**INTERVIEW QUESTIONS FOR OBJECT-ORIENTED PROGRAMMING**

**Assignment 2**

**1.Explain OOPS?**

As the name suggests, [Object-Oriented Programming](https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/) or OOPs refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks assigned by you. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming. The main aim 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.

Let us discuss prerequisites by polishing concepts of method declaration and message passing. Starting off with the method declaration, it consists of six components:

* [**Access Modifier**](https://www.geeksforgeeks.org/access-modifiers-java/): Defines the **access type** of the method i.e. from where it can be accessed in your application. In Java, there are 4 types of access specifiers:
  + **public:** Accessible in all classes in your application.
  + **protected:** Accessible within the package in which it is defined and in its **sub class(es) (including subclasses declared outside the package)**.
  + **private:** Accessible only within the class in which it is defined.
  + **default (declared/defined without using any modifier):** Accessible within the same class and package within which its class is defined.
* **The return type**: The data type of the value returned by the method or void if it does not return a value.
* **Method Name**: The rules for field names apply to method names as well, but the convention is a little different.
* **Parameter list**: Comma-separated list of the input parameters that are defined, preceded by their data type, within the enclosed parentheses. If there are no parameters, you must use empty parentheses ().
* **Exception list**: The exceptions you expect the method to throw. You can specify these exception(s).
* **Method body**: It is the block of code, enclosed between braces, that you need to execute to perform your intended operations.

[**Message Passing**](https://www.geeksforgeeks.org/message-passing-in-java/)**:** Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

Now that we have covered the basic prerequisites, we will move on to the 4 pillars of OOPs which are as follows. But, let us start by learning about the different characteristics of an Object-Oriented Programming Language.

OOPS concepts are as follows:

1. [Class](https://www.geeksforgeeks.org/classes-objects-java/)
2. [Object](https://www.geeksforgeeks.org/classes-objects-java/)
3. [Method](https://www.geeksforgeeks.org/methods-in-java/)and [method passing](https://www.geeksforgeeks.org/message-passing-in-java/)
4. Pillars of OOPs
   * [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
   * [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
   * [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
   * [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)
     + Compile-time polymorphism
     + Runtime polymorphism

**2. Explain an abstraction? Real life example.**

**Abstraction in Java** is another [OOPs principle](https://www.scientecheasy.com/2020/07/oops-concepts-in-java.html/) that manages complexity. It is a process of hiding complex internal implementation details from the user and providing only necessary functionality to the users.

In other words, abstraction in Java is a technique by which we can hide the data that is not required to a user.

It hides all unwanted data so that users can work only with the required data. It removes all non-essential things and shows only important things to users.

That is, every user will get the required data and will not get confused with unnecessary data.

Let’s take some real time examples to understand the concept of java abstraction.

**Realtime Examples of Abstraction in Java**

**1**. Let’s first take **ATM machine** as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

**2.Making coffee with a coffee machine** is a good example of abstraction. You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get.

**3. Explain encapsulation? Real life example.**

The process of binding data and corresponding [methods](https://www.scientecheasy.com/2020/06/java-methods.html/) (behavior) together into a single unit is called **encapsulation in Java**.

In other words, encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes.

Thereby, we can keep variables and methods safes from outside interference and misuse.

Every Java class is an example of encapsulation because we write everything within the class only that binds variables and methods together and hides their complexity from other classes.

Another example of encapsulation is a capsule. Basically, capsule encapsulates several combinations of medicine.

Consider the below real time example: Encapsulation: As a driver you know how to start the car by pressing the start button and internal details of the starting operations are hidden from you. So the entire starting process is hidden from you otherwise we can tell starting operation is encapsulated from you.

**4.Explain the relationship among abstraction and encapsulation?**

| **Parameter** | **Abstraction** | **Encapsulation** |
| --- | --- | --- |
| Use for | Abstraction solves the problem and issues that arise at the design stage. | Encapsulation solves the problem and issue that arise at the implementation stage. |
| Focus | Abstraction allows you to focus on what the object does instead of how it does it | Encapsulation enables you to hide the code and data into a single unit to secure the data from the outside world. |
| Implementation | You can use abstraction using Interface and Abstract Class. | You can implement encapsulation using Access Modifiers (Public, Protected & Private.) |
| Focuses | Focus mainly on what should be done. | Focus primarily on how it should be done. |
| Application | During design level. | During the Implementation level. |

**5.Explain polymorphism?**

**Polymorphism** is an object-oriented programming concept that refers to the ability of a variable, function, or object to take on multiple forms. In a programming language exhibiting polymorphism, class objects belonging to the same hierarchical tree (inherited from a common parent class) may have functions with the same name, but with different behaviors.

The classic example is of the **Shape class** and all the classes that are inherited from it, such as:

* Rectangle
* Triangle
* Circle

**6. Explain Inheritance?**

**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](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) 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.

### **Why use inheritance in java**

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) 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.

**7. How composition is better than inheritance?**

Favoring[Composition](https://www.geeksforgeeks.org/composition-in-java/) over [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)is a principle in [object-oriented programming (OOP).](https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/)Classes should achieve polymorphic behavior and code reuse by their composition rather than inheritance from a base or parent class. To get the higher design flexibility, the design principle says that composition should be favored over inheritance.

Inheritance should only be used when subclass ‘is a’ superclass. Don’t use inheritance to get code reuse. If there is no ‘is a’ relationship, then use composition for code reuse.

Reasons to Favour Composition over Inheritance in Java and OOP:

1. The fact that Java does not support multiple inheritances is one reason for favoring composition over inheritance in Java. Since you can only extend one class in Java, but if you need multiple features, such as reading and writing character data into a file, you need Reader and Writer functionality. It makes your job simple to have them as private members, and this is called Composition.
2. Composition offers better test-ability of a class than Inheritance. If one class consists of another class, you can easily construct a Mock Object representing a composed class for the sake of testing. This privilege is not given by inheritance.
3. Although both Composition and Inheritance allow you to reuse code, one of Inheritance’s disadvantages is that it breaks encapsulation. If the subclass depends on the action of the superclass for its function, it suddenly becomes fragile. When super-class behavior changes, sub-class functionality can be broken without any modification on its part.
4. In the timeless classic Design Patterns, several object-oriented design patterns listed by Gang of Four: Elements of Reusable Object-Oriented Software, favor Composition over Inheritance. Strategy design pattern, where composition and delegation are used to modify the behavior of Context without touching context code, is a classical example of this. Instead of getting it by inheritance, because Context uses composition to carry strategy, it is simple to have a new implementation of strategy at run-time.
5. Another reason why composition is preferred over inheritance is flexibility. If you use Composition, you are flexible enough to replace the better and updated version of the Composed class implementation. One example is the use of the comparator class, which provides features for comparison.

**8. Which OOPS concept is used as a reuse mechanism?**

Inheritance is the feature that provides a reuse mechanism.

This mechanism provides reusability to the user. While abstraction, encapsulation, and dynamic binding have different functionalities in the OOP paradigm

**9. Which OOPS concept exposes only the necessary information to the calling functions?**

* Data hiding is a technique used in object-oriented programming.
* It means hiding the internal details.
* Data hiding makes sure that the internal details are restricted to class members.
* Data integrity is maintained in data hiding.
* Data hiding reduces the complexities and increases the robustness.
* Another main advantage is that it reduces the interdependencies between two software.

**10. Explain a class? Create a class.**

In the real world, you often have many objects of the same kind. For example, your bicycle is just one of many bicycles in the world. Using object-oriented terminology, we say that your bicycle object is an [*instance*](javascript:var%20meth=openWin;%20meth('instance');) of the class of objects known as bicycles. Bicycles have some state (current gear, current cadence, two wheels) and behavior (change gears, brake) in common. However, each bicycle's state is independent of and can be different from that of other bicycles.

When building bicycles, manufacturers take advantage of the fact that bicycles share characteristics, building many bicycles from the same blueprint. It would be very inefficient to produce a new blueprint for every individual bicycle manufactured.

In object-oriented software, it's also possible to have many objects of the same kind that share characteristics: rectangles, employee records, video clips, and so on. Like the bicycle manufacturers, you can take advantage of the fact that objects of the same kind are similar and you can create a blueprint for those objects. A software blueprint for objects is called a [*class*](javascript:var%20meth=openWin;%20meth('class');).

**11. Using above created class, Write in brief abstraction and encapsulation**

**12. Explain difference among class and object?**

|  |  |  |
| --- | --- | --- |
| No. | Object | Class |
| 1) | Object is an instance of a class. | Class is a blueprint or template from which objects are created. |
| 2) | Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. | Class is a group of similar objects. |
| 3) | Object is a physical entity. | Class is a logical entity. |
| 4) | Object is created through new keyword mainly e.g. Student s1=new Student (); | Class is declared using class keyword e.g. class Student {} |
| 5) | Object is created many times as per requirement. | Class is declared once. |
| 6) | Object allocates memory when it is created. | Class doesn't allocated memory when it is created. |
| 7) | There are many ways to create object in java such as new keyword, new Instance () method, clone () method, factory method and deserialization. | There is only one way to define class in java using class keyword. |

**13. Define access modifiers?**

Access modifiers are keywords used to specify the accessibility of a class (or type) and its members. These modifiers can be used from code inside or outside the current application.  
  
Access modifiers in .NET are used to control the accessibility of each of the members of a type from different possible areas of code. This can be handled from within the current assembly or outside it. An assembly represents a logical unit of functionality and consists of types and resources located in one or more files. The purpose of using access modifiers is to implement encapsulation, which separates the interface of a type from its implementation. With this, the following benefits can be derived:

* Prevention of access to the internal data set by users to invalid state.
* Provision for changes to internal implementation of the types without affecting the components using it.
* Reduction in complexity of the system by reducing the interdependencies between software components.

**14. Explain an object? Create an object of above class.**

The object is a basic building block of an [OOPs](https://www.javatpoint.com/java-oops-concepts) language. In Java, we cannot execute any program without creating an object. There is various way to create an [object in Java](https://www.javatpoint.com/object-and-class-in-java) that we will discuss in this section, and also learn how to create an object in Java.

[Java](https://www.javatpoint.com/java-tutorial) provides five ways to create an object.

* Using new Keyword
* Using clone() method
* Using newInstance() method of the Class class
* Using newInstance() method of the Constructor class
* Using Deserialization

Using the **new** keyword is the most popular way to create an object or instance of the class. When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created **object** and also returns the **reference** of that object to that memory. The new keyword is also used to create an array. The syntax for creating an object is:

1. ClassName object = new ClassName();

**15. Give real life examples of object.**

An object represents data (attributes) and its related methods (behaviour) as a single unit. Take the example of a car. When we describe a car as an object, there are a number of physical factors which contribute to define its state such as its brand name, colour, speed, size, number of seats, etc. In addition, the car has several functional definitions. It transports people to different locations by controlling its speed and direction, an accelerator to increase or decrease its speed and brakes to stop it. Thus we can say that car is an object that combines its attributes and behaviour as a single unit

**16. Explain a Constructor.**

A constructor is a special method of a class or structure in object-oriented programming that initializes a newly created object of that type. Whenever an object is created, the constructor is called automatically.

A constructor is like an instance method that usually has the same name as the class, and can be used to set the values of the members of an object, either to default or to user-defined values. However, although it resembles it, a constructor is not a proper method since it doesn’t have a return type. Instead of performing a task by executing code, the constructor initializes the object, and it cannot be static, final, abstract, and synchronized.

**17. Define the various types of constructors?**

Types of Constructor

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor.

1. Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

1. Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

**18. Whether static method can use non static members?**

Static methods cannot use non-static variables because static methods can be invoked on classes that have not been instantiated (created). Member non-static variables belong to an instance of a class and only are allocated and assigned values when a class instance is created.

Until then, the variables don’t exist and thus cannot be referenced.

**19. Explain Destructor?**

In Java, when we create an object of the class it occupies some space in the memory (heap). If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming. To resolve this problem, we use the **destructor**. In this section, we will discuss the alternate option to the **destructor in Java**. Also, we will also learn how to use the **finalize()** method as a destructor.

The **destructor** is the opposite of the constructor. The constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object.

Remember that **there is no concept of destructor in Java**. In place of the destructor, Java provides the garbage collector that works the same as the destructor. The [garbage collector](https://www.javatpoint.com/Garbage-Collection) is a program (thread) that runs on the [JVM](https://www.javatpoint.com/jvm-java-virtual-machine). It automatically deletes the unused objects (objects that are no longer used) and free-up the memory. The programmer has no need to manage memory, manually. It can be error-prone, vulnerable, and may lead to a memory leak.

It is a special method that automatically gets called when an object is no longer used. When an object completes its life-cycle the garbage collector deletes that object and deallocates or releases the memory occupied by the object.

It is also known as **finalizers** that are non-deterministic. In [Java](https://www.javatpoint.com/java-tutorial), the allocation and deallocation of objects handled by the **garbage collector**. The invocation of finalizers is not guaranteed because it invokes implicitly.

**20. Explain an Inline function?**

An inline function is **one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate set of instructions in memory**. This eliminates call-linkage overhead and can expose significant optimization opportunities.

**21. Explain a virtual function?**

A virtual function or virtual method in an OOP language is a function or method used to override the behavior of the function in an inherited class with the same signature to achieve the polymorphism.

When the programmers switch the technology from [C++](https://www.javatpoint.com/cpp-tutorial) to [Java](https://www.javatpoint.com/java-tutorial), they think about where is the virtual function in Java. In C++, the virtual function is defined using the **virtual** keyword, but in Java, it is achieved using different techniques. See [Virtual function in C++](https://www.javatpoint.com/cpp-virtual-function).

Java is an object-oriented programming language; it supports OOPs features such as [polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java), abstraction, [inheritance](https://www.javatpoint.com/inheritance-in-java), etc. These concepts are based on objects, classes, and member functions.

By default, all the instance methods in Java are considered as the Virtual function except final, static, and private methods as these methods can be used to achieve polymorphism.

**22. Explain a friend function?**

Java does not have the friend keyword like [c++](https://www.codingninjas.com/codestudio/library/c" \t "_blank), which is used to access the non-public members of a class. Nevertheless, we can achieve this functionality.

Before we try to implement friend functionality, we need to see what it is precisely in c++ and why we need it, and what we can do using this keyword.

A friend class gets access to private and protected members of another class in which it is declared a friend. When granting access to a class, you need to specify that the access is granted to a class using the**'friend'** keyword ➡ **friend class x class;**

**23. Explain function overloading?**

Function Overloading in Java occurs when there are functions having the same name but have different numbers of parameters passed to it, which can be different in data like int, double, float and used to return different values are computed inside the [respective overloaded method](https://www.educba.com/method-overloading-in-java/). Function overloading is used to reduce complexity and increase the efficiency of the program by involving more functions that are segregated and can be used to distinguish among each other with respect to their individual functionality. Overloaded functions are related to compile-time or [static polymorphism](https://www.educba.com/polymorphism-in-java/). There is also a concept of type conversion, which is basically used in overloaded functions used to calculate the conversion of type in variables.

**Syntax**

Overloaded functions have the same name but different types of arguments or parameters assigned to them. They can be used to calculate mathematical or logical operations within the number of assigned variables in the method. The syntax of the overloaded function can be given below, where there are up to N number of variables assigned.

### Working of Function Overloading

Function overloading works by calling different functions having the same name, but the different number of arguments passed to it. There are many coding examples that can be shown in order to identify the benefits and disadvantages of function overloading properly.

**24. Explain a base class, sub class, super class?**

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

**25. Write in brief linking of base class, sub class and base object, sub object.**

**26. Explain an abstract class?**

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial)

. It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### **Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor)

and static methods also.

* It can have final methods which will force the subclass not to change the body of the method.

**27. Explain operator overloading?**

**28. Define different types of arguments? (Call by value/Call by reference)**

| Call By Value | Call By Reference |
| --- | --- |
| While calling a function, we pass values of variables to it. Such functions are known as “Call By Values”. | While calling a function, instead of passing the values of variables, we pass address of variables(location of variables) to the function known as “Call By References. |
| In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function. | In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function. |
| With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function. | With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them. |

**29. Explain the super keyword?**

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

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

**30. Explain method overriding?**

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.

### Usage of Java Method Overriding

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### 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).

|  |  |
| --- | --- |
| Overriding | Overloading |
| Implements “runtime polymorphism” | Implements “compile time polymorphism” |
| The method call is determined at runtime based on the object type | The method call is determined at compile time |
| Occurs between superclass and subclass | Occurs between the methods in the same class |
| Have the same signature (name and method arguments) | Have the same name, but the parameters are different |
| On error, the effect will be visible at runtime | On error, it can be caught at compile time |

**31. Difference among overloading and overriding?**

**32. Whether static method can use non-static members?**

Static methods cannot use non-static variables because static methods can be invoked on classes that have not been instantiated (created). Member non-static variables belong to an instance of a class and only are allocated and assigned values when a class instance is created.

Until then, the variables don’t exist and thus cannot be referenced.

**33. Explain a base class, sub class, super class?**

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

**34. Write in brief linking of base class, sub class and base object, sub object.**

**35. Explain an interface?**

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [abstraction](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

**36. Explain exception handling?**

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

In this tutorial, we will learn about Java exceptions, it's types, and the difference between checked and unchecked exceptions.

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

**37. Explain the difference among structure and a class?**

**Class**

* It is defined using ‘class’ keyword.
* When data is defined in a class, it is stored in memory as a reference.
* It gets memory allocated only when an object of that class is created.
* The reference type (before creating an object) is allocated on heap memory.
* They can have constructors and destructors.
* It can use inheritance to inherit properties from base class.
* The ‘protected’ access modifier can be used with the data members defined inside the class.

Structure

* The ‘struct’ keyword is used to define a structure.
* Every member in the structure is provided with a unique memory location.
* When the value of one data member is changed, it doesn’t affect other data members in structure.
* It helps initialize multiple members at once.
* Total size of the structure is equivalent to the sum of the size of every data member.
* It is used to store various data types.
* It takes memory for every member which is present within the structure.
* A member can be retrieved at a time.
* It supports flexible arrays.
* Its instance can be created without a keyword.
* It doesn’t support protected access modifier.
* It doesn’t support inheritance.
* It doesn’t have a constructor or destructor.
* The values allocated to structures are stored in stack memory.

**38. Explain the default access modifier in a class?**

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

**39. Explain a pure virtual function?**

Pure virtual function can be defined as the functions that are declared inside a base class without any relative definition. Hence, each derived class must re-declare it as a pure virtual function or redefine it. They are also called as “do-nothing” functions as their definitions are empty and they are of the form, virtual void display() = 0;

The display () in the above declaration is a pure virtual function with no definition relative to the base class. The assigned operator does not specify that zero is assigned to this function, instead it is used to tell the compiler that the declared function is a pure virtual function and that it will not have a definition. Moreover, a class consisting of pure virtual function is called abstract class or pure abstract class. This abstract class cannot declare any object of its own. Doing so may lead to an error from the compiler “cannot create instance of an abstract class”. Besides this, even if the base class declares a pure virtual function, this function cannot perform any operation and cannot be used to declare the objects as well.

**40. Explain dynamic or run time polymorphism?**

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. 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.

Dynamic Polymorphism in Java

In Java, **polymorphism** is a concept of object-oriented programming that allows us to perform a single action in different forms. In this section, we will discuss only the **dynamic polymorphism in Java.**

**Dynamic polymorphism** is a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time. It is also known as [**runtime polymorphism**](https://www.javatpoint.com/runtime-polymorphism-in-java) or **dynamic method dispatch**. We can achieve dynamic polymorphism by using the [**method overriding**](https://www.javatpoint.com/method-overriding-in-java).

In this process, an overridden method is called through a 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

Properties of Dynamic Polymorphism

* It decides which method is to execute at runtime.
* It can be achieved through dynamic binding.
* It happens between different classes.
* It is required where a subclass object is assigned to a super-class object for dynamic polymorphism.
* Inheritance involved in dynamic polymorphism.

**41. Do we require a parameter for constructors?**

**42. Explain static and dynamic binding?**

**Static Binding** When compiler acknowledges all the information required to call a function or all the values of the variables during compile time, it is called “static binding”. As all the required information are known before runtime, it increases the program efficiency and it also enhances the speed of execution of a program. Static Binding makes a program very efficient, but it declines the program flexibility, as ‘values of variable’ and ‘function calling’ are predefined in the program. Static binding is implemented in a program at the time of coding. Overloading a function or an operator are the example of compile time polymorphism i.e. static binding.

**Dynamic Binding** Calling a function or assigning a value to a variable, at run-time is called “Dynamic Binding”. Dynamic binding can be associated with run time ‘polymorphism’ and ‘inheritance’ in OOP. Dynamic binding makes the execution of program flexible as it can be decided, what value should be assigned to the variable and which function should be called, at the time of program execution. But as this information is provided at run time it makes the execution slower as compared to static binding.

**43. How many instances can be created for an abstract class?**

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure. For an abstract class in the OOP paradigm, we cannot instantiate it.

**44. Explain the default access specifiers in a class definition?**

It is not a keyword. Any Java members such as class or methods or data members when not specified with any access modifier they are by default considered as **default access modifiers.**These methods or data members are **only accessible within the same package** and they cannot be accessed from outside the package.

It provides more visibility than a private access modifier. But this access modifier is more restricted than protected and public access modifiers.Let us consider an example for the default access modifier.Here, we have two different packages p1 and p2. In the p1 package, we have class A1 where we declared a default variable and a default method. Now we are trying to access this variable and method from outside the package that is from package p2 which has a class A2.

When we try to access these variables and methods from outside the package we get a Compile time error.Hence, we conclude that the **default access modifier members**can be accessed only within the same package and cannot be accessed from outside the package. And they have more visibility than private access modifier but is more restricted than protected and public access modifiers.

**45. Which OOPS concept is used as reuse mechanism?**

Inheritance is the feature that provides a reuse mechanism.

This mechanism provides reusability to the user. While abstraction, encapsulation, and dynamic binding have different functionalities in the OOP paradigm

**46. Define the Benefits of Object Oriented Programming?**

• Code reusability New objects can be derived from old objects, allowing for improvement and refinement of the code at each stage and also preserving parts of the code for other programs. This is used to develop many class libraries using class codes that have already been written, for example, Microsoft Foundation Classes (MFC).

Code Modularity Everything in OOP is an object; these objects can be interchanged or removed to meet the users’ needs .

• Easier maintenance Inheritance usually reduces maintenance because of the ‘domino effect it has on derived classes when a change is made in a base class.

• Design stability Once a stable base class has been developed, the new classes that are derived may have fewer less errors and bugs.

• Improved communication between developers and users Objects can be broken down into real life entities, hence it is easier it communicate ideas.

• Seamless transition from design to implementation This is mainly because communications are improved.

**47. Explain method overloading?**

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the [program](https://www.javatpoint.com/java-programs).

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

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

**48. Explain the difference among early binding and late binding?**

| Early Binding | Late Binding |
| --- | --- |
| It is a compile-time process | It is a run-time process |
| The method definition and method call are linked during the compile time. | The method definition and method call are linked during the run time. |
| Actual object is not used for binding. | Actual object is used for binding. |
| For example: Method overloading | For example: Method overriding |
| Program execution is faster | Program execution is slower |

**49. Explain early binding? Give examples?**

Early binding benefits include:

* The compiler can perform optimization, which results in more efficient applications
* Early-bound objects are faster, have better code readability and are easily maintained.
* Using Visual Studio's integrated development environment (IDE) as a development tool, early binding aids in rapid application development (RAD) by providing quick access to object properties and methods - reducing typographical errors.
* Errors are revealed during compilation, which reduces runtime error severity and frequency.

During early binding, the C# compiler performs syntax and type checks to ensure that the correct parameter amount and type are passed to the method or property. Early binding also checks for the return value, which minimizes execution time and runtime errors.  
  
For example, a C# automation client application needs to calculate a value with a defined Microsoft Excel formula. The application may create an object of type Microsoft Excel and call its required method after initializing the object properties. Because Microsoft Excel is a specific type of object, the created and assigned variable is an early-bound object.  
  
While working with early-bound objects, it is necessary to include the type library where the definitions of early-bound object types are defined. Additionally, an early-bound object cannot be used to assign an object of another type after its declaration.

**50.Explain loose coupling and tight coupling?**

**Tight coupling :** In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.

**Example :**If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI(Remote Method Invocation).

|  |
| --- |
| // Java program to illustrate  // tight coupling concept  class Subject {      Topic t = new Topic();      public void startReading()      {          t.understand();      }  }  class Topic {      public void understand()      {          System.out.println("Tight coupling concept");      }  } |

**Explanation:** In the above program the Subject class is dependents on Topic class. In the above program Subject class is tightly coupled with Topic class it means if any change in the Topic class requires Subject class to change. For example, if Topic class understand() method change to gotit() method then you have to change the startReading() method will call gotit() method instead of calling understand() method.

**Loose coupling :**In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling.  
**Example :**If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

|  |
| --- |
| // Java program to illustrate  // loose coupling concept  **public** **interface** Topic  {  **void** understand();  }  **class** Topic1 **implements** Topic {  **public** **void** understand()      {          System.out.println("Got it");      }  } **class** Topic2 **implements** Topic {  **public** **void** understand()      {          System.out.println("understand");      }  } **public** **class** Subject {  **public** **static** **void** main(String[] args)      {          Topic t = **new** Topic1();          t.understand();      }  } |

**Explanation :** In the above example, Topic1 and Topic2 objects are loosely coupled. It means Topic is an interface and we can inject any of the implemented classes at run time and we can provide service to the end user.

**51. Give an example among tight coupling and loose coupling.**

**Example :**If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. **The examples of Loose coupling are Interface, JMS.**

|  |
| --- |
| // Java program to illustrate  // loose coupling concept  **public** **interface** Topic  {  **void** understand();  }  **class** Topic1 **implements** Topic {  **public** **void** understand()      {          System.out.println("Got it");      }  } **class** Topic2 **implements** Topic {  **public** **void** understand()      {          System.out.println("understand");      }  } **public** **class** Subject {  **public** **static** **void** main(String[] args)      {          Topic t = **new** Topic1();          t.understand();      }  } |

**Explanation :** In the above example, Topic1 and Topic2 objects are loosely coupled. It means Topic is an interface and we can inject any of the implemented classes at run time and we can provide service to the end user.

**Example :**If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. **The best example of tight coupling is RMI(Remote Method Invocation).**

|  |
| --- |
| // Java program to illustrate  // tight coupling concept  class Subject {      Topic t = new Topic();      public void startReading()      {          t.understand();      }  }  class Topic {      public void understand()      {          System.out.println("Tight coupling concept");      }  } |

**Explanation:** In the above program the Subject class is dependents on Topic class. In the above program Subject class is tightly coupled with Topic class it means if any change in the Topic class requires Subject class to change. For example, if Topic class understand() method change to gotit() method then you have to change the startReading() method will call gotit() method instead of calling understand() method.

**52. Write in brief abstract class.**

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).Before learning the Java abstract class, let's understand the abstraction in Java first.A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### **Points to Remember**

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.

In java, the following some *important observations*about abstract classes are as follows:

1. An instance of an abstract class can not be created.
2. Constructors are allowed.
3. We can have an abstract class without any abstract method.
4. There can be a **final method** in abstract class but any abstract method in class(abstract class) can not be declared as final  or in simpler terms final method can not be abstract itself as it will yield an error: “Illegal combination of modifiers: abstract and final”
5. We can define static methods in an abstract class
6. We can use the **abstract keyword** for declaring ***top-level classes (Outer class) as well as inner classes*** as abstract
7. If a**class** contains at least **one abstract method**then compulsory should declare a class as abstract
8. If the**Child class** is unable to provide implementation to all abstract methods of the**Parent class**then we should declare that **Child class as abstract**so that the next level Child class should provide implementation to the remaining abstract method

**53. Define the Benefits of oops over pop?**

1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.

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

3)OOPs provides 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.

**54. Explain Generalization and Specialization?**

The process of extracting common characteristics from two or more classes and combining them into a generalized superclass is called Generalization. The common characteristics can be attributes or methods.

The Generalization is the process that does the grouping entities into broader or distributed categories based on certain common attributes. All the common attributes bind together to form a higher-level component or element is called a generalized entity. The two different types of entities in a university's database, for example, can be Students and Professors where the students come under the professor entity.

Generalization is represented by a triangle followed by a line.

Specialization is the reverse process of Generalization means creating new subclasses from an existing class.

Specialization is the process of dividing a parent-level entity into narrower categories accordingly to all the possible child categories. By having the behavior of the opposite of the generalization process, specialization requires the separation of entities based on certain uncommon attributes.

Specialization is quite useful in a situation where you block out the unnecessary data so you can locate or identify the specific information whenever it requires. specialized kinds of entities can be specialized further due to the higher chance of further modularity.

Let’s take an example of a Bank Account; A Bank Account is of two types – A current Account and Saving Account. Current Account and Saving Account inherits the common/ generalized properties like Account Number, Account Balance, etc. from a Bank Account and also have their own specialized properties like interest rate, etc.

**55. Write in brief Association, Aggregation and Composition?**

**Association**

Association relationship is a structural relationship in which different objects are linked within the system. It exhibits a binary relationship between the objects representing an activity. It depicts the relationship between objects, such as a teacher, can be associated with multiple teachers.

It is represented by a line between the classes followed by an arrow that navigates the direction, and when the arrow is on both sides, it is then called a bidirectional association. We can specify the multiplicity of an association by adding the adornments on the line that will denote the association.

**Aggregation**

Aggregation is a subset of association, is a collection of different things. It represents has a relationship. It is more specific than an association. It describes a part-whole or part-of relationship. It is a binary association, i.e., it only involves two classes. It is a kind of relationship in which the child is independent of its parent.

For example:

Here we are considering a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.

**Composition**

The composition is a part of aggregation, and it portrays the whole-part relationship. It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted. It exists between similar objects.

As you can see from the example given below, the composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

**56. Write in brief Object Composition vs. Inheritance.**

| S.NO | Inheritance | Composition |
| --- | --- | --- |
| 1. | In inheritance, we define the class which we are inheriting(super class) and most importantly it cannot be changed at runtime | Whereas in composition we only define a type which we want to use and which can hold its different implementation also it can change at runtime. Hence, Composition is much more flexible than Inheritance. |
| 2. | Here we can only extend one class, in other words more than one class can’t be extended as java do not support multiple inheritance. | Whereas composition allows to use functionality from different class. |
| 3. | In inheritance we need parent class in order to test child class. | Composition allows to test the implementation of the classes we are using independent of parent or child class. |
| 4. | Inheritance cannot extend final class. | Whereas composition allows code reuse even from final classes. |
| 5. | It is an **is-a** relationship. | While it is a **has-a** relationship. |

**57. Explain cohesion?**

**58. Explain “black-box-reuse” and “white-box-reuse”?**

White-box:

pros:

* simple (very natural concept)
* you have more control over things

cons:

* requires intrinsic knowledge on component internals
* can be difficult to implement (OO inheritance constraints)  
  sometimes it leads to broken\incorrect inheritance chains

Black-box:

pros:

* low coupling (gives late binding and other goodies)

cons:

* not obvious (code is much harder to understand)
* interfaces are more fragile than classes (i.e. interfaces vs inheritance)

**59. Explain “this”**

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.

Here is given the 6 usage of java this keyword.

[this can be used to refer current class instance variable.](https://www.javatpoint.com/this1)

[this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)

[this() can be used to invoke current class constructor.](https://www.javatpoint.com/this3)

[this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)

[this can be passed as argument in the constructor call.](https://www.javatpoint.com/this5)

[this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)

**60. Write in brief static member and member functions.**

**61. How will you relate unrelated classes or how will you achieve polymorphism without using the base class?**

**62. Explain the Diamond problem?**

In **Java, the diamond problem** is related to multiple inheritance. Sometimes it is also known as the **deadly diamond problem** or **deadly diamond of death**. In this section, we will learn **what is the demand problem in Java** and **what is the solution to the diamond problem**.

The diamond problem is a common problem in Java when it comes to inheritance. Inheritance is a very popular property in an object-oriented programming language, such as [C++](https://www.javatpoint.com/cpp-tutorial), [Java](https://www.javatpoint.com/java-tutorial), etc. There are different types of inheritance such as, single, multiple, multi-level, and hybrid inheritance. But remember that **Java does not support the multiple inheritance** because of the diamond problem.

As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super-class A, then it is called simple inheritance, and Java supports them.

Multi-level inheritance allows a child class to inherit properties from a class that can inherit properties from some other classes. For example, class C can inherit its property from B class which itself inherits from A class. Java also supports them.

What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**. In the above figure, we find that class D is trying to inherit form class B and class C, that is not allowed in Java.

It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPPs languages. It is sometimes referred to as the **deadly diamond of death**.

**63. Explain the solution for diamond problem?**

The solution to the diamond problem is **default methods** and **interfaces**. We can achieve multiple inheritance by using these two things.

The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.

The advantage of interfaces is that it can have the same default methods with the same name and signature in two different interfaces. It allows us to implement these two interfaces, from a class. We must override the default methods explicitly with its interface name.

**64. Explain the need of abstract class?**

Abstract classes permit providing a *partial* set of default implementations of methods in a class. Since they're incomplete, they can't be instantiated and used as they stand, but they can be subclassed to add the missing details in a way that's specific to that particular implementations, and those subclasses can be instantiated.

Without abstract classes, you would have to provide dummy implementations of the methods you intend to override ... which could be done, but then there'd be the risk of forgetting to implement one of them. Having some methods remain entirely abstract ensures that the real implementations have to fill in the gaps, or continue to be abstract themselves and force their descendents to do so.

It's not something the language couldn't live without. But it's *very useful*. You'll discover just how useful as you become more proficient in Java and OO design.

**65. Why can’t we instantiate abstract class?**

1. Since an abstract class is nothing on its own, e.g. vehicle, we want to create an object of an concrete implementation, like Car, Bike, etc.
2. The constructor of an abstract class gets called during object chaining.
3. We can never directly create an object of an abstract class, even if it contains a constructor and all methods are implemented.

But from the compiler's perspective, why does Java enforce these rules?

**66. Can abstract class have constructors?**

You would define a constructor in an abstract class if you are in one of these situations:

* you want to perform some initialization (to fields of the abstract class) before the instantiation of a subclass actually takes place
* you have defined final fields in the abstract class but you did not initialize them in the declaration itself; in this case, you MUST have a constructor to initialize these fields

Note that:

* you may define more than one constructor (with different arguments)
* you can (should?) define all your constructors protected (making them public is pointless anyway)
* your subclass constructor(s) can call one constructor of the abstract class; it may even **have to** call it (if there is no no-arg constructor in the abstract class)

In any case, don't forget that if you don't define a constructor, then the compiler will automatically generate one for you (this one is public, has no argument, and does nothing).

**67. How many instances can be created for an abstract class?**

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure. For an abstract class in the OOP paradigm, we cannot instantiate it.

**68. Which keyword can be used for overloading?**

The **operator** keyword declares a function specifying what operator-symbol means when applied to instances of a class. This gives the operator more than one meaning, or "overloads" it. The compiler distinguishes between the different meanings of an operator by examining the types of its operands.

**69. Explain the default access specifiers in a class definition?**

It is not a keyword. Any Java members such as class or methods or data members when not specified with any access modifier they are by default considered as **default access modifiers.**These methods or data members are **only accessible within the same package** and they cannot be accessed from outside the package.

It provides more visibility than a private access modifier. But this access modifier is more restricted than protected and public access modifiers.Let us consider an example for the default access modifier.Here, we have two different packages p1 and p2. In the p1 package, we have class A1 where we declared a default variable and a default method. Now we are trying to access this variable and method from outside the package that is from package p2 which has a class A2.

When we try to access these variables and methods from outside the package we get a Compile time error.Hence, we conclude that the **default access modifier members**can be accessed only within the same package and cannot be accessed from outside the package. And they have more visibility than private access modifier but is more restricted than protected and public access modifiers.

**70. Define all the operators that cannot be overloaded?**

**71. Explain the difference among structure and a class?**

**Class**

* It is defined using ‘class’ keyword.
* When data is defined in a class, it is stored in memory as a reference.
* It gets memory allocated only when an object of that class is created.
* The reference type (before creating an object) is allocated on heap memory.
* They can have constructors and destructors.
* It can use inheritance to inherit properties from base class.
* The ‘protected’ access modifier can be used with the data members defined inside the class.

Structure

* The ‘struct’ keyword is used to define a structure.
* Every member in the structure is provided with a unique memory location.
* When the value of one data member is changed, it doesn’t affect other data members in structure.
* It helps initialize multiple members at once.
* Total size of the structure is equivalent to the sum of the size of every data member.
* It is used to store various data types.
* It takes memory for every member which is present within the structure.
* A member can be retrieved at a time.
* It supports flexible arrays.
* Its instance can be created without a keyword.
* It doesn’t support protected access modifier.
* It doesn’t support inheritance.
* It doesn’t have a constructor or destructor.
* The values allocated to structures are stored in stack memory.

**72. Explain the default access modifier in a class?**

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

**73. Can you list out the different types of constructors?**

Types of Constructor

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor.

1. Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

1. Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

**74. Explain a ternary operator?**

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

**75. Do We Require Parameter For Constructors?**

**76. Explain Sealed Modifiers?**

When applied to a class, the sealed modifier **prevents other classes from inheriting from it**. In the following example, class B inherits from class A , but no class can inherit from class B . You can also use the sealed modifier on a method or property that overrides a virtual method or property in a base class.

 I will discuss what is a sealed modifier, how to use it and what’s its impact on your application’s performance.

First of all, let’s start with a definition; sealed is a modifier that, if it’s applied to a class makes it **non-inheritable** and if applied to virtual methods or properties makes them **non-ovveridable**.

An example of its usage is **specialized class/method or property**in which potential alterations can make them stop working as expected (for example, the Pens class of the System.Drawing namespace).

Because a sealed class cannot be inherited, it cannot be used as **base class** and by consequence, an **abstract class** cannot use the sealed modifier. It’s also important to mention that **structs are implicitly sealed**.

**77. Explain The Difference Between New And Override.**

**78. How Can We Call The Base Method Without Creating An Instance?**

yes, it is possible

1)if it is a static method.

2)by inheriting from that class.

3)from derived classes using base keyword.

**79. Define The Various Types Of Constructors?**

Types of Constructor

In Java, constructors can be divided into 3 types:

1. No-Arg Constructor

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor.

1. Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).

1. Default Constructor

If we do not create any constructor, the Java compiler automatically create a no-arg constructor during the execution of the program. This constructor is called default constructor.

**80. Define Manipulators?**

**Manipulators** are helping functions that can modify the [input/output](https://www.geeksforgeeks.org/basic-input-output-c/) stream. It does not mean that we change the value of a variable, it only modifies the I/O stream using insertion (<<) and extraction (>>) operators.

* Manipulators are special functions that can be included in the I/O statement to alter the format parameters of a stream.
* Manipulators are operators that are used to format the data display.
* To access manipulators, the file iomanip.h should be included in the program.

**81. Can you give some examples of tokens?**

In Java, a program is a collection of classes and methods, while methods are a collection of various expressions and statements. Tokens in Java are the small units of code which a [Java compiler](https://www.edureka.co/blog/just-in-time-compiler/) uses for constructing those statements and expressions. Java supports 5 types of tokens which are:

1. [Keywords](https://www.edureka.co/blog/tokens-in-java/#keywords)
2. [Identifiers](https://www.edureka.co/blog/tokens-in-java/#identifiers)
3. [Literals](https://www.edureka.co/blog/tokens-in-java/#literals)
4. [Operators](https://www.edureka.co/blog/tokens-in-java/#operators)
5. [Special Symbols](https://www.edureka.co/blog/tokens-in-java/#specialsymbols)

Let’s now talk about each of them one by one.

**Keywords**

[Keywords in Java](https://www.edureka.co/blog/java-keywords/) are predefined or reserved words that have special meaning to the Java compiler. Each keyword is assigned a special task or function and cannot be changed by the user. You cannot use keywords as variables or identifiers as they are a part of Java syntax itself. A keyword should always be written in lowercase as Java is a case sensitive language. Java supports various keywords, some of them are listed below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 01. abstract | 02. boolean | 03. byte | 04. break | 05. class |
| 06. case | 07. catch | 08. char | 09. continue | 10. default |
| 11. do | 12. double | 13. else | 14. extends | 15. final |
| 16. finally | 17. float | 18. for | 19. if | 20. implements |
| 21. import | 22. instanceof | 23. int | 24. interface | 25. long |
| 26. native | 27. new | 28. package | 29. private | 30. protected |
| 31. public | 32. return | 33. short | 34. static | 35. super |
| 36. switch | 37. synchronized | 38. this | 39. throw | 40. throws |
| 41. transient | 42. try | 43. void | 44. volatile | 45. while |
| 46. assert | 47. const | 48. enum | 49. goto | 50. strictfp |

**82. Explain structured programming and its disadvantage?**

In structured programming design, programs are broken into different functions these functions are also known as modules, subprogram, subroutines and procedures.

Each function is design to do a specific task with its own data and logic. Information can be passed from one function to another function through parameters. A function can have local data that cannot be accessed outside the function’s scope. The result of this process is that all the other different functions are synthesized in an another function. This function is known as main function. Many of the high level languages supported structure programming.

Structured programming minimized the chances of the function affecting another. It supported to write clearer programs. It made global variables to disappear and replaced by the local variables. Due to this change one can save the memory allocation space occupied by the global variable. Its organization helped to understand the programming logic easily. So that one can easily understand the logic behind the programs. It also helps the new comers of any industrial technology company to understand the programs created by their senior workers of the industry. It also made debugging easier.

Functional abstraction was introduced with structured programming. Abstraction simply means that how able one can or we can say that it means the ability to look at something without knowing about its inner details. In structured programming, it is important to know that a given function satisfies its requirement and performs a specific task. Weather How that task is performed is not important.

Advantages of structured programming

The following are the different advantages of structured programming

1. It is user friendly and easy to understand.
2. Similar to English vocabulary of words and symbols.
3. It is easier to learn.
4. They require less time to write.
5. They are easier to maintain.
6. These are mainly problem oriented rather than machine based.
7. Program written in a higher level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator.
8. It is independent of machine on which it is used i.e. programs developed in high level languages can be run on any computer.

**83. When to use interface over abstract class.**

* If the functionality we are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated classes.
* Interfaces are a good choice when we think that the API will not change for a while.
* Interfaces are also good when we want to have something similar to multiple inheritances since we can implement multiple interfaces.
* If we are designing small, concise bits of functionality, use interfaces. If we are designing large functional units, use an abstract class.

**84. Explain a private constructor? Where will you use it?**

Private constructors**allow us to restrict the instantiation of a class**. Simply put, they prevent the creation of class instances in any place other than the class itself.

Public and private constructors, used together, allow control over how we wish to instantiate our classes – this is known as constructor delegation.

**85. Can you override private virtual methods?**

We can not even declare private virtual method, We can't use the virtual modifier with the static, abstract, private or override modifiers.

**86. Can you allow class to be inherited, but prevent from being over-ridden?**

Use 'private' access specified to prevent a method from being overridden in the derived class.  
The 'sealed' keyword can be applied to methods along with the 'override' keyword. That is, the keyword prevents further overriding of the method - class base declares a method M to be virtual. class d1 (which is derived from base) overrides the method along with 'sealed override' keywords. class d2 (which is derived from d1) cannot override M in this case. However if you apply the 'sealed' keyword without override keyword you get a com-piler error

**87. Why can’t you specify accessibility modifiers for methods inside interface?**

Interface methods are contract with the outside world which specifies that class implementing this interface does a certain set of things.

Interface members are always public because the purpose of an interface is to enable other types to access a class or struct.

Interfaces can have access specifiers like protected or internal etc. Thus limiting 'the outside world' to a subset of 'the whole outside world'.

**88. Can static members use non static members? Give reasons.**

**89. Define different ways a method can be overloaded?**

Method overloading in Java can be achieved in different ways. As we have understood that it is the list of parameters that differentiate the two methods with the same name in Java. The different ways of method overloading in Java can be achieved by varying parameters list in one of the below way.

Here are some of the different ways of method overloading in Java, along with example of method overloading in Java:

1. Number of parameters
2. The data type of parameters
3. The sequence of Data type of parameters

**The Number of Parameters**

As it is clear from the name that there will be a different number of parameters in the overloaded methods which will decide which method to execute seeing the method call statement. Below is the method overloading in Java, where the number of parameters varies.

Method1: add(int, int)

Method2: add(int, int, int)

Method calling statement add(20,30) will execute the method1 and Method calling statement add(10,20,30) will execute the method2.

**The Data Type of Parameters**

In this type of method overloading in Java, the two java methods of a class have the same name, and the number of parameters can be the same or different, but the data type of parameters will differ.

Method1: add(int, int)

Method2: add(int, float)

Method3: add(float, float)

In the above methods, the method name is the same, but the data type of the parameters is different. So, method calling statement add(2, 5.5) will execute the method2. Similarly, add(7.5, 10.25) will execute the method3.

**The Sequence of the Data Type of Parameters**

In this type of method overloading in Java, both the method name and the number of parameters is the same, but the difference lies in the sequence of data types of these parameters. Below is the example of overloaded methods:

Method1: add(int, float)

Method2: add(float, int)

Here, calling the statement to add(100, 75.5) will call the method1 to execute, and add(55.25, 150) will execute method2.

**90. Can we have an abstract class without having any abstract method?**

**91. Explain the default access modifier of a class?**

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

**92. Can function overriding be explained in same class?**

No, function overriding can only be defined in different class because it treated as base and derived or parent child relationship

Note: the override keyword is used in function overriding to override method.

**93. Does function overloading depends on Return Type?**

No, it does not depend on return type. because if return type is different and function name as well as parameter is also same.

Then it will give compile time error.

**94. Can abstract class have a constructor?**

As we all know abstract classes also do have a constructor. So if we do not define any constructor inside the abstract class then JVM (Java Virtual Machine) will give a default constructor to the abstract class

**95. Define rules of Function overloading and function overriding**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Function Overloading** | **Function Overriding** |
| 1. | In function overloading, two or more functions can own the same name, but the parameters will be different. | Function overriding permit us to redefine a method with the same name and signature |
| 2. | There is no requirement of the inheritance concept here. | In function overriding, we need an inheritance concept. |
| 3. | In the case of function overloading, the signatures should be different. | In the case of function overriding, the signatures should be the same. |
| 4. | We can use it as an example of compile time polymorphism. | We can use it as an example of run time polymorphism. |
| 5. | It happens during compile time. | It occurs during the run time. |
| 6. | A function has the ability to load multiple times. | A function can be overridden only a single time. |