Curious Freaks - OOPS & Java Imp Interview Quesitons
Checkout curiousfreakss.com for Placement Support

1. **Object oriented programming**
   It is the programming paradigm that is defined using objects. Objects can be considered as real-world instances of entities like class, that have some characteristics and behaviors.

2. **Main features of object oriented programming**
   - Inheritance
   - Encapsulation
   - Polymorphism
   - Data Abstraction

3. **Classes**
   A class can be understood as a template or a blueprint, which contains some values, known as member data or member, and some set of rules, known as behaviors or functions. So when an object is created, it automatically takes the data and functions that are defined in the class.Also one can create as many objects as they want based on a class.

4. **Objects**
   An object refers to the instance of the class, which contains the instance of the members and behaviors defined in the class template. In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object. So the objects consume space and have some characteristic behavior.

5. **Difference between class and structure**
   Class: User defined blueprint from which objects are created.It consists of methods or set of instructions that are to be performed on the objects.

   Structure: A structure is basically a user defined collection of variables which are of different data types.

6. **Advantages of object oriented programming**
   - OOPs is very helpful in solving very complex problems.
   - Highly complex programs can be created, handled, and maintained easily using object-oriented programming.
   - OOPs promote code reuse, thereby reducing redundancy.
   - OOPs also help to hide the unnecessary details with the help of Data Abstraction.
   - OOPs are based on a bottom-up approach, unlike the Structural programming paradigm, which uses a top-down approach.
   - Polymorphism offers a lot of flexibility in OOPs.

7. **Data abstraction**
   Data abstraction is a very important feature of OOPs that allows displaying only the important information and hiding the implementation details. For example, while riding a bike, you know that if you raise the accelerator, the speed will increase, but you don't know how it actually happens. This is data abstraction as the implementation details are hidden from the rider.

8. **How to achieve data abstraction?**
   Data abstraction can be achieved through
   - Abstract class
   - Abstract method

   Abstraction using classes: A class can decide which data member will be visible to the outside world and which is not using access specifier.

   Abstraction in header files: For example, consider the pow() method present in math.h header file.Whenever we need to calculate power of a number, we simply call the function and pass the numbers as arguments without knowing the underlying algorithm.

9. **Encapsulation**
   Encapsulation refers to binding the data and the code that works on that together in a single unit. For example, a class. Encapsulation also allows data-hiding as the data specified in one class is hidden from other classes.

   Example: In a company there are different sections like the account section, finance section, sales section, etc.The finance section handles all the financial transactions and keep records of all the data related to finance.Similarly, the sales section handles all the sales related activities and keep record of all these sales.Now, there may arise a situation when for some reason an official from finance section, needs all the data about sales in a particular month.In this case, he is not allowed to directly access the data of sales section.He will first have to contact some other officer in the sale section and then request him to give the particular data.This is what encapsulation is.

10. **Access specifiers**
    Access specifiers or access modifiers are keywords that determine the accessibility of methods, classes, etc in OOPs. These access specifiers allow the implementation of encapsulation. The most common access specifiers are public, private and protected.

11. **Role of access specifiers**
    It plays an important role in implementing encapsulation.
    - The data member should be labeled as private using private access specifier.

- The member function which manipulates the data member should be labeled as public.

12. **Inheritance**
    The capability of a class to derive properties and characteristics from another class.With the usage of inheritance, we can avoid the chances of error and data redundancy.

    Example: Let's take three different vehicles - a car, truck, or bus. These three are entirely different from one another with their own specific characteristics and behavior. But in all three, you will find some common elements, like steering wheel, accelerator, clutch, brakes, etc. Though these elements are used in different vehicles, still they have their own features which are common among all vehicles. This is achieved with inheritance. The car, the truck, and the bus have all inherited the features like steering wheel, accelerator, clutch, brakes, etc, and used them as their own. Due to this, they did not have to create these components from scratch, thereby facilitating code reuse.

13. **Types of inheritance**
    - Single inheritance
    - Multiple inheritance
    - Multilevel inheritance
    - Hierarchical inheritance
    - Hybrid inheritance

14. **Limitations of inheritance**
    - Increases the time and effort required to execute a program as it requires jumping back and forth between different classes
    - The parent class and the child class get tightly coupled
    - Any modifications to the program would require changes both in the parent as well as the child class
    - Needs careful implementation else would lead to incorrect results

15. **Superclass**
    A superclass or base class is a class that acts as a parent to some other class or classes.

16. **Subclass**
    A class that inherits from another class is called the subclass.

17. **Polymorphism**
    Polymorphism is composed of two words - “poly” which means “many”, and “morph” which means “shapes”. Therefore Polymorphism refers to something that has many shapes.Polymorphism is the ability of data to be processed in more than one form.

In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms.

18. **Compile time polymorphism**
    Compile time polymorphism, also known as Static Polymorphism, refers to the type of Polymorphism that happens at compile time. What it means is that the compiler decides what shape or value has to be taken by the entity.

    Compile time polymorphism is achieved by method overloading or operator overloading.

    Method Overloading:When there are multiple functions with same name but different parameters, then these functions are set to be overloaded.Functions can be overloaded by change in number of arguments or change in type of arguments.

    Operator Overloading:A single operator '+' when placed between integer operands, adds them and when placed between string operands, concatenates them.

19. **Runtime polymorphism**
    Runtime polymorphism, also known as Dynamic Polymorphism, refers to the type of Polymorphism that happens at the run time. What it means is it can't be decided by the compiler. Therefore what shape or value has to be taken depends upon the execution.

    Runtime polymorphism is achieved by function overriding.

    Function Overriding:It occurs when a derived class has a definition for one of the member functions of the base class.That base function is said to be overridden.

20. **Difference between abstract class and interface**
    Interface and abstract class both are special types of classes that contain only method declaration, not their implementation. But the interface is entirely different from an abstract class.The main difference between the two is that when an interface is implemented, the subclass must define all its methods and provide its implementation.Whereas for an abstract class is inherited, the subclass does not need to provide the definition of its abstract method until and unless the subclasses using it.Also, an abstract class can contain abstract methods as well as non abstract methods.

JAVA

1. **public static void main(String args[]) in Java**
    - public: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
    - static: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as main() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
    - void: It is the return type of the method. Void defines the method which will not return any value.
    - main: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
    - String args[]: It is the parameter passed to the main method.

2. **Why Java is platform independent**
   Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

3. **Java is not 100% Object-oriented**
   Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

4. **Wrapper classes**
   Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

5. **Constructors**
   In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.
   There are two types of constructors:

Default Constructor: In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.

Parameterized Constructor: The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

6. **Singleton class**
   Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

7. **Difference between equals() and ==**
   Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.
   "==" or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. *public boolean equals(Object o)* is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

8. **Difference between a local variable and an instance variable**
   In Java, a local variable is typically used inside a method, constructor, or a block and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won't be even aware of that variable.
   Whereas, an instance variable in Java, is a variable which is bounded to its object itself. These variables are declared within a class, but outside a method. Every object of that class will create its own copy of the variable while using it. Thus, any changes made to the variable won't reflect in any other instances of that class and will be bound to that particular instance only.

9. **Final keyword**
   final is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

- **final variable**
  When the final keyword is used with a variable then its value can't be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.
- **final method**
  When a method is declared final then it can't be overridden by the inheriting class.
- **final class**
  When a class is declared as final in Java, it can't be extended by any subclass but it can extend other class.

10. Why does Java not make use of pointers

Pointers are quite complicated and unsafe to use by beginner programmers. Java focuses on code simplicity, and the usage of pointers can make it challenging. Pointer utilization can also cause potential errors. Moreover, security is also compromised if pointers are used because the users can directly access memory with the help of pointers.
Thus, a certain level of abstraction is furnished by not including pointers in Java. Moreover, the usage of pointers can make the procedure of garbage collection quite slow and erroneous. Java makes use of references as these cannot be manipulated, unlike pointers.

11. Difference between constructors and methods

Methods	Constructors
Used to represent the behavior of an object	Used to initialize the state of an object
Must have a return type	Do not have any return type
Needs to be invoked explicitly	Is invoked implicitly
No default method is provided by the compiler	A default constructor is provided by the compiler if the class has none
Method name may or may not be same as class name	Constructor name must always be the same as the class name

12. Difference between static and non-static methods

Static Method	Non-static method
The static keyword must be used before the method name	No need to use the static keyword before the method name
They can't access any non-static	It can access any static method and

instance variables or methods	any static variable without creating an instance of the class

13. **Collection class in Java**

In Java, the collection is a framework that acts as an architecture for storing and manipulating a group of objects. Using Collections you can perform various tasks like searching, sorting, insertion, manipulation, deletion, etc. Java collection framework includes the following:

- Interfaces
- Classes
- Methods

14. **Super keyword**

- The super keyword is used to access hidden fields and overridden methods or attributes of the parent class.
- Following are the cases when this keyword can be used:

a. Accessing data members of parent class when the member names of the class and its child subclasses are the same.
b. To call the default and parameterized constructor of the parent class inside the child class.
c. Accessing the parent class methods when the child classes have overridden them.

15. **Garbage collection**

The main objective of this process is to free up the memory space occupied by the unnecessary and unreachable objects during the Java program execution by deleting those unreachable objects.This ensures that the memory resource is used efficiently, but it provides no guarantee that there would be sufficient memory for the program execution. Heap memory is cleaned in garbage collection process.