Quiz

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

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows us to define one interface and have multiple implementations. Polymorphism allows us to code to an interface that reduces coupling, increases reusability, and makes our code easier to read. Polymorphism is achieved using late binding that depends on the runtime type of the object. It allows to code using abstract classes and interfaces and not depend on not concrete types.

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

This principle states that software entities should be open for extension, but closed for modification. As a result, when the business requirements change then the entity can be extended, but not modified. Java Swing Framework for developing GUI applications in Java is an example of code that adheres to the open close principle. If needed to create a new JFrame, do not change the existing code for that class, extend it with a new class that implements requirements.

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. It is possible in the case where the compiler has enough information to decide which method body is going to be invoked as a result of a given method call. Binding of all the static, private and final methods is done at compile-time where inheritance is not polymorphic.

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

In the late binding or dynamic binding, the compiler doesn't decide the method to be called, means the compiler does not have enough information at compile time to know which method body should be called. Binding of the message to a method body at runtime depending on the concrete type of the object or the runtime type of the object. It is deferred to the runtime because it needs the concrete type of the object to decide which version of the method is going to be invoked. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method, thus it invokes the most overridden method in the inheritance hierarchy.

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

Coding to interfaces is a technique to write classes based on an interface; interface that defines what the behavior of the object should be. It involves creating an interface first, defining its methods and then creating the actual class with the implementation. Programing our code to depend on abstract classes and interfaces rather than on concrete types. When we program to an interface our code is extensible and adheres to the open close principle.

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

A Factory 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. Like a constructor of a class a factory encapsulates the logic of object creation. Unlike a constructor a factory

is not required to return a new object instance of the class every time it is invoked. It can be used to restrict access to a constructor and to enforce relationships to prevent arbitrary creation of objects. It can also be used to control how many instances of a given type are created

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.