Concepts Revisit

1. Classes/Objects

A class is a blueprint or template for creating objects. It defines the properties (attributes or fields) and behaviors (methods) that objects of that class will have. An object, on the other hand, is an instance of a class, representing a specific entity with its unique state and behavior.
The class does not occupy any memory space till the time an object is instantiated.

1.1 Constructor :

Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally.
Constructors have the same name as class or structure.
Constructors don't have a return type. (Not even void)
Constructors are only called once, at object creation.

Three types,

1. Parameterized
2. Non-parameterized
3. Copy constructor

1.1.1 Copy Constructor
A Copy constructor is an overloaded constructor used to declare and initialize an object from another object. There is only a user defined copy constructor in Java(C++ has a default one too).
initialize the data members of the class using another object of the same class. It copies the values of the data variables of one object of a class to the data members of another object of the same class.

```
class Student {
   String name;
   int age;
```

```
    Student(Student s2) {
        this.name = s2.name;
        this.age = s2.age;
    }
}
```

2. Inheritance

Inheritance is a feature of OOPs which allows classes to inherit common properties from other classes. For example, if there is a class such as 'vehicle', other classes like 'car', 'bike', etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

1. **Single Inheritance**
2. **Hierarchical Inheritance**: Deriving more than one class from a base class
3. **Multilevel Inheritance:** Deriving more classes from a derived class

3. Polymorphism

Polymorphism is the ability of objects to take on different forms or behave in different ways depending on the context in which they are used.
It enables flexibility and extensibility in the code by allowing methods to be overridden in subclasses and providing different implementations for the same method signature.

3.1 Compile Time Polymorphism :

The polymorphism which is implemented at the compile time is known as compile-time polymorphism. Example - Method Overloading

Method Overloading : Method overloading is a technique which allows you to have more than one function with the same function name but with different functionality. Method overloading can be possible on the following basis:

1. The type of the parameters passed to the function.
2. The number of parameters passed to the function.

3.2 Runtime Polymorphism :

Runtime polymorphism is also known as **dynamic polymorphism**. Function overriding is an example of RP. Function overriding means when the child class contains the method which is already present in the parent class. Hence, **the child class overrides the method of the parent class**. In case of function overriding, parent and child classes both contain the same function with a different definition. The call to the function is **determined** at runtime is runtime polymorphism.

4. Access Modifiers

Public	Anywhere, inside/outside package
Default	Anywhere but within same package
Protected	Within class and in inherited class
Private	Within class only

5. Encapsulation

Encapsulation is the process of combining data and functions into a single unit called class. In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class. In simpler words, attributes of the class are kept private and public getter and setter methods are provided to manipulate these attributes. Thus, encapsulation makes the concept of data hiding possible.

6. Abstraction

Abstraction is the process of hiding the implementation details of a class and showing only the essential features of the object. Abstract classes and methods define a blueprint for subclasses without providing implementations. This allows for the creation of generalized frameworks and promotes code reusability.

Data binding **:** Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.

Abstraction is achieved in 2 ways :

- Abstract class
- Interfaces (Pure Abstraction)

1. **Abstract Class**

- 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 and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

2. **Interfaces**

- All the fields in interfaces are public, static and final by default.
- All methods are public & abstract by default.
- A class that implements an interface must implement all the methods declared in the interface.
- Interfaces support the functionality of multiple inheritance.

Static Keyword

Static can be :

1. **Variable** (also known as a class variable)

   Only one copy of variable that's shared

2. **Method** (also known as a class method)
3. **Block**: static {} It is executed only once when the class is loaded into memory.
4. It is mainly used for static initializations.
5. **Static class:**

   Used for utility classes and helper classes
   It can only contain static members and functions.
   It cannot be instantiated or extend
   It cannot extend non-static classes

7. Association

Association is a relationship between two separate classes that establishes a connection between their objects. It represents a more general relationship than aggregation and composition. Association can be one-to-one, one-to-many, or many-to-many, and it can be bidirectional or unidirectional.

```
class Person {
 }
```

```
class Company {
    private Person owner;
}
```

The above example is **One-One relationship**
One-Many relationship will be:

```
class Company {
    private List<Person> employees = new ArrayList<>();
}
```

Many-Many will be:

```
class Course {
      private List<Student> students = new ArrayList<>();
}
class Student {
      private List<Course> courses= new ArrayList<>();
}
```

7.1 Composition

Composition in OOP is a mechanism to establish **has-a** relation within objects. Composition is achieved by defining classes that contain instances of other classes as members.
The objects cannot exist independently.

```
class CarEngine {}
class Car {
   private CarEngine;
}
```

7.2 Aggregation

Same just the instances as members can exist independently. It represents a **whole-part relationship**, where the part (component) can belong to multiple wholes (containers) and can be shared among them.

When should I select what?
Composition: Each Library (container) has a collection of Books (component). A Book is a fundamental part of a Library's inventory and cannot exist without the Library. Therefore, Library and Book exhibit a composition relationship.

Aggregation: Each Library (container) may have one or more Librarians (component), but a Librarian can work in multiple Libraries or exist independently. Therefore, Library and Librarian exhibit an aggregation relationship.

8. Singleton Pattern

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. It is useful when exactly one object is needed to coordinate actions across the system.

9. Factory Pattern

10. Builder Pattern

11. Decorator Pattern

Interview Questions

Why use OOPs?

- OOPs allows clarity in programming thereby allowing simplicity in solving complex problems
- Code can be reused through inheritance thereby reducing redundancy
- Data and code are bound together by encapsulation
- OOPs allows data hiding, therefore, private data is kept confidential
- Problems can be divided into different parts making it simple to solve
- The concept of polymorphism gives flexibility to the program by allowing the entities to have multiple forms

Can you call the base class method without creating an instance?

Yes, you can call the base class without instantiating it if:

- It is a static method
- The base class is inherited by some other subclass

An **object** is a physical entity	A **class** is a logical entity
Objects take memory space when they are created	A class does not take memory space when created
Objects can be declared as and when required	Classes are declared just once

Problems with inheritance

Increases the time and effort required to execute a program as it requires jumping back and forth between different classes

The parent class and the child class get tightly coupled

23. What is operator overloading?

Operator overloading refers to implementing operators using user-defined types based on the arguments passed along with it.

if you raise the accelerator, the speed will increase, but you don't know how it actually happens. This is data abstraction as the implementation details are hidden from the rider.

34. What are virtual functions?

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

35. What are pure virtual functions?

Pure virtual functions or abstract functions are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

Class	Method'
A class is basically a template that binds the code and data together into a single unit. Classes consist of methods, variables, etc	Callable set of instructions also called a procedure or function that are to be performed on the given data

45. What is an exception?

An exception is a kind of notification that interrupts the normal execution of a program. Exceptions provide a pattern to the error and transfer the error to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

46. What is exception handling?

Exception handling in Object-Oriented Programming is a very important concept that is used to manage errors. An exception handler allows errors to be thrown and caught and implements a centralized mechanism to resolve them.

47. What is the difference between an error and an exception?

Error	Exception
Errors are problems that should not be encountered by applications	Conditions that an application might try to catch

49. What is a finally block?

A finally block consists of code that is used to execute important code such as **closing a connection**, etc. This block executes when the try block exits. It also makes sure that finally block executes even in case some unexpected exception is encountered.

50. What are the limitations of OOPs?

- Usually not suitable for small problems
- Requires intensive testing
- Takes more time to solve the problem
- Requires proper planning
- The programmer should think of solving a problem in terms of objects

Difference between a Structure and a Class:

Structure: In languages like C, C++, and Swift, a structure is a user-defined data type that groups together variables of different data types under a single name. By default, members of a structure are public.
Class: In languages like C++, Java, and Python, a class is also a user-defined data type, but it not only holds data but also defines operations (functions) that can be performed on that data. Classes can have constructors, destructors, access specifiers, inheritance, etc. By default, members of a class are private.
Key Differences:
Inheritance: Classes support inheritance, allowing one class to inherit properties and behavior from another. Structures typically do not support inheritance.
Access Specifiers: Classes have access specifiers like public, private, and protected to control the access to their members, while structures generally don't have access control.
Default Visibility: Members of a class are private by default, whereas members of a structure are public by default.

Difference between Immutable and Mutable strings:

Immutable Strings: In languages like Python, immutable strings cannot be changed after they are created. Operations that appear to modify a string actually create a new string with the modified content. Once created, the content of an immutable string cannot be altered.
Mutable Strings: Mutable strings, on the other hand, can be modified after creation. Languages like C and C++ allow manipulation of string content directly, allowing changes to the characters within the string.

Up-casting and down-casting:

Up-casting: Up-casting refers to the casting of a derived class reference or pointer to a base class reference or pointer. It involves converting a subtype to a supertype. This is generally safe and implicit in languages like Java and C#. For example, if we have a hierarchy where Dog is a subclass of Animal, up-casting allows treating a Dog object as an Animal object.
Down-casting: Down-casting is the opposite operation, where a base class reference or pointer is cast to a derived class reference or pointer. It involves

converting a supertype to a subtype. Down-casting is not always safe and typically requires explicit casting. For example, in Java, you may need to explicitly cast an Animal object back to a Dog object if you originally up-casted it.

Differentiate between shallow and deep copy:

Shallow Copy: A shallow copy of an object creates a new object but does not create copies of the nested objects within it. Instead, it copies references to the nested objects. Therefore, changes made to the nested objects in the copied object will affect the original object and vice versa.
Deep Copy: A deep copy, on the other hand, creates a new object and recursively copies all objects referenced by the original object, creating new instances of the nested objects as well. This results in a completely independent copy where changes made to the copied object or its nested objects do not affect the original object. However, deep copying can be more resource-intensive and complex, especially for objects with complex structures or circular references.