OOPs (Object-Oriented Programming System)

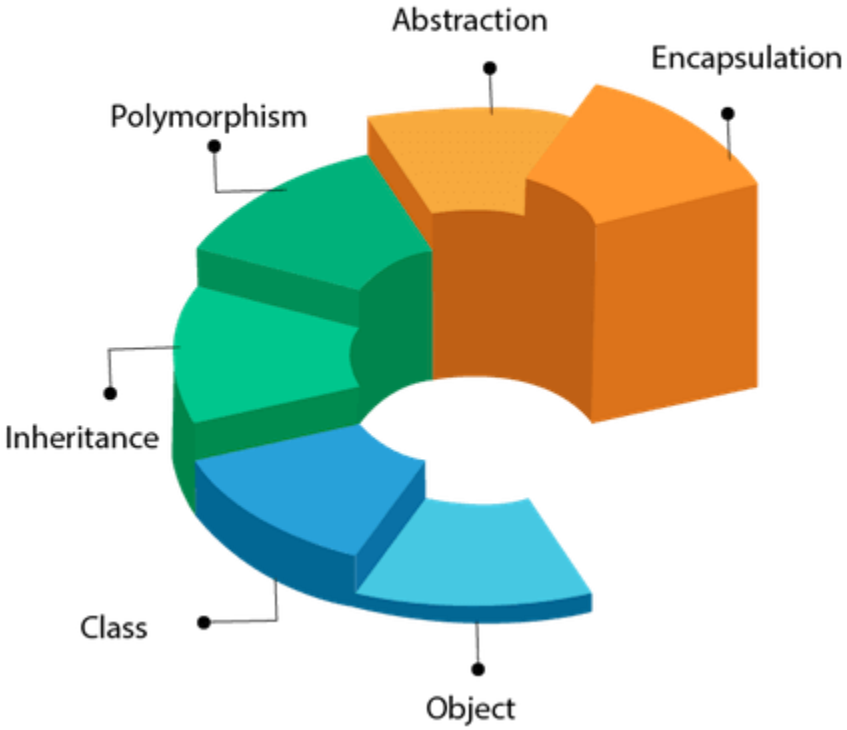
Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- **Object**
- **Class**
- **Inheritance**
- **Polymorphism**
- **Abstraction**
- **Encapsulation**

Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- **Coupling**
- **Cohesion**
- **Association**
- **Aggregation**
- **Composition**

OOPs (Object-Oriented Programming System)



Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the

details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One

- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be undirectional or bidirectional.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

3) OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

1) What do you understand by OOP?

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts.

In other words, It is a programming paradigm that revolves around the object rather than function and procedure.

In other words, it is an approach for developing applications that emphasize on objects. An object is a real word entity that contains data and code. It allows binding data and code together.

2) Name any seven widely used OOP languages.

There are various OOP languages but the most widely used are:

- Python
- Java
- Go
- Dart
- C++
- C#
- Ruby

---

3) What is the purpose of using OOPs concepts?

The aim of OOP is to implement real-world entities like inheritance, hiding, polymorphism in programming. The main purpose of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

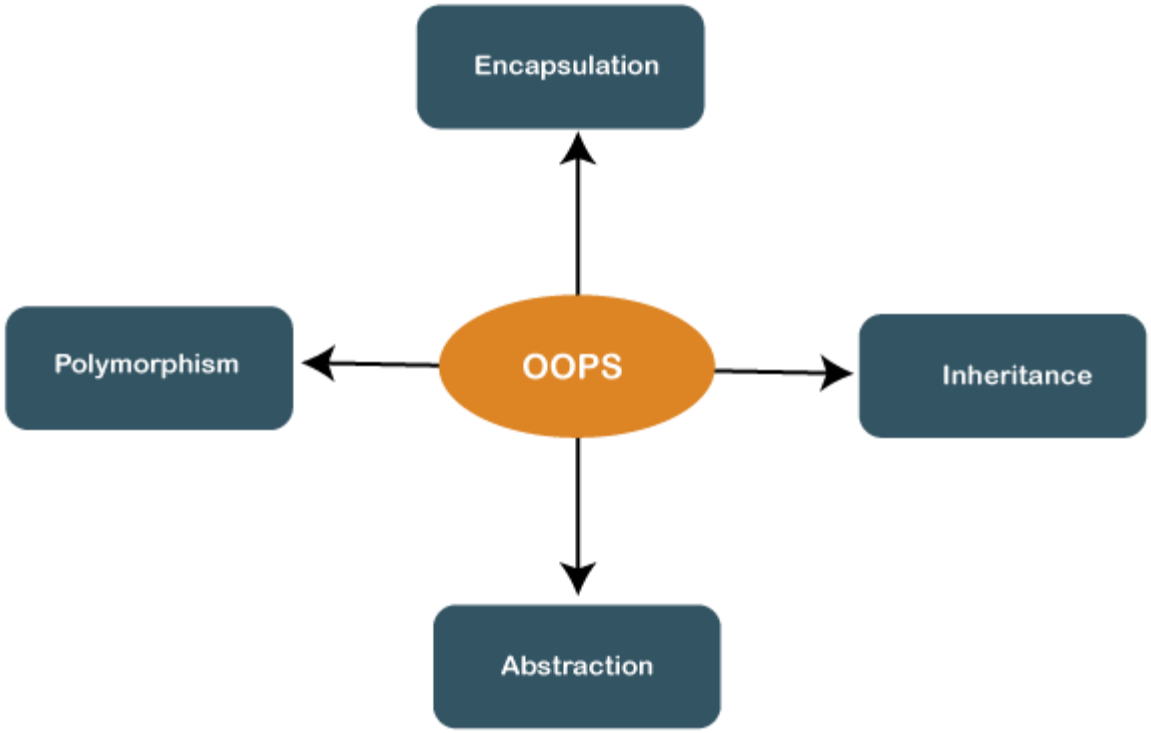
---

4) What are the four main features of OOPs?

The OOP has the following four features:

- Inheritance

- Encapsulation
- Polymorphism
- Data Abstraction



5) Why OOP is so popular?

OOPs, programming paradigm is considered as a better style of programming. Not only it helps in writing a complex piece of code easily, but it also allows users to handle and maintain them easily as well. Not only that, the main pillar of OOPs - Data Abstraction, Encapsulation, Inheritance, and Polymorphism, makes it easy for programmers to solve complex scenarios. As a result of these, OOPs is so popular.

6) What are the advantages and disadvantages of OOP?

Advantages of OOP

- It follows a bottom-up approach.
- It models the real word well.
- It allows us the reusability of code.
- Avoids unnecessary data exposure to the user by using the abstraction.

- OOP forces the designers to have a long and extensive design phase that results in better design and fewer flaws.
- Decompose a complex problem into smaller chunks.
- Programmer are able to reach their goals faster.
- Minimizes the complexity.
- Easy redesign and extension of code that does not affect the other functionality.

Disadvantages of OOP

- Proper planning is required.
- Program design is tricky.
- Programmer should be well skilled.
- Classes tend to be overly generalized.

7) What are the limitations of OOPs?

- Requires intensive testing processes.
- Solving problems takes more time as compared to Procedure Oriented Programming.
- The size of the programs created using this approach may become larger than the programs written using the procedure-oriented programming approach.
- Software developed using this approach requires a substantial amount of pre-work and planning.
- OOP code is difficult to understand if you do not have the corresponding class documentation.
- In certain scenarios, these programs can consume a large amount of memory.
- Not suitable for small problems.
- Takes more time to solve problems.

8) What are the differences between object-oriented programming and structural programming?

Object-oriented Programming	Structural Programming

It follows a bottom-up approach.	It follows a top-down approach.
It provides data hiding.	Data hiding is not allowed.
It is used to solve complex problems.	It is used to solve moderate problems.
It allows reusability of code that reduces redundancy of code.	Reusability of code is not allowed.
It is based on objects rather than functions and procedures.	It provides a logical structure to a program in which the program is divided into functions.
It provides more security as it has a data hiding feature.	It provides less security as it does not support the data hiding feature.
More abstraction more flexibility.	Less abstraction less flexibility.
It focuses on data.	It focuses on the process or logical structure.

9) What do you understand by pure object-oriented language? Why Java is not a pure object-oriented programming language?

The programming language is called pure object-oriented language that treats everything inside the program as an object. The primitive types are not supported by the pure OOPs language. There are some other features that must satisfy by a pure object-oriented language:

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction
- All predefined types are objects
- All user-defined types are objects
- All operations performed on objects must be only through methods exposed to the objects.

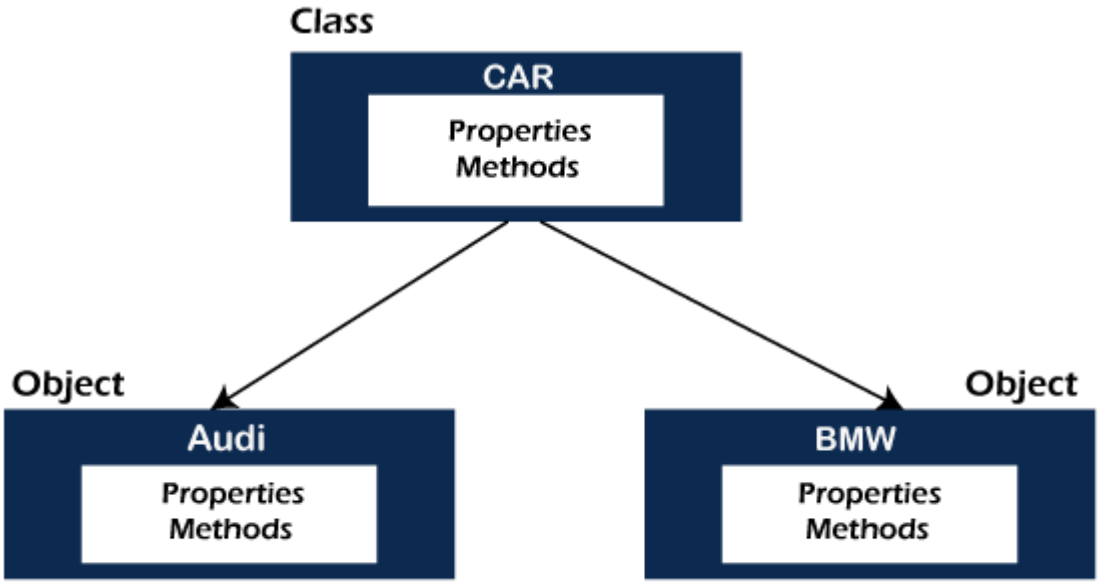
Java is not a pure object-oriented programming language because pre-defined data types in Java are not treated as objects. Hence, it is not an object-oriented language.

10) What do you understand by class and object? Also, give example.

Class: A class is a blueprint or template of an object. It is a user-defined data type. Inside a class, we define variables, constants, member functions, and other functionality. It does not consume memory at run time. Note that classes are not considered as a data structure. It is a logical entity. It is the best example of data binding.

Object: An object is a real-world entity that has attributes, behavior, and properties. It is referred to as an instance of the class. It contains member functions, variables that we have defined in the class. It occupies space in the memory. Different objects have different states or attributes, and behaviors.

The following figure best illustrates the class and object.



11) What are the differences between class and object?

Class	Object

It is a logical entity.	It is a real-world entity.
It is conceptual.	It is real.
It binds data and methods together into a single unit.	It is just like a variable of a class.
It does not occupy space in the memory.	It occupies space in the memory.
It is a data type that represents the blueprint of an object.	It is an instance of the class.
It is declared once.	Multiple objects can be declared as and when required.
It uses the keyword class when declared.	It uses the new keyword to create an object.
A class can exist without any object.	Objects cannot exist without a class.

12) What are the key differences between class and structure?

Class	Structure
Class is a group of common objects that shares common properties.	The structure is a collection of different data types.
It deals with data members and member functions.	It deals with data members only.
It supports inheritance.	It does not support inheritance.
Member variables cannot be initialized directly.	Member variables can be initialized directly.
It is of type reference.	It is of a type value.

It's members are private by default.	It's members are public by default.
The keyword class defines a class.	The keyword struct defines a structure.
An instance of a class is an object.	An instance of a structure is a structure variable.
Useful while dealing with the complex data structure.	Useful while dealing with the small data structure.

13) What is the concept of access specifiers when should we use these?

In OOPs language, **access specifiers** are reserved keyword that is used to set the accessibility of the classes, methods and other members of the class. It is also known as **access modifiers**. It includes **public, private,** and **protected**. There is some other access specifier that is language-specific. Such as Java has another access specifier **default**. These access specifiers play a vital role in achieving one of the major functions of OOP, i.e. encapsulation. The following table depicts the accessibility.

Specifiers	Within Same Class	In Derived Class	Outside the Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

14) What are the manipulators in OOP and how it works?

Manipulators are helping functions. It is used to manipulate or modify the input or output stream. The modification is possible by using the **insertion** (<<) and **extraction** (>>) operators. Note that the modification of input or output stream does not mean to change the values of variables. There are two types of manipulators with **arguments** or **without arguments.**

The example of manipulators that do not have arguments is **endl, ws, flush,** etc. Manipulators with arguments are **setw(val), setfill(c), setbase(val), setiosflags(flag).** Some other manipulators are **showpos, fixed, scientific, hex, dec, oct,** etc.

---

15) What are the rules for creating a constructor?

- It cannot have a return type.
- It must have the same name as the Class name.
- It cannot be marked as static.
- It cannot be marked as abstract.
- It cannot be overridden.
- It cannot be final.

---

16) What are the differences between the constructor and the method in Java?

Constructor	Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.
It creates an instance of a class.	It is used to execute Java code.
It is invoked implicitly when we create an object of the class.	It gets executed when we explicitly called it.

It cannot be inherited by the subclass.	It can be inherited by the subclass.
It does not have any return type.	It must have a return type.
It cannot be overridden in Java.	It can be overridden in Java.
It cannot be declared as static.	It can be declared as static.
The Java compiler provides a default constructor if you don't have any constructor in a class.	Java compiler does not provide any method by default.

17) How does procedural programming be different from OOP differ?

Procedural Oriented Programming	Object-Oriented Programming
It is based on functions.	It is based on real-world objects.
It follows a top-down approach.	It follows a bottom-up approach.
It is less secure because there is no proper way to hide data.	It provides more security.
Data is visible to the whole program.	It encapsulates the data.
Reuse of code is not allowed.	The code can be reused.
Modification and extension of code are not easy.	We can easily modify and extend code.
Examples of POP are C, VB, FORTRAN, Pascal, etc.	Examples of OOPs are C++, Java, C#, .NET, etc.

18) What are the differences between error and exception?

Basis of Comparison	Exception	Error
Recoverable/ Irrecoverable	Exception can be recovered by using the try-catch block.	An error cannot be recovered.
Type	It can be classified into two categories i.e. checked and unchecked.	All errors in Java are unchecked.
Occurrence	It occurs at compile time or run time.	It occurs at run time.
Package	It belongs to java.lang.Exception package.	It belongs to java.lang.Error package.
Known or unknown	Only checked exceptions are known to the compiler.	Errors will not be known to the compiler.
Causes	It is mainly caused by the application itself.	It is mostly caused by the environment in which the application is running.
Example	Checked Exceptions: SQLException, IOException Unchecked Exceptions: ArrayIndexOutOfBoundException, NullPointerException, ArithmaticException	Java.lang.StackOverFlow, java.lang.OutOfMemoryError

19) What are the characteristics of an abstract class?

An abstract class is a class that is declared as abstract. It cannot be instantiated and is always used as a base class. The characteristics of an abstract class are as follows:

- Instantiation of an abstract class is not allowed. It must be inherited.
- An abstract class can have both abstract and non-abstract methods.
- An abstract class must have at least one abstract method.
- You must declare at least one abstract method in the abstract class.
- It is always public.

- It is declared using the **abstract**

The purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share.

---

20) Is it possible for a class to inherit the constructor of its base class?

No, a class cannot inherit the constructor of its base class.

---

21) Identify which OOPs concept should be used in the following scenario?

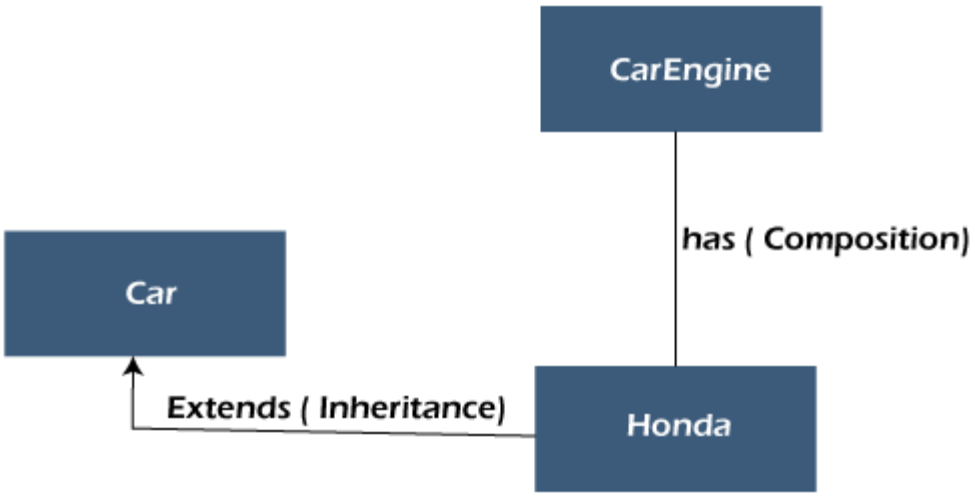
A group of 5 friends, one boy never gives any contribution when the group goes for the outing. Suddenly a beautiful girl joins the same group. The boy who never contributes is now spending a lot of money for the group.

Runtime Polymorphism

---

22) What is composition?

Composition is one of the vital concepts in OOP. It describes a class that references one or more objects of other classes in instance variables. It allows us to model a has-a association between objects. We can find such relationships in the real world. For example, a car has an engine. the following figure depicts the same



The main benefits of composition are:

- Reuse existing code
- Design clean APIs
- Change the implementation of a class used in a composition without adapting any external clients.

23) What are the differences between copy constructor and assignment operator?

The copy constructor and the assignment operator (=) both are used to initialize one object using another object. The main difference between the two is that the copy constructor allocates separate memory to both objects i.e. existing object and newly created object while the assignment operator does not allocate new memory for the newly created object. It uses the reference variable that points to the previous memory block (where an old object is located).

Syntax of Copy Constructor

```
1. class_name (const class_name &obj)
2. {
3. //body
4. }
```

Syntax of Assignment Operator

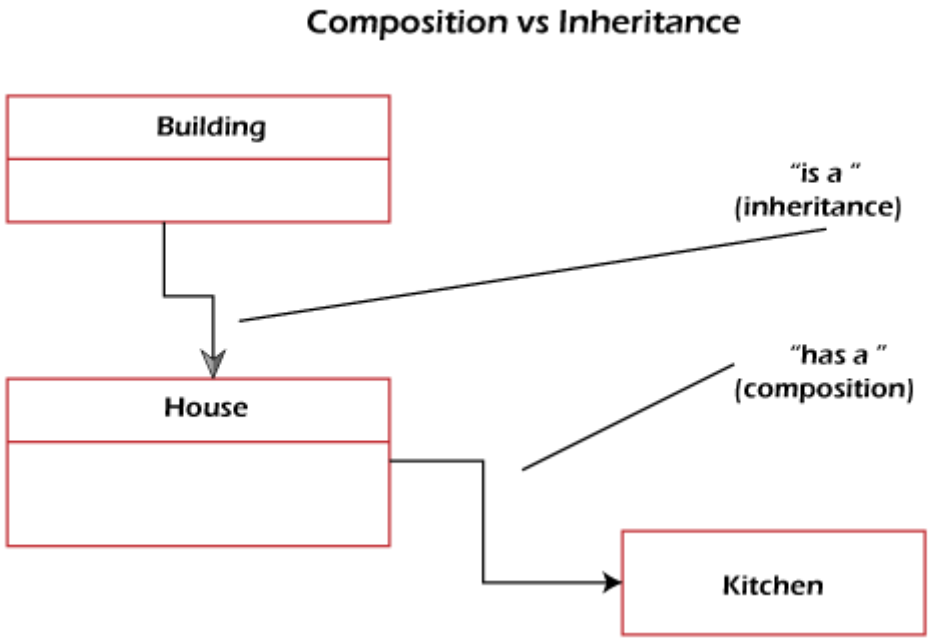
```
1. class_name obj1, obj2;
2. obj1=obj2;
```

Copy Constructor	Assignment Operator
It is an overloaded constructor.	It is an operator.
It creates a new object as a copy of an existing object.	It assigns the value of one object to another object both of which already exist.

The copy constructor is used when a new object is created with some existing object.	It is used when we want to assign an existing object to a new object.
Both the objects use separate memory locations.	Both objects share the same memory but use the two different reference variables that point to the same location.
If no copy constructor is defined in the class, the compiler provides one.	If the assignment operator is not overloaded then the bitwise copy will be made.

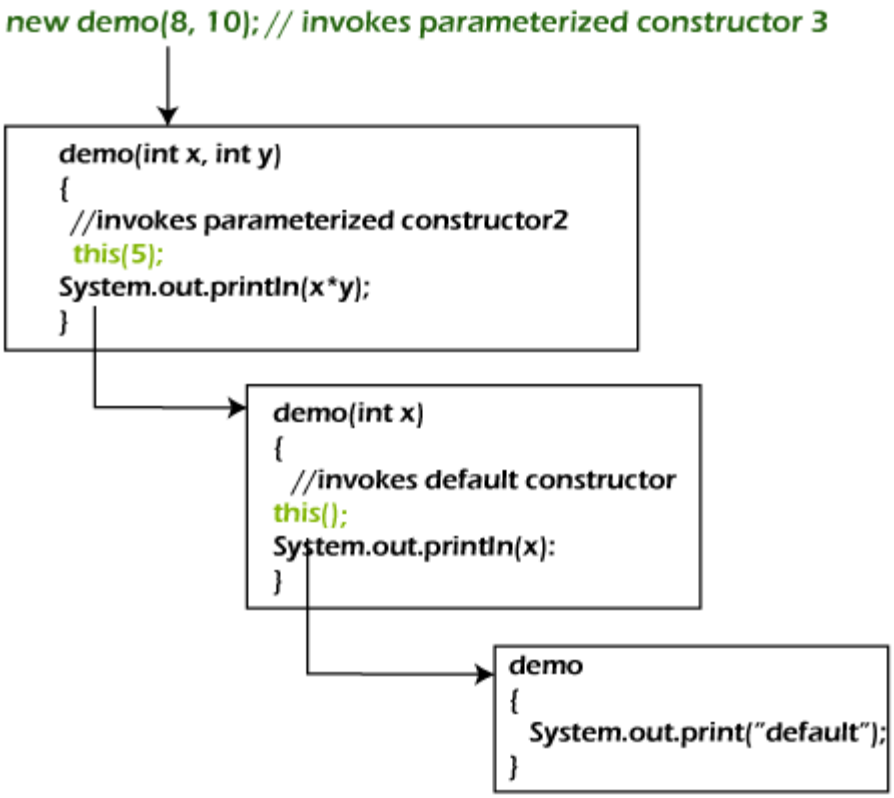
24) What is the difference between Composition and Inheritance?

Inheritance means an object inheriting reusable properties of the base class. Compositions mean that an object holds other objects. In Inheritance, there is only one object in memory (derived object) whereas, in Composition, the parent object holds references of all composed objects. From a design perspective, inheritance is "is a" relationship among objects whereas Composition is "has a" relationship among objects.



25) What is constructor chaining?

In OOPs, constructor chaining is a sequence of invoking constructors (of the same class) upon initializing an object. It is used when we want to invoke a number of constructors, one after another by using only an instance. In other words, if a class has more than one constructor (overloaded) and one of them tries to invoke another constructor, this process is known as constructor chaining. In C++, it is known as constructor delegation and it is present from C++ 11.



26) What are the limitations of inheritance?

- The main disadvantage of using inheritance is two classes get tightly coupled. That means one cannot be used independently of the other. If a method or aggregate is deleted in the Super Class, we have to refactor using that method in SubClass.
- Inherited functions work slower compared to normal functions.
- Need careful implementation otherwise leads to improper solutions.

27) What are the differences between Inheritance and Polymorphism?

Inheritance	Polymorphism
Inheritance is one in which a derived class inherits the already existing class's features.	Polymorphism is one that you can define in different forms.
It refers to using the structure and behavior of a superclass in a subclass.	It refers to changing the behavior of a superclass in the subclass.
It is required in order to achieve polymorphism.	In order to achieve polymorphism, inherence is not required.
It is applied to classes.	It is applied to functions and methods.
It can be single, hybrid, multiple, hierarchical, multipath, and multilevel inheritance.	There are two types of polymorphism compile time and run time.
It supports code reusability and reduces lines of code.	It allows the object to decide which form of the function to be invoked at run-time (overriding) and compile-time (overloading).

28) What is Coupling in OOP and why it is helpful?

In programming, separation of concerns is known as **coupling**. It means that an object cannot directly change or modify the state or behavior of other objects. It defines how closely two objects are connected together. There are two types of coupling, **loose** coupling, and **tight** coupling.

Objects that are independent of one another and do not directly modify the state of other objects is called loosely coupled. Loose coupling makes the code more flexible, changeable, and easier to work with.

Objects that depend on other objects and can modify the states of other objects are called tightly coupled. It creates conditions where modifying the code of one object

also requires changing the code of other objects. The reuse of code is difficult in tight coupling because we cannot separate the code.

Since using loose coupling is always a good habit.

---

29) Name the operators that cannot be overload.

1. Scope Resolution Operator (::)
2. Ternary Operator (? :)
3. Member Access or Dot Operator (.)
4. Pointer to Member Operator (.*)
5. sizeof operator

---

30) What is the difference between new and override?

The new modifier instructs the compiler to use the new implementation instead of the base class function. Whereas, Override modifier helps to override the base class function.

virtual: indicates that a method may be overridden by an inheritor

override: Overrides the functionality of a virtual method in a base class, providing different functionality.

new: Hides the original method (which doesn't have to be virtual), providing different functionality. This should only be used where it is absolutely necessary.

When you hide a method, you can still access the original method by upcasting to the base class. This is useful in some scenarios, but dangerous.

---

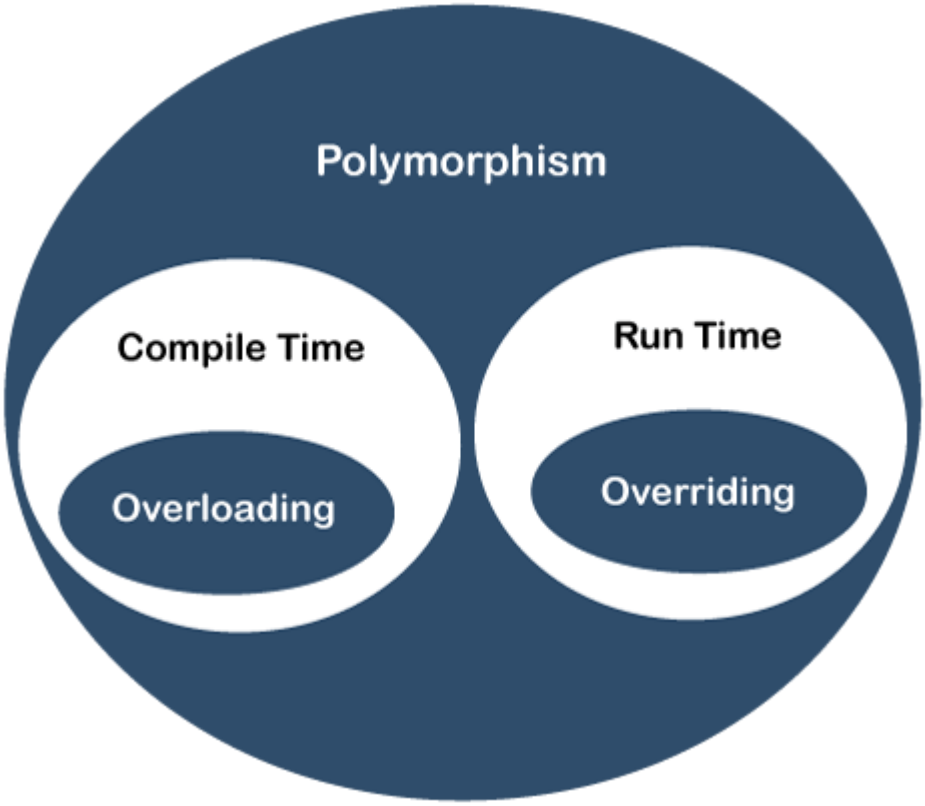
31) Explain overloading and overriding with example?

Overloading

Overloading is a concept in OOP when two or more methods in a class with the same name but the method signature is different. It is also known as **compile-time polymorphism**. For example, in the following code snippet, the method **add()** is an overloaded method.

```
1. public class Sum
```

```
2.  {
3.  int a, b, c;
4.  public int add();
5.  {
6.  c=a+b;
7.  return c;
8.  }
9.  add(int a, int b);
10. {
11. //logic
12. }
13. add(int a, int b, int c);
14. {
15. //logic
16. }
17. add(double a, double b, double c);
18. {
19. //logic
20. }
21. //statements
22. }
```



Overriding

If a method with the same method signature is presented in both child and parent class is known as method **overriding**. The methods must have the same number of parameters and the same type of parameter. It overrides the value of the parent class method. It is also known as **runtime polymorphism**. For example, consider the following program.

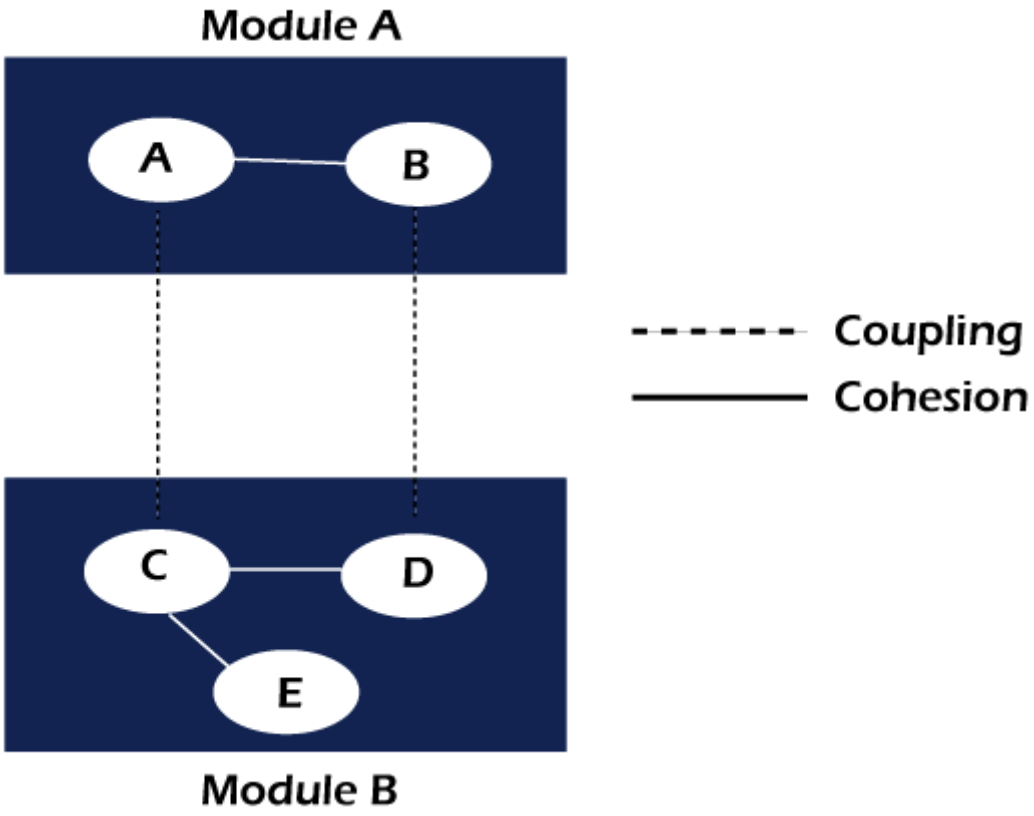
```
1. class Dog
2. {
3. public void bark()
4. {
5. System.out.println("woof ");
6. }
7. }
8. class Hound extends Dog
9. {
10. public void sniff()
11. {
12. System.out.println("sniff ");
```

```
13. }
14. //overrides the method bark() of the Dog class
15. public void bark()
16. {
17. System.out.println("bowl");
18. }
19. }
20. public class OverridingExample
21. {
22. public static void main(String args[])
23. {
24. Dog dog = new Hound();
25. //invokes the bark() method of the Hound class
26. dog.bark();
27. }
28. }
```

32) What is Cohesion in OOP?

In OOP, **cohesion** refers to the degree to which the elements inside a module belong together. It measures the strength of the relationship between the module and data. In short, cohesion represents the clarity of the responsibilities of a module. It is often contrasted with coupling.

It focuses on a how single module or class is intended. Higher the cohesiveness of the module or class, better is the object-oriented design.



There are two types of cohesion, i.e. **High** and **Low**.

- High cohesion is associated with several required qualities of software including **robustness, reliability**, and **understandability**.
- Low cohesion is associated with unwanted qualities such as being difficult to **maintain, test, reuse,** or even **understand**.

High cohesion often associates with loose coupling and vice versa.

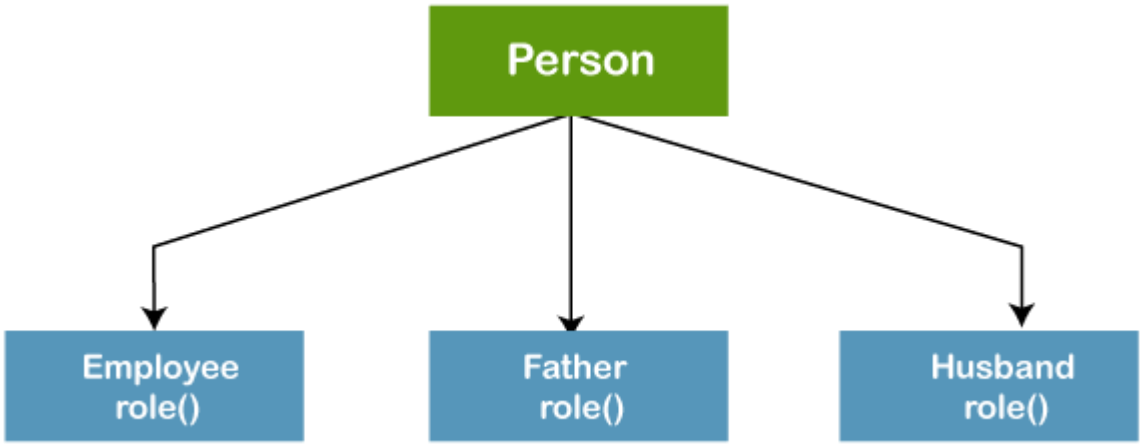
---

33) Give a real-world example of polymorphism?

The general meaning of Polymorphism is one that has different forms. The best real-world example of polymorphism is a **person** that plays different roles at different palaces or situations.

- At home a person can play the role of father, husband, and son.
- At the office the same person plays the role of boss or employee.
- In public transport, he plays the role of passenger.
- In the hospital, he can play the role of doctor or patient.

- At the shop, he plays the role of customer.



Hence, the same person possesses different behavior in different situations. It is called polymorphism.

---

34) What is the difference between a base class and a superclass?

The base class is the root class- the most generalized class. At the same time, the superclass is the immediate parent class from which the other class inherits.

---

35) What is data abstraction and how can we achieve data abstraction?

It is one of the most important features of OOP. It allows us to show only essential data or information to the user and hides the implementation details from the user. A real-world example of abstraction is driving a car. When we drive a car, we do not need to know how the engine works (implementation) we only know how ECG works.

There are two ways to achieve data abstraction

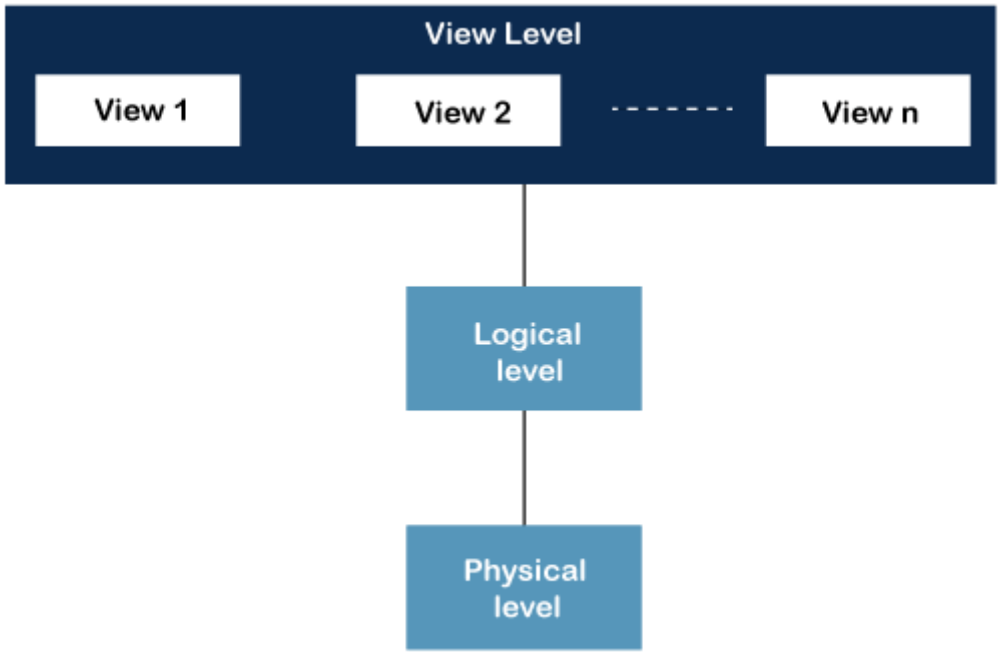
- **Abstract class**
- **Abstract method**

---

36) What are the levels of data abstraction?

There are **three** levels of data abstraction:

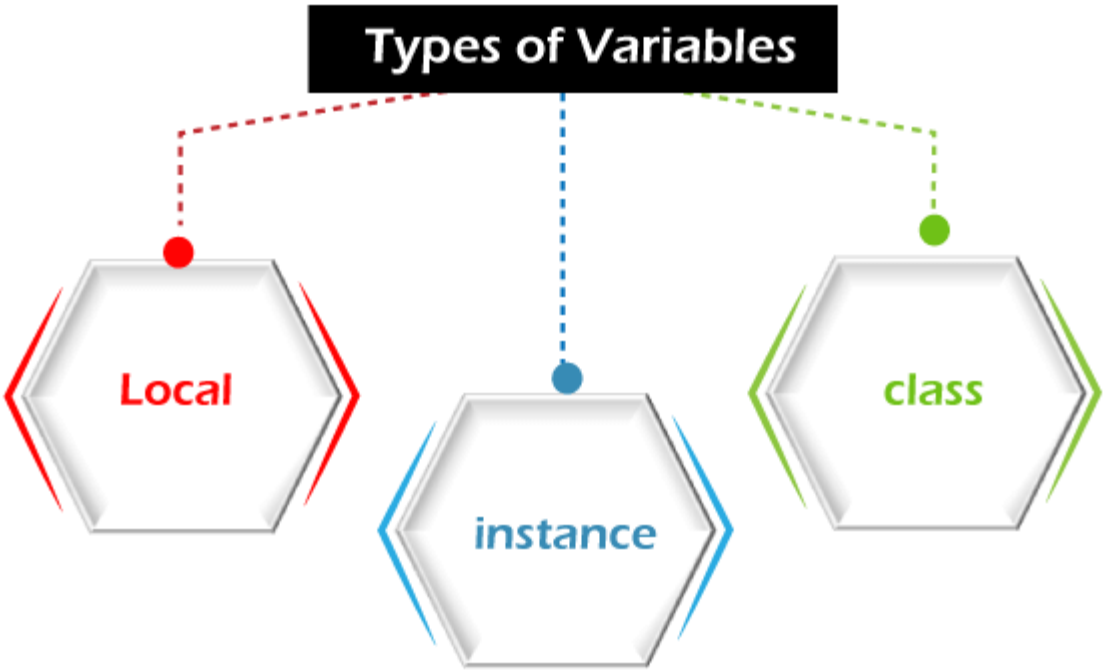
- **Physical Level:** It is the lowest level of data abstraction. It shows how the data is actually stored in memory.
- **Logical Level:** It includes the information that is actually stored in the database in the form of tables. It also stores the relationship among the data entities in relatively simple structures. At this level, the information available to the user at the view level is unknown.
- **View Level:** It is the highest level of data abstraction. The actual database is visible to the user. It exists to ease the availability of the database by an individual user.



Levels of data abstraction

37) What are the types of variables in OOP?

There are three types of variables:



Instance Variable: It is an object-level variable. It should be declared inside a class but must be outside a method, block, and constructor. It is created when an object is created by using the new keyword. It can be accessed directly by calling the variable name inside the class.

Static Variable: It is a class-level variable. It is declared with keyword **static** inside a class but must be outside of the method, block, and constructor. It stores in static memory. Its visibility is the same as the instance variable. The default value of a static variable is the same as the instance variable. It can be accessed by calling the **class_name.variable_name**.

Local Variable: It is a method-level variable. It can be declared in method, constructor, or block. Note that the use of an access modifier is not allowed with local variables. It is visible only to the method, block, and constructor in which it is declared. Internally, it is implemented at the stack level. It must be declared and initialized before use.

Another type of variable is used in object-oriented programming is the **reference** variable.

Reference Variable: It is a variable that points to an object of the class. It points to the location of the object that is stored in the memory.

---

38) Is it possible to overload a constructor?

Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. For example:

```
1. public class Demo
2. {
3. Demo()
4. {
5. //logic
6. }
7. Demo(String str) //overloaded constructor
8. {
9. //logic
10.}
11. Demo(double d) //overloaded constructor
12.{
13. //logic
14.}
15. //statements
16.}
```

---

39) Can we overload the main() method in Java also give an example?

Yes, we can also overload the main() method in Java. Any number of main() methods can be defined in the class, but the method signature must be different. Consider the following code.

```
1. class OverloadMain
2. {
3. public static void main(int a)  //overloaded main method
4. {
5. System.out.println(a);
6. }
7. public static void main(String args[])
8. {
9. System.out.println("main method invoked");
```

```
10. main(6);
11. }
12. }
```

40) Consider the following scenario:

If a class Demo has a static block and a main() method. A print statement is presented in both. The question is which one will first execute, static block or the main() method, and why?

JVM first executes the static block on a priority basis. It means JVM first goes to static block even before it looks for the main() method in the program. After that main() method will be executed.

```
1. class Demo
2. {
3. static              //static block
4. {
5. System.out.println("Static block");
6. }
7. public static void main(String args[])  //static method
8. {
9. System.out.println("Static method");
10. }
11. }
```

41) What is the difference between an object-oriented programming language and object-based programming language?

There are the following basic differences between the object-oriented language and object-based language.

- Object-oriented languages follow all the concepts of OOPs whereas, the object-based language doesn't follow all the concepts of OOPs like inheritance and polymorphism.
- Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.

- Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.

Note:- An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

42) What is an object?

Any entity that has state and behavior is known as an object. In Java, Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the **new** keyword.

43) What will be the initial value of an object reference which is defined as an instance variable?

All object references are initialized to null in Java.

44) What is the constructor?

The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

More Details.

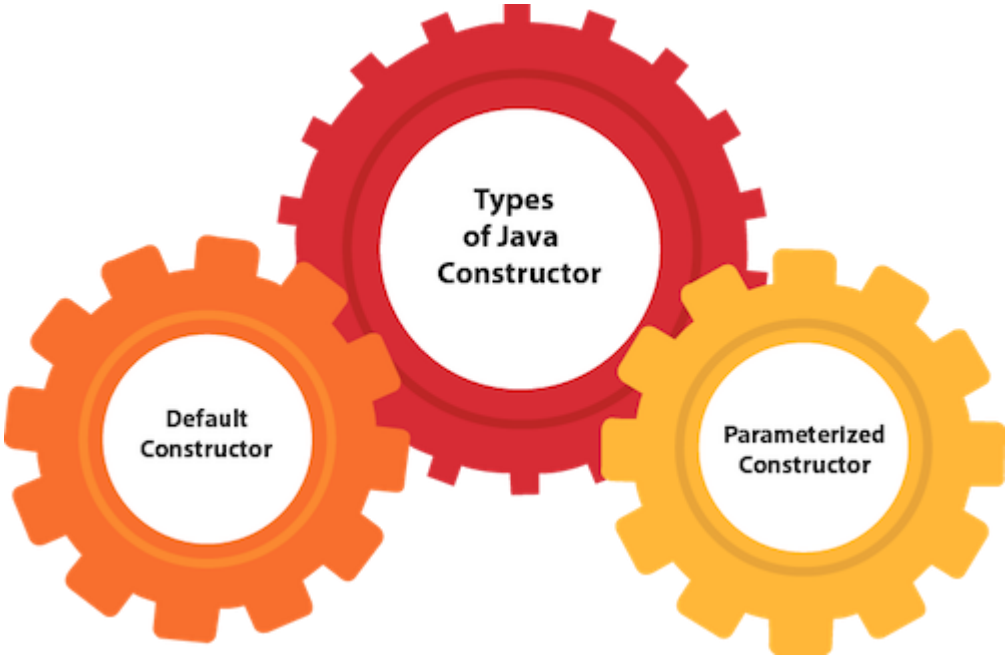
---

45) How many types of constructors are used in Java?

Based on the parameters passed in the constructors, there are two types of constructors in Java.

- **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
- **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say

that the constructors which can accept the arguments are called parameterized constructors.



46) What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

```
1. class Student3{
2. int id;
3. String name;
4. 
5. void display(){System.out.println(id+" "+name);}
6. 
7. public static void main(String args[]){
8. Student3 s1=new Student3();
9. Student3 s2=new Student3();
10. s1.display();
11. s2.display();
12. }
13. }
```

Test it Now

Output:

```
0 null
0 null
```

Explanation: In the above class, you are not creating any constructor, so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

More Details.

---

47) Does constructor return any value?

Ans: yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor). More Details.

---

48)Is constructor inherited?

No, The constructor is not inherited.

---

49) Can you make a constructor final?

No, the constructor can't be final.

---

50) Can we overload the constructors?

Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. Consider the following example.

```
1.  class Test
2.  {
3.      int i;
4.      public Test(int k)
5.      {
6.          i=k;
7.      }
8.      public Test(int k, int m)
9.      {
10.         System.out.println("Hi I am assigning the value max(k, m) to i");
11.        if(k>m)
12.        {
13.            i=k;
14.        }
15.        else
16.        {
17.            i=m;
18.        }
19.    }
20. }
21. public class Main
22. {
23.    public static void main (String args[])
24.    {
25.        Test test1 = new Test(10);
26.        Test test2 = new Test(12, 15);
27.        System.out.println(test1.i);
28.        System.out.println(test2.i);
29.    }
30. }
31.
```

In the above program, The constructor Test is overloaded with another constructor. In the first call to the constructor, The constructor with one argument is called, and i will be initialized with the value 10. However, In the second call to the constructor, The constructor with the 2 arguments is called, and i will be initialized with the value 15.

51) What do you understand by copy constructor in Java?

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- **By constructor**
- **By assigning the values of one object into another**
- **By clone() method of Object class**

In this example, we are going to copy the values of one object into another using java constructor.

```
1. //Java program to initialize the values from one object to another
2. class Student6{
3.     int id;
4.     String name;
5.     //constructor to initialize integer and string
6.     Student6(int i,String n){
7.     id = i;
8.     name = n;
9.     }
10.    //constructor to initialize another object
11.    Student6(Student6 s){
12.    id = s.id;
13.    name =s.name;
14.    }
15.    void display(){System.out.println(id+" "+name);}
16.
17.    public static void main(String args[]){
18.    Student6 s1 = new Student6(111,"Karan");
19.    Student6 s2 = new Student6(s1);
20.    s1.display();
21.    s2.display();
22.   }
23. }
```

Test it Now

Output:

```
111 Karan
111 Karan
```

52) What is the output of the following Java program?

```
1.  public class Test
2.  {
3.      Test(int a, int b)
4.      {
5.          System.out.println("a = "+a+" b = "+b);
6.      }
7.      Test(int a, float b)
8.      {
9.          System.out.println("a = "+a+" b = "+b);
10.     }
11.     public static void main (String args[])
12.     {
13.         byte a = 10;
14.         byte b = 15;
15.         Test test = new Test(a,b);
16.     }
17. }
```

The output of the following program is:

```
a = 10 b = 15
```

Here, the data type of the variables a and b, i.e., byte gets promoted to int, and the first parameterized constructor with the two integer parameters is called.

---

53) What is the output of the following Java program?

```
1.  class Test
2.  {
3.      int i;
4.  }
5.  public class Main
```

```
6.  {
7.      public static void main (String args[])
8.      {
9.          Test test = new Test();
10.         System.out.println(test.i);
11.     }
12. }
```

The output of the program is 0 because the variable i is initialized to 0 internally. As we know that a default constructor is invoked implicitly if there is no constructor in the class, the variable i is initialized to 0 since there is no constructor in the class.

---

54) What is the output of the following Java program?

```
1.  class Test
2.  {
3.      int test_a, test_b;
4.      Test(int a, int b)
5.      {
6.      test_a = a;
7.      test_b = b;
8.      }
9.      public static void main (String args[])
10.     {
11.         Test test = new Test();
12.         System.out.println(test.test_a+" "+test.test_b);
13.     }
14. }
```

There is a **compiler error** in the program because there is a call to the default constructor in the main method which is not present in the class. However, there is only one parameterized constructor in the class Test. Therefore, no default constructor is invoked by the constructor implicitly.

55) What is the static variable?

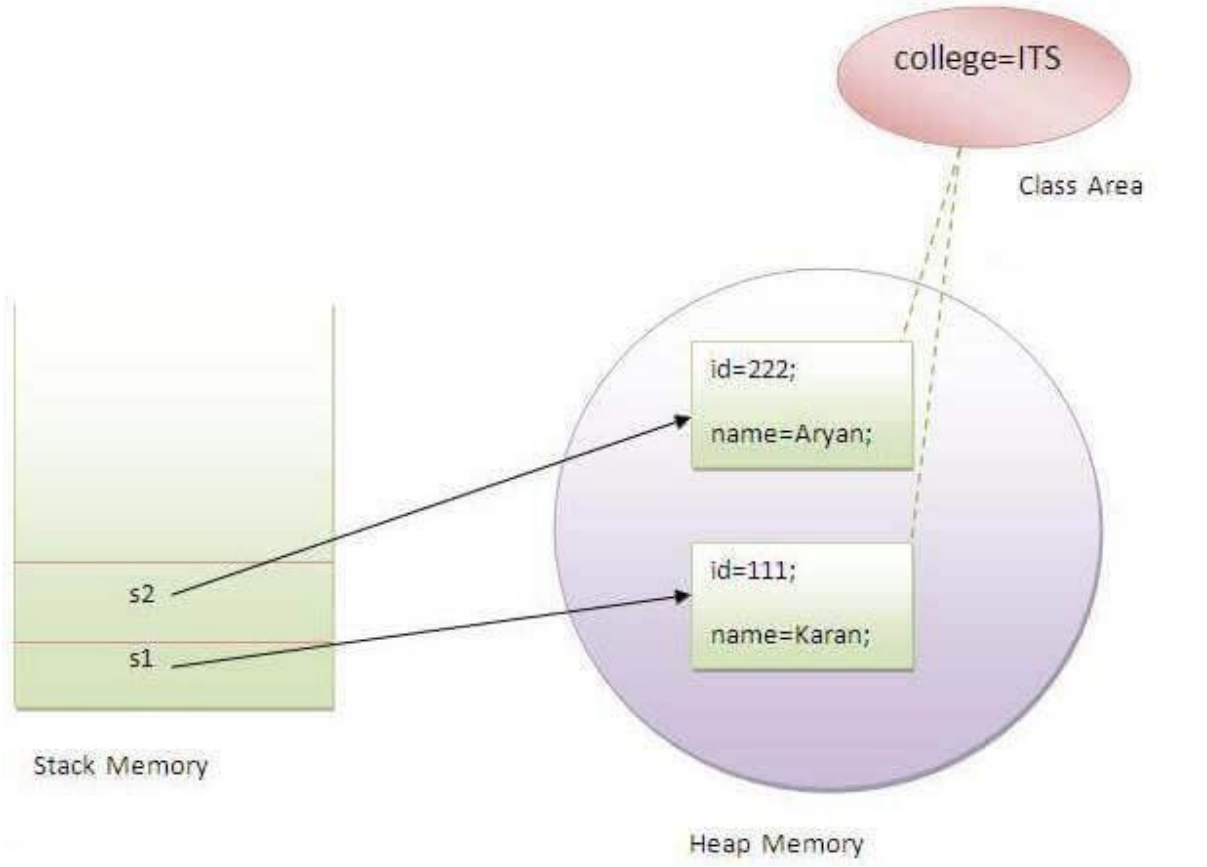
The static variable is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc. Static variable gets memory only once in the class area at the time of

class loading. Using a static variable makes your program more memory efficient (it saves memory). Static variable belongs to the class rather than the object.

```
1.  //Program of static variable
2.
3.  class Student8{
4.    int rollno;
5.    String name;
6.    static String college ="ITS";
7.
8.    Student8(int r,String n){
9.    rollno = r;
10.   name = n;
11.   }
12. void display (){System.out.println(rollno+" "+name+" "+college);}
13.
14. public static void main(String args[]){
15. Student8 s1 = new Student8(111,"Karan");
16. Student8 s2 = new Student8(222,"Aryan");
17.
18. s1.display();
19. s2.display();
20. }
21.}
```

Test it Now

```
Output:111 Karan ITS
       222 Aryan ITS
```



More Details.

56) What is the static method?

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

More Details.

57) What are the restrictions that are applied to the Java static methods?

Two main restrictions are applied to the static methods.

- The static method can not use non-static data member or call the non-static method directly.

- this and super cannot be used in static context as they are non-static.

58) Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation. More Details.

59) Can we override the static methods?

No, we can't override static methods.

60) What is the static block?

Static block is used to initialize the static data member. It is executed before the main method, at the time of classloading.

```
1. class A2{
2.   static{System.out.println("static block is invoked");}
3.   public static void main(String args[]){
4.    System.out.println("Hello main");
5.   }
6. }
```

Test it Now

```
Output: static block is invoked
        Hello main
```

More Details.

61) Can we execute a program without main() method?

Ans) No, It was possible before JDK 1.7 using the static block. Since JDK 1.7, it is not possible. More Details.

62) What if the static modifier is removed from the signature of the main method?

Program compiles. However, at runtime, It throws an error "NoSuchMethodError."

63) What is the difference between static (class) method and instance method?

static or class method	instance method
1)A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
2)We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.
4)For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

64) Can we make constructors static?

As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

65) Can we make the abstract methods static in Java?

In Java, if we make the abstract methods static, It will become the part of the class, and we can directly call it which is unnecessary. Calling an undefined method is completely useless therefore it is not allowed.

66) Can we declare the static variables and methods in an abstract class?

Yes, we can declare static variables and methods in an abstract method. As we know that there is no requirement to make the object to access the static context, therefore,

we can access the static context declared inside the abstract class by using the name of the abstract class. Consider the following example.

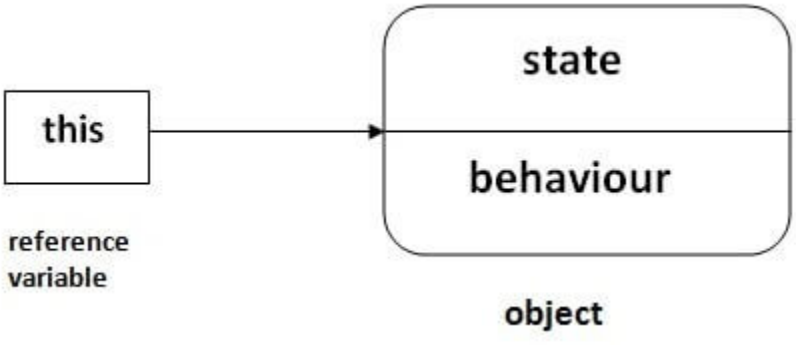
```
1. abstract class Test
2. {
3.     static int i = 102;
4.     static void TestMethod()
5.     {
6.         System.out.println("hi !! I am good !!");
7.     }
8. }
9. public class TestClass extends Test
10.{
11.    public static void main (String args[])
12.    {
13.        Test.TestMethod();
14.        System.out.println("i = "+Test.i);
15.    }
16.}
```

Output

```
hi !! I am good !!
i = 102
```

67) What is **this** keyword in java?

The **this** keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors. It can also be returned from the method as the current class instance.



More Details.

---

68) What are the main uses of this keyword?

There are the following uses of **this** keyword.

- **this** can be used to refer to the current class instance variable.
- **this** can be used to invoke current class method (implicitly)
- **this()** can be used to invoke the current class constructor.
- **this** can be passed as an argument in the method call.
- **this** can be passed as an argument in the constructor call.
- **this** can be used to return the current class instance from the method.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2) this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

4) this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

5) this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

6) this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

---

69) Can we assign the reference to **this** variable?

No, this cannot be assigned to any value because it always points to the current class object and this is the final reference in Java. However, if we try to do so, the compiler error will be shown. Consider the following example.

```
1.  public class Test
2.  {
3.      public Test()
4.      {
5.          this = null;
6.          System.out.println("Test class constructor called");
7.      }
8.      public static void main (String args[])
9.      {
10.         Test t = new Test();
11.     }
12. }
```

Output

```
Test.java:5: error: cannot assign a value to final variable this
        this = null;
        ^
1 error
```

70) Can **this** keyword be used to refer static members?

Yes, It is possible to use this keyword to refer static members because this is just a reference variable which refers to the current class object. However, as we know that,

it is unnecessary to access static variables through objects, therefore, it is not the best practice to use this to refer static members. Consider the following example.

```
1. public class Test
2. {
3.     static int i = 10;
4.     public Test ()
5.     {
6.         System.out.println(this.i);
7.     }
8.     public static void main (String args[])
9.     {
10.        Test t = new Test();
11.    }
12.}
```

Output

```
10
```

71) How can constructor chaining be done using this keyword?

Constructor chaining enables us to call one constructor from another constructor of the class with respect to the current class object. We can use this keyword to perform constructor chaining within the same class. Consider the following example which illustrates how can we use this keyword to achieve constructor chaining.

```
1. public class Employee
2. {
3.     int id,age;
4.     String name, address;
5.     public Employee (int age)
6.     {
7.         this.age = age;
8.     }
9.     public Employee(int id, int age)
10.    {
11.        this(age);
12.        this.id = id;
```

```
13.    }
14.    public Employee(int id, int age, String name, String address)
15.    {
16.        this(id, age);
17.        this.name = name;
18.        this.address = address;
19.    }
20.    public static void main (String args[])
21.    {
22.        Employee emp = new Employee(105, 22, "Vikas", "Delhi");
23.        System.out.println("ID: "+emp.id+" Name:"+emp.name+" age:"+emp.age
    +" address: "+emp.address);
24.    }
25.
26.}
```

Output

```
ID: 105 Name:Vikas age:22 address: Delhi
```

72) What are the advantages of passing this into a method instead of the current class object itself?

As we know, that this refers to the current class object, therefore, it must be similar to the current class object. However, there can be two main advantages of passing this into a method instead of the current class object.

- this is a final variable. Therefore, this cannot be assigned to any new value whereas the current class object might not be final and can be changed.
- this can be used in the synchronized block.

73) What is the Inheritance?

Inheritance is a mechanism by which one object acquires all the properties and behavior of another object of another class. It is used for Code Reusability and Method Overriding. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields

in your current class also. Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

There are five types of inheritance in Java.

- **Single-level inheritance**
- **Multi-level inheritance**
- **Multiple Inheritance**
- **Hierarchical Inheritance**
- **Hybrid Inheritance**

Multiple inheritance is not supported in Java through class.

More Details.

---

74) Why is Inheritance used in Java?

There are various advantages of using inheritance in Java that is given below.

- Inheritance provides code reusability. The derived class does not need to redefine the method of base class unless it needs to provide the specific implementation of the method.
- Runtime polymorphism cannot be achieved without using inheritance.
- We can simulate the inheritance of classes with the real-time objects which makes OOPs more realistic.
- Inheritance provides data hiding. The base class can hide some data from the derived class by making it private.
- Method overriding cannot be achieved without inheritance. By method overriding, we can give a specific implementation of some basic method contained by the base class.

---

75) Which class is the superclass for all the classes?

The object class is the superclass of all other classes in Java.

---

76) Why is multiple inheritance not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since the compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have the same method or different, there will be a compile time error.

```
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
8.
9.  Public Static void main(String args[]){
10.  C obj=new C();
11.  obj.msg();//Now which msg() method would be invoked?
12.}
13.}
```

Test it Now

```
Compile Time Error
```

77) What is aggregation?

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship. For example, The aggregate class Employee having various fields such as age, name, and salary also contains an object of Address class having various fields such as Address-Line 1, City, State, and pin-code. In other words, we can say that Employee (class) has an object of Address class. Consider the following example.

Address.java

```
1. public class Address {
2. String city,state,country;
3.
```

```
4.  public Address(String city, String state, String country) {
5.      this.city = city;
6.      this.state = state;
7.      this.country = country;
8.  }
9.
10. }
```

Employee.java

```
1.  public class Emp {
2.  int id;
3.  String name;
4.  Address address;
5.
6.  public Emp(int id, String name,Address address) {
7.      this.id = id;
8.      this.name = name;
9.      this.address=address;
10. }
11.
12. void display(){
13. System.out.println(id+" "+name);
14. System.out.println(address.city+" "+address.state+" "+address.country);
15. }
16.
17. public static void main(String[] args) {
18. Address address1=new Address("gzb","UP","india");
19. Address address2=new Address("gno","UP","india");
20.
21. Emp e=new Emp(111,"varun",address1);
22. Emp e2=new Emp(112,"arun",address2);
23.
24. e.display();
25. e2.display();
26.
27. }
```

```
28.}
```

Output

```
111 varun
gzb UP india
112 arun
gno UP india
```

78) What is composition?

Holding the reference of a class within some other class is known as composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. In other words, we can say that composition is the particular case of aggregation which represents a stronger relationship between two objects. Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

79) What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

80) Why does Java not support pointers?

The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe(unsecured) and complex to understand.

81) What is super in java?

The **super** keyword in Java is a reference variable that is used to refer to the immediate parent class object. Whenever you create the instance of the subclass, an instance of the parent class is created implicitly which is referred by super reference variable. The super() is called in the class constructor implicitly by the compiler if there is no super or this.

```
1. class Animal{
2. Animal(){System.out.println("animal is created");}
```

```
3.  }
4.  class Dog extends Animal{
5.  Dog(){
6.  System.out.println("dog is created");
7.  }
8.  }
9.  class TestSuper4{
10. public static void main(String args[]){
11. Dog d=new Dog();
12. }
13. }
```

Test it Now

Output:

```
animal is created
dog is created
```

More Details.

---

82) How can constructor chaining be done by using the super keyword?

```
1.  class Person
2.  {
3.      String name,address;
4.      int age;
5.      public Person(int age, String name, String address)
6.      {
7.          this.age = age;
8.          this.name = name;
9.          this.address = address;
10.     }
11. }
12. class Employee extends Person
13. {
14.     float salary;
15.     public Employee(int age, String name, String address, float salary)
16.     {
17.         super(age,name,address);
```

```
18.         this.salary = salary;
19.     }
20. }
21. public class Test
22. {
23.     public static void main (String args[])
24.     {
25.         Employee e = new Employee(22, "Mukesh", "Delhi", 90000);
26.         System.out.println("Name: "+e.name+" Salary: "+e.salary+" Age: "+e.a
    ge+" Address: "+e.address);
27.     }
28. }
```

Output

```
Name: Mukesh Salary: 90000.0 Age: 22 Address: Delhi
```

83) What are the main uses of the super keyword?

There are the following uses of super keyword.

- super can be used to refer to the immediate parent class instance variable.
- super can be used to invoke the immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

84) What are the differences between this and super keyword?

There are the following differences between this and super keyword.

- The super keyword always points to the parent class contexts whereas this keyword always points to the current class context.
- The super keyword is primarily used for initializing the base class variables within the derived class constructor whereas this keyword primarily used to differentiate between local and instance variables when passed in the class constructor.
- The super and this must be the first statement inside constructor otherwise the compiler will throw an error.

85) What is the output of the following Java program?

```
1. class Person
2. {
3.     public Person()
4.     {
5.         System.out.println("Person class constructor called");
6.     }
7. }
8. public class Employee extends Person
9. {
10.    public Employee()
11.    {
12.        System.out.println("Employee class constructor called");
13.    }
14.    public static void main (String args[])
15.    {
16.        Employee e = new Employee();
17.    }
18.}
```

Output

```
Person class constructor called
Employee class constructor called
```

Explanation

The super() is implicitly invoked by the compiler if no super() or this() is included explicitly within the derived class constructor. Therefore, in this case, The Person class constructor is called first and then the Employee class constructor is called.

---

86) Can you use this() and super() both in a constructor?

No, because this() and super() must be the first statement in the class constructor.

Example:

```
1. public class Test{
2.     Test()
```

```
3.      {
4.          super();
5.          this();
6.          System.out.println("Test class object is created");
7.      }
8.      public static void main(String []args){
9.      Test t = new Test();
10.     }
11. }
```

Output:

```
Test.java:5: error: call to this must be first statement in constructor
```

87)What is object cloning?

The object cloning is used to create the exact copy of an object. The clone() method of the Object class is used to clone an object. The **java.lang.Cloneable** interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

```
1. protected Object clone() throws CloneNotSupportedException
2.
```

More Details.

88) What is method overloading?

Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature. We can achieve method overloading in two ways.

- **By Changing the number of arguments**
- **By Changing the data type of arguments**

Method overloading increases the readability of the program. Method overloading is performed to figure out the program quickly.

More Details.

89) Why is method overloading not possible by changing the return type in java?

In Java, method overloading is not possible by changing the return type of the program due to avoid the ambiguity.

```
1. class Adder{
2. static int add(int a,int b){return a+b;}
3. static double add(int a,int b){return a+b;}
4. }
5. class TestOverloading3{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));//ambiguity
8. }}
```

Test it Now

Output:

```
Compile Time Error: method add(int, int) is already defined in class Adder
```

More Details.

---

90) Can we overload the methods by making them static?

No, We cannot overload the methods by just applying the static keyword to them(number of parameters and types are the same). Consider the following example.

```
1. public class Animal
2. {
3.     void consume(int a)
4.     {
5.         System.out.println(a+" consumed!!");
6.     }
7.     static void consume(int a)
8.     {
9.         System.out.println("consumed static "+a);
10.    }
11.    public static void main (String args[])
12.    {
13.        Animal a = new Animal();
```

```
14.     a.consume(10);
15.     Animal.consume(20);
16.  }
17.}
```

Output

```
Animal.java:7: error: method consume(int) is already defined in class Animal
    static void consume(int a)
                ^
Animal.java:15: error: non-static method consume(int) cannot be referenced
from a static context
        Animal.consume(20);
               ^
2 errors
```

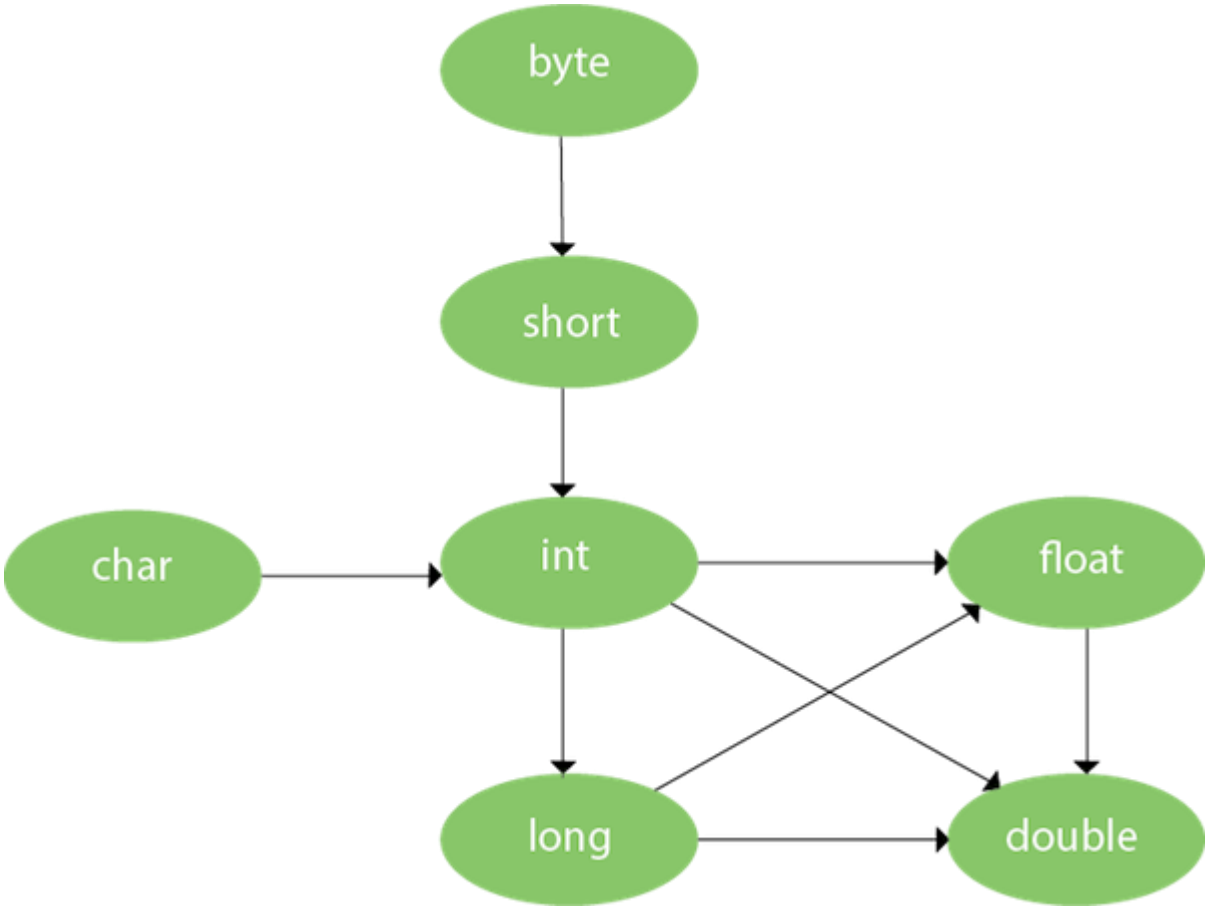
91) Can we overload the main() method?

Yes, we can have any number of main methods in a Java program by using method overloading.

More Details.

92) What is method overloading with type promotion?

By Type promotion is method overloading, we mean that one data type can be promoted to another implicitly if no exact matching is found.



As displayed in the above diagram, the byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on. Consider the following example.

```
1. class OverloadingCalculation1{
2.   void sum(int a,long b){System.out.println(a+b);}
3.   void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5.   public static void main(String args[]){
6.   OverloadingCalculation1 obj=new OverloadingCalculation1();
7.   obj.sum(20,20);//now second int literal will be promoted to long
8.   obj.sum(20,20,20);
9.   }
10.}
```

Test it Now

Output

```
40
60
```

93) What is the output of the following Java program?

```
1. class OverloadingCalculation3{
2.   void sum(int a,long b){System.out.println("a method invoked");}
3.   void sum(long a,int b){System.out.println("b method invoked");}
4.
5.   public static void main(String args[]){
6.   OverloadingCalculation3 obj=new OverloadingCalculation3();
7.   obj.sum(20,20);//now ambiguity
8.   }
9. }
```

Output

```
OverloadingCalculation3.java:7: error: reference to sum is ambiguous
obj.sum(20,20);//now ambiguity
     ^
      both method sum(int,long) in OverloadingCalculation3
      and method sum(long,int) in OverloadingCalculation3 match
1 error
```

Explanation

There are two methods defined with the same name, i.e., sum. The first method accepts the integer and long type whereas the second method accepts long and the integer type. The parameter passed that are a = 20, b = 20. We can not tell that which method will be called as there is no clear differentiation mentioned between integer literal and long literal. This is the case of ambiguity. Therefore, the compiler will throw an error.

94) What is method overriding:

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods.

Rules for Method overriding

- The method must have the same name as in the parent class.
- The method must have the same signature as in the parent class.
- Two classes must have an IS-A relationship between them.

More Details.

95) Can we override the static method?

No, you can't override the static method because they are the part of the class, not the object.

96) Why can we not override static method?

It is because the static method is the part of the class, and it is bound with class whereas instance method is bound with the object, and static gets memory in class area, and instance gets memory in a heap.

97) Can we override the overloaded method?

Yes.

98) Difference between method Overloading and Overriding.

Method Overloading	Method Overriding
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes that have IS-A relationship between them.
3) In this case, the parameters must be different.	In this case, the parameters must be the same.
4) Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5) In Java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method	Return type must be same or covariant in method overriding.

overloading. But you must have to change the parameter.	

99) Can we override the private methods?

No, we cannot override the private methods because the scope of private methods is limited to the class and we cannot access them outside of the class.

100) Can we change the scope of the overridden method in the subclass?

Yes, we can change the scope of the overridden method in the subclass. However, we must notice that we cannot decrease the accessibility of the method. The following point must be taken care of while changing the accessibility of the method.

- The private can be changed to protected, public, or default.
- The protected can be changed to public or default.
- The default can be changed to public.
- The public will always remain public.

101) Can we modify the throws clause of the superclass method while overriding it in the subclass?

Yes, we can modify the throws clause of the superclass method while overriding it in the subclass. However, there are some rules which are to be followed while overriding in case of exception handling.

- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception, but it can declare the unchecked exception.
- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

102) What is the output of the following Java program?

```
1. class Base
```

```
{
    void method(int a)
    {
        System.out.println("Base class method called with integer a = "+a);
    }

    void method(double d)
    {
        System.out.println("Base class method called with double d ="+d);
    }
}

class Derived extends Base
{
    @Override
    void method(double d)
    {
        System.out.println("Derived class method called with double d ="+d);
    }
}

public class Main
{
    public static void main(String[] args)
    {
        new Derived().method(10);
    }
}
```

Output

```
Base class method called with integer a = 10
```

Explanation

The method() is overloaded in class Base whereas it is derived in class Derived with the double type as the parameter. In the method call, the integer is passed.

103) Can you have virtual functions in Java?

Yes, all functions in Java are virtual by default.

---

104) What is covariant return type?

Now, since java5, it is possible to override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type. The covariant return type specifies that the return type may vary in the same direction as the subclass.

```
1. class A{
2. A get(){return this;}
3. }
4.
5. class B1 extends A{
6. B1 get(){return this;}
7. void message(){System.out.println("welcome to covariant return type");}
8.
9. public static void main(String args[]){
10. new B1().get().message();
11. }
12. }
```

Test it Now

```
Output: welcome to covariant return type
```

More Details.

---

105) What is the output of the following Java program?

```
1. class Base
2. {
3.     public void baseMethod()
4.     {
5.         System.out.println("BaseMethod called ...");
6.     }
7. }
8. class Derived extends Base
9. {
```

```
10.    public void baseMethod()
11.    {
12.        System.out.println("Derived method called ...");
13.    }
14. }
15. public class Test
16. {
17.    public static void main (String args[])
18.    {
19.        Base b = new Derived();
20.        b.baseMethod();
21.    }
22. }
```

Output

```
Derived method called ...
```

Explanation

The method of Base class, i.e., baseMethod() is overridden in Derived class. In Test class, the reference variable b (of type Base class) refers to the instance of the Derived class. Here, Runtime polymorphism is achieved between class Base and Derived. At compile time, the presence of method baseMethod checked in Base class, If it presence then the program compiled otherwise the compiler error will be shown. In this case, baseMethod is present in Base class; therefore, it is compiled successfully. However, at runtime, It checks whether the baseMethod has been overridden by Derived class, if so then the Derived class method is called otherwise Base class method is called. In this case, the Derived class overrides the baseMethod; therefore, the Derived class method is called.

106) What is the final variable?

In Java, the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its value. In other words, we can say that the final variable once assigned to a value, can never be changed after that. The final variable which is not assigned to any value can only be assigned through the class constructor.


```
1.  class Bike9{
2.   final int speedlimit=90;//final variable
3.   void run(){
4.    speedlimit=400;
5.   }
6.   public static void main(String args[]){
7.   Bike9 obj=new  Bike9();
8.   obj.run();
9.   }
10. }//end of class
```

Test it Now

```
Output:Compile Time Error
```

More Details.

---

107) What is the final method?

If we change any method to a final method, we can't override it. More Details.

```
1.  class Bike{
2.    final void run(){System.out.println("running");}
3.  }
4.
5.  class Honda extends Bike{
6.    void run(){System.out.println("running safely with 100kmph");}
7.
8.    public static void main(String args[]){
```

```
9.    Honda honda= new Honda();
10.   honda.run();
11.   }
12.}
```

Test it Now

```
Output:Compile Time Error
```

---

108) What is the final class?

If we make any class final, we can't inherit it into any of the subclasses.

```
1.  final class Bike{}
2.
3.  class Honda1 extends Bike{
4.    void run(){System.out.println("running safely with 100kmph");}
5.
6.    public static void main(String args[]){
7.    Honda1 honda= new Honda1();
8.    honda.run();
9.    }
10.}
```

Test it Now

```
Output:Compile Time Error
```

More Details.

---

109) What is the final blank variable?

A final variable, not initialized at the time of declaration, is known as the final blank variable. We can't initialize the final blank variable directly. Instead, we have to initialize it by using the class constructor. It is useful in the case when the user has some data which must not be changed by others, for example, PAN Number. Consider the following example:

```
1.  class Student{
2.  int id;
3.  String name;
4.  final String PAN_CARD_NUMBER;
5.  ...
6.  }
```

More Details.

110) Can we initialize the final blank variable?

Yes, if it is not static, we can initialize it in the constructor. If it is static blank final variable, it can be initialized only in the static block. More Details.

111) Can you declare the main method as final?

Yes, We can declare the main method as public static final void main(String[] args){}.

112) What is the output of the following Java program?

```
1. class Main {
2.  public static void main(String args[]){
3.    final int i;
4.    i = 20;
5.    System.out.println(i);
6.  }
7. }
```

Output

```
20
```

Explanation

Since i is the blank final variable. It can be initialized only once. We have initialized it to 20. Therefore, 20 will be printed.

113) What is the output of the following Java program?

```
1. class Base
2. {
3.    protected final void getInfo()
4.    {
5.        System.out.println("method of Base class");
6.    }
7. }
```

```
8.
9.  public class Derived extends Base
10. {
11.     protected final void getInfo()
12.     {
13.         System.out.println("method of Derived class");
14.     }
15.     public static void main(String[] args)
16.     {
17.         Base obj = new Base();
18.         obj.getInfo();
19.     }
20. }
```

Output

```
        Derived.java:11:  error:  getInfo()  in  Derived  cannot  override
getInfo() in Base
    protected final void getInfo()
                         ^
  overridden method is final
1 error
```

Explanation

The getDetails() method is final; therefore it can not be overridden in the subclass.

---

114) Can we declare a constructor as final?

The constructor can never be declared as final because it is never inherited. Constructors are not ordinary methods; therefore, there is no sense to declare constructors as final. However, if you try to do so, The compiler will throw an error.

---

115) Can we declare an interface as final?

No, we cannot declare an interface as final because the interface must be implemented by some class to provide its definition. Therefore, there is no sense to make an interface final. However, if you try to do so, the compiler will show an error.

---

116) What is the difference between the final method and abstract method?

The main difference between the final method and abstract method is that the abstract method cannot be final as we need to override them in the subclass to give its definition.

117) What is the difference between compile-time polymorphism and runtime polymorphism?

There are the following differences between compile-time polymorphism and runtime polymorphism.

SN	compile-time polymorphism	Runtime polymorphism
1	In compile-time polymorphism, call to a method is resolved at compile-time.	In runtime polymorphism, call to an overridden method is resolved at runtime.
2	It is also known as static binding, early binding, or overloading.	It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch.
3	Overloading is a way to achieve compile-time polymorphism in which, we can define multiple methods or constructors with different signatures.	Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class.
4	It provides fast execution because the type of an object is determined at compile-time.	It provides slower execution as compare to compile-time because the type of an object is determined at run-time.
5	Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time.	Run-time polymorphism provides more flexibility because all the things are resolved at runtime.

118) What is Runtime Polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

```
1. class Bike{
2.   void run(){System.out.println("running");}
3. }
4. class Splendor extends Bike{
5.   void run(){System.out.println("running safely with 60km");}
6.   public static void main(String args[]){
7.     Bike b = new Splendor();//upcasting
8.     b.run();
9.   }
10.}
```

Test it Now

Output:

```
running safely with 60km.
```

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

More details.

---

119) Can you achieve Runtime Polymorphism by data members?

No, because method overriding is used to achieve runtime polymorphism and data members cannot be overridden. We can override the member functions but not the data members. Consider the example given below.

```
1. class Bike{
2.  int speedlimit=90;
3. }
4. class Honda3 extends Bike{
```

```
5.  int speedlimit=150;
6.  public static void main(String args[]){
7.   Bike obj=new Honda3();
8.   System.out.println(obj.speedlimit);//90
9.   }
```

Test it Now

Output:

```
90
```

More details.

---

120) What is the difference between static binding and dynamic binding?

In case of the static binding, the type of the object is determined at compile-time whereas, in the dynamic binding, the type of the object is determined at runtime.

Static Binding

```
1.  class Dog{
2.   private void eat(){System.out.println("dog is eating...");}
3.
4.   public static void main(String args[]){
5.    Dog d1=new Dog();
6.    d1.eat();
7.   }
8.  }
```

Dynamic Binding

```
1.  class Animal{
2.   void eat(){System.out.println("animal is eating...");}
3.  }
4.
5.  class Dog extends Animal{
6.   void eat(){System.out.println("dog is eating...");}
7.
8.   public static void main(String args[]){
9.    Animal a=new Dog();
```

```
10.  a.eat();
11. }
12.}
```

More details.

---

121) What is the output of the following Java program?

```
1.  class BaseTest
2.  {
3.    void print()
4.    {
5.      System.out.println("BaseTest:print() called");
6.    }
7.  }
8.  public class Test extends BaseTest
9.  {
10.  void print()
11.  {
12.    System.out.println("Test:print() called");
13.  }
14.  public static void main (String args[])
15.  {
16.    BaseTest b = new Test();
17.    b.print();
18.  }
19.}
```

Output

```
Test:print() called
```

Explanation

It is an example of Dynamic method dispatch. The type of reference variable b is determined at runtime. At compile-time, it is checked whether that method is present in the Base class. In this case, it is overridden in the child class, therefore, at runtime the derived class method is called.

---

122) What is Java instanceOf operator?

The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has a null value, it returns false. Consider the following example.

```
1. class Simple1{
2.  public static void main(String args[]){
3.  Simple1 s=new Simple1();
4.  System.out.println(s instanceof Simple1);//true
5.  }
6. }
```

Test it Now

Output

```
true
```

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

123) What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it. Abstraction lets you focus on what the object does instead of how it does it.

In Java, there are two ways to achieve the abstraction.

- **Abstract Class**
- **Interface**

More details.

124) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

More details.

---

125) What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method. Consider the following example.

```
1. abstract class Bike{
2.   abstract void run();
3. }
4. class Honda4 extends Bike{
5. void run(){System.out.println("running safely");}
6. public static void main(String args[]){
7.  Bike obj = new Honda4();
8.  obj.run();
9. }
10.}
```

Test it Now

Output

```
running safely
```

More details.

---

126) Can there be an abstract method without an abstract class?

No, if there is an abstract method in a class, that class must be abstract.

---

127) Is the following program written correctly? If yes then what will be the output of the program?

```
1. abstract class Calculate
2. {
3.     abstract int multiply(int a, int b);
4. }
```

```
5.
6.  public class Main
7.  {
8.      public static void main(String[] args)
9.      {
10.         int result = new Calculate()
11.         {
12.             @Override
13.             int multiply(int a, int b)
14.             {
15.                 return a*b;
16.             }
17.         }.multiply(12,32);
18.         System.out.println("result = "+result);
19.     }
20. }
```

Yes, the program is written correctly. The Main class provides the definition of abstract method multiply declared in abstract class Calculation. The output of the program will be:

Output

```
384
```

128) Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

129) Is it possible to instantiate the abstract class?

No, the abstract class can never be instantiated even if it contains a constructor and all of its methods are implemented.

130) What is the interface?

- The interface is a blueprint for a class that has static constants and abstract methods.
- It can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

More details.

---

131) Can you declare an interface method static?

No, because methods of an interface are abstract by default, and we can not use static and abstract together.

---

132) Can the Interface be final?

No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

---

133) What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

```
1. public interface Serializable{
2. }
```

---

134) What are the differences between abstract class and interface?

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

Abstract class	Interface
An abstract class can have abstract and non-abstract methods.	The interface has only abstract methods. Since Java8, it can have default and static method also.
An abstract class can have instance variables.	An interface cannot have instance variables.
An abstract class can have the constructor.	The interface cannot have the constructor.
An abstract class can have static methods.	The interface cannot have static methods.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
The abstract class **can provide the implementation of the interface**.	The Interface **can't provide the implementation of the abstract class**.
The **abstract keyword** is used to declare an abstract class.	The **interface keyword** is used to declare an interface.
An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
An **abstract class** can be extended using keyword **extends**	An **interface class** can be implemented using keyword **implements**
A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example: **public abstract class Shape{** **public abstract void draw();** **}**	Example: **public interface Drawable{** **void draw();** **}**

135) Can we define private and protected modifiers for the members in interfaces?

No, they are implicitly public.

136) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

137) How to make a read-only class in Java?

A class can be made read-only by making all of the fields private. The read-only class will have only getter methods which return the private property of the class to the main method. We cannot modify this property because there is no setter method available in the class. Consider the following example.

```
1.  //A Java class which has only getter methods.
2. public class Student{
3. //private data member
4. private String college="AKG";
5. //getter method for college
6. public String getCollege(){
7. return college;
8. }
9. }
```

138) How to make a write-only class in Java?

A class can be made write-only by making all of the fields private. The write-only class will have only setter methods which set the value passed from the main method to the

private fields. We cannot read the properties of the class because there is no getter method in this class. Consider the following example.

```
1.  //A Java class which has only setter methods.
2. public class Student{
3. //private data member
4. private String college;
5. //getter method for college
6. public void setCollege(String college){
7. this.college=college;
8. }
9. }
```

139) What are the advantages of Encapsulation in Java?

There are the following advantages of Encapsulation in Java?

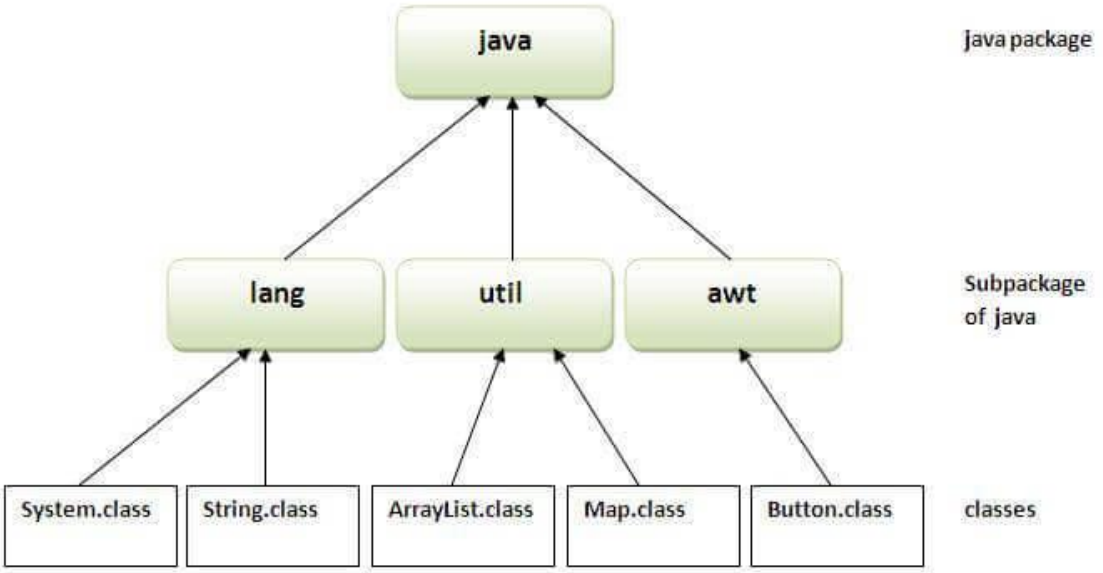
- By providing only the setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
- It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is easy to test. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

140) What is the package?

A package is a group of similar type of classes, interfaces, and sub-packages. It provides access protection and removes naming collision. The packages in Java can be categorized into two forms, inbuilt package, and user-defined package. There are many built-in packages such as Java, lang, awt, javax, swing, net, io, util, sql, etc. Consider the following example to create a package in Java.

```
1. //save as Simple.java
2. package mypack;
```

```
3.  public class Simple{
4.   public static void main(String args[]){
5.      System.out.println("Welcome to package");
6.    }
7.  }
```



More details.

141) What are the advantages of defining packages in Java?

By defining packages, we can avoid the name conflicts between the same class names defined in different packages. Packages also enable the developer to organize the similar classes more effectively. For example, one can clearly understand that the classes present in java.io package are used to perform io related operations.

142) How to create packages in Java?

If you are using the programming IDEs like Eclipse, NetBeans, MyEclipse, etc. click on **file->new->project** and eclipse will ask you to enter the name of the package. It will create the project package containing various directories such as src, etc. If you are using an editor like notepad for java programming, use the following steps to create the package.

- **Define a package** package_name**. Create the class with the name** class_name **and save this file with** your_class_name.java**.**

- **Now compile the file by running the following command on the terminal.**
    1. javac -d . your_class_name.java

    The above command creates the package with the name **package_name** in the present working directory.

- **Now, run the class file by using the absolute class file name, like following.**
    1. java package_name.class_name

---

143) How can we access some class in another class in Java?

There are two ways to access a class in another class.

- **By using the fully qualified name:** To access a class in a different package, either we must use the fully qualified name of that class, or we must import the package containing that class.
- **By using the relative path**, We can use the path of the class that is related to the package that contains our class. It can be the same or subpackage.

---

144) Do I need to import java.lang package any time? Why?

No. It is by default loaded internally by the JVM.

---

145) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or the same class multiple times. Neither compiler nor JVM complains about it. However, the JVM will internally load the class only once no matter how many times you import the same class.

---

146) What is the static import?

By static import, we can access the static members of a class directly, and there is no to qualify it with the class name.

More details.

147) What is wrapper class?

Wrapper classes are classes that allow primitive types to be accessed as objects.

148) What is autoboxing and unboxing?

The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

149) Difference between Object and Instance?

- ✓ Object is a real copy of class but instance is a virtual copy of class.
- ✓ A blueprint for a house design is like a class description. All the houses built from that blueprint are objects of that class. A given house is an instance.
- ✓ Instance is logical but object is physical means occupies some memory.
- ✓ We can create an instance for abstract class as well for interface, but we cannot create an object for those.
- ✓ Object is an instance of class and instance means representative of class i.e. object.
- ✓ Instance refers to reference of an object.
- ✓ Object is actually pointing to memory address of that instance.
- ✓ You can't store an instance but you can store an object.
- ✓ A single object can have more than one instance.
- ✓ You can't pass instance over the layers but you can pass the object over the layers.

150) Full form of oops?

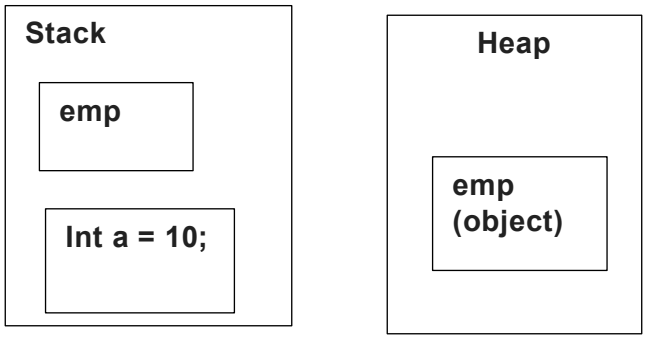
Object-oriented Programming System.

151) Difference between Reference and Object?

Employee emp; // reference variable

Employee emp = new Employee(); // Object

Int a = 10;



152) What are the advantages of Java inner classes?

There are two types of advantages of Java inner classes.

- Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of the outer class including private.
- Nested classes are used to develop a more readable and maintainable code because it logically groups classes and interfaces in one place only.
- Code Optimization: It requires less code to write.

---

153) What is a nested class?

The nested class can be defined as the class which is defined inside another class or interface. We use the nested class to logically group classes and interfaces in one place so that it can be more readable and maintainable. A nested class can access all the data members of the outer class including private data members and methods. The syntax of the nested class is defined below.

```
1. class Java_Outer_class{
2.  //code
3.  class Java_Nested_class{
4.   //code
```

```
5.  }
6. }
7.
```

There are two types of nested classes, static nested class, and non-static nested class. The non-static nested class can also be called as inner-class

More details.

---

154) What are the disadvantages of using inner classes?

There are the following main disadvantages of using inner classes.

- **Inner classes increase the total number of classes used by the developer and therefore increases the workload of JVM since it has to perform some routine operations for those extra classes which result in slower performance.**
- **IDEs provide less support to the inner classes as compare to the top level classes and therefore it annoys the developers while working with inner classes.**

---

155) What are the types of inner classes (non-static nested class) used in Java?

There are mainly three types of inner classes used in Java.

Type	Description
Member Inner Class	**A class created within class and outside method.**
Anonymous Inner Class	**A class created for implementing an interface or extending class. Its name is decided by the java compiler.**
Local Inner Class	**A class created within the method.**

---

156) Is there any difference between nested classes and inner classes?

Yes, inner classes are non-static nested classes. In other words, we can say that inner classes are the part of nested classes.

More details.

---

157) Can we access the non-final local variable, inside the local inner class?

No, the local variable must be constant if you want to access it in the local inner class.

More details.

---

158) How many class files are created on compiling the OuterClass in the following program?

```
1.  public class Person {
2.  String name, age, address;
3.  class Employee{
4.    float salary=10000;
5.  }
6.  class BusinessMen{
7.    final String gstin="£4433drt3$";
8.  }
9.  public static void main (String args[])
10. {
11.   Person p = new Person();
12. }
13. }
```

3 class-files will be created named as Person.class, Person$BusinessMen.class, and Person$Employee.class.

---

159) What are anonymous inner classes?

Anonymous inner classes are the classes that are automatically declared and instantiated within an expression. We cannot apply different access modifiers to them.

Anonymous class cannot be static, and cannot define any static fields, method, or class. In other words, we can say that it a class without the name and can have only one object that is created by its definition. Consider the following example.

```
1. abstract class Person{
2.   abstract void eat();
3. }
4. class TestAnonymousInner{
5.  public static void main(String args[]){
6.   Person p=new Person(){
7.   void eat(){System.out.println("nice fruits");}
8.   };
9.   p.eat();
10. }
11.}
```

Test it Now

Output:

```
nice fruits
```

Consider the following example for the working of the anonymous class using interface.

```
1. interface Eatable{
2.  void eat();
3. }
4. class TestAnnonymousInner1{
5.  public static void main(String args[]){
6.  Eatable e=new Eatable(){
7.   public void eat(){System.out.println("nice fruits");}
8.  };
9.  e.eat();
10. }
11.}
```

Test it Now

Output:

```
nice fruits
```

160) What is the nested interface?

An Interface that is declared inside the interface or class is known as the nested interface. It is static by default. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The external interface or class must refer to the nested interface. It can't be accessed directly. The nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class. The syntax of the nested interface is given as follows.

```
1. interface interface_name{
2.  ...
3.  interface nested_interface_name{
4.   ...
5.  }
6. }
7.
```

More details.

---

161) Can a class have an interface?

Yes, an interface can be defined within the class. It is called a nested interface.

More details.

---

162) Can an Interface have a class?

Yes, they are static implicitly.

More details.

163) What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management.

More details.

164) What is gc()?

The gc() method is used to invoke the garbage collector for cleanup processing. This method is found in System and Runtime classes. This function explicitly makes the Java Virtual Machine free up the space occupied by the unused objects so that it can be utilized or reused. Consider the following example for the better understanding of how the gc() method invoke the garbage collector.

```
1. public class TestGarbage1{
2.  public void finalize(){System.out.println("object is garbage collected");}
3.  public static void main(String args[]){
4.   TestGarbage1 s1=new TestGarbage1();
5.   TestGarbage1 s2=new TestGarbage1();
6.   s1=null;
7.   s2=null;
8.   System.gc();
9.  }
10.}
```

Test it Now

```
object is garbage collected
object is garbage collected
```

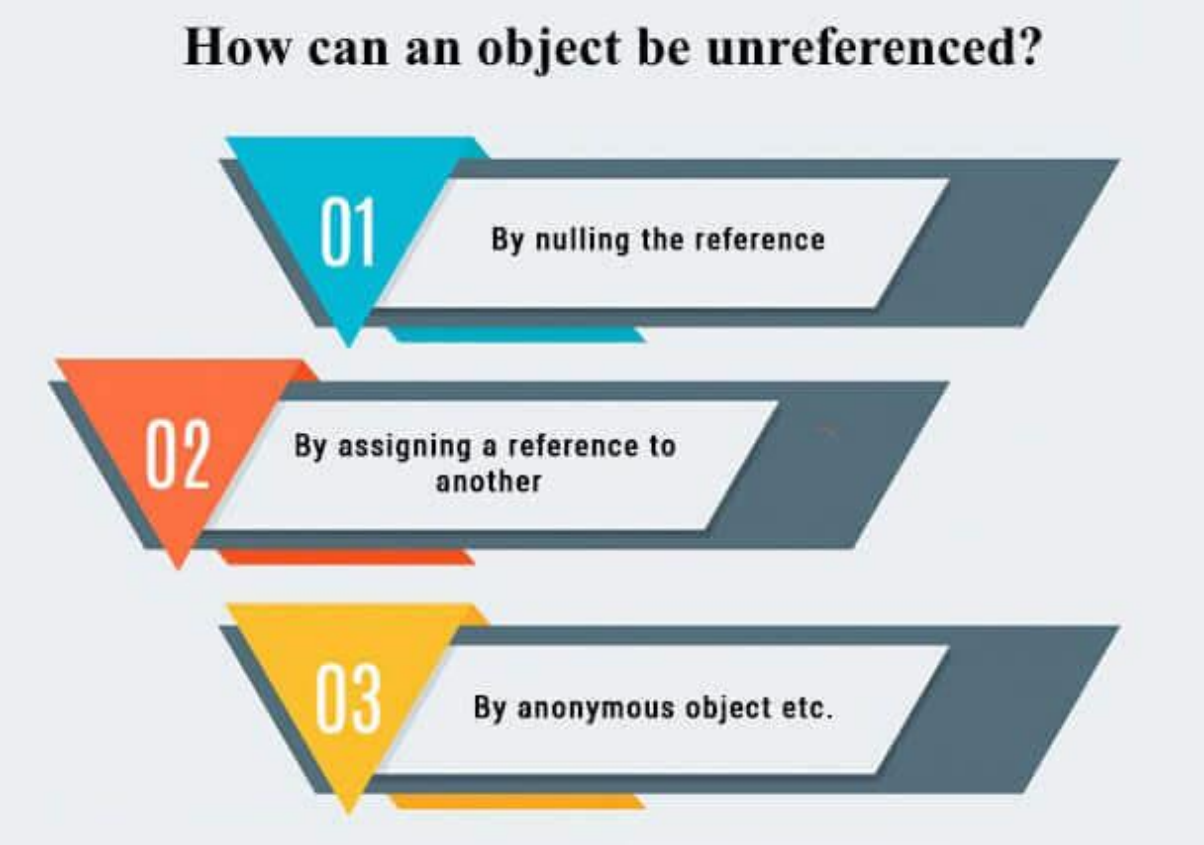
165) How is garbage collection controlled?

Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

166) How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.



1) By nulling a reference:

1. Employee e=**new** Employee();
2. **e=**null**;**

2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. **Employee e2=**new **Employee();**
3. e1=e2;//now the first object referred by e1 is available for garbage collection

3) By anonymous object:

1. **new** Employee();

---

167) What is the purpose of the finalize() method?

The finalize() method is invoked just before the object is garbage collected. It is used to perform cleanup processing. The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created an object without new, you can use the finalize method to perform cleanup processing (destroying remaining objects). The cleanup processing is the process to free up all the resources, network which was previously used and no longer needed. It is essential to remember that it is not a reserved keyword, finalize method is present in the object class hence it is available in every class as object class is the superclass of every class in java. Here,

we must note that neither finalization nor garbage collection is guaranteed. Consider the following example.

```
1. public class FinalizeTest {
2.     int j=12;
3.     void add()
4.     {
5.         j=j+12;
6.         System.out.println("J="+j);
7.     }
8.     public void finalize()
9.     {
10.        System.out.println("Object is garbage collected");
11.    }
12.    public static void main(String[] args) {
13.        new FinalizeTest().add();
14.        System.gc();
15.        new FinalizeTest().add();
16.    }
17.}
18.
```

168) Can an unreferenced object be referenced again?

Yes,

169) What kind of thread is the Garbage collector thread?

Daemon thread.

170) What is the difference between final, finally and finalize?

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method, and variable. The final class	Finally is used to place important code, it will be	Finalize is used to perform clean up processing just

	can't be inherited, final method can't be overridden, and final variable value can't be changed.	executed whether an exception is handled or not.	before an object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

171) What is the purpose of the Runtime class?

Java Runtime class is used to interact with a java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory, etc. There is only one instance of java.lang.Runtime class is available for one java application. The Runtime.getRuntime() method returns the singleton instance of Runtime class.

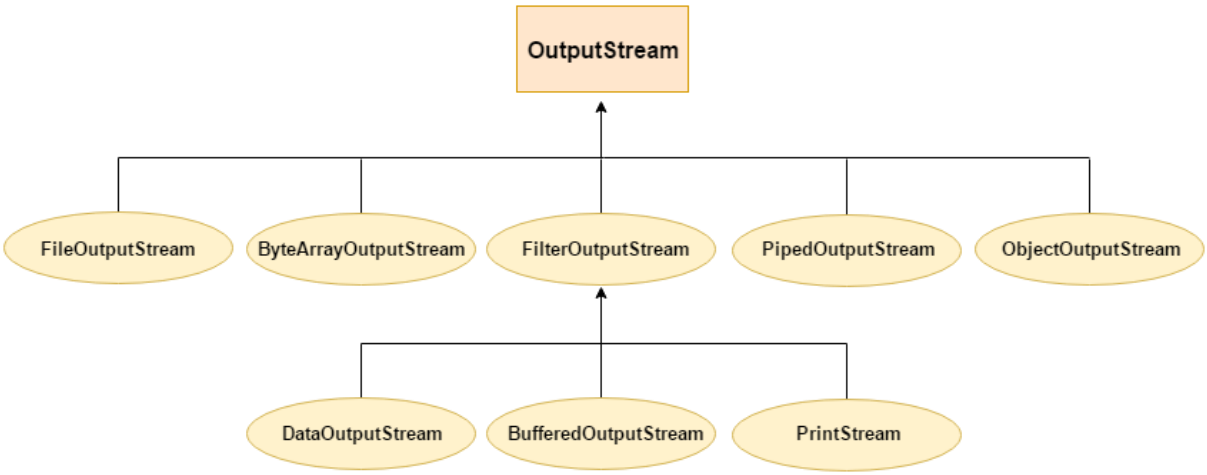
172) How will you invoke any external process in Java?

By Runtime.getRuntime().exec(?) method. Consider the following example.

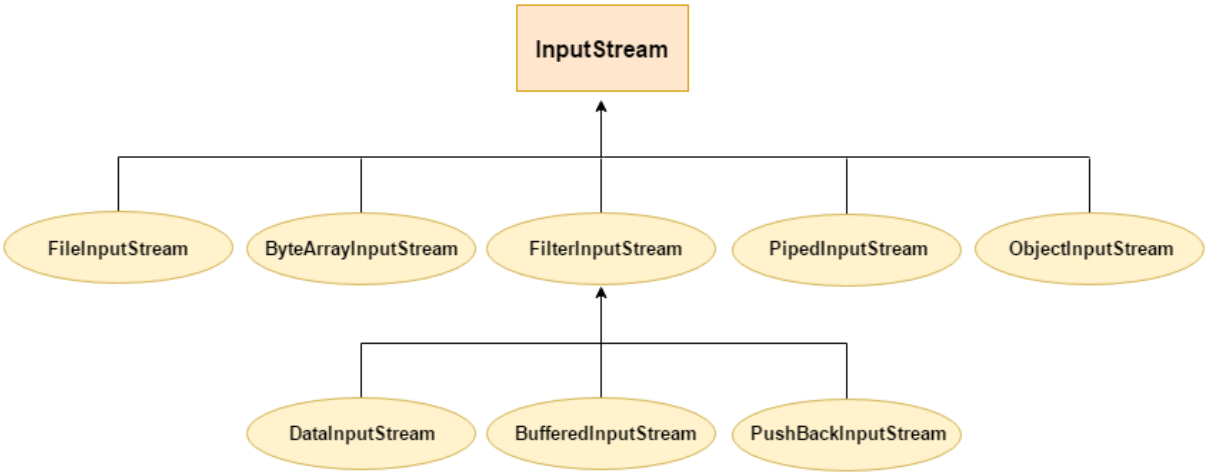
```
1. public class Runtime1{
2.  public static void main(String args[])throws Exception{
3.   Runtime.getRuntime().exec("notepad");//will open a new notepad
4.  }
5. }
```

173) Give the hierarchy of InputStream and OutputStream classes.

OutputStream Hierarchy



InputStream Hierarchy



174) What do you understand by an IO stream?

The stream is a sequence of data that flows from source to destination. It is composed of bytes. In Java, three streams are created for us automatically.

- **System.out: standard output stream**
- **System.in: standard input stream**
- **System.err: standard error stream**

175) What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented. The ByteStream classes are used to perform input-output of 8-bit bytes whereas the CharacterStream classes

are used to perform the input/output for the 16-bit Unicode system. There are many classes in the ByteStream class hierarchy, but the most frequently used classes are FileInputStream and FileOutputStream. The most frequently used classes CharacterStream class hierarchy is FileReader and FileWriter.

---

176) What are the super most classes for all the streams?

All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

---

177) What are the FileInputStream and FileOutputStream?

Java FileOutputStream is an output stream used for writing data to a file. If you have some primitive values to write into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through the FileOutputStream class. However, for character-oriented data, it is preferred to use FileWriter than FileOutputStream. Consider the following example of writing a byte into a file.

```
1. import java.io.FileOutputStream;
2. public class FileOutputStreamExample {
3.     public static void main(String args[]){
4.           try{
5.            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.            fout.write(65);
7.            fout.close();
8.            System.out.println("success...");
9.           }catch(Exception e){System.out.println(e);}
10.     }
11. }
```

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video, etc. You can also read character-stream data. However, for reading streams of characters, it is

recommended to use FileReader class. Consider the following example for reading bytes from a file.

```
1. import java.io.FileInputStream;
2. public class DataStreamExample {
3.     public static void main(String args[]){
4.         try{
5.           FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.           int i=fin.read();
7.           System.out.print((char)i);
8.
9.           fin.close();
10.        }catch(Exception e){System.out.println(e);}
11.       }
12.     }
13.
```

178) What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

Java BufferedOutputStream class is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. Whereas, Java BufferedInputStream class is used to read information from the stream. It internally uses the buffer mechanism to make the performance fast.

179) How to set the Permissions to a file in Java?

In Java, FilePermission class is used to alter the permissions set on a file. Java FilePermission class contains the permission related to a directory or file. All the permissions are related to the path. The path can be of two types:

- **D:\\IO\\-: It indicates that the permission is associated with all subdirectories and files recursively.**
- **D:\\IO*: It indicates that the permission is associated with all directory and files within this directory excluding subdirectories.**

Let's see the simple example in which permission of a directory path is granted with read permission and a file of this directory is granted for write permission.

```
1. package com.javatpoint;
2. import java.io.*;
3. import java.security.PermissionCollection;
4. public class FilePermissionExample{
5.     public static void main(String[] args) throws IOException {
6.      String srg = "D:\\IO Package\\java.txt";
7.      FilePermission file1 = new FilePermission("D:\\IO Package\\-", "read");
8.      PermissionCollection permission = file1.newPermissionCollection();
9.      permission.add(file1);
10.         FilePermission file2 = new FilePermission(srg, "write");
11.         permission.add(file2);
12.        if(permission.implies(new FilePermission(srg, "read,write"))) {
13.         System.out.println("Read, Write permission is granted for the path "+sr
    g );
14.          }else {
15.         System.out.println("No Read, Write permission is granted for the path "
    +srg);          }
16.    }
17.}
```

Output

```
Read, Write permission is granted for the path D:\IO Package\java.txt
```

180) What are FilterStreams?

FilterStream classes are used to add additional functionalities to the other stream classes. FilterStream classes act like an interface which read the data from a stream, filters it, and pass the filtered data to the caller. The FilterStream classes provide extra functionalities like adding line numbers to the destination file, etc.

181) What is an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another. Many Filter classes that allow a user to make a chain using multiple input streams. It generates a combined effect on several filters.

182) In Java, How many ways you can take input from the console?

In Java, there are three ways by using which, we can take input from the console.

- Using BufferedReader class: **we can take input from the console by wrapping System.in into an InputStreamReader and passing it into the BufferedReader. It provides an efficient reading as the input gets buffered. Consider the following example.**

```
1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;
4. public class Person
5. {
6.     public static void main(String[] args) throws IOException
7.     {
8.       System.out.println("Enter the name of the person");
9.         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
10.        String name = reader.readLine();
11.        System.out.println(name);
12.    }
13. }
```

- Using Scanner class: **The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using a regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces. Consider the following example.**

```
1. import java.util.*;
2. public class ScannerClassExample2 {
3.     public static void main(String args[]){
4.         String str = "Hello/This is JavaTpoint/My name is Abhishek.";
5.         //Create scanner with the specified String Object
6.         Scanner scanner = new Scanner(str);
7.         System.out.println("Boolean Result: "+scanner.hasNextBoolean());
```

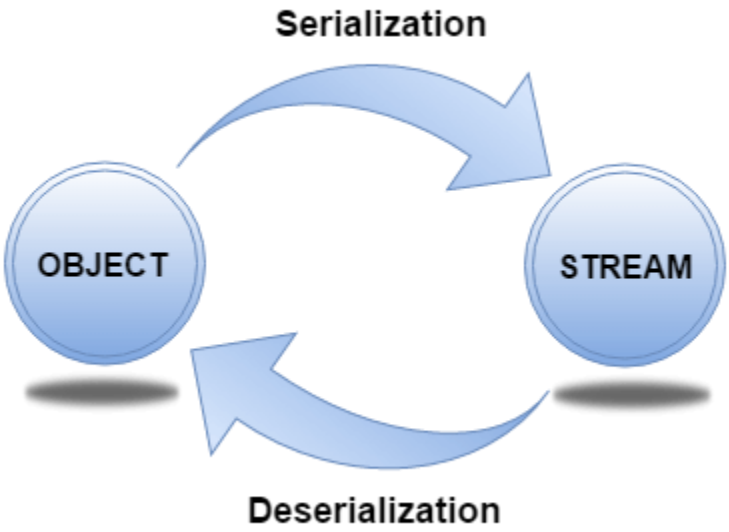
```
8.          //Change the delimiter of this scanner
9.          scanner.useDelimiter("/");
10.         //Printing the tokenized Strings
11.         System.out.println("---Tokenizes String---");
12.       while(scanner.hasNext()){
13.           System.out.println(scanner.next());
14.       }
15.         //Display the new delimiter
16.         System.out.println("Delimiter used: " +scanner.delimiter());

17.         scanner.close();
18.         }
19. }
20.
```

- Using Console class: **The Java Console class is used to get input from the console. It provides methods to read texts and passwords. If you read the password using the Console class, it will not be displayed to the user. The java.io.Console class is attached to the system console internally. The Console class is introduced since 1.5. Consider the following example.**

```
1. import java.io.Console;
2. class ReadStringTest{
3. public static void main(String args[]){
4. Console c=System.console();
5. System.out.println("Enter your name: ");
6. String n=c.readLine();
7. System.out.println("Welcome "+n);
8. }
9. }
```

183) What is serialization?

Serialization in Java is a mechanism of writing the state of an object into a byte stream. It is used primarily in Hibernate, RMI, JPA, EJB and JMS technologies. It is mainly used to travel object's state on the network (which is known as marshaling). Serializable interface is used to perform serialization. It is helpful when you require to save the state of a program to storage such as the file. At a later point of time, the content of this file can be restored using deserialization. It is also required to implement RMI(Remote Method Invocation). With the help of RMI, it is possible to invoke the method of a Java object on one machine to another machine.



More details.

184) How can you make a class serializable in Java?

A class can become serializable by implementing the Serializable interface.

185) How can you avoid serialization in child class if the base class is implementing the Serializable interface?

It is very tricky to prevent serialization of child class if the base class is intended to implement the Serializable interface. However, we cannot do it directly, but the serialization can be avoided by implementing the writeObject() or readObject() methods in the subclass and throw NotSerializableException from these methods. Consider the following example.

```
1. import java.FileInputStream;
2. import java.io.FileOutputStream;
3. import java.IOException;
4. import java.io.NotSerializableException;
5. import java.ObjectInputStream;
6. import java.io.ObjectOutputStream;
7. import java.Serializable;
8. class Person implements Serializable
9. {
10.    String name = " ";
```

```
11.    public Person(String name)
12.    {
13.        this.name = name;
14.    }
15. }
16. class Employee extends Person
17. {
18.    float salary;
19.    public Employee(String name, float salary)
20.    {
21.        super(name);
22.        this.salary = salary;
23.    }
24.    private void writeObject(ObjectOutputStream out) throws IOException
25.    {
26.        throw new NotSerializableException();
27.    }
28.    private void readObject(ObjectInputStream in) throws IOException
29.    {
30.        throw new NotSerializableException();
31.    }
32.
33. }
34. public class Test
35. {
36.    public static void main(String[] args)
37.            throws Exception
38.    {
39.        Employee emp = new Employee("Sharma", 10000);
40.
41.        System.out.println("name = " + emp.name);
42.        System.out.println("salary = " + emp.salary);
43.
44.        FileOutputStream fos = new FileOutputStream("abc.ser");
45.        ObjectOutputStream oos = new ObjectOutputStream(fos);
46.
47.        oos.writeObject(emp);
```

```
48.
49.         oos.close();
50.         fos.close();
51.
52.         System.out.println("Object has been serialized");
53.
54.         FileInputStream f = new FileInputStream("ab.txt");
55.         ObjectInputStream o = new ObjectInputStream(f);
56.
57.         Employee emp1 = (Employee)o.readObject();
58.
59.         o.close();
60.         f.close();
61.
62.         System.out.println("Object has been deserialized");
63.
64.         System.out.println("name = " + emp1.name);
65.         System.out.println("salary = " + emp1.salary);
66.     }
67. }
```

186) Can a Serialized object be transferred via network?

Yes, we can transfer a serialized object via network because the serialized object is stored in the memory in the form of bytes and can be transmitted over the network. We can also write the serialized object to the disk or the database.

187) What is Deserialization?

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

```
1. import java.io.*;
2. class Depersist{
3.  public static void main(String args[])throws Exception{
4.
```

```
5.  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));

6.  Student s=(Student)in.readObject();
7.  System.out.println(s.id+" "+s.name);
8.
9.  in.close();
10. }
11.}
```

```
211 ravi
```

188) What is the transient keyword?

If you define any data member as transient, it will not be serialized. By determining transient keyword, the value of variable need not persist when it is restored. More details.

189) What is Externalizable?

The Externalizable interface is used to write the state of an object into a byte stream in a compressed format. It is not a marker interface.

190) What is the difference between Serializable and Externalizable interface?

No.	Serializable	Externalizable
1)	The Serializable interface does not have any method, i.e., it is a marker interface.	The Externalizable interface contains is not a marker interface, It contains two methods, i.e., writeExternal() and readExternal().
2)	It is used to "mark" Java classes so that objects of these classes may get the certain capability.	The Externalizable interface provides control of the serialization logic to the programmer.
3)	It is easy to implement but has the higher performance cost.	It is used to perform the serialization and often result in better performance.

4)	No class constructor is called in serialization.	We must call a public default constructor while using this interface.

191) Give a brief description of Java socket programming?

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connectionless. Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket, and DatagramPacket classes are used for connectionless socket programming. The client in socket programming must know two information:

- **IP address of the server**
- **port number**

---

192) What is Socket?

A socket is simply an endpoint for communications between the machines. It provides the connection mechanism to connect the two computers using TCP. The Socket class can be used to create a socket.

---

193) What are the steps that are followed when two computers connect through TCP?

There are the following steps that are performed when two computers connect through TCP.

- **The ServerSocket object is instantiated by the server which denotes the port number to which, the connection will be made.**
- **After instantiating the ServerSocket object, the server invokes accept() method of ServerSocket class which makes server wait until the client attempts to connect to the server on the given port.**
- **Meanwhile, the server is waiting, a socket is created by the client by instantiating Socket class. The socket class constructor accepts the server port number and server name.**

- **The Socket class constructor attempts to connect with the server on the specified name. If the connection is established, the client will have a socket object that can communicate with the server.**
- **The accept() method invoked by the server returns a reference to the new socket on the server that is connected with the server.**

194) Write a program in Java to establish a connection between client and server?

Consider the following program where the connection between the client and server is established.

File: MyServer.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyServer {
4. public static void main(String[] args){
5. try{
6. ServerSocket ss=new ServerSocket(6666);
7. Socket s=ss.accept();//establishes connection
8. DataInputStream dis=new DataInputStream(s.getInputStream());
9. String  str=(String)dis.readUTF();
10. System.out.println("message= "+str);
11. ss.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

File: MyClient.java

```
1. import java.io.*;
2. import java.net.*;
3. public class MyClient {
4. public static void main(String[] args) {
5. try{
6. Socket s=new Socket("localhost",6666);
7. DataOutputStream dout=new DataOutputStream(s.getOutputStream());
```

```
8. dout.writeUTF("Hello Server");
9. dout.flush();
10. dout.close();
11. s.close();
12. }catch(Exception e){System.out.println(e);}
13. }
14. }
```

195) How do I convert a numeric IP address like 192.18.97.39 into a hostname like java.sun.com?

By InetAddress.getByName("192.18.97.39").getHostName() where 192.18.97.39 is the IP address. Consider the following example.

```
1. import java.io.*;
2. import java.net.*;
3. public class InetDemo{
4. public static void main(String[] args){
5. try{
6. InetAddress ip=InetAddress.getByName("195.201.10.8");
7.
8. System.out.println("Host Name: "+ip.getHostName());
9. }catch(Exception e){System.out.println(e);}
10. }
11. }
12.
```

196) What is the reflection?

Reflection is the process of examining or modifying the runtime behavior of a class at runtime. The java.lang.Class class provides various methods that can be used to get metadata, examine and change the runtime behavior of a class. The java.lang and java.lang.reflect packages provide classes for java reflection. It is used in:

- **IDE (Integrated Development Environment), e.g., Eclipse, MyEclipse, NetBeans.**
- **Debugger**
- **Test Tools, etc.**

197) What is the purpose of using java.lang.Class class?

The java.lang.Class class performs mainly two tasks:

- **Provides methods to get the metadata of a class at runtime.**
- **Provides methods to examine and change the runtime behavior of a class.**

198) What are the ways to instantiate the Class class?

There are three ways to instantiate the Class class.

- **forName() method of Class class**: The forName() method is used to load the class dynamically. It returns the instance of Class class. It should be used if you know the fully qualified name of the class. This cannot be used for primitive types.

- **getClass() method of Object class**: It returns the instance of Class class. It should be used if you know the type. Moreover, it can be used with primitives.

- **the .class syntax**: If a type is available, but there is no instance then it is possible to obtain a Class by appending ".class" to the name of the type. It can be used for primitive data type also.

199) What is the output of the following Java program?

```
1. class Simple{
2.  public Simple()
3.  {
4.    System.out.println("Constructor of Simple class is invoked");
5.  }
6.  void message(){System.out.println("Hello Java");}
7. }
8.
9. class Test1{
10. public static void main(String args[]){
```

```
11. try{
12. Class c=Class.forName("Simple");
13. Simple s=(Simple)c.newInstance();
14. s.message();
15. }catch(Exception e){System.out.println(e);}
16. }
17. }
```

Output

```
Constructor of Simple class is invoked
Hello Java
```

Explanation

The newInstance() method of the Class class is used to invoke the constructor at runtime. In this program, the instance of the Simple class is created.

---

200) What is the purpose of using javap?

The javap command disassembles a class file. The javap command displays information about the fields, constructors and methods present in a class file.

Syntax

javap fully_class_name

---

201) Can you access the private method from outside the class?

Yes, by changing the runtime behavior of a class if the class is not secured.

More details.

202)What are wrapper classes?

Wrapper classes are classes that allow primitive types to be accessed as objects. In other words, we can say that wrapper classes are built-in java classes which allow the conversion of objects to primitives and primitives to objects. The process of converting primitives to objects is called autoboxing, and the process of converting objects to

primitives is called unboxing. There are eight wrapper classes present in **java.lang** package is given below.

Primitive Type	Wrapper class
boolean	**Boolean**
char	**Character**
byte	**Byte**
short	**Short**
int	**Integer**
long	**Long**
float	**Float**
double	**Double**

203)What are autoboxing and unboxing? When does it occur?

The autoboxing is the process of converting primitive data type to the corresponding wrapper class object, eg., int to Integer. The unboxing is the process of converting wrapper class object to primitive data type. For eg., integer to int. Unboxing and autoboxing occur automatically in Java. However, we can externally convert one into another by using the methods like valueOf() or xxxValue().

It can occur whenever a wrapper class object is expected, and primitive data type is provided or vice versa.

- **Adding primitive types into Collection like ArrayList in Java.**
- **Creating an instance of parameterized classes ,e.g., ThreadLocal which expect Type.**
- **Java automatically converts primitive to object whenever one is required and another is provided in the method calling.**

- **When a primitive type is assigned to an object type.**

204) What is the output of the below Java program?

```
1. public class Test1
2. {
3.   public static void main(String[] args) {
4.   Integer i = new Integer(201);
5.   Integer j = new Integer(201);
6.   if(i == j)
7.   {
8.     System.out.println("hello");
9.   }
10.  else
11.  {
12.    System.out.println("bye");
13.  }
14.  }
15.}
```

Output

```
bye
```

Explanation

The Integer class caches integer values from -127 to 127. Therefore, the Integer objects can only be created in the range -128 to 127. The operator **==** will not work for the value greater than 127; thus **bye** is printed.

205) What is object cloning?

The object cloning is a way to create an exact copy of an object. The clone() method of the Object class is used to clone an object. The java.lang.Cloneable interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException. The clone() method is defined in the Object class. The syntax of the clone() method is as follows:

protected Object clone() throws CloneNotSupportedException

206) What are the advantages and disadvantages of object cloning?

Advantage of Object Cloning

- You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
- It is the easiest and most efficient way of copying objects, especially if we are applying it to an already developed or an old project. Just define a parent class, implement Cloneable in it, provide the definition of the clone() method and the task will be done.
- Clone() is the fastest way to copy the array.

Disadvantage of Object Cloning

- To use the Object.clone() method, we have to change many syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone(), etc.
- We have to implement the Cloneable interface while it does not have any methods in it. We have to use it to tell the JVM that we can perform a clone() on our object.
- Object.clone() is protected, so we have to provide our own clone() and indirectly call Object.clone() from it.
- Object.clone() does not invoke any constructor, so we do not have any control over object construction.
- If you want to write a clone method in a child class, then all of its superclasses should define the clone() method in them or inherit it from another parent class. Otherwise, the super.clone() chain will fail.
- Object.clone() supports only shallow copying, but we will need to override it if we need deep cloning.

207) What is a native method?

A native method is a method that is implemented in a language other than Java. Natives methods are sometimes also referred to as foreign methods.

208) What is the purpose of the strictfp keyword?

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable. The precision may differ from platform to platform that is why java programming language has provided the strictfp keyword so that you get the same result on every platform. So, now you have better control over the floating-point arithmetic.

---

209) What is the purpose of the System class?

The purpose of the System class is to provide access to system resources such as standard input and output. It cannot be instantiated. Facilities provided by System class are given below.

- **Standard input**
- **Error output streams**
- **Standard output**
- **utility method to copy the portion of an array**
- **utilities to load files and libraries**

There are the three fields of Java System class, i.e., static printstream err, static inputstream in, and standard output stream.

---

210) What comes to mind when someone mentions a shallow copy in Java?

Object cloning.

---

211) What is a singleton class?

Singleton class is the class which can not be instantiated more than once. To make a class singleton, we either make its constructor private or use the static getInstance method. Consider the following example.

```
1. class Singleton{
2.     private static Singleton single_instance = null;
3.     int i;
```

```
    private Singleton ()
    {
       i=90;
    }
    public static Singleton getInstance()
    {
       if(single_instance == null)
       {
          single_instance = new Singleton();
       }
       return single_instance;
    }
}
public class Main
{
  public static void main (String args[])
  {
     Singleton first = Singleton.getInstance();
     System.out.println("First instance integer value:"+first.i);
     first.i=first.i+90;
     Singleton second = Singleton.getInstance();
     System.out.println("Second instance integer value:"+second.i);
  }
}
```

212) Write a Java program that prints all the values given at command-line.

Program

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

for(int i=0;i<args.length;i++)
System.out.println(args[i]);

```

7. }
8. **}**

1. compile by > javac A.java
2. **run by > java A sonoo jaiswal 1 3 abc**

Output

```
sonoo
jaiswal
1
3
abc
```

213) Which containers use a border layout as their default layout?

The Window, Frame and Dialog classes use a border layout as their default layout.

214) Which containers use a FlowLayout as their default layout?

The Panel and Applet classes use the FlowLayout as their default layout.

215) What are peerless components?

The lightweight component of Swing is called peerless components. Spring has its libraries, so it does not use resources from the Operating System, and hence it has lightweight components.

216) is there is any difference between a Scrollbar and a ScrollPane?

The Scrollbar is a Component whereas the ScrollPane is a Container. A ScrollPane handles its events and performs its scrolling.

217) What is a lightweight component?

Lightweight components are the one which does not go with the native call to obtain the graphical units. They share their parent component graphical units to render them. For example, Swing components, and JavaFX Components.

---

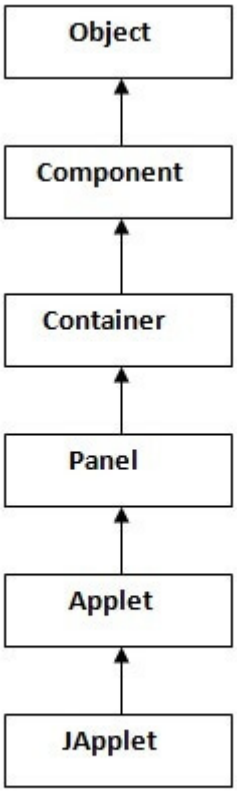
218) What is a heavyweight component?

The portable elements provided by the operating system are called heavyweight components. AWT is limited to the graphical classes provided by the operating system and therefore, It implements only the minimal subset of screen elements supported by all platforms. The Operating system dependent UI discovery tools are called heavyweight components.

---

219) What is an applet?

An applet is a small java program that runs inside the browser and generates dynamic content. It is embedded in the webpage and runs on the client side. It is secured and takes less response time. It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os, etc. However, the plugins are required at the client browser to execute the applet. The following image shows the architecture of Applet.



When an applet is created, the following methods are invoked in order.

- **init()**
- **start()**
- **paint()**

When an applet is destroyed, the following functions are invoked in order.

- **stop()**
- **destroy()**

220) Can you write a Java class that could be used both as an applet as well as an application?

Yes. Add a main() method to the applet.

221) What is Locale?

A Locale object represents a specific geographical, political, or cultural region. This object can be used to get the locale-specific information such as country name, language, variant, etc.

```
1. import java.util.*;
2. public class LocaleExample {
3. public static void main(String[] args) {
4. Locale locale=Locale.getDefault();
5. //Locale locale=new Locale("fr","fr");//for the specific locale
6.
7. System.out.println(locale.getDisplayCountry());
8. System.out.println(locale.getDisplayLanguage());
9. System.out.println(locale.getDisplayName());
10. System.out.println(locale.getISO3Country());
11. System.out.println(locale.getISO3Language());
12. System.out.println(locale.getLanguage());
13. System.out.println(locale.getCountry());
14.
15. }
16. }
```

Output:

```
United States
English
English (United States)
USA
eng
en
US
```

---

222)How will you load a specific locale?

By ResourceBundle.getBundle(?) method.

---

223) What is a JavaBean?

JavaBean is a reusable software component written in the Java programming language, designed to be manipulated visually by a software development environment, like

JBuilder or VisualAge for Java. t. A JavaBean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides the easy maintenance. Consider the following example to create a JavaBean class.

```
1. //Employee.java
2. package mypack;
3. public class Employee implements java.io.Serializable{
4. private int id;
5. private String name;
6. public Employee(){}
7. public void setId(int id){this.id=id;}
8. public int getId(){return id;}
9. public void setName(String name){this.name=name;}
10. public String getName(){return name;}
11. }
```

224) What is the purpose of using the Java bean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides the easy maintenance.

225) What do you understand by the bean persistent property?

The persistence property of Java bean comes into the act when the properties, fields, and state information are saved to or retrieve from the storage.

226) What is RMI?

The RMI (Remote Method Invocation) is an API that provides a mechanism to create the distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.

227) What is the purpose of stub and skeleton?

Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes the method on the stub object, it does the following tasks:

- **It initiates a connection with remote Virtual Machine (JVM).**
- **It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM).**
- **It waits for the result.**
- **It reads (unmarshals) the return value or exception.**
- **It finally, returns the value to the caller.**

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

- **It reads the parameter for the remote method.**
- **It invokes the method on the actual remote object.**
- **It writes and transmits (marshals) the result to the caller.**

---

228) What are the steps involved to write RMI based programs?

There are 6 steps which are performed to write RMI based programs.

- **Create the remote interface.**
- **Provide the implementation of the remote interface.**
- **Compile the implementation class and create the stub and skeleton objects using the rmic tool.**
- **Start the registry service by the rmiregistry tool.**
- **Create and start the remote application.**
- **Create and start the client application.**

---

229) What is the use of HTTP-tunneling in RMI?

HTTP tunneling can be defined as the method which doesn't need any setup to work within the firewall environment. It handles the HTTP connections through the proxy servers. However, it does not allow outbound TCP connections.

---

230) What is JRMP?

JRMP (Java Remote Method Protocol) can be defined as the Java-specific, stream-based protocol which looks up and refers to the remote objects. It requires both client and server to use Java objects. It is wire level protocol which runs under RMI and over TCP/IP.

---

231) Can RMI and CORBA based applications interact?

Yes, they can. RMI is available with IIOP as the transport protocol instead of JRMP.

232) Anonymous Array in Java?

An array in Java **without any name** is known as **an anonymous array**. It is an array just for creating and using instantly. Using an anonymous array, we can pass an array with user values without the referenced variable.

Examples:

```
// anonymous int array
new int[] { 1, 2, 3, 4};

// anonymous char array
new char[] {'x', 'y', 'z'};

// anonymous String array
new String[] {"Geeks", "for", "Geeks"};

// anonymous multidimensional array
new int[][] { {10, 20}, {30, 40, 50} };
```

```
// Java program to illustrate the
// concept of anonymous array

class Test {
    public static void main(String[] args)
    {
          // anonymous array
          sum(new int[]{ 1, 2, 3 });
    }

    public static void sum(int[] a)
    {
        int total = 0;

        // using for-each loop
        for (int i : a)
            total = total + i;

        System.out.println("The sum is: " + total);
    }
}
```

233) Array Index out of bound exception?

If length of the array is negative equal to the array size or greater than the array size while traversing the array.

234) What is array in java?

An array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type.

235) What is toString() method?

It is an object class method. It is always call when we create a new object.

236) Which operator is support operator overloading in Java?

+(Plus).

Exa:- 9+10 = 19

"java"+9 = java9

237) What is Recursion in Java?

Recursion in java is a process in which a method calls itself continuously.

238) What is Local class in Java?

A class i.e., created inside a method, is called local inner class in java.

Local Classes are classes that are defined inside a block, which is a group of zero or more statement between balanced braces.

A local class is declared locally within a block of Java code rather than as a member of class.

239) Object class method in java?

Object class is present in **java.lang** package. The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

- tostring()
- hashCode()
- equals(Object obj)
- getClass()
- finalize()
- clone()
- wait()
- notify()
- notifyAll()

240) What is Private Constructor?

If you make any class constructor private, you cannot create the instance of that class from outside the class.

241) Example of Multilevel Inheritance in java?

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class BabyDog extends Dog{
8. void weep(){System.out.println("weeping...");}
9. }
10. class TestInheritance2{
11. public static void main(String args[]){
12. BabyDog d=new BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
16. }}
```

Output:

```
weeping...
barking...
eating...
```

242) Example of static block in java?

When a block is decorated or associated with the word static, it is called a static block. Static Block is known as the static clause. A static block can be used for the static initialization of a class. The code that is written inside the static block run once, when the class is getting loaded into the memory.

```
public class Test {
	static {
		System.out.println("Inside the static block");
```

```
	}
	public static void main(String[] args) {
		System.out.println("Inside the main method");
	}

}
```

Output:-

```
Inside the static block

Inside the main method
```

243) Encapsulation Example in java?

A Java Bean class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

244) How can we make class Immutable?

Declare the class as final so it can't be extended.

Make all fields private so that direct access is not allowed.

245) can we overload a main() method in java?

Yes, we can overload the main() method, but only public static void main(String[] args) will be used when your class is launched by the JVM. For example:-

```
public class Test {

	public static void main(String[] args) {
		System.out.println("original main");
	}

	public static void main(String args) {
		System.out.println("overloaded main");
	}

	public static void main(String args,String args2) {
		System.out.println("overloaded main");
	}

}
```

246) can we override a main() method in java?

No, We can't override the static method because static method is a class method and the scope of this method within the same class itself. Hence, it does not makes sense to override it.

247) What is the difference between while loop and do while loop?

The difference between while and do while loop is that in the while loop the condition is checked prior to executing any statements whereas in the case of do while loop, statements are run at least once, and then the condition is verified.

248) What is the difference between for loop and while loop?

For Loop	While Loop
The for loop used only when we already knew the number of iterations.	The while loop used only when the number of iteration are not exactly known.
If the condition is not put up in "For Loop" , then loop iterates infinite times.	If the condition is not put in "While Loop" it provides compilation errors.

249) What is the syntax of Inheritance?

```
Class SubClassName extends SuperClassName {

        //method and fields

}
```

250) What is the syntax of for loop, while loop and do while loop?

Java Simple for Loop

Syntax:

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

Java for-each Loop

Syntax:

```
for(data_type variable : array_name){
//code to be executed
}
```

Java Labeled For Loop

Syntax:

```
labelname:
for(initialization; condition; increment/decrement){
//code to be executed
}
```

Java Infinitive for Loop

Syntax:

```
for(;;){
//code to be executed
}
```

Java While Loop

Syntax:

```
while (condition){
//code to be executed
I ncrement / decrement statement
```

4. }

Java Infinitive While Loop

If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

```
1. while(true){
2. //code to be executed
3. }
```

Java do-while Loop

Syntax:

```
1. do{
2. //code to be executed / loop body
3. //update statement
4. }while (condition);
```

Java Infinitive do-while Loop

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

Syntax:

```
1. do{
2. //code to be executed
3. }while(true);
```

Java If-else Statement

Java if Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

```
1. if(condition){
2. //code to be executed
```

```
3. }
```

Java if-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
1. if(condition){
2. //code if condition is true
3. }else{
4. //code if condition is false
5. }
```

Using Ternary Operator

We can also use ternary operator (? :) to perform the task of if...else statement.

Java if-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
1. if(condition1){
2. //code to be executed if condition1 is true
3. }else if(condition2){
4. //code to be executed if condition2 is true
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. else{
11. //code to be executed if all the conditions are false
12. }
```

Java Nested if statement

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```

Java Switch Statement

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
  code to be executed if all cases are not matched;
}
```

Java Break Statement

Syntax:

```
jump-statement;
break;  //The Java break statement is used to break loop or switch statement.
```

Java Continue Statement

Syntax:

jump-statement;

continue; **//It continues the current flow of the program and skips the remaining code**

251) Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

252) Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- **Code Reusability**
- **Code Optimization**

253)new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

254)What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

255) Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

```
new Calculation();//anonymous object

new Calculation().fact(5);
```

256) Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created.

Every time an object is created using the new() keyword, at least one constructor is called.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Q257) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

258) Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

259)Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

---

260)Is there Constructor class in Java?

Yes.

---

261)What is the purpose of Constructor class?

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

262)Java static keyword

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes your program memory efficient (i.e., it saves memory).

Example:- Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

263)Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Inheritance represents the IS-A relationship **which is also known as a *parent-child* relationship.**

264)Why use inheritance in java

- **For Method Overriding (so runtime polymorphism can be achieved).**
- **For Code Reusability.**

Terms used in Inheritance

- Class: **A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.**

- **Sub Class/Child Class**: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class**: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability**: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

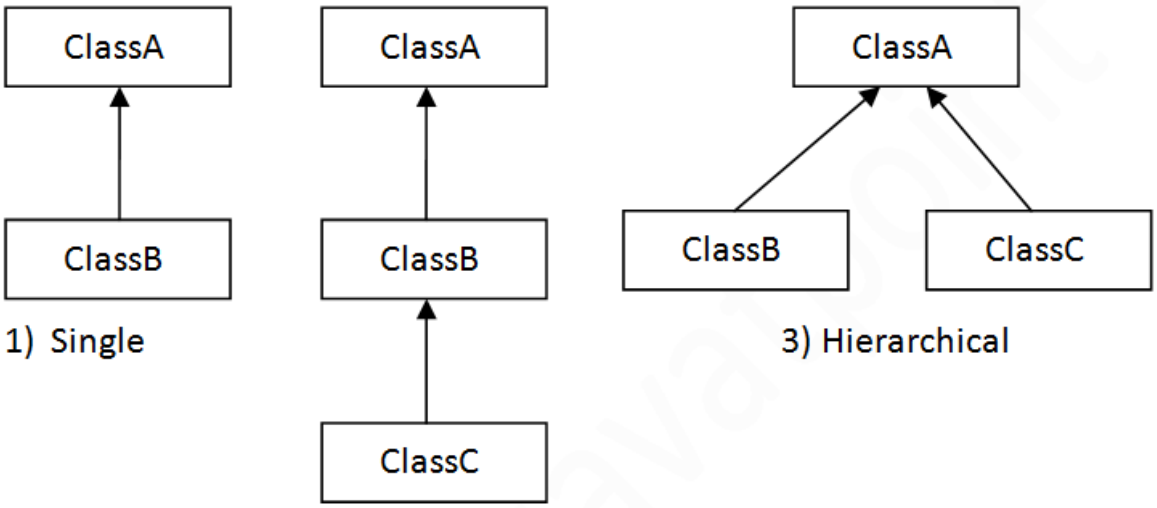
-> As displayed in the below example, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

```
1. class Employee{
2.  float salary=40000;
3. }
4. class Programmer extends Employee{
5.  int bonus=10000;
6.  public static void main(String args[]){
7.    Programmer p=new Programmer();
8.    System.out.println("Programmer salary is:"+p.salary);
9.    System.out.println("Bonus of Programmer is:"+p.bonus);
10.       }
11.       }
```

Types of inheritance in java

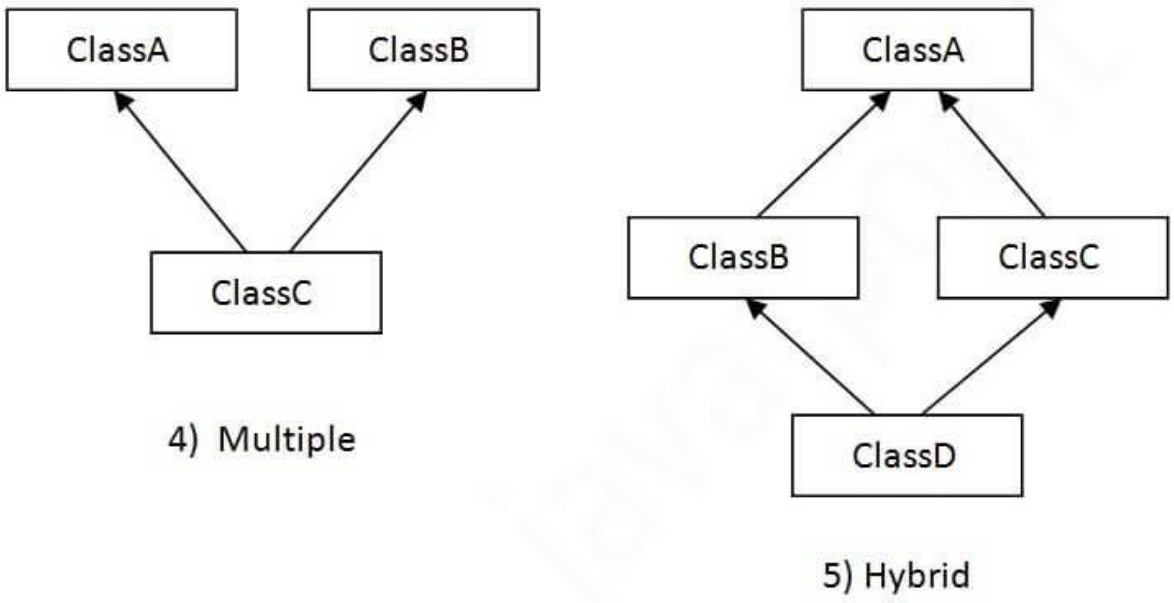
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



Note: Multiple inheritance is not supported in Java through class.

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



.

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10.       d.bark();
11.       d.eat();
12.       }}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class BabyDog extends Dog{
8. void weep(){System.out.println("weeping...");}
9. }
10.       class TestInheritance2{
11.       public static void main(String args[]){
12.       BabyDog d=new BabyDog();
13.       d.weep();
14.       d.bark();
15.       d.eat();
16.       }}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class Cat extends Animal{
8. void meow(){System.out.println("meowing...");}
9. }
10. class TestInheritance3{
11. public static void main(String args[]){
12. Cat c=new Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}
```

Output:

```
meowing...
eating...
```

Q265) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
8.
9.  public static void main(String args[]){
10.          C obj=new C();
11.          obj.msg();//Now which msg() method would be invoked?
12.       }
13.       }
```

Test it Now

```
Compile Time Error
```

266) Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
1. class Employee{
2. int id;
3. String name;
4. Address address;//Address is a class
5. ...
6. }
```

In such case, Employee has an entity reference address, so relationship is Employee HAS-A address.

267)Why use Aggregation?

- **For Code Reusability.**

268)When use Aggregation?

- Code reuse is also best achieved by aggregation when there is no is-a relationship.

- Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Understanding meaningful example of Aggregation

In this example, Employee has an object of Address, address object contains its own informations such as city, state, country etc. In such case relationship is Employee HAS-A address.

Address.java

```
1. public class Address {
2. String city,state,country;
3.
4. public Address(String city, String state, String country) {
5.     this.city = city;
6.     this.state = state;
7.     this.country = country;
8. }
9.
10.         }
```

Emp.java

```
1. public class Emp {
2. int id;
3. String name;
4. Address address;
5.
6. public Emp(int id, String name,Address address) {
7.     this.id = id;
8.     this.name = name;
9.     this.address=address;
10.         }
11.
12.         void display(){
13.         System.out.println(id+" "+name);
14.         System.out.println(address.city+" "+address.state+" "+address.coun
    try);
15.         }
16.
17.         public static void main(String[] args) {
18.         Address address1=new Address("gzb","UP","india");
19.         Address address2=new Address("gno","UP","india");
20.
```

```
21.         Emp e=new Emp(111,"varun",address1);
22.         Emp e2=new Emp(112,"arun",address2);
23.
24.         e.display();
25.         e2.display();
26.
27.         }
28.         }
```

Test it Now

```
Output:111 varun
       gzb UP india
       112 arun
       gno UP india
```

269)Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. **By changing number of arguments**
2. **By changing the data type**

In java, Method Overloading is not possible by changing the return type of the method only.

Q270) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
1. class Adder{
2. static int add(int a,int b){return a+b;}
3. static double add(int a,int b){return a+b;}
4. }
```

```
5. class TestOverloading3{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));//ambiguity
8. }}
```

Test it Now

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.

271)Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
1. class TestOverloading4{
2. public static void main(String[] args){System.out.println("main with String []");}
3. public static void main(String args){System.out.println("main with String");
   }
4. public static void main(){System.out.println("main without args");}
5. }
```

Test it Now

Output:

```
main with String[]
```

272) Example of Method Overloading with TypePromotion

```
1. class OverloadingCalculation1{
2.   void sum(int a,long b){System.out.println(a+b);}
3.   void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5.   public static void main(String args[]){
6.   OverloadingCalculation1 obj=new OverloadingCalculation1();
7.   obj.sum(20,20);//now second int literal will be promoted to long
8.   obj.sum(20,20,20);
9.
10.        }
11.      }
```

Test it Now

```
Output:40
       60
```

273) Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```
1. class OverloadingCalculation3{
2.   void sum(int a,long b){System.out.println("a method invoked");}
3.   void sum(long a,int b){System.out.println("b method invoked");}
4.
5.   public static void main(String args[]){
6.   OverloadingCalculation3 obj=new OverloadingCalculation3();
7.   obj.sum(20,20);//now ambiguity
8.   }
9. }
```

Test it Now

```
Output:Compile Time Error
```

274) Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Rules for Java Method Overriding

1. **The method must have the same name as in the parent class**
2. **The method must have the same parameter as in the parent class.**
3. **There must be an IS-A relationship (inheritance).**

275) Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

276) Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

---

277) Can we override java main method?

No, because the main is a static method.

278) Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

279) Final Keyword In Java

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

2) Java final method

If you make any method as final, you cannot override it.

3) Java final class

If you make any class as final, you cannot extend it.

Q280) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
1. class Bike{
2.   final void run(){System.out.println("running...");}
3. }
4. class Honda2 extends Bike{
5.    public static void main(String args[]){
6.     new Honda2().run();
7.    }
8. }
```

Que281) Can we initialize blank final variable?

Yes, but only in constructor. For example:

Q282) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Q283) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

Q284) Can we declare a constructor final?

No, because constructor is never inherited.

285) Runtime Polymorphism in Java

Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

286) Java instanceof

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

287) What is a Diamond Problem in Java?

Talking about Multiple inheritance is when a child class is inherits the properties from more than one parents and the methods for the parents are same (Method name and parameters are exactly the same) then child gets confused about which method will be called. This problem in Java is called the Diamond problem.

```
// Java Program to demonstrate
// Diamond Problem
```

```
import java.io.*;

// Parent Class1
class Parent1 {
	void fun() { System.out.println("Parent1"); }
}

// Parent Class2
class Parent2 {
	void fun() { System.out.println("Parent2"); }
}

// Inherting the Properties from
// Both the classes
class test extends Parent1, Parent2 {
	// main function
	public static void main(String[] args)
	{
		test t = new test();
		t.fun();
	}
}
```

NOTE:- Java does not support the multiple inheritance **because of the diamond problem.**