Home Quiz - Answers

1. Explain polymorphism and why it is important.
   1. Polymorphism is one of the principles of OOP which promotes having multiple forms for a single contract. Polymorphism allows us to have multiple implementations decided on runtime for a certain operation.
   2. This is important because we will be able to scale our application by adding new implementations everytime we have a new requirement
2. Explain the open close principle and give an example.
   1. The Open Closed principle is one of the 5 SOLID design principles which guides to have an application to be closed for modification but open for extension
   2. For instance, if we have a Shape class that computes areas of a rectangle and a circle, if we have to change the code when we need to add one additional shape, like a triangle, that is not going to be a good design. Instead, we should have an interface and polymorphically be determined later which concrete classes need to be executed -
   3. Which means we only need to add a new class with an implementation, everytime we need to add new feature - not alter old code
3. Explain early binding and when it is possible.
   1. Early binding refers to events that occur at compile time. In essence, early binding occurs when all information needed to call a function is known at compile time.
   2. Examples of early binding include static fields, local and instance variables
   3. This are class elements that do not support polymorphism
4. Explain late binding and why it is needed.
   1. Late binding refers to function calls that are not resolved until run time.
   2. The main advantage to late binding is flexibility. Unlike early binding, late binding allows you to create programs that can respond to events occurring while the program executes polymorphically
   3. Late binding is needed when we want to inject new concrete implementation later at run time
5. Explain programming to an interface and what are the advantages of doing so.
   1. Programming to an interface is a technique to write classes based on an interface;
   2. 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
   3. The advantage includes
      1. Clients remained decoupled and unaware of specific class they are using
      2. Depending on the context, different implementation classes can be polymorphically provided without having to change client code - makes the app evolve
      3. Helps parallelize work.
6. Explain Factory design pattern and why is it important
   1. A factory design pattern is one of the creational design patterns that define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate
   2. Factory Method Pattern allows the sub-classes to choose the type of objects to create
   3. It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code
7. List at three advantages of using a Factory method over using the constructor
   1. Constructors don't have meaningful names
   2. Static factory methods can return the same type that implements the method(s), a subtype, and also primitives, so they offer a more flexible range of returning types
   3. Static factory methods can be controlled-instanced methods, with theSingleton pattern being the most glaring example of this feature
8. Explain Template Method design pattern and how it is useful
   1. The template method is a method in a superclass, usually an abstract superclass, and defines the skeleton of an operation in terms of a number of high-level steps. These steps are themselves implemented by additional helper methods in the same class as the template method
   2. It lets subclasses implement varying behavior
   3. 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
9. Explain Listener design pattern and give an example of its application
   1. A Listener design pattern is a design pattern commonly named as an Observer pattern, which is one of the behavioral design pattern, in which an object, named the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any state changes, usually by calling one of their methods.
   2. Usually used in EventListeners to wait for an event and registered an action to be performed when an action is triggered
10. Explain the Façade design pattern and give an example of how it is useful for information hiding in subsystem design
    1. Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system
    2. This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes
    3. It helps in information hiding as it creates an external class to only publish public methods and hides internal details
11. Explain the Singleton design pattern and show how you implement it.
    1. Singleton Design Pattern involves a single class which is responsible to create an object while making sure that only single object gets created.
    2. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.
    3. Implementations of the Singleton have these two steps in common:
       1. Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
       2. Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.