1. Define byte code, abstract class

Bytecode: All Java programs are compiled into class files that contain bytecodes, which is the machine language of the Java virtual machine. Bytecode is the compiled format for Java programs. Once a Java program has been converted to bytecode, it can be transferred across a network and executed by Java Virtual Machine (JVM). Bytecode files generally have a *.class* extension.

Abstract class: an abstract class is one that needs to be extended and its method implemented. It cannot be instantiated.that is it should satisfy the following conditions. When it is used:

```
abstract class Shape
{
        ……
        ……
        abstract void draw();
}
```

We cannot use abstract classes to instantiate objects directly. For example,

```
Shape s = new Shape()
```

is illegal because Shape is an abstract class.

The abstract methods of an abstract class must be defined in its subclass.

We cannot declare abstract constructors or abstract static methods.

2. Define variable. Explain scope of variables.

A variable is an identifier that denotes a storage location used to store a data value. A variable may take different values at different times during the execution of the program, unlike constants that remain unchanged.

A block that begins with an opening curly brace and ends with a closing curly brace, defines a scope. Each time when a block is created, a new scope is defined. The scope determines the lifetime of the object too. There are two general categories of scopes:

Local: variables declared inside the method are known as local variables. They must be initialized before its usage. They are declared inside the scope are not visible to the code that is defined outside the scope. Thus, when we are declare the variable within a scope we are localizing the variable and protecting it from unauthorized access or modification. They are created when execution enters the method and are destroyed when the method is exited.

Global: variables that are declared in the outer scope are visible to the inner scope.

Instance variables: these are created when the objects are instantiated using ‘new’ keyword and are therefore associated with objects. They take different values for each object.

Class variables: these are global to the whole class and belong to the entire set of objects of that class.

Static variables: these are created when the class is loaded and continue to exist as long as the class is loaded.

Example:

```
public class DemoScope
{
    public static void main(String args[ ])
    {
        int x;                          //known to all code within main
        x = 10;
        if ( x = = 10)
        {                               //start of new scope
            int y = 20;                 //known only to this block
            System.out.println("x and y: " +x+ " " +y);
            x = y * 2;
        }
        // y = 100;                     // error! Y not known here
        //x is still known here
        System.out.println("x is: " +x);
    }
}
```

3. When do we declare a method final?

A method is declared as final when we wish to prevent the subclasses from overriding the members of the superclass. We declare them as final using the keyword 'final'. Making a method final ensures that the functionality defined in this method will never be altered in any way.

Example:

```
class Bike
{
    final void run()
    {
        System.out.println("running");
    }
}
class Honda extends Bike
{
    void run()
    {
```

```
        //error! Cant override
        System.out.println("running safely with 100kmph");
    }
    public static void main(String args[])
    {
        Honda h= new Honda();
        h.run();
    }
}
```

4. What is typecasting? Explain widening and narrowing with example.

Storing a value of one data type into a variable of another data type is known as type casting. We must cast the value to be stored by preceding it with the type name in parenthesis. The syntax is:

type variable1 = (type) variable2;

Casting is often necessary when a method returns a type different than the one we require.

Widening: the process of assigning a smaller type to a larger one is known as widening or promotion.

Example:

```
class Simple
{
    public static void main(String args[])
    {
        int a = 10;
        float f = a;
        System.out.println(a);
        System.out.println(f);
    }
}
```

Narrowing/promotion: the process of assigning a larger type to a smaller one is known as narrowing. Narrowing may result in loss of information.

Example:

```
class Simple1
{
    public static void main(String args[])
    {
        float f = 10.5f;
        int a = (int)f;
        System.out.println(f);
        System.out.println(a);
    }
}
```

5. Define object. How are objects created in java?

Object can be defined in different ways:

- Object is a run time entity.
- Object is an entity that has state and behavior
- Object is an instance of a class

Object has three characteristics:

- **State:** represents data (value of an object).
- **Behavior:** represents the behavior (functionality) of an object
- **Identity:** it is typically implemented via a unique ID. The value of the ID is not visible to the external user. But it is used internally by the JVM to identify each object uniquely.

An object is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object.

Objects are created using new operator. The new operator creates an object of the specified class and returns a reference to that object. Here is an example of reating an object of type Rectangle class. Refer to the figure below.

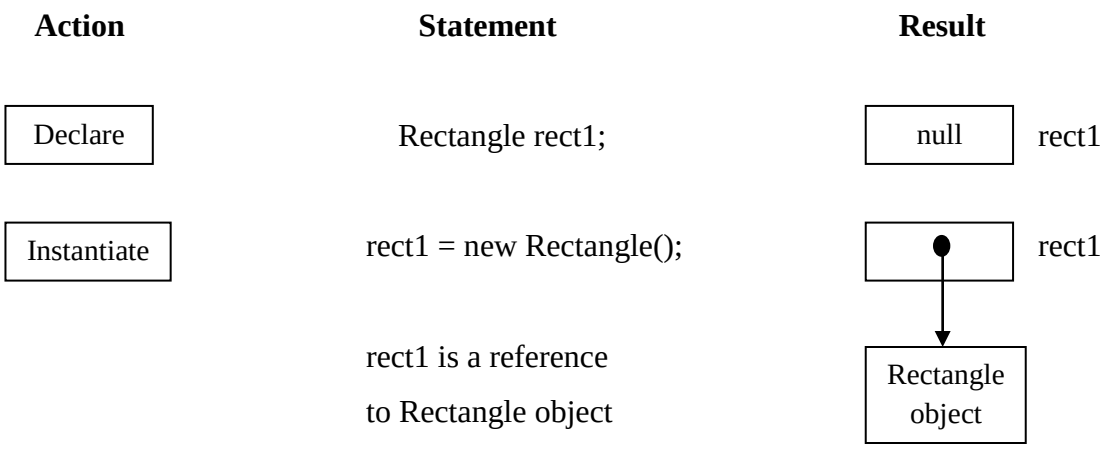


Figure: Creating objects

The statements can be combined into one as shown below:

```
Rectangle rect1 = new Rectangle();
```

6. What are the two ways of giving values to the variable?

The process of giving values to a variable is known as initialization. The ones that are not initialized are automatically set to zero.

- **By using an assignment statement**

A simple method of giving a value to a variable is through the assignment statement which is as follows:

variableName = value;

For example:

```
intialValue = 0;
finalValue = 100;
yes = ‘x’;
p = q = r = 0;
```

It is also possible to assign a value to a variable at the time of its declaration which is of the form:

type variableName = value;

Example:

```
int finalValue = 100;
char yes = ‘x’;
double total = 75.36;
```

The below statements are also valid:

```
float x, y, z;          //declares three float variables
int m = 5, n = 10;      //declares and initializes two int variables
int m, n = 10;          //declares m and n and initializes n
```

- **By using a read statement**

We may also give values to the variables interactively through the keyboard using readLine() method as illustrated in the program below:

```
import java.io.DataInputStream
class Reading
{
    public static void main(String args[])
    {
        DataInputStream in = new DataInputStream();
        int i = 0;
        float f = 0.0f;
        try
        {
```

```
                System.out.println("enter an integer number: ");
                i = Integer.parseInt(in.readLine());
                System.out.println("enter an floating point number: ");
                f = Float.valueOf(in.readLine()).floatValue();
            }
            Catch(Exception e) { }
            System.out.println("integer number = "+i);
            System.out.println("float number = "+f);
        }
    }
```

7. Define class and write down its syntax?

A class is a user defined data type with a template that serves to define its properties. It is also defined as a template from which objects are created. Once the class has been defined, we can create variables of that class using declarations that are similar to the basic type declarations.

```
class classname [extends superclassname]
{
    [ fileds declaration; ]
    [ methods declaration; ]
}
```

8. What is a constructor? Explain with an example.

All objects that are created must be given initial values. There are approaches to do this. One approach is to use the dot operator to access the instance variables and then assigns values to them individually. This will be difficult to initialize all the variables of all the objects. The second approach is to use the method to initialize each object individually like rect1.getData(15, 10);.

Constructor is a special type of method supported by java. A constructor enables an object to initialize itself when it is created. They have same name as the class name. They do not specify any return type, not even void. This is because they return the instance of the class itself.

Example:

```
class Rectangle
{
    int length, width;
    Rectangle( int x, int y)                    //defining constructor
```

```
        {
                length = x;
                width = y;
        }
        int rectArea()
        {
                return (length * width);
        }
}
class RectangleArea
{
        public static void main(String args[])
        {
                Rectangle rect1 = new rectangle(15, 10);       //calling constructor
                int area1 = rect1.rectArea();
                System.out.println("Area is = "+area1);
        }
}
```

9. Explain command line arguments.

Command line arguments are parameters that are supplied to the application program at the time of invoking it for execution. We can write java programs that can receive and use the arguments provided in the command line. We have the main() method:

public static void main(String args[])

here args is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements. We can simply access the array elements and use them in the program as we wish.

For example consider the command line:

java Test BASIC FORTRAN C++ JAVA

This command line contains four arguments. These are assigned to the array args as follows:

BASIC	→	args[0]
FORTRAN	→	args[1]
C++	→	args[2]
JAVA	→	args[3]

The individual elements of an array are accessed by using an index or subscript like args[i]. the value I denotes the position of the elements inside the array. For example, args[2] denotes the third element and represents C++. Here is an example demonstrating the use of command line arguments:

```
class Test
{
        public static void main(String args[])
        {
                int count = 0;
                String s;
                count = args.length;
                System.out.println("number of arguments: "+count);
                while (i<count)
                {       s = args[i];
                        i = i + 1;
                        System.out.println(s);
                }
        }
}
```

We compile and run this program with the command line as follows:

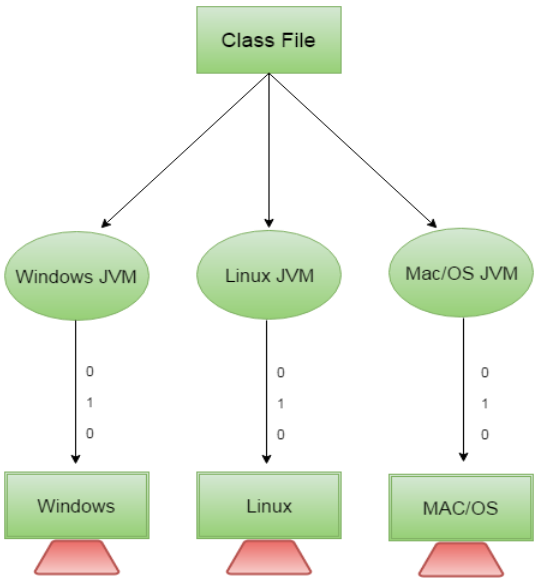
```
javac Test.java
java Test Java is Simple Object-Oriented Distributed Robust
```

Upon execution the command line arguments Simple, Object-Oriented and so on are passed to the program through the array args. The element args[0] conatins Simple, args[1] contains Object-Oriented, and so on. These elements are accessed using the loop variable i as an index. The index i is incremented using a while loop until all the arguments are accessed. The number of arguments is obtained by the statement

```
count = args.length;
```

10. Java is platform independent. Explain.

Platform independent means the execution of the program is not restricted by the type of OS environment provided, thereby, make it possible to process the program at any type of environment available. In java, when the source code is complied, it generates the .class file comprising the bytecodes. Bytecode is platform independent because it can be run on multiple platforms i.e, Write Once Run Anywhere. The figure demonstrates the platform independency of java.



11. What are static variables and static methods? Explain with example.

Class contains two sections. One declares variables and the other declares methods. These variables and methods are called instance variables and instance methods because every time the class is instantiated, a new copy of each of them is created. They are accessed by dot operator.

Suppose if we want to define a member that is common to all objects and accessed without using a particular object, such members are declared as static and are called as static members. The member belongs to the class as a whole rather than the objects created from the class, and therefore the static variables and static methods are often referred to as class variables and class methods. They can be defined as follows:

```
static int count;
static int max(int x, int y);
```

static variables: static variables are used when we want to have a common property to all instances of a class. The static variable gets memory only once in class area at the time of class loading.

Example:

```
class Student
{
        int rollno;
        String name;
        static String college = "xxxxx";
        Student(int r, int n)
        {
                rollno=r;
                name = n;
        }
        void display()
        {
                System.out.println(rollno+" "+name+" "+college);
        }
        public static void main(String args[])
        {
                Student s1 = new Student(111,"abc");
                Student s2 = new Student(222,"xyz");
                s1.display();
                s2.display();
        }
}
```

static methods: like static variables, static methods can be called without using the objects. These methods do not directly affect an instance of that class and are usually declared as class methods or static methods. Java class libraries contain a large number of class methods. For example, Math class of java library:

```
float x = Math.sqrt(25.0);
```

Example:

```
class mathOperation
{
    static float mul(float x, float y)
    {
        return x*y;
    }
    static float divide(float x, float y)
    {
        return x/y;
    }
}
class mathApplication
{
    public static void main(String args[])
    {
        float a = mathOperation.mul(4.0,5.0);
        float b = mathOperation.divide(a,2.0);
        System.out.println("b = "+b);
    }
}
```

Static methods have several restrictions:

- They can only call other static methods.
- They can only access static data
- They cannot refer to this or super in any way.

12. Explain the features of java.

- **Compiled and interpreted:** first java compiler translates source code into bytecode instructions. Bytecodes are not machine instructions and therefore in the second stage, java interpreter generates machine code that can be directly executed by the machine that is running the java program. Hence java is both compiled and interpreted language.
- **Platform independent and portable:** Platform independent means the execution of the program is not restricted by the type of OS environment provided, thereby, make it possible to process the program at any type of environment available. In java, when the source code is complied, it generates the .class file comprising the bytecodes. Bytecode is platform independent because it can be run on multiple platforms. Java ensures portability in two ways: first, bytecode can be implemented on any machine. And secondly, the size of the primitive data types is machine-independent.
- Robust and secure

- Distributed
- Simple, small and familiar
- Multithreaded and interactive
- High performance
- Dynamic and extensible
- Ease of development
- Scalability and performance
- Monitoring and manageability
- Desktop client

  (for explanation refer text book pg no. 12 to pg no. 14)

13. Describe the structure of java program.

A java program may contain many classes of which only one class defines a main method. Classes contain data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements. To write a java program, we first define classes and put them together. A java program may contain one or more sections as shown in the figure below:

Documentation Section	Suggested
Package Statement	Optional
Import Statements	Optional
Interface Statements	Optional
Class Definitions	Essential

Main Method Class
{
Main Method Definition
}
	Essential

Figure: general structure of java program

Documentation Section: this section comprises a set f comment lines giving the name of the program. Comments must explain why and what of classes and how of algorithms. This would help in maintain the program. Java also uses another style of comment /*…..*/ known as documentation comment.

Package Statement: this is the first statement allowed in java file. It declares a package name and informs the compiler that the classes defined here belong to this package. The package statement is optional. That is, our classes do not have to be part of package.

Example: package student;

Import Statements: the next statement may be a number of import statements. This is similar to #include statement in C.

Example: import student.test;

This statement instructs the interpreter to load the test class contained in the package student. Using import statements, we can have access to classes that are part of other named packages.

Interface Statements: an interface is like a class but includes a group of method declarations. This is an optional section and is used only when we wish to implement the multiple inheritance feature in the program.

Class Definitions: a java program may contain multiple class definitions. Classes are the primary and essential elements of a java program. These classes are used to map the objects of real-worlds problems.

Main Method Class: main method is the starting point of every java program. Hence this is an essential section. A simple java program may contain only this part. The main method creates objects of various classes and establishes communications between them. On reaching the end of the main, the program terminates and the control passes back to the operating system.

14. List the differences between c++ and java

Comparison Index	**C++**	**Java**
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.

Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are

		virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.

15. Explain methods of String class.

Method Call	**Task performed**	**Example**
s2 = s1.toLowerCase();	Converts the string s1 to all lowercase	String s1 = "MLACW"; s2 = s1.toLowerCase(); output: s2 = mlacw
s2 = s1.toUpperCase();	Converts the string s1 to all uppercase	String s1 = "java"; s2 = s1.toUpperCase(); output: s2 = JAVA
s2 = s1.replace('x', 'y');	Replace all appearances of x with y	s1 = "semester"; s2 = s1.replace('s', 'z'); s1 = "zemezter"
s2 = s1.trim();	Remove white spaces at the beginning and end of the string s1	String s1 = " MLACW "; s2 = s1.trim(); s2 = "MLACW"

s1.equals(s2);	Returns 'true' if s1 is equal to s2	String s1 = "bca";
		String s2 = "bca";
		if (s1.equals(s2))
		return true;
		else
		return false;

s1.length();	Gives the length of s1	String s1 = "iqac"; int len = s1.length(); len = 4

s1.charAt(n);	Gives the nth character of s1	String s1 = “iqac”; s1.charAt(2); character at position 2 in s1 is ‘a’
s1.compareTo(s2);	Returns negative if s1<s2, positive if s1>s2 and zero if s1 is equal to s2	▪ If suppose: String s1 = “iqac”; String s2 = “bca”; s1.compareTo(s2); then the value returned is 7 ▪ If suppose: String s1 = “bca”; String s2 = “iqac”; s1.compareTo(s2); then the value returned is -7 ▪ If suppose: String s1 = “iqac”; String s2 = “iqac”; s1.compareTo(s2); then the value returned is 0
s1.concat(s2);	Concatenates s1 and s2	String s1 = “iqac”; String s2 = “bca”; s1.compareTo(s2); the concatenated string will be “iqacbca”
s1.substring(n);	Gives substring starting from nth character	String s1 = “bca5thsemester” s1.substring(3); the substring will be “5thsemester”
s1.substring(n, m);	Gives substring starting from n^{th} character upto m^{th} character (not including m^{th})	String s1 = “bca5thsemester” s1.substring(6,9); the substring will be “sem”
s1.indexOf(‘x’)	Gives the position of the first occurrence of ‘x’ in the string s1	String s1 = “iqac”; s1.indexOf(‘c’); here ‘c’ occurs at position 3

s1.indexOf('x', n)	Gives the position of 'x' that occurs after n^{th} position in the string s1	String s1 = "bca5thsemester" s1.indexOf('s', 7) here 's' is occurring at 10^{th} position after position 7.
p.toString()	Creates a string representation of object p	

16. Explain JVM.

All language compilers translate source code into machine code for a specific computer. Java compiler also does the same thing. Java compiler produces an intermediate code known as bytecode for a machine that does not exist. The machine is called 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 figure below illustrates the process of compiling a java program into bytecode which is also referred to as virtual machine code.

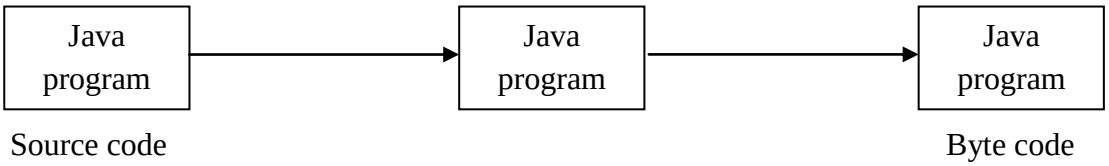


Figure: process of compilation

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 known as shown in the figure below. Note that interpreter is different for different machines.

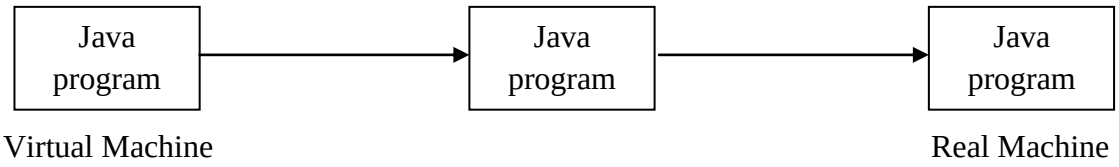


Figure: process of converting bytecode into machine code

Here is a diagram that illustrates how java works on a typical computer. The java object framework acts as an intermediary between the user programs and the virtual machine which in turn acts as the intermediary between the operating system and the java object framework.

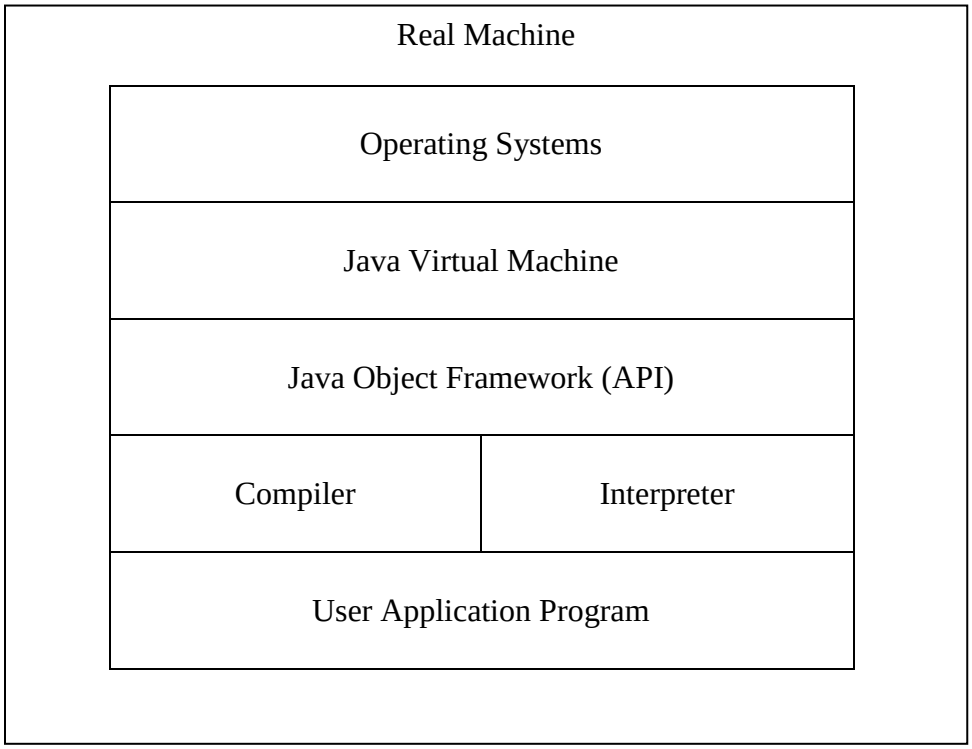


Figure: layers of interactions for java programs

17. Explain method overriding with example.

There may be occasions when we want an object to respond to the same method but have different behavior when that method is called. In such situations we should override the method defined in the superclass. This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass. Then when the method is called, the method defined in the subclass is invoked and executed instead of one in the superclass. This is known as overrinding.

Example-1:

```
class Superclass
{
        int x;
        Superclass(int x)
        {
                this.x = x;
        }
        void display()
        {
                System.out.println("Super x = "+x);
        }
}
class Subclass extends Superclass
{
```

```
        int y;
        Subclass(int x, int y)
        {
                super(x);
                this.y = y;
        }
        void display()
        {
        System.out.println("Super x = "+x);
        System.out.println("Sub y = "+y);
        }
}
class OverrideTest
{
        public static void main(String args[])
        {
                Subclass s1 = new Subclass(100, 200);
                s1.display();
        }
}
```

Example-2:

```
class Bank
{
        int getRateOfInterest()
        {
                return 0;
        }
}
class SBI extends Bank
{
        int getRateOfInterest()
        {
                return 8;
        }
}
class ICICI extends Bank
{
        int getRateOfInterest()
        {
                return 7;
        }
}
class AXIS extends Bank
{
```

```
        int getRateOfInterest()
        {
                return 9;
        }
}
class TestOverride
{
        public static void main(String args[])
        {
                SBI s = new SBI();
                ICICI i = new ICICI();
                AXIS a = new AXIS();
                System.out.println("SBI Rate of Interest: "+s. getRateOfInterest());
                System.out.println("ICICI Rate of Interest: "+i. getRateOfInterest());
                System.out.println("AXIS Rate of Interest: "+a. getRateOfInterest());
        }
}
```

18. Explain method overloading with example.

In java, it is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism.

To create an overloaded method, all we have to do is to provide several different method definitions in the class, all with the same name, but with different parameter lists. The difference may either be in the number or type of arguments. That is, each parameter list should be unique. Note that the method's return type does not play any role in this. Here is an example of creating an overloaded method.

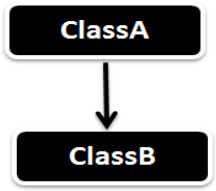
Example:

```
class Area1
{
        void AreaOfCircle(double r)
        {
                double area=3.14*r*r;
                System.out.println("Area of circle is :"+area);
        }
        void AreaOfSquare(double l)
        {
                double area = l*l;
                System.out.println("Area of square is :"+area);
```

```
    }
    void AreaOfRectangle(double l,double b)
    {
        double area = l*b;
        System.out.println("Area of rectangle is :"+area);
    }
    void AreaOfTriangle(double b,double h)
    {
        double area=0.5*b*h;
        System.out.println("Area of triangle is :"+area);
    }
    public static void main(String args[])
    {
        Area1 a1 = new Area1();
        a1. AreaOfCircle(5);
        a1. AreaOfSquare(3.5);
        a1. AreaOfRectangle(7,17);
        a1. AreaOfTriangle (4,13);
    }
}
```

19. Explain single inheritance in java with an example program. How it is different from multilevel inheritance?

Single inheritance: Single Inheritance enables a single derived class (Sub class) to inherit properties and behavior from a single parent class (Super class). The diagram below represents single inheritance in java where Class B extends only one class Class A. Here Class B will be the Sub class and Class A will be one and only Super class.

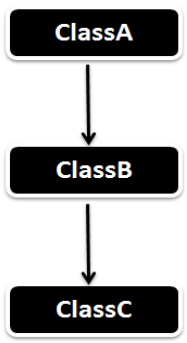


Example:

```
class Room
{
    int length;
    int breadth;
    Room(int x, int y)
    {
        length = x;
        breadth = y;
    }
```

```
        int area()
        {
                return (length * breadth);
        }
}
class BedRoom extends Room
{
        int height;
        BedRoom(int x, int y, int z)
        {
                super(x, y);
                height = z;
        }
        int volume()
        {
                return (length * breadth * height);
        }
}
Class TestInheritance
{
        public static void main(String args[])
        {
                BedRoom r1 = new BedRoom(14, 12, 10);
                int a1 = r1.area();
                int vol = r1.volume();
                System.out.println("Area = "+a1);
                System.out.println("Volume = "+vol);
        }
}
```

Multilevel inheritance: In Java Multilevel Inheritance sub class will be inheriting a parent class and as well as the sub class act as the parent class to other class. Lets now look into the below flow diagram, we can see Class B inherits the property of Class A and again Class B act as a parent for Class C. In Short Class A parent for Class B and Class B parent for Class C.



The Class C inherits the members of Class B directly as it is explicitly derived from it, whereas the members of Class A are inherited indirectly into Class C (via Class B). So the Class B acts as a direct superclass and Class A acts as a indirect superclass for Class C.

Example:

```
class Animal
{
        void eat()
        {       System.out.println("eating…..");       }
}
class Dog extends Animal
{
        void bark()
        {       System.out.println("baring…..");       }
}
class BabyDog extends Animal
{
        void weep()
        {       System.out.println("weeping…..");              }
}
class TestInheritance
{
        public static void main(String args[])
        {
                BabyDog d = new BabyDog();
                d.weep();
                d.bark();
                d.eat();
        }
}
```

20. Compare:

i. while and do while

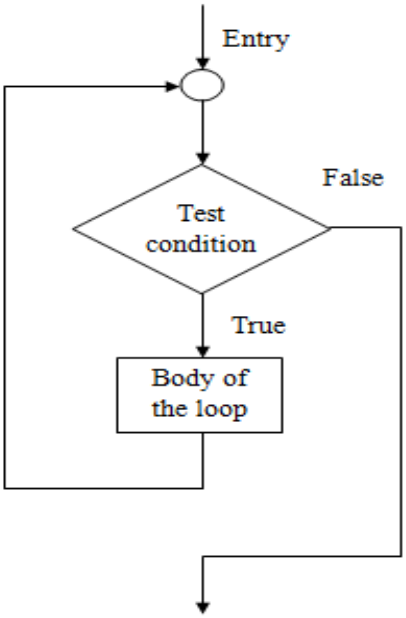
while	**do while**
It is an entry controlled loop statement. The test condition is evaluated and if condition is true, then the body is executed. The process of repeated execution of the body of the loop continues until the test condition becomes false. Then the control	In do while, the program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This

is transferred out of the loop.

Syntax:

```
Initialization;
while (test condition)
{
        Body of the loop
}
```

Flowchart:



Example:

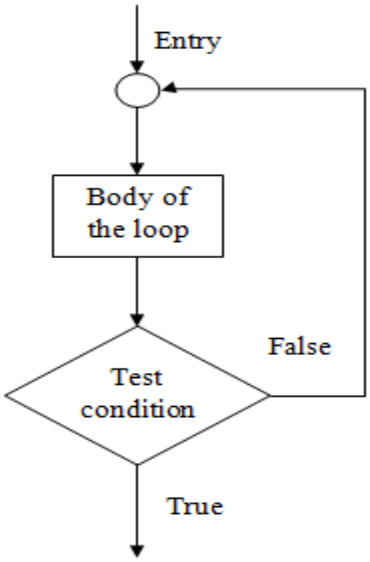
```
public class WhileExample
{
        public static void main(string args[])
        {
                int i = 1;
                while(i <=10)
                {
                        System.out.println(i);
                        i++;
                }
        }
}
```

process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

Syntax:

```
Initialization;
do
{
        Body of the loop
}
while (test condition);
```

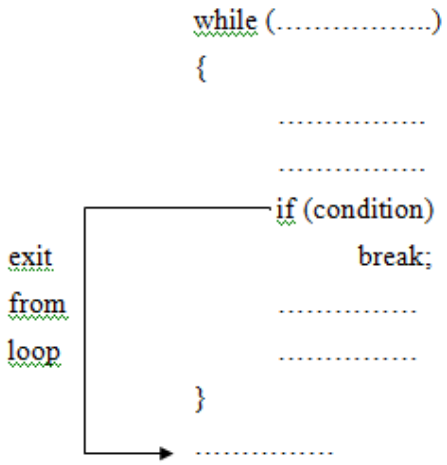
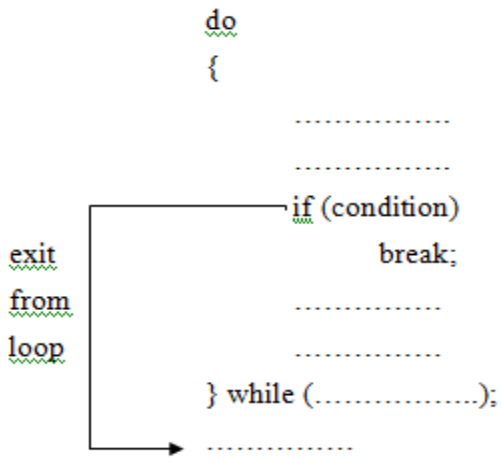
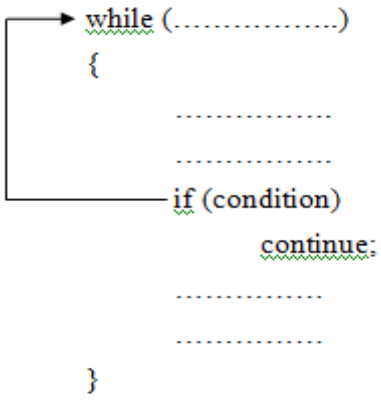
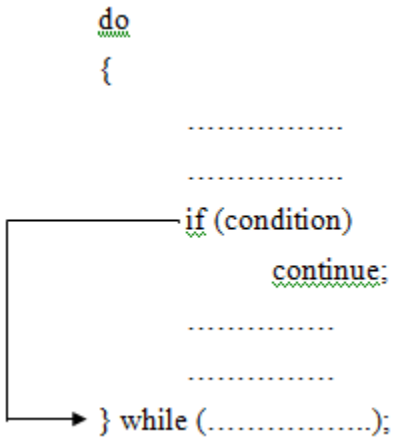
Flowchart:



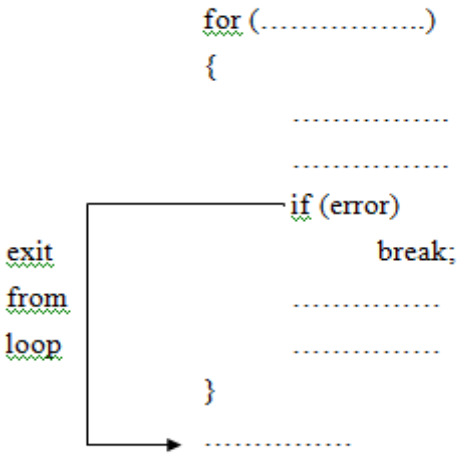
Example:

```
public class DoWhileExample
{
        public static void main(string args[])
        {
                int i = 1;
                do
                {
                        System.out.println(i);
                        i++;
                } while(i <=10);
        }
}
```

ii. break and continue

break	continue
Break is used to **terminate** the execution of the loop. It **breaks** the iteration. When this statement is executed, control will come out from the loop and executes the statement immediate after loop. Break is used with loops as well as switch case. **Syntax:** **(i)**  **(ii)** 	Continue is not used to terminate the execution of loop. It **skips** the iteration. When this statement is executed, it will not come out of the loop but moves/jumps to the next iteration of loop. Continue is only used in loops, it is not used in switch case. **Syntax:** **(i)**  **(ii)** 

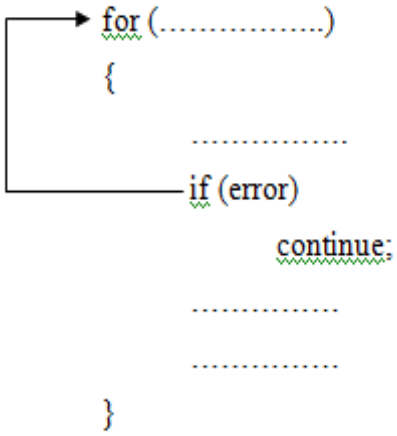
(iii)



Example:

```
public class BreakExample
{
        public static void main(string args[])
        {
                for(int i = 1; i<=10; i++)
            {
                if(i == 5)
                {
                        break;
                        i++;
                }
                System.out.println(i);
            }
        }
}
```

(iii)



Example:

```
public class ContinueExample
{
        public static void main(string args[])
        {
                for(int i = 1; i<=10; i++)
            {
                if(i == 5)
                {
                        continue;
                        i++;
                }
                System.out.println(i);
            }
        }
}
```