**Quiz-1**

**Question 1)** Explain polymorphism and why it is important.

**Answer:** Polymorphism is a capability of object to act in many different forms.

It is simply gained by Inheritance. In object-oriented programming, with the help of polymorphism we can use the same method name with different signature in the class. If we are using the inheritance feature, in that case we can use the same methods of all the inherited classes. And it will help in re-usability of code. With the help of Polymorphism, we do not have to write the similar features for all the classes.

**Question 2)** Explain the open close principle and give an example.

**Answer:** The Open-Closed Principle (OCP) states that software entities (classes, modules, methods, etc.) should be open for extension, but closed for modification.

For example, let's take a class to represent a 3d object (cuboid). We have created a class that will calculate the total volume of given list of objects.

Diagram

Description automatically generated

Now we need to add one more 3d object. Now the application class needs to be modified to calculate the volume of a sphere.

Diagram

Description automatically generated

Now we can resolve this issue by using an open close principle. We can define a stable class which is well tested and if alter is required we can do it by extending the base class.

Diagram

Description automatically generated

**Question 3)** Explain early binding and when it is possible.

**Answer:** The binding which can be resolved at compile time by the compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time. The method definition and method call are linked during the compile time. Actual object is not used for binding. For example: Method overloading.

**Question 4)** Explain late binding and why it is needed.

**Answer:** Late binding or dynamic binding where the compiler doesn’t decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.

**Question 5)** Explain programming to an interface and what are the advantages of doing so.

**Answer:** Programming to an interface is a technique by which developers can expose certain methods of an object to other objects in the system. The developers who receive implementations of these interfaces can code to the interface in place of coding to the object itself. It also means focusing our design on what the code is doing, not how it does it. This is a vital distinction that pushes our design towards correctness and flexibility.

The advantages of using interfaces in Java are as follows:

1. Without bothering about the implementation part, we can achieve the security of the implementation.
2. In Java, multiple inheritance is not allowed, however, we can use an interface to make use of it as we can implement more than one interface.

**Question 6)** Explain Factory design pattern and why is it important

**Answer:** The factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes or implemented in a base class and optionally overridden by derived classes—rather than by calling a constructor.

Advantage of Factory Design Pattern:

1. Factory Method Pattern allows the sub-classes to choose the type of objects to create.
2. It promotes loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class so that it will work with any classes that implement that interface or that extend that abstract class.

**Question 7)** List at three advantages of using a Factory method over using the constructor

**Answer:** There are advantages of using a factory method over using the constructor. Some are listed below:

* We can create objects without specifying their concrete classes
* Factory Method Pattern allows the sub-classes to choose the type of objects to create.
* It promotes loose-coupling by eliminating the need to bind application-specific classes into the code.

**Question 8)** Explain Template Method design pattern and how it is useful

**Answer:** The template method design pattern is to define an algorithm as a skeleton of operations and leave the details to be implemented by the child classes. The overall structure and sequence of the algorithm are preserved by the parent class. It is useful for the following reasons:

1. Let subclasses implement varying behavior (through method overriding)
2. Avoid duplication in the code, the general workflow structure is implemented once in the abstract class’s algorithm, and necessary variations are implemented in the subclasses.
3. Control at what points subclassing is allowed. As opposed to a simple polymorphic override, where the base method would be entirely rewritten allowing radical change to the workflow, only the specific details of the workflow are allowed to change.

**Question 9)** Explain Listener design pattern and give an example of its application

**Answer:** The listener is a behavioral design pattern that lets us define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing/listening to.

Let's take an example of text editor object.

Diagram

Description automatically generated

Here the event manager contains the list of subscribers. Any event listener object can be added to the subscriber list. The editor notifies on events like edit, open, and save action. The event manager calls the update of all subscriber objects.

**Question 10)** Explain the Façade design pattern and give an example of how it is useful for information hiding in subsystem design

**Answer:** Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

In this example, the Facade pattern simplifies interaction with a complex video conversion framework. We done need to understand the complex component working together. We uses the abstract operations define on façade class.

Diagram

Description automatically generated

**Question 11)** Explain the Singleton design pattern and show how you implement it.

**Answer:** The singleton pattern is a design pattern that restricts the instantiation of a class to one object.

Example code:

*class Singleton*

*{*

*private static Singleton obj = new Singleton();*

*private Singleton() {}*

*public static Singleton getInstance()*

*{*

*return obj;*

*}*

*}*

The above class creates a single instance of class Singleton. The object cannot be instantiated from other classes. The static method is created to return the instance reference.