statement-2;     }
 }
 else
{     statement-3;     }
statement-x;
```

Example of Nested if statement

```
class largest
{
public static void main(String args[])
{
int x=10, y=30, z=20;
System.out.println("Largest number is=");
if(x>y)
{
   if(x>z)
  {
  System.out.println(x);
  }
  else
  {
  System.out.println(z);
  }
}
```

Concept Diagram:




```
else
    {
     if(z>y)
     {
    System.out.println(z);
     }
    else
     {
    System.out.println(y);
     }
}
}
}
```

The Else-If Ladder:

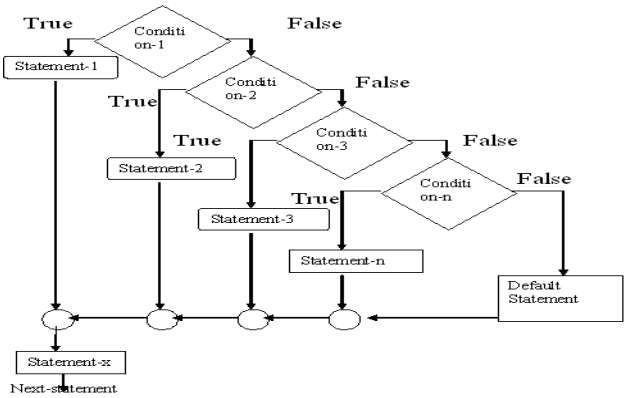
This construct is known as if else if ladder. The conditions are evaluated from the top to downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the statement –x by skipping the rest of the ladder. When all the n conditions become false then the final else if containing the default- statement will be executed.

Syntax:

```
if(condition1)
 statement-1;
      else if(condition2)
         statement-2;
              else if(condition3)
                statement-3;
                        ……….
                            ……….
                          else if(condition-n)
                               statement-n;
                                    else
                               default- statement ;

statement-x;
```

Concept Diagram:



Example of Else-if Ladder statement

```
if (marks > 79)
grade=”Honours”;
else if(marks > 59)
          grade=”First Class”;
          else if(marks > 49)
                    grade=”Second Class”;
                    else if(marks >39)
                              grade=”Pass Class”;
                              else
                              grade=”Fail”;
System.out.println(“Grade=”+grade);
```

Q. Describe break and continue statement with example

Break: 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.

Exampl:

```
public class TestBreak
{
public static void main(String args[])
{
int [] numbers = {10, 20, 30, 40, 50};
for(int x : numbers )
{
if( x == 30 )
{
break;
}
System.out.println( “value of x=“+x );
}
}
}
```

Output:

```
value of x=10
value of x=20
```

continue: The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop. A labeled continue statement skips the current iteration of an outer loop marked with the given label.

Example:

```
public class TestContinue
{
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");
}
}
}
```

Output:

```
10
20
40
50
```

Q. State syntax and describe working of ‘for each’ version of for loop with example

The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Syntax:

```
for(data_type variable : array | collection)
{  }
```

Example:

```
class ForEachExample1
{
public static void main(String args[])
{
int arr[]={12,13,14,44};
for(int i:arr)
{
System.out.println(i);
}
}
}
```

Q. Illustrate with example the use of switch statement

Java has a built-in multi way decision statement known as switch. The switch case statement is useful when user have multiple choices. Switch statement tests the value of a given variable against a list of case values & when match is found, a block of statements associated with that case is executed.

Syntax:

```
switch(expression)
{
case value1:
        block1;
        break;
case value2:
        block2;
        break;
-----
----
default :
        Default block;
        break;
}
statement n;
```

Example:

```
class switchdemo
{
public static void main ( string args[ ] )
{
int a=Integer.parseInt(args[0]);
switch (a)
{
case 1:
        System.out.println("Poor");
        break;
case 2:
        System.out.println("work Hard");
        break;
case 3:
        System.out.println("Good");
        break;
case 4:
        System.out.println("Very Good");
        break;
case 5:
        System.out.println("Excellent");
        break;
default:
        System.out.println(" Invalid value entered")
}
}
}
```

Programs:

Q. Write a program to check whether an entered number is prime or not

```
class PrimeNo
{
public static void main(String args[])
 {
Int num=Integer.parseInt(args[0]);
int flag=0;
for(int i=2;i<num;i++)
{
if(num%i==0)
{
System.out.println(num + " is not a prime number");
flag=1;
break;
}
}
if(flag==0)
System.out.println(num + " is a prime number");
```

```
}
}
```

Q. Write a program to print prime numbers in the given range.

```
class primenum
{
public static void main(String args[])
{
int i,num=0,n=20;
System.out.println("value of n="+n);
while(num<=n)
{
i=2;
while(i<=num)
{
if(num%i==0)
break;
i++;
}
if(i==num)
System.out.println(num+" is prime");
num++;
}
}
}
```

Q. Write a program to find sum of digits of number.

```
import java.io.*;
class SumOfDigits
{
public static void main(String args[])
{
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
int n;
try
{
System.out.println("Enter the number");
n = Integer.parseInt(b.readLine());
int sum = 0, n1;
while (n > 0)
{
n1 = n%10; n = n/10;
sum = sum+n1;
}
System.out.println("The sum of the digits of the number is :"+sum);
}
catch(Exception e)
{ System.out.println(e); }
 }  }
```

OR

```
class SumOfDigits
{
public static void main(String args[])
{
 int n = 1234; //Student may have assumed any other number
int sum = 0, n1;
while (n > 0)
{
n1 = n%10;
n = n/10;
sum = sum+n1;
}
System.out.println("The sum of the digits of the number is :"+sum);
}
}
```

Q. Write a program to print reverse of a number.

```
class Reverse1
{
public static void main(String args[])
{
int num = Integer.parseInt(args[0]); //take argument as command line
int remainder, result=0;
while(num>0)
{
remainder = num%10;
result = result * 10 + remainder;
num = num/10;
}
System.out.println("Reverse number is : "+result);
}
}
```

Q. Write a program to generate Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89 ...

```
class FibonocciSeries
{
public static void main(String args[])
{
int Num1=0;
int Num2=1;
int Num3=0;
System.out.println(Num1);
System.out.println(Num2);
for(int i=0;i<=20;i++)
{
Num3=Num1+Num2;
System.out.println(Num3);
Num1=Num2;
Num2=Num3;
```

```
        }
        }
}
```

Q. Write a program to accept a number and print its factorial.

```
import java.io.*;
import java.lang.*;
class Factorialn
{
public static void main(String args[])
{
DataInputStream in=new DataInputStream(System.in);
int n, fact=1;
try
{
System.out.print("Enter Number:");
n=Integer.parseInt(in.readLine());
do
{
fact=fact*n;
n--;
}while(n>=1);
System.out.println("Factorial of Number="+fact);
}
catch(Exception e)
{
System.out.println("Error");
}
}
}
```

Q. Program to check whether entered number is palindrome or not?

```
class palindrome
{
public static void main(String args[])
{
int num=Integer.parseInt(args[0]);
int n=num, reverse=0,remainder;
while(num>0)
{
remainder=num%10;
reverse=reverse*10+remainder;
num=num/10;
}
if(reverse==n)
System.out.println(n+" is a Palindrome number");
else
System.out.println(n+" is not a Palindrome number");
```

```
}
}
```

Q. Program to calculate percentage of 4 subjects and print in suitable format.

```
import java.lang.*;
import java.io.*;
class percentage
{
public static void main(String args[])
{
float sub1, sub2, sub3, sub4, per;
DataInputStream in=new DataInputStream(System.in);

try
{
System.out.print("Enter marks of subject1:");
sub1=Float.parseFloat(in.readLine());
System.out.print("Enter marks of subject2:");
sub2=Float.parseFloat(in.readLine());
System.out.print("Enter marks of subject3:");
sub3=Float.parseFloat(in.readLine());
System.out.print("Enter marks of subject4:");
sub4=Float.parseFloat(in.readLine());
per=(sub1+sub2+sub3+sub4)/4;
System.out.print("Percentage="+per);
}

catch(Exception e)
{
System.out.println("Error");
}
}
}
```

Q. Write a program to check whether the given number is divisible by 5 or not.

```
import java.io.*;
class test
{
public static void main(String args[])
{
BufferedReader br=new BufferedReader( new InputStreamReader(System.in));
int n;
try
{
System.out.println("Enter a number :");
n=Integer.parseInt(br.readLine());
if(n%5==0)
```

```
System.out.println(n + " is divisible by 5 ");
else
System.out.println(n + " is not divisible by 5 ");
}
catch(IOException e)
{ System.out.println("I/O error"); }
}
}
```

Write a program to print the following output (4m, W-11)

```
* * * *
* * *
* *
*
```

```
class triangle
{
public static void main(String args[])
{
int num=5;
while(num>0)
{
for(int j=1;j<=num;j++)
{
System.out.print(" "+"*"+" ");
}
System.out.print("\n");
num--;
}
}
}
```

Q. Define a class having one 3-digit as a data number. Initialize and display reverse of that number

```
class reverse
{
public static void main(String args[])
{
Int num = 253;
int remainder, result=0;
while(num>0)
 {
 remainder = num%10;
result = result * 10 + remainder;
num = num/10;
}
System.out.println("Reverse number is : "+result);
 }
}
```