# Why we need to use OOP ? Some major OOP languages ?

Object Oriented Programming (OOP) is a programming language model that focuses on objects and operations on objects, rather than logical operations. In OOP, programs are designed by making objects interact with each other. Languages ​​such as Java, C , C#, Python, PHP, JavaScript, Ruby, Perl, Smalltalk, Objective-C are the main object-oriented programming languages. In OOP theory, the realization of 4 basic features is considered mandatory, and if even one is missing, this language is not considered pure OOP. These:

* Encapsulation
* Inheritance
* Polymorphism
* It is Abstraction.

## Encapsulation

Encapsulation is one of the core concepts of OOP. By its general definition, it allows to set limits on how much of the data, classes and methods can be viewed and modified by the user. There are three access modifiers: public (public), private (private), and protected (protected). Public ones can be viewed and changed by anyone, so they are the most unsafe class type. While writing a program, it is not recommended to use Public methods to change the internal structure of the program. The public modifier is used for data that is requested to be added or modified by an external user. Protected modifier is a more secure access modifier than public modifier. They can be viewed or accessed within the same class. They can also be viewed or accessed by superclasses, derived classes, and classes contained within the same package. Private is the most secure access modifier. Private ones can only be seen or accessed by the class they are in. Classes can be private, as well as their properties and the data they hold.

## Inheritance

Inheritance is a mechanism that enables to create a parent-child relationship between one class and another class and to use common methods and properties on these classes. It is one of the core concepts of OOP. It allows other classes to be built on top of existing classes. There are 5 types of Inheritance.

• Single Inheritance: A subclass carries all the features of a single superclass.

• Multiple Inheritance: A subclass carries all the properties of more than one superclass.

• Multilevel Inheritance: After subclassing a class, it is called creating a subclass of this subclass.

• Hierarchical Inheritance: It is called base class (base class) of a superclass to more than one subclass.

• Hybrid Inheritance: An Inheritance type that contains 2 or more of the other Inheritance types.

## Polymorphism

Polymorphism is a feature of the programming language in OOP that specifies its ability to handle different types of data and classes differently. To be more specific is the ability to redefine methods and derived classes. For example, suppose there is a class called shape; Thanks to polymorphism, the programmer can determine the areas of different shapes with different methods. Regardless of what the shape is, the program will give the user the right space.

## Abstraction

Abstraction is one of the core concepts of OOP. An abstract superclass is created if it is desired to create a superclass that has the common features and functions of subclasses but does not yet have an object. The methods of the abstract class can be defined as templates or created as abstract methods to be overwritten by its subclasses. A class with an abstract method automatically becomes abstract itself and no objects are created from abstract classes.

# Interface vs Abstract class ?

1. A class can inherit more than one interface, but only one abstract class can inherit a class.
2. Empty methods can be defined in Interface, but both empty methods and filled methods can be defined in abstract classes.
3. Using abstract classes provides an advantage in terms of speed.
4. When we write a new method in the Interface, it is necessary to fill in this method one by one in all the classes we implement from this interface, but the situation is different in abstract classes, when we define a method and fill it in, all classes derived from our abstract class gain this feature.
5. Interfaces help to provide multiple inheritance, while abstract classes do not support multiple inheritance.
6. All the objects in the Interface must be "public", while in the Abstract classes not all the elements must be "public".
7. Interface does not contain constructor methods. An abstract class can contain constructor methods.
8. Interface methods cannot be static. Abstract class non-abstract methods can be defined as static.

# Why wee need equals and hashcode ? When to override ?

Equals is similar to the == operator to test for object identity rather than object equality. HashCode is a method in which a class implicitly or explicitly divides the data stored in a class instance into a single hash value, which is a 32-bit signed integer. The key difference between equals and hashCode in Java is that Equals is used to compare two objects whereas hashCode is used in hash generation to decide in which group an object should be categorized.

# Diamon problem in Java ? How to fix it?

For example, we have two classes, B and C. Let these two classes be derived from class A. Classes B and C use a method in class A by overriding it. Let's say the D class is derived from B and C together, that is, they made Multiple Inheritance.

If D class will use this method without overriding, will it use it as it comes from B or will it use it as it comes from C? This problem is called the Diamond Problem. It was named because the shape resembles a diamond.

It is not possible for us to encounter such a problem in languages ​​such as Java, because these languages ​​do not support Multiple Inheritance, rather they support Multiple Inheritance over Interfaces, as you know, interfaces are structures that do not contain any code behaviorally, only contain certain method signatures. Even if we derive that class from n interfaces with the same method signature, we do not encounter any problems because there is only one method that we need to implement.

# Why we need Garbagge Collector ? How does it run ?

Garbage Collector can be defined as a program used to automatically manage memory by removing object allocation. In the Java language, we know that new objects are created and memory allocated using the new operator. Memory allocated to an object using a new operator remains allocated until references use that object. As soon as the references cease to exist, the memory occupied by the object is reclaimed. Then Java automatically handles the de-allocation or destruction of the objects and we don't need to destroy the object explicitly. This technique is Garbage Collection in Java, where programmers do not need to explicitly handle the freeing of objects.

During garbage collection, the Garbage Collector searches the Heap memory and then 'marks' objects that cannot be reached. Then it destroys them. But the problem arises as the number of objects increases. As objects increase, so does the time spent on Garbage Collection as you search for unreachable objects. However, it doesn't affect much since most objects are short-lived. The above behavior is 'Generation Garbage Collection' and is expected to improve JVM performance. In this approach, the entire Stack space is split into Young Generation, Old or Tenured Generation, and Persistent Generation.

**#1) Young Generation Stack Area:** All new objects are created in this area. When the field is full, Minor GC takes place where all dead objects are destroyed. Small GC operation is fast and fast as most objects die. Objects that survive in the younger generation are carried over to the older generation.

**#2) Old Generation Stack Space**: This generation stores long surviving objects. When the threshold age set for the younger generation is met, the object is carried over to the older generation. When the old generation field is populated, a Major GC is performed.

Major GC is slow as the objects contained here are living objects. Sometimes the entire Stack area is cleared, containing both young and old generations. This is called 'Full GC'.

#3) **Persistent Generation L** There was a Persistent Generation (Perm Gen) until Java 7. Metadata held by Perm Gen was used by the JVM. The JVM used this metadata to define the classes and methods used in the application. Perm Gen was removed in Java 8.

# Java ‘static’ keyword usage ?

Variables created using the static keyword are called "class variables", not object variables. These variables do not belong to the object, but carry the information of the class. Class variables take up memory even if no objects have been created from the class in which they are defined. Object variables, on the other hand, only take up memory when an object is defined. Another point where these two types of variables differ is that there is only one instance of class variables. So no matter how many objects of that class are created, there is only one class variable in memory and no matter how we access it, we access the same class variable.

# Immutability means ? Where, How and Why to use it ?

There are things that are immutable in Java. All wrapper classes such as String, Boolean, Integer are immutable. Imable classes once mom and good morning. Compliance specifications to this standard can be specified as follows:

**Simple design;** Edits on the object after the object is a tool will make each object a different state. We may be sent to reach the object, or we may have to document that the object may encounter it. This high immutable class, being single and being, gives us great transition.

**Security;** They buy over the Immutable initial purchase configuration and stay fixed in value and don't have to run certain features. This ensures thread safety. In addition, immutable classes can also lead to performance and memory features. So the value of the string is not updated as in the code below. The target is 100000 new data entries in the stack. This can make the memory incredibly bloated. Being thread-safe can also be the cause of performance issues.

# Composition and Aggregation means and differences ?

**Aggregation :** The relationship between two objects is shown by drawing a line in Unified Modeling Language (UML). Connection is a relationship. UML helps to get a visual representation of the system. It is different from normal programming languages. Relationship also defines the multiplicity of objects. These are one-to-one, one-to-many, and many-to-many.

**Composition** is a more specific form of aggregation. Explains the property. E.g; a book object consists of a sheet object or pages. The multiplicity is also used to indicate the number of objects. Explains that a class has more than one student. The highlighted diamond symbol represents the composition in UML. This is a collection, but more specific, as the book has one page or many pages. If the book object is destroyed, the sheet objects are also destroyed. Page objects cannot exist without a book object. Therefore, composition is a more specific form of aggregation, which means possession.

|  |  |
| --- | --- |
| Aggregation | Composition |
| * An aggregation is a relationship between two objects that defines a "one" relationship. * UML Symbol * Gathering is indicated by a diamond. * Functionality * In aggregation, if the owning object is destroyed, it will not affect the containing object. | * Composition is the most specific type of aggregation that means possession. * The composition is indicated by a highlighted diamond. * In the composition, if the owning object is destroyed, it will affect the host object. |

# Cohesion and Coupling means and differences ?

**Coupling:** Represents more than one class/object. Two more sub-concepts emerge here; Loose technology In Object-oriented programming languages, a class is a class with a class/containing in its class, or a class with a class (link) to another class. In the concept of strict; User classes are also available, which help on a related class. It is safe in modularity and easy to maintain. It is explained in detail in a simple way in this software. That is, converting what is built on a class into service classes.

**Cohesion:** Procedures within classes, where the relationship between classes depends, and whether they are true/connected. A class must perform a single task with the Single responsibility principle. If those who do many different jobs in a class, this job is good in different classes, it is people who are in different classes in this job. They are not going to use data collection fields in a class. Stickiness example classes are a bane that threatens modularity and software maintenance.

# Heap and Stack means and differences ?

Stack and Heap are logical structures in memory (ram). Types such as int, short, byte, long, decimal, double, float, which we call primitive type, are called value type and kept in the stack. These values must be known before the runtime, and the operating system allocates a certain place in the stack before the program runs. If this section is exceeded by the person who wrote the code, a stack overflow error may be encountered. The data on the stack is in LIFO logic. It is sorted and no operation can be performed with a value in between. Class type variables are reference types, the model (reference) they refer to is stored in the stack and their values are stored in the heap.

|  |  |
| --- | --- |
| Stack | Heap |
| * Stack is cool. It is easy to use. * It is stored in RAM in the computer. Just like Heap. * The created variables are automatically destroyed when they leave the stack scope. * Reaching it is quite fast compared to heap. * When the usage is high on the stack, the space may not be enough. For example, assigning 21 elements to a 20-dimensional array... * Created variables can be used without pointers. * It is created at compile time. * If you know the exact size of the place you will use, it will be convenient for you to use Stack. | * Heap is a bit coy. But if you take it easy, the room will take your load. * It is kept in RAM in the computer. Just like Stack. * Heap variables created inside a block cannot be destroyed automatically when they leave the block, this must be done manually. * It is quite slow compared to Stack. * It creates memory problems if not used properly. * Variables are used with pointers. * It is created at runtime. * If you don't know the exact size you need, using Heap is perfect for you. |

# Exception means ? Type of Exceptions ?

Exceptions are events that occur during the execution of programs that disrupt the normal code flow. For example, division by zero, non-index array access, etc.

## What are the Exception Types?

Exception classes derive from the Object class like other Java classes. A subclass of the Object class is the Throwable class. In this class, the Exception class derives. The Exception is divided into two parts:

## Check Exception

It is the exception type that the IDE warns the developer. Checked Exceptions are all handled at compile time. As an example, we are writing a code and the IDE will give a warning if an error is likely to occur here. This error is called Check Exception.

## Unchecked Exception

It is the exception type that the IDE does not warn the developer. It is an exception class derived from the RuntimeException class.

# How to summarize 'clean code' as short as possible ?

It is a reader-oriented development style where clean code is easy to understand and develop new code on.

# What is the method of hiding in Java ?

Method hiding can be defined as, "if a subclass defines a static method with the same signature as a static method in the super class, in such a case, the method in the subclass hides the one in the superclass." The mechanism is known as method hiding. It happens because static methods are resolved at compile time.

# What is the difference between abstraction and polymorphism in Java ?

|  |  |
| --- | --- |
| Abstraction | Polymorphism |
| Polymorphism is the ability of an object to behave in more than one way. | Inheritance is creating a new class using the properties and methods of an existing class. |
| Polymorphism is used for what form of method objects will call at compile time and at run time. | Inheritance is used for code reusability. |
| Polymorphism is applied in the methods. | Inheritance is implemented in classes. |
| Polymorphism can be divided into overloading and overriding. | Inheritance can be divided into single-level, multi-level, hierarchical, hybrid and multiple inheritance. |