<u>Quiz 1 (April 2023)</u>

1. (T) A bidirectional association between two domain objects A and B results in an instance variable of type A placed inside class B and vice versa.
2. (T) Unlike an association, a dependency between two domain objects A and B is a temporary relationship and does not result in a property in the resulting class.
3. (T) Per the Subtype Abstraction Principle, a variable whose static type is A can hold a runtime refence to an object of type A or subtype of A.
4. (T) Polymorphism is the ability of a variable to take on multiple forms and a method to exist with multiple implementations.
5. (T) According to the Open Close Principle, code should be closed for modifications and open for extension. New requirements result in new classes being added without modifying existing ones.
6. (T) Per the Single Responsibility Principle, a class should be designed to do one thing and one thing only. A request for a class to do something that is not amongst its responsibilities is delegated to the right entity that knows how to handle it.
7. (T) When you program to an interface you do not depend on the concrete classes and your code adheres to the Open Close Principle.
8. (T) In the Object Creation Factory, the Client does not depend directly on the Concrete Product. The Client is programmed to the interface Product and uses the Object Creation Factory to inject a dependency at runtime.
9. (T) Like instance methods, static methods defined on a type A are inherited by the subclasses of A. Unlike instance methods static method calls do not depend on the runtime type of the object. Static inheritance is not polymorphic.
10. (F) In the case of static method calls, final method calls and private method calls, the compiler has enough information to bind the message to the method at compile time and does not need to wait till the runtime to resolve the binding. The compiler must wait for the runtime to resolve the binding in all the other cases.
11. (T) Unlike a constructor, a factory method is not required to return a new object instance of the class every time it is invoked. It can be used to enforce a relationship and to prevent arbitrary creation of objects.
12. (T) A factory method can be used to control how many instances of a given type are created.

13. List three advantages of using a Factory method over using the constructor.

1) Encapsulation: The client code does not have to be aware of the concrete implementation of the objects being created, only the interface that they implement.
2) Flexibility: It allows you to easily switch between different implementations of objects without changing the client code.
3) Extensibility: It makes it easy to add new types of objects to the system, as you only need to create a new subclass and add it to the factory.
4) Testability: By separating the creation of objects from their use, it makes it easier to test the client code in isolation.
5) Consistency: It ensures that objects are created in a consistent way, using a centralized factory class.

14. Explain the Template Method design pattern and show the role of polymorphism in implementing it.
    ***Ans**: The Template Method pattern promotes code reuse by encapsulating the common parts of an algorithm in a base class, while allowing for flexibility and extensibility through polymorphism.*

15. Explain the Listener design pattern and give an example of its application.
    ***Ans:** The Listener design pattern is a behavioral pattern that involves creating an object (the listener) that is notified when an event occurs in another object (the source).*

```
//Code
JButton button = new JButton("Click me!");
button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
      System.out.println("Button clicked!");
      }
});
```

16. Explain the Facade design pattern and give an example of how it is useful for information hiding in subsystem design.

Ans: *The Facade design pattern is a structural pattern that provides a unified interface to a set of interfaces in a subsystem. It is used to simplify and unify a complex system by providing a single interface to a higher-level component, rather than exposing the underlying complexity of the subsystem. The Facade acts as a gateway to the subsystem, allowing clients to interact with it without needing to know the details of its implementation.*

A common example of the Facade pattern is the use of an operating system's graphical user interface (GUI). The GUI acts as a Facade for the underlying system, providing an easy-to-use interface that hides the complexity of the system's inner workings.

17. Explain The Singleton design pattern and show how you implement it.

Ans: *The Singleton design pattern is a creational design pattern that ensures that only one instance of a class can be created and that it provides a global point of access to that instance.*

```
public class Singleton {
    private static Singleton instance = null;
    private Singleton() {
        // Private constructor to prevent instantiation from outside the class
    }
    public static Singleton getInstance() {
        if (instance == null) { instance = new Singleton(); }
        return instance;
    }
}
```