UGANDA CHRISTIAN UNIVERSITY

A Centre of Excellence in the Heart of Africa

NAME: **NKURAIJA GARY MATTHEW**

REGISTRATION NO.: **S19B23/537**

ACCESS NO.: **A87132**

COURSE: **Bachelor of Science in Computer Science**

COURSE UNIT: **SOFTWARE CONSTRUCTION**

LECTURE: **Mr. Simon Lubambo**

QUESTION: **Write short notes about design patterns**

Many software design problems have established solutions known as design patterns that have worked well for developers. Design patterns first appeared in architecture, where they were used to build effective, aesthetically pleasing, and practical structures. They are used in software development to address typical issues with software architecture and design, like controlling complexity, improving maintainability, and encouraging code reuse.

The three categories of design patterns are creational, structural, and behavioral. Creational patterns concentrate on the production of objects and the mechanisms that produce them, trying to produce products that are suited for the circumstances. Object relationships are controlled by structural patterns, which also specify how they relate to one another. Behavioral patterns handle object communication and control their interaction.

Singleton, Factory Method, Abstract Factory, Builder, and Prototype are examples of creational patterns. A class can only have one instance thanks to singleton, making it the best choice for coordinating system-wide operations. Factory Method builds objects without giving their specific class, allowing runtime class determination. Without mentioning specific classes, an Abstract Factory generates related object families. Builder separates the construction of complicated objects from their representation, allowing for various depictions of the same construction process. By replicating existing objects, prototype generates new ones.

Adapter, Bridge, Composite, Decorator, Facade, Flyweight, and Proxy are examples of structural patterns. An adapter transforms a class's interface into a different interface that customers anticipate, allowing disparate classes to work together. Bridge separates an abstraction from its implementation, enabling independent development of both. For hierarchical structures, Composite interprets object groups as individual items. Each objects receive behavior additions through decorators, allowing for runtime behavior addition and removal. Facade provides a more user-friendly interface for a complicated system. Flyweight reduces memory use by using several instances of the same item. Proxy provides a substitute or stand-in object that stands in for the real item.

Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor are examples of behavioral patterns. Requests are passed up an object chain using Chain of Responsibility until Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor are examples of behavioral patterns. Behavioral patterns include Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, and Visitor. Chain of Responsibility distributes requests along an object chain till. Requests are passed up an object chain in a chain of responsibility until one handles the request. A request is contained as an object by a command, enabling request queuing or reporting. The phrase "interpreter" describes a language's grammar and offers an appropriate interpreter. Sequential access to an aggregate object's elements is made possible by iterators without exposing the representation's underlying data.

Mediator facilitates interaction modification by producing an object that has a collection of objects' interactions. The internal state of an object is captured and restored via Memento. When one object's state changes, the observer creates a one-to-many dependency between them, automatically updating and notifying dependents. Based on its internal state, an object can change its behavior thanks to state.

A strategy defines a collection of algorithms, encapsulating each one and enabling runtime selection and replacement.

Template Method delegated various steps to subclasses while defining the basic structure of a method in an operation. Visitor designates a fresh action to be applied to an object structure's constituent parts without altering the structure's classes.