JAVA Programming

Dr. Abu Nowshed Chy
Assistant Professor, Dept. of CSE, CU

1. What is Object Oriented Programming (OOP)?
Ans:

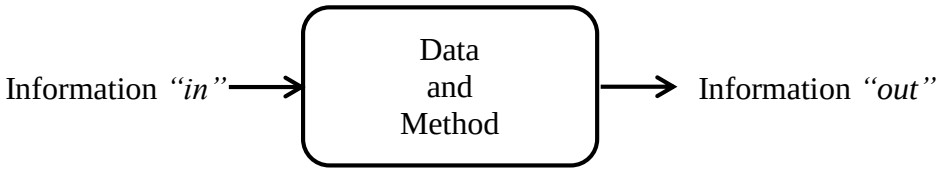
Object Oriented Programming (OOP):
Object Oriented Programming (OOP) is an approach that provides a way of modularizing programs by creating partitioned memory areas for both data and functions that can be used as templates for creating copies of such modules on demand.

2. Describe the three important features of OOP.
Or
What are some features that you would expect to find in an OOP language? Are these features present in Java? Briefly explain to illustrate your answer.
Ans:

The three important features of OOP are-

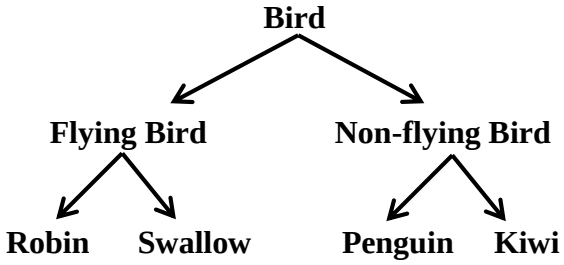
- Encapsulation.
- Inheritance.
- Polymorphism.

Encapsulation:
The wrapping up of data and methods into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it. This insulation of the data from direct access by the program is called data hiding.

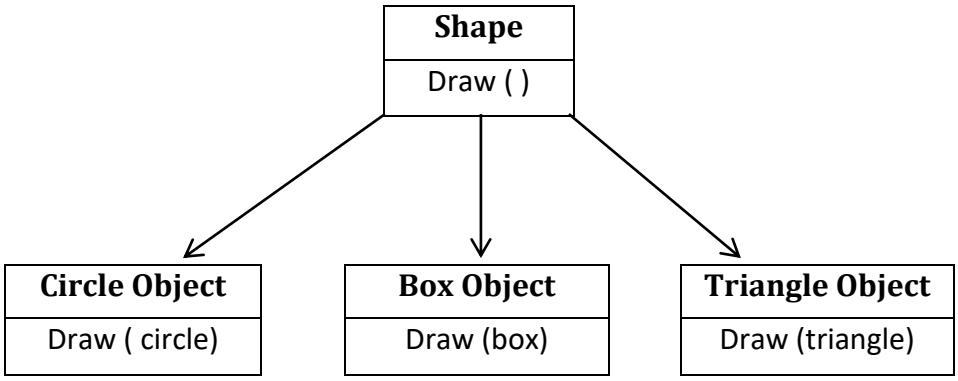


Encapsulation – Objects as *"black boxes"*

Inheritance:
Inheritance is the process by which objects of one class acquire the properties of objects of another class. This is important because it supports the concept of hierarchical classification.



Polymorphism:
Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behaviour in different instances. The behaviour depends upon the types of data used in the operation.



3. List five benefits and five application areas of Object-Oriented Programming.
Ans:

The principal advantages / benefits of OOP:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple objects to coexist without any interference.
- Object-Oriented systems can be easily upgraded from small to large systems.
- It is easy to partition the work in a project based on objects.
- Software complexity can be easily managed.

Application areas of OOP:

- Real-time Systems.
- Object-Oriented databases.
- Artificial Intelligence and Expert Systems.
- Neural Networks and Parallel Programming.
- Simulation and Modeling.
- Decision Support and Office Automation Systems.

4. What is Java? What are the benefits of Java?

Or

Briefly describe about the Java features.

Ans:

Java:

- Java is a general-purpose, Object Oriented, a platform-independent programming language developed by Sun Microsystems of the USA in 1991.
- Java programs consist of one or more classes.
- Java is platform-independent. Compiled Java programs can run on many different machines.

Benefits of Java / list of Java Buzzwords / Java Features:

- Compiled and Interpreted.
- Platform-Independent and Portable.
- Object-Oriented.
- Robust and Secure.
- Distributed.
- Familiar, Simple, and Small.
- Multithreaded and Interactive.
- High Performance.
- Dynamic and Extensible.

5. What are the differences between Java and C?

Ans:

Differences between Java and C:

- Java does not include the C unique statements keywords **goto, sizeof,** and **typedef.**
- Java does not contain the data types **struct, union,** and **enum**.
- Java does not support an explicit pointer type.
- Java does not have a pre-processor.
- Java does not support any mechanism for defining variable arguments to functions.
- Java adds many features required for Object-Oriented Programming.

6. What are the differences between Java and C++?

Ans:

Differences between Java and C++:

- Java does not support multiple inheritance of classes. This is accomplished using a new feature called “interface”.
- Java does not use pointers.
- Java does not support operator overloading.
- Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
- Java has replaced the destructor function with a *finalize( )* function.

7. What is Java Development Kit? What are the uses of `javac, java,` and `jdb`?
Ans:

Java environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as *Java Development Kit (JDK)* and the classes and methods are part of the *Java Standard Library (JSL)*, also known as the *Application Programming Interface (API).*

Java Development Kit (JDK):
The Java Development Kit (JDK) comes with a collection of tools that are used for developing and running Java programs. They include:

- appletviewer (for viewing Java applets)
- javac (Java compiler)
- java (Java interpreter)
- javap (Java disassembler)
- javah (for header files)
- javadoc (for creating HTML dcouments)
- jdb (Java debugger)

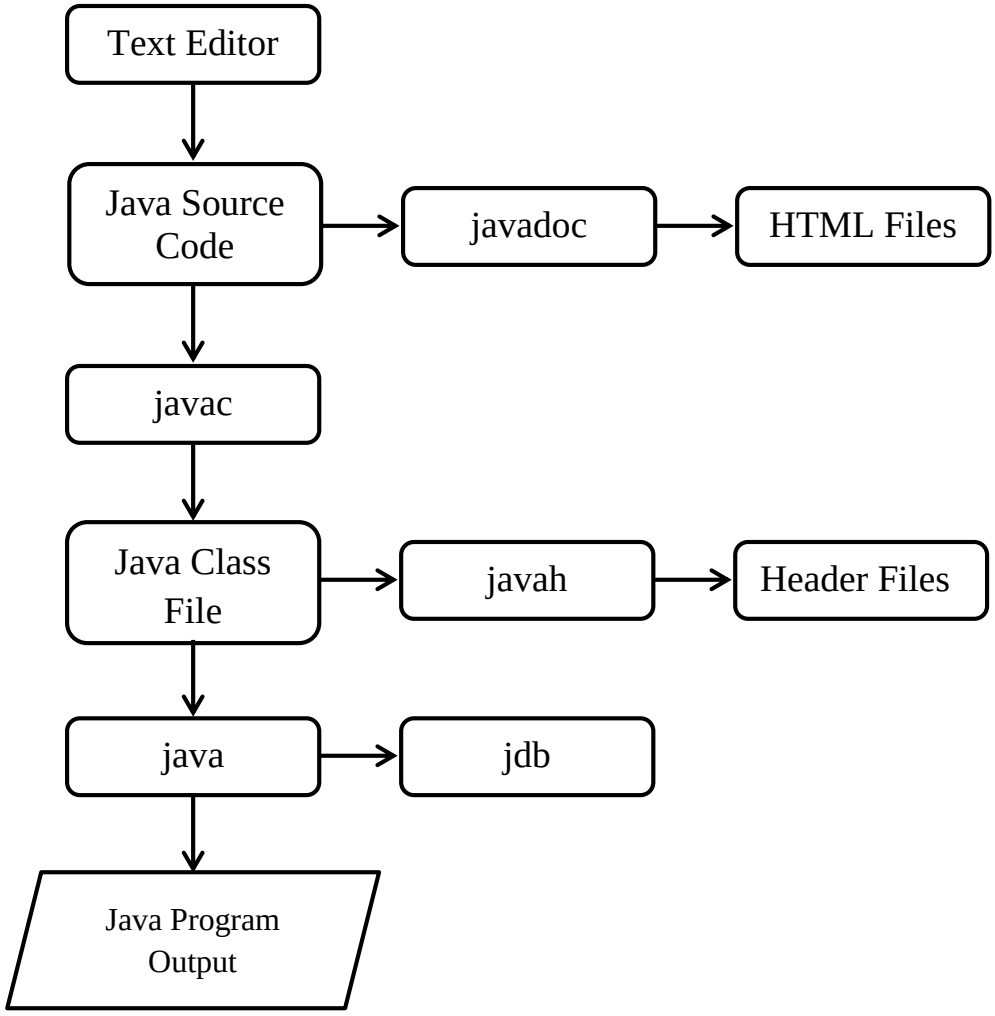
Tool	Description / Uses
`appletviewer`	Enables us to run Java applets.
`java`	Java interpreter, which runs applets and applications by reading and interpreting byte-code files.
`javac`	The Java compiler, which translates Java source-code to byte-code files that the interpreter can understand.
`javadoc`	Creates HTML format documentation from Java source-code files.
`javah`	Produces header files for use with native methods.
`javap`	Java disassembler, which enables us to convert byte-code files into a program description.
`jdb`	Java debugger, which helps us to find errors in our programs.

8. What do you mean by BYTECODE?
Ans:

Bytecode is an intermediate form of java programs. Bytecode consists of an optimized set of instructions that are not specific to the processor. We get Bytecode after compiling the java program using a compiler called javac. The Bytecode is to be executed by the java runtime environment which is called as Java Virtual Machine (JVM).

9. Show the process of building and running Java application.
Ans:

The process of building and running Java application programs illustrated in below figure:



To create a Java program we need to create a source code file using a text editor. The source code is then compiled using the Java compiler `javac` and executed using the Java interpreter `java`. The Java debugger `jdb` is used to find errors.

10. What is the main method in Java? Explain all keywords associated with the main method.

Or

The line: `public static void main (String args[ ])` defines a method named main. Here, what do public, static, and void means?

Ans:

The main method in Java: **`public static void`** `main (String args[])`

`Public`	The keyword `public` is an access specifier that declares the `main` method as unprotected and therefore making it accessible to all other classes.
`Static`	The keyword `static` declares the `main` method as one that belongs to the entire class and not a part of any objects of the class. The `main` must always be declared as `static` since the interpreter uses this method before any objects are created.
`Void`	The type modifier `void` states that the `main` method does not return any value.
`String args[]`	`String args[ ]` declares a parameter named `args`, which contains an array of objects of the class type `String`.

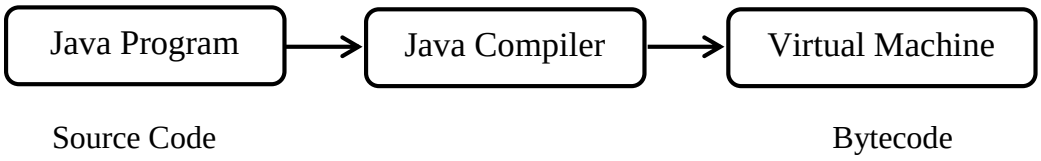
11. How does Java achieve architecture neutrality?

Or

What is Java Virtual Machine? Do you consider Java Interpreter as the implementation of Java Virtual Machine?

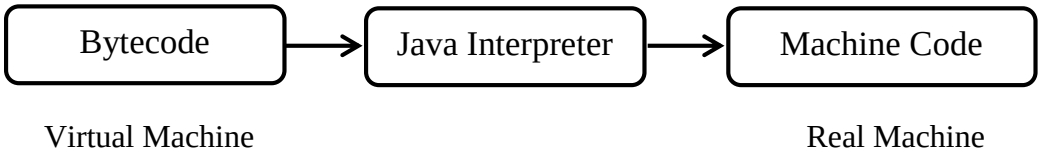
Ans:

The Java compiler produces an intermediate code known as bytecode 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 the major functions of a real computer.



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 as shown in the below figure.



Process of converting bytecode into machine code

12. Define: Command Line Arguments.
Ans:

Command Line Arguments:
Command-line arguments are parameters that are supplied to the application program at the time of invoking it for execution. The parameters must follow the program name on the command line e.g.,

```
program-name  parameter 1  parameter 2 …… parameter n
```

13. What is the scope of a variable?
Or
Discuss the scope of different types of variables in Java.
Ans:

Java variables are actually classified into three kinds:

- instance variables.
- class variables.
- local variables.

Instance and class variables are declared inside a class. Instance variables are created when the objects are instantiated and therefore they are associated with the objects. On the other hand, class variables are global to a class and belong to the entire set of objects that the class creates.

Variables declared and used inside methods are called local variables. They are called so because they are not available for use outside the method definition. These variables are visible to the program only from the beginning of its program block to the end of the program block.

14. Define Data Type Casting. Describe Java's way of data type casting.
Ans:

Data Type Casting:
The process of converting one data type to another is called casting. The syntax is:

```
type variable1 = (type) varibale2;
```

Casting is often necessary when a method returns a type different than the one we require. Casting into a smaller type may result in a loss of data. Casting a floating-point value to an integer will result in a loss of a fractional part.

Java's Automatic Conversion:
When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

- The two types are compatible.
- The destination is larger than the source type.

The process of assigning a smaller type to a larger one is known as widening or promotion conversion and that of assigning a larger type to a smaller one is known as narrowing conversion. Narrowing conversion may result in the loss of information.

15. What are the use of *instanceof* operator and *dot* operator?
Ans:

***instanceof* Operator:**
The *instanceof* operator is an object reference operator and returns `true` if the object on the left-hand side is an instance of the class given on the right side. This operator allows us to determine whether the object belongs to a particular class or not.

Example:

```
person instanceof  student
```

is `true` if the object `person` belongs to the class `student`; otherwise, it is `false`.

***dot* Operator:**
The dot (**.**) operator is used to access the instance variables and methods of class objects.

Examples:

```
person1.age          // Reference to the variable age
person1.salary( )  //  Reference to the method salary( )
```

It is also used to access classes and sub-packages from a package.

16. What is the relationship between class and object in Java? Show with example.
Or
"A class is a logical construct while an object has physical reality"- why?
Ans:

When we create a class, we are creating a new data type. Then we can use this type to create objects of that type. The data or variables defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called a member of the class. That is, a class is a logical framework that defines the relationship between its members. When we declare an object of a class, we are creating an instance of that class. Thus, a class is a logical construct while an object has physical reality (i.e.; an object occupies spaces in memory).

Class Example:

```
class Box{
     float width, height, depth;
     float volume(){
          return width*height*depth;
     }
}
```

Object Creation of that class:

```
Box mybox = new Box( );
```

17. Show how objects are created in Java?

Or

Discuss the role of *new* operator.

Ans:

Obtaining objects of a class is a two-step process. *First,* we must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can refer to an object. *Second,* we must acquire an actual, physical copy of the object and assign it to that variable. We can do it by using the `new` operator. It dynamically allocates memory for an object during execution and returns a reference to it. This reference is the address in memory for an object allocated by `new`. This reference is then stored in the variable.

```
Box mybox;                // declare
mybox = new Box( );   // instantiate
```

Both statements can be combined into one as shown below:

```
Box mybox = new Box( );
```

18. What is a constructor? Write down the properties of the constructor in Java?

Ans:

Constructor:

A constructor initialized an object immediately upon creation. It has the following properties:

- It has the same name as the class itself.
- Syntactically, it is similar to a method.
- No return type (not even void) is required. Because it returns the instance of the class itself.
- Once defined, the constructor is automatically called after the object is created, before the new operator completes.

19. Explain why constructors have no return type, not even void in one point?

Ans:

A Constructor has no return type, not even void, because, it returns the instance of the class itself.

20. What are the differences between a method and a constructor?

Ans:

A constructor is a member function of a class that is used to create objects of that class. It has the same name as the class itself, has no return type, and is invoked using the `new` operator.

A method is an ordinary member function of a class. It has its own name, a return type (which may be void), and is invoked using the `dot` operator.

21. How does constructor can be overloaded in a Java class? Show with example.
Ans:

In a simple constructor like –

```
class Box {
     float width, height, depth;

     Box(float w, float h, float d) {
          width = w;
          height = h;
          depth = d;
     }

     float volume() {
          return width * height * depth;
     }
}
```

Here, the Box constructor requires three parameters. Therefore, Box my1=new Box( ); is invalid.

But it may happen that, one may want just a Box object but does not care about initial values or may initialize a cube object by specifying the same value for all three dimensions, the Box( ) should be overloaded.

```
class Box {
     float width, height, depth;

     Box(float w, float h, float d) {
          width = w;      // all three dimension specified
          height = h;
          depth = d;
     }

     Box( ) {
          width = -1;     // used -1 to uninitialized Box
          height = -1;
          depth = -1;
     }

     Box(float len) {
          width = height = depth = len; // same value for all
     }

     float volume() {
          return width * height * depth;
     }
}
```

```
public class Box_Final {
     public static void main(String[] args) {
          Box my1 = new Box(15, 64, 32);
          Box my2 = new Box( );
          Box my3 = new Box(7);

          float area1 = my1.volume();
          System.out.println(area1);
          float area2 = my2.volume();
          System.out.println(area2);
          float area3 = my3.volume();
          System.out.println(area3);

     }
}
```

22. How can you access the constructor of the super class? Depict with an example.
Ans:

A subclass can call a constructor defined by it superclass by use of the following form of super:

```
super(parameter-list);
```

Here, the `parameter-list` specifies any parameters needed by the constructor in the superclass. `super()` must always be the first statement executed inside a subclass constructor.

For example:

```
class Box {
     float width, height, depth;

     Box(float w, float h, float d) {
          width = w;
          height = h;
          depth = d;
     }
}


class BoxWeight extends Box {
     float weight;

     BoxWeight(float w, float h, float d, float m) {
          super(w,h,d);
          weight = m;
     }
}
```

23. Show the use of a parameterized constructor with a Java program.
Ans:

To set different dimensions for different objects, parameterized constructors may be used.

```
class Box {
     float width, height, depth;

     Box(float w, float h, float d) {
          width = w;
          height = h;
          depth = d;
     }

     float volume() {
          return width * height * depth;
     }
}

public class Box_Final {
     public static void main(String[] args) {
          Box my1 = new Box(15, 64, 32);
          Box my2 = new Box(10, 54, 24);

          float area1 = my1.volume();
          System.out.println(area1);
          float area2 = my2.volume();
          System.out.println(area2);
     }
}
```

24. What are the differences between methods overloading and methods overriding?
Ans:

Differences between methods overloading and methods overriding:

- **Method Overloading:**
  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.

  Method overriding:
  If we want an object to respond to the same method but have different behaviour then we can do it 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 the one in the superclass. This is known as method overriding.

- In method overloading, the number of the overloaded method is not fixed. You can create any number of overloaded methods. But in method overriding, the method of subclass can override the method of superclass exactly once.
- The argument lists of overloaded methods are different from one another but they must be identical in the overriding method.
- There is no restriction to the return type of the overloaded methods. But the return type of the overriding method must be identical to the return type of the overridden method.

25. What is method overloading? Why it is so important in Object Oriented Programming? Explain with a Java Program.

Ans:

Method Overloading:

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.

It is so important in Object-Oriented Programming like Java because with this feature Java implements polymorphism. 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.

Example:

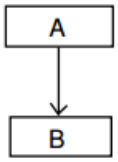
```
class TestDemo{
     void test(){
          System.out.println("No Parameters");
     }

     void test(int a){
          System.out.println("a = " +a);
     }

     void test(int a, double b){
          System.out.println("a="+ a +"  b="+b);
     }
}

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

          TestDemo my1=new TestDemo();

          my1.test();
          my1.test(5);
          my1.test(5,8.5);
     }
}
```

26. Define: Inheritance. Explain different types of inheritance with examples in brief.
Ans:

Inheritance:
The mechanism of deriving a new class from an existing class is called inheritance. The existing class is known as the superclass and the new class is known as the subclass.
There are different types of inheritance:

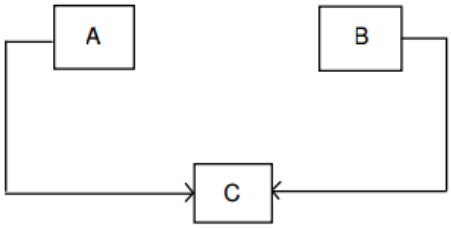
1. Single inheritance (Only one superclass)
2. Multiple inheritance (Several superclass)
3. Hierarchical inheritance (One superclass, many subclass)
4. Multi-level inheritance (Derived from a derived class)

Single inheritance:
A class derived from only one superclass is called single inheritance.



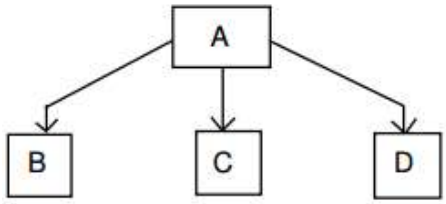

```
class A {.........}
class B extends A {.........}
```

Multiple inheritance:
A class derived from several superclasses is called multiple inheritance.



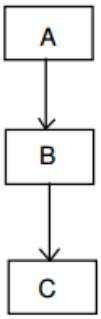

```
interface A {.........}
interface B {.........}
class C implements A, B {.........}
```

Hierarchical inheritance:
In hierarchical inheritance, one superclass may be inherited by more than one subclass.




```
class A {.........}
class B extends A {.........}
class C extends A {.........}
class D extends A {.........}
```

Multi-level inheritance:
The mechanism of deriving a class from another derived class is called multilevel inheritance.




```
class A {.........}
class B extends A {.........}
class C extends B {.........}
```

27. What are the conditions to use the keyword super? Give example.
Ans:

Constructor definitions in both superclass and subclass are the same. To avoid duplicate definitions, the **`super`** keyword can be used. **`super`** can be used to call a superclass constructor and to access a member of the superclass that has been hidden by a member of a subclass.

The keyword **`super`** is used subject to the following conditions:

- **`super`** may only be used within a subclass constructor method.
- The call to the superclass constructor must appear as the first statement within the subclass constructor.
- The parameters in the **`super`** call must match the order and type of the instance variable declared in the superclass.

28. What is the difference between parameter and argument? When `static` is used?
Ans:

The parameter is a variable defined by a method that receives a value when it is called. On the other hand, the argument is the value that is passed to a method when it is invoked.

Static:
If we want to define a member that is common to all objects and accessed without using a particular object, that is, the member belongs to the class as a whole rather than the objects created from the class, then `static` can be used. When a member is declared `static` it can be accessed before any objects are created. `static` methods have the following restrictions:

- Can call other `static` methods.
- Must access `static` data.
- Can't refer to "`this`" or "`super`"

29. Define Method Overriding. Discuss with a Java Program.
Ans:

Method Overriding:
If we want an object to respond to the same method but have different behaviour then we can do it 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 the one in the superclass. This is known as method overriding.

Example:

```
class A{
     int i,j;

     A(int a, int b){
          i=a;
          j=b;
     }

     void show(){
          System.out.println("i="+ i +" j="+j);
     }
}

class B extends A{    // inheriting A
     int k;

     B(int a, int b, int c){
          super(a,b);     // pass values to superclass
          k=c;
     }

     void show(){
          //super.show();
          System.out.println("k= "+k);
     }
}

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

          B my1=new B(1,2,3);
          my1.show();
     }
}
```

30. How does methods and variables of the super class can be accessed by the subclass? Explain with an example.

Or

With an example shown, how inheritance can be used in a Java program?

Ans:

If we wish to access the superclass version of an overridden function, we can do it by using **super**.

```
class A{
      int i,j;

      A(int a, int b){
            i=a;
            j=b;
      }

      void show(){
            System.out.println("i="+ i +" j="+j);
      }
}

class B extends A{    // inheriting A
      int k;

      B(int a, int b, int c){
            super(a,b);     // pass values to superclass
            k=c;
      }

      void show(){
            super.show();
            System.out.println("k= "+k);
      }
}

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

            B my1=new B(1,2,3);
            my1.show();
      }
}
```

In class B, the superclass version of `show()` is invoked within the subclass version.

31. How can you prevent method overriding?
Ans:

Making a variable and method **final** means that the functionality of that variable and method cannot be altered in any way. We can make a method **final** to avoid override.

```
class a{
     final void show( ){
          System.out.println("This is final");
     }
}

class b extends a{
 void show( ){    // error
          System.out.println("Error");
     }
}
```

32. How can we prevent a class being further sub-classed for security reasons?
Or
Write short notes on: 1) final keyword
Ans:

We can prevent a class from being further sub-classed i.e.; avoiding inheritance by defining the class as **final**. This is achieved in Java using the keyword **final** as follows:

```
final class Aclass{.........}
final class Bclass extends SomeClass{.........}
```

Any attempt to inherit these classes will cause an error and the compiler will not allow it. Declaring a class **final** prevents any unwanted extensions to the class.

33. Describe the garbage collection of Java.
Or
What is the purpose of garbage collection in Java, and when is it used?
Ans:

The purpose of garbage collection in Java is to identify and discard objects that are no longer needed by a program so that their resources can be reclaimed and reused. A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used. Garbage collection only occurs sporadically during the execution of your program.

34. Write the advantage of using `finalize( )` in a class written in Java?
Ans:

We know that Java run-time is an automatic garbage collecting system. It automatically frees up the memory resources used by the objects. But still, objects may hold some non-object resources such as file descriptors or window system fonts. The garbage collector cannot free these resources. To free these resources, we can use a **`finalizer`** method. The **`finalizer`** method is simply **`finalize( ).`** Java calls that method whenever it is about to reclaim the space for that object.

35. How can we make overriding compulsory?
Or
Write short notes on: i) abstract method
Or
What are the usefulness of abstract classes in Java?
Ans:

In certain situations, if the superclass defines the method in only structural form but the implementation needs to be done by its subclass, then the superclass should be declared as **`abstract`**. While using **`abstract`** classes, we must satisfy the following conditions:

- **`abstract`** classes cannot be instantiated with the `new` operator.
- **`abstract`** constructors and **`abstract static`** methods are not possible.
- The **`abstract`** methods of an **`abstract`** class must be defined in its subclass.

Example:

```
abstract class A{
     abstract void call();

     void call2(){
          System.out.println("OK");
     }
}

class B extends A{
     void call(){
          System.out.println("Implemented"); }
}

public class Final {
     public static void main(String[] args) {
          B my1=new B();

          my1.call();
          my1.call2();
     }
}
```

36. What are the differences between abstract class and final class?
Ans:

- An **`abstract`** class is a class that must be sub-classed thus making overriding compulsory and a **`final`** class is a class that can't be sub-classed thus prevent overriding .
- **`abstract`** class can't create the instance or objects whereas the **`final`** class can create the instance or objects.

- Example:

Final Class:

```
final class Aclass{……}
final class Bclass extends Someclass{……}
```

Abstract Class:

```
abstract class A{
   abstract void call();
   ... ... ... ... ...
}
```

37. What are Java Access Specifiers? Explain their functionalities.
Ans:

Java's access specifiers are public, private, and protected.

Public:
A class that is declared as the public has a global scope and an instance of this class can be created from anywhere within or outside of the class i.e. by other classes in the same packages as well as classes in other packages. Only one non-inner class in any file can be defined with the public keyword. The public specifier can be used for variables, methods, and classes.

Protected:
A protected field is visible within the same class, same package, and in subclasses in other packages. This can be used for methods, variables, and inner classes.

Private Access:
Private fields enjoy the highest level of protection. They are accessible only within their own class. They cannot be inherited by subclasses and therefore not accessible in subclasses.

38. How does a super class object can reference a subclass object in Java? Explain with a Java Program.

Ans:

A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass. This aspect of inheritance is quite useful in a variety of situations. Let us consider the following example:

```
class Box{
      float width, height, depth;
      Box(){
            width=-1; //  use -1 to indicate an uninitialized box
            height=-1;
            depth=-1;
      }
      float volume(){
            return width*height*depth;
      }
}

class BoxWeight extends Box {
      float weight;

      BoxWeight(float w, float h, float d, float m){
            width=w;
            height=h;
            depth=d;
            weight=m;
      }
}

public class Box_Final {
      public static void main(String[] args) {
            BoxWeight my1=new BoxWeight(3,5,7,8);
            Box my2 = new Box();

            float volume1=my1.volume();
            System.out.println(volume1);
            System.out.println(my1.weight);

            my2=my1; // assigning my1 reference to my2 reference.
            volume1=my2.volume();
            System.out.println(volume1);
            System.out.println(my2.weight); // Error, my2 does not define weight.
      }
}
```

39. How can you declare and initiate an array in Java?
Ans:

In Java, the creation of an array involves three steps:

1. Declare the array.
2. Create memory locations.
3. Put values into the memory locations.

Declaration of Arrays:
Arrays in Java may be declared in two forms:

Form1:

```
type arrayname[ ];
```

Form2:

```
type[ ] arrayname;
```

Examples:

```
int number[ ];
int[ ] counter;
```

Creation of Arrays:
Java allows us to create arrays using new operator only, as shown below:

```
arrayname = new type[size];
```

Examples:

```
number = new int[5];
```

Combining the two steps- declaration and creation-into one we get:

```
int number = new int[5];
```

Initialization of Arrays:
The final step is to put values into the array created. This process is known as initialization.

```
arrayname[subscript] = value;
```

Examples:

```
number[0] = 35;
```

We can also initialize arrays automatically in the same way as the ordinary variables when they are declared, as shown below:

```
type arrayname[ ] = {list of values};
```

Examples:

```
int number[ ] = {35, 40, 50};
```

40. What is a vector? Write down the advantages of vectors over arrays. What is its major constraint?

Ans:

Vector:

A vector is a generic dynamic array that can hold objects of any type and any number. The objects do not have to be homogeneous. Vectors are created like arrays easily:

```
Vector intVect = new Vector( ); // declaring without size
Vector list = new Vector(3);  // declaring with size
```

Advantages:

Vectors possess a number of advantages over arrays:

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

Constraint:

A major constraint in using vectors is that we cannot directly store simple data types in a vector; we can only store objects. Therefore, we need to convert simple types to objects. This can be done using the wrapper class.

41. Describe wrapper class. Write the importance of that.

Or

What are the applications of wrapper classes?

Ans:

Wrapper class:

- In Java, there is a wrapper class for every primitive data-type. Basically, the Wrapper classes encapsulate a primitive type within an object. For example, the wrapper class for `int` is `Integer`, the class for `float` is `Float`, and so on.
- The type wrappers are `Double`, `Float`, `Long`, `Integer`, `Short`, `Byte`, `Character`, and `Boolean`. These classes offer a wide array of methods that allow you to fully integrate the primitive types into Java's object hierarchy.

Importance / applications of wrapper classes:

- Wrapper classes provide a mechanism to "wrap" primitive values in an object so that the primitives can be included in activities reserved for objects, like being added to Collections, or returned from a method with an object return value.

- Wrapper classes provide a mechanism to "wrap" primitive values in an object so that the primitives can be included in activities reserved for objects, like being added to Collections, or returned from a method with an object return value.

42. Does Java support multiple inheritance of classes? How can we accomplish in Java?
Ans:

No, Java does not support multiple inheritances of classes. But we can accomplish this by using interfaces.

43. What are the major differences and similarities between an interface and a class?
Or
What is the major difference between an interface and a class?
Ans:

An interface is basically a kind of class. Like classes, interfaces contain methods and variables but with a major difference. The major difference between class and interface is that interfaces define the only abstract method and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants.

44. Summarize the rules associated for defining an interface with example?
Ans:

An interface is basically a kind of class. The syntax for defining an interface is very similar to that for defining a class. The general form of an interface definition is:

```
access interface InterfaceName {
variables declaration;
methods declaration;
}
```

Here, `access` is either public or not used, the **`interface`** is the keyword and `InterfaceName` is any valid Java variable.

Variables are declared as follows:

```
static final type VariableName = value;
```

All variables are declared as constant.

Methods declaration will contain only a list of methods without any body statement. For example:

```
return-type methodname1 (parameter_list);
```

Here is an example of an interface definition that contains two variables and one method.

```
interface Item{
     static final int code = 1001;
     static final String name = "Fan";
     void display (  );
}
```

45. Give an example where interfaces can be used to support multiple inheritance.

Or

Develop a standalone Java program to show where interface can be used to support multiple inheritances.

Ans:

Implementing multiple inheritance:

```
class Student{
     int rollNumber;

     void getNumber(int n){
          rollNumber=n;
     }

     void putNumber(){
          System.out.println("Roll NO:" +rollNumber);
     }
}


class Test extends Student{
     float part1, part2;

     void getMArks(float m1, float m2){
          part1=m1;
          part2=m2;
     }

     void putMarks(){
          System.out.println("Marks Obtained:");
          System.out.println("part1 = " +part1);
          System.out.println("part2 = " +part2);
     }
}


interface Sports{
     float sportWt = 6.0F;
     void putWt();
}
```

```
class Results extends Test implements Sports{
     float total;
     public void putWt(){
          System.out.println("Sports Wt = :" +sportWt);
     }

     void display(){
          total = part1 + part2 + sportWt;
          putNumber();
          putMarks();
          putWt();
          System.out.println("Total Score = " +total);
     }
}



class Hybrid {
     public static void main(String[] args) {
          Results student1 = new Results();
          student1.getNumber(1234);
          student1.getMArks(27.5F, 33.0F);
          student1.display();
     }

}
```

46. What is *this* reference?

Ans:-

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the ***this*** keyword. ***this*** can be used inside any method to refer to the current object. That is, ***this*** is always a reference to the object on which the method was invoked.

```
Box(double w, double h, double d){
     this.width=w;
     this.height=h;
     this.depth = d;
}
```

The use of ***this*** in this case is redundant, but perfectly correct.

47. What do you mean by Dynamic Initialization?
Ans:-

Java is a flexible programming language that allows the dynamic initialization of variables. Initializing a variable at run time is called dynamic initialization. It utilizes memory efficiently and provides the flexibility of using different formats of data at the run time considering the situation. In java, we can declare the variable at any place before it is used.

```
/* DynamicInitializationDemo.java */
public class DynamicInitializationDemo
{
    public static void main(String[] args)
    {
        //dynSqrt will be initialized when Math.sqrt
        //will be executed at run time
        double dynSqrt = Math.sqrt (16);
        System.out.println("sqrt of 16 is : " + dynSqrt);
    }
}
```

48. Differentiate exit-controlled loop and entry-controlled loop.
Ans:

Entry-controlled loop:
In the entry-controlled loop, the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.
Example: while loop.

Exit-controlled loop:
In the case of an exit-controlled loop, the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time.
Example: do-while loop.

49. Compare *do-while* with *while* loop.
Ans:

Comparison between *while* and *do…while*:

while	**do … while**
The ***while*** is an entry-controlled loop that is the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed.	The ***do…while*** is an exit-controlled loop that is the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time i.e., body of the loop executed at least once though the conditions are not satisfied.
General form: *while (test condition)* *{* *body of the loop* *}*	General form: *do* *{* *body of the loop* *}* *while (test condition)*

50. Show the comparison of *for, while* and *do-while loop*
Ans:

for	**while**	**do-while**

`for(n=1;n<=10;n++)`
`{`
`  ----------`
`  ----------`
`}`
	`n=1;`
	`while (n<=10)`
	`{`
	`  ----------`
	`  ----------`
	`  n=n+1;`
	`}`
		`n = 1;`
		`do{`
		`  ----------`
		`  ----------`
		`  n=n+1;`
		`  }`
		`while (n<=10);`

51. What is the minimum number of times that do-while loop can be executed?
Ans:

Since the test condition is evaluated at the bottom of the loop, the do…while construct provides an exit-controlled loop, and therefore the body of the loop is executed at least once.

52. Enlist the difference between Procedural Programming and Object Oriented Programming Language.

Ans:

Procedural Programming Language	**Object Oriented Programming Language(OOP)**
The procedural programming language executes a series of procedures sequentially.	In object-oriented programming approach, there is a collection of objects.
This is a top-down programming approach.	This is a bottom-up programming approach.
The major focus is on procedures or functions.	The main focus is on objects.
Data reusability is not possible.	Data reusability is one of the important feature of OOP.
Data binding is not possible.	Data binding can be done by making it private.