ADS

Assignment: Lecture 2
By: Cesar Mamani

Point 1

Designing object-oriented (OO) software can be tricky, especially when looking to make it reusable. Finding the right level of detail in the objects and structuring the interface and relationships without losing generalizability for future issues or extensions is a challenge. Experienced OO designers are known for their ability to find recurring patterns in classes, making it easy to add and remove features easily.

In this sense, design patterns are useful tools for OO designers. A design pattern consists of four essential elements: the name, which is an identifier for the pattern; the problem, which describes the context in which the pattern is applied; the solution, which provides a template with a short description, including relationships, collaborations, and responsibilities; and the consequences, which include advantages, disadvantages, impact, portability, and extensibility of the pattern.

Point 2

The book also mentions the MVC (Model-View-Controller) design pattern, which consists of three components: the Model, which represents objects; the View, which is the graphical interface; and the Controller, which defines how the graphical interface reacts to user events. The MVC pattern makes use of other design patterns, such as Observable, Composite, and Strategy.

The chapter summarizes the 23 most common design patterns, each with different levels of detail and abstraction. These patterns are identified by their purpose and scope. Regarding the goal, there are structural patterns, which focus on assembling class objects into larger structures while maintaining flexibility and efficiency; creational patterns, which provide mechanisms for creating objects and increase flexibility and code reuse; and behavioral patterns, which are responsible for the communication of states between objects.

In terms of scope, there are class patterns, which deal with the relationships between parent and child classes; and object patterns, which focus on the relationship between individual objects.

The granularity of an object can be determined by considering its size. Patterns such as Facade, Flyweight, Abstract Factory, Builder, Visitor, and Command can be used to break objects down into smaller units.

Point 3

Another fundamental distinction is the difference between class inheritance and interface inheritance. When you inherit from a class, you extend the parent's behaviors, and your modifications are based on the parent's implementation. On the other hand, interface inheritance describes the ability of an object to be used instead of another object, as long as

it satisfies the same interface. The use of interfaces provides a clear visualization of the methods of the code and hides the implementation and the types of objects that use it.

In many cases, it is preferable to use composition instead of inheritance. Composition involves building complex objects by combining simpler objects. This weaker and more flexible relationship between objects allows for greater modularity, component reuse, and easier switching of instances at runtime. In contrast, inheritance creates a stronger, more static relationship that can limit flexibility and reuse.