Question 1 (2 points) Explain polymorphism and why it is important.

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows to define one interface and have multiple implementations. The word "poly" means many and "morphs" means forms, So, it means many forms.

Importance:

1. It helps the programmer to reuse the codes, i.e., classes once written, tested and implemented can be reused as required. Saves a lot of time.
2. Single variable can be used to store multiple data types.
3. Easy to debug the codes.

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

In software development, object-oriented design helps to write flexible, scalable, and reusable code. It is recommended that the developers follow SOLID principles when writing a code. One of the five SOLID principles is the open/closed principle. The principle states that software entities like class, modules, functions, etc.; should be able to extend a class behavior without modifying it. This principle separates the existing code from modified mode to provide better stability, maintainability and minimizes the changes in the code.

Example:

```
package Quiz1;
    abstract class Calculator{
        public abstract double add();
        public abstract double sub();
        public abstract double mul();
        public abstract double div();
}

class Scientific extends Calculator{
    public double add() {
        return 0;
    }

    public double sub() {
        return 0;
    }

    public double mul() {
        return 0;
    }

    public double div() {
        return 0;
    }

    public double sin(){
        return 0;
    }
```

```
    public double cos(){
        return 0;
    }

    public static void main(String[] args) {
        Calculator calculator= new Scientific();
        calculator.add();
        calculator.sub();
    }
}
```

Question 3 (2 points) Explain early binding and when it is possible.

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.

Example:

```
public class Dog {
    private void eat(){
        System.out.println("Dog eat meats.");
    }

    public static void main(String[] args) {
        Dog dog= new Dog();
        dog.eat();
    }
}
```

Question 4 (2 points) Explain late binding and why it is needed.

Late binding: In the late binding or dynamic binding, 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.

```
class Animal{
    public void animalEat(){
        System.out.println("Animals must eat");
    }
}

public class Dog extends Animal{
    @Override
    public void animalEat(){
        System.out.println("Dog eats meat");
    }

    public static void main(String[] args) {
        Animal dog= new Dog();
        dog.animalEat();
    }
}
```

Question 5 (2 points) Explain programming to an interface and what are the advantages of doing so.

Interface is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement interface use implements keyword.

Advantages and uses:

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

Question 6 (2 points) Explain Factory design pattern and why is it important

A Factory Pattern or Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance of the class. The Factory Method Pattern is also known as Virtual Constructor.

Importance:

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

Question 7 (2 points) List at three advantages of using a Factory method over using the constructor

1. Readable Names: One of the serious limitations of the constructor is that cannot give it an explicit name, the name must be same as the name of the class. If class return two different types of objects for a different purpose, factory method with more readable names.

2. Polymorphism: Another serious limitation of a constructor is that it always returns the same type of object. You cannot vary the type of constructed object; it must be same as the name of the contractor. But the factory method can return an object of different subclasses

3. Coupling: Factory methods promote the idea of coding using Interface then implementation which results in more flexible code, but constructor ties your code to a particular implementation.

Question 8 (2 points) Explain Template Method design pattern and how it is useful

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.

The template method is used for the following reasons.

- It lets subclasses implement varying behavior (through overriding of the hook methods).
- It avoids duplication in the code: the general workflow of the algorithm is implemented once in the abstract class's template method, and necessary variations are implemented in the subclasses.
- It controls the point(s) at which specialization is permitted. If the subclasses were to simply override the template method, they could make radical and arbitrary changes to the workflow. In contrast, by overriding only the hook methods, only certain specific details of the workflow can be changed, and the overall workflow is left intact.

Question 9 (2 points) Explain Listener design pattern and give an example of its application

Not required

Question 10 (2 points) Explain the Façade design pattern and give an example of how it is useful for information hiding in subsystem design

Not required

Question 11 (2 points) Explain The Singelton design pattern and show how you implement it.

The singleton pattern is a design pattern that restricts the instantiation of a class to one object. The singleton pattern is one of the simplest design patterns. Sometimes we need to have only one instance of our class for example a single DB connection shared by multiple objects as creating a separate DB connection for every object may be costly. Similarly, there can be a single configuration manager or error manager in an application that handles all problems instead of creating multiple managers.