Quiz 1

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

Polymorphism is one of the most significant features of OOP which perform a single action in different ways. It provides reusability to the code. The classes that are written, tested and implemented can be reused multiple times.

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

The principle states that software entities like class, modules, function etc. should be able to extend a class behavior without modifying it. In short open for extension and closed for modification. This is achieved by using inheritance and polymorphism.

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 early binding. During this, the method definition and the method call are linked. Method overloading is example of early binding.

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

The binding which can be resolved at run time by the compiler is late binding. The method definition and the method call are linked during run time. Method overriding is example of late binding.

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

Interface contains different methods which are abstract in nature, and which can be implemented in other classes. It helps to achieve multiple inheritance, decrease coupling and improve testability.

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

Factory Method Pattern says that just define an interface or abstract class for creating an object but let the subclass decide which class to instantiate. It promotes the loose coupling by eliminating the need to bind class to the code.

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

Factory Method Pattern allows the sub-classes to choose the type of objects to create. It has a descriptive name, whereas a constructor has a fixed name. It can return a subtype of the type or an object implementing an interface.

It doesn't have to return an instance every time. It solves the problem of overloaded constructors with same number of parameters.

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

Template Method is a behavioral design pattern that allows us to defines a skeleton of an algorithm in a base class and let subclasses override the steps without changing the overall algorithm's structure. In this pattern parts of the task the template method wants to accomplish are declared as abstract and differed to concrete types to define them. The concrete types define the specifies of the subtasks (the abstract methods) which will then be invoked by the template method using polymorphism.

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

The listeners pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically. The object which is being watched is called the subject. The objects which are watching the state changes are called listeners.

For example, we can define a listener for a button in a user interface. If the button is selected, the listener is notified and performs a certain action.

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

A facade is an entry point to a sub system; it simplifies the task of using the subsystem and hides the details of how the classes of the subsystem interact with each other. When we use a subsystem through its facade our code is decoupled from the subsystem details. This allows the subsystem to change without affecting other parts of the system.

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

The Singleton restricts the number of instances that can be created from a given type to only one instance.

Make constructor private. Make a private constant static instance of this Singleton class. Write a static/factory method that returns the object of the singleton class that we have created as a class-member instance. We can also mark a static member as public to access constant static instance directly. But we would like to access class/instance members via methods only. So, the singleton class is different from a normal Java class in terms of instantiation. For a normal class, we use a constructor, whereas for singleton class we use the getInstance() method.