OOPS ASSIGNMENT-3

Name - Avdhesh Kumar Sharma
Class - BTech CSE AIML - 4A
Rollno - 2K21CSUN04008

Sol 1 → Design patterns are a set of best practices or reusable solutions to common software design problems that been tested and refined overtime. These patterns are intended to provide a standard approch to solving a particular type of problem in software development, making it easier to maintain, extend and reuse the code.

(i) **Singleton pattern** :- It is a creational pattern that ensures that only one instance of a class is created and the instance is globally accessible throughout the application. This pattern is useful in situations where there is a need to restrict the instantiation of a class to a single object, such as database connections, logging and caching.

(ii) **Factory Method pattern** :- Provides an interface for creating objects in a superclass, but allows subclasses to after the type of objects that will be created. This pattern is useful in situations where there are multiple classes that can be instantiated, and the decision to choose which class is instantiable based on some conditions or parameters.

(3) Adapter pattern :- It is a sturctual pattern that allows 2 incompatible interfaces to work together by creating a bridge b/w them. This pattern is useful in situations where are multiple objects that need to be notified when some event occurs.

(4) Observer pattern :- It is a behavioural pattern that defines a one-to-many dependancy between objects, so that when one object changes state, all its depend -ants are notified and updated automatically.

(5) Decoder pattern :- It is a sturctural pattern that allows new functionality to be added to an existing object dynamically, without changing its structure.

(6) Stratergy pattern :- It is a behavioral pattern that allows different algorithms or stratergies to be selected at runtime based on some conditions or parameters.

Sol2 - (1) Single Responsibility Principle (SRP)- A class should have only one reason to change. This principle states that each class should have only one responsibily and should focus on doing one thing well.

(2) Open-Closed Principle (OCP) - It states that software entities should be open for extension but closed for.

modification, meaning that new functionality should be added by writing new code rather than modifying existing code.

(3) Liskov Subsitution Principle (LSP) - It states that a subclass should be able to be used in place of its superclass without causing any unexpected behaviour or breating the program.

(4) Interface Segregation principle (ISP) - It states that classes should not be forced to depend on methods they do not use, and that interfaces should be designed to be cohesive and focused on a single responsibility.

(5) Dependancy Inversion Principle (DIP) - It states that high-level modules should not depend on low level modules, and that both should depend on abstractions.

Sol3 - Let us consider a case study of a simple online shopping system that allows customers to browse products, add them to their cart, and checkout.

(1) SRP

(a) 'Product' class : Responsible for defining the properties of a product, such as its name, price,

and description.

(b) 'Cart' class :- Responsible for managing the items a customer has added to their cart, such as adding & removing items, and calculating the total price of the cart.

(c) 'Checkout' class :- Responsible for preprocessing the customer's payment information and competing the ~~others~~ orders.

(2) OCP -

(a) 'Product' class - Implement an interface 'Product Info' which contains methods such as 'getName()', 'getPrice()', and 'getDescription()'. This interface can be extended in the future if we need to add more methods for products.

(b) 'Cart' class - Implement an abstract class 'Shopping Cart' which defines the basic functionality of a shopping cart, such as adding and removing items.

(c) 'checkout' class :- Implements an interface 'Payment gateway' which defines the methods for processing payment information.

(3) LSP -

(a) 'Product' class :- ensures that all subclasses of 'Product' have the same behaviour, such as

returning the name, price, and description of the product.

(b) 'Cart' class :- ensure that all subclasses of 'cart' have the same behaviour, such as adding and removing items from the cart, and calculating the total price of the cart.

(4) ISP

(a) 'Product Info' interface :- Contains methods for retrieving the name, price & description of a product. This interface can be implemented by the 'product' class.

(b) 'Cart Item' interface :- Contains methods for retrieving the product, quantity, and subtotal of a cart item.

(c) 'Order' Interface :- contains methods for retrieving the customer information, cart items and total price of an order.

(5) DIP

(a) 'Cart' class : Instead of creating a concrete implementation of a payment gateway within the 'cart' class, we can inject a 'payment Gateway' dependancy into the custructor or method of the 'cart' class. This allows us to easily swap out different payment gateway implementations without modifying the

'Cont' class.