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**

Design patterns are reusable solutions to common software design problems that have been found to be effective by software developers. The concept of design patterns originated in the field of architecture, where architects developed patterns for building structures that were efficient, aesthetically pleasing, and functional. In software development, design patterns are used to address common problems in software architecture and design, such as managing complexity, improving maintainability, and increasing code reuse.

Design patterns are divided into three categories: creational, structural, and behavioral. Creational patterns are used to create objects and deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. Structural patterns are used to manage relationships between objects, defining how they relate to one another. Behavioral patterns are used to manage communication between objects, defining how they interact with one another.

Creational patterns include the Singleton pattern, the Factory Method pattern, the Abstract Factory pattern, the Builder pattern, and the Prototype pattern. The Singleton pattern is used to ensure that there is only one instance of a class, making it useful for situations where only one object is needed to coordinate actions across the system. The Factory Method pattern is used to create objects without specifying the exact class of object that will be created, allowing the class to be determined at runtime. The Abstract Factory pattern is used to create families of related objects without specifying their concrete classes. The Builder pattern is used to separate the construction of a complex object from its representation, allowing different representations of the same construction process. The Prototype pattern is used to create new objects by cloning an existing object.

Structural patterns include the Adapter pattern, the Bridge pattern, the Composite pattern, the Decorator pattern, the Facade pattern, the Flyweight pattern, and the Proxy pattern. The Adapter pattern is used to convert the interface of a class into another interface that clients expect, allowing incompatible classes to work together. The Bridge pattern is used to decouple an abstraction from its implementation, allowing the two to vary independently. The Composite pattern is used to treat a group of objects as if they were a single object, making it useful for hierarchical structures. The Decorator pattern is used to add behavior to an individual object, allowing behavior to be added and removed at runtime. The Facade pattern is used to provide a simplified interface to a complex system, making it easier to use. The Flyweight pattern is used to minimize memory usage by sharing object instances that are identical. The Proxy pattern is used to provide a surrogate or placeholder object that acts as a stand-in for the real object.

Behavioral patterns include the Chain of Responsibility pattern, the Command pattern, the Interpreter pattern, the Iterator pattern, the Mediator pattern, the Memento pattern, the Observer pattern, the State pattern, the Strategy pattern, the Template Method pattern, and the Visitor pattern. The Chain of Responsibility pattern is used to pass requests along a chain of objects until one object handles the request. The Command pattern is used to encapsulate a request as an object, allowing the request to be queued or logged. The Interpreter pattern is used to define a grammar for a language and provide an interpreter for that language. The Iterator pattern is used to provide a way to access the elements of an aggregate object sequentially, without exposing its underlying representation. The Mediator pattern is used to define an object that encapsulates how a set of objects interact, making it easier to modify the interactions. The Memento pattern is used to capture and restore an object's internal state. The

Observer pattern is used to define a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically. The State pattern is used to allow an object to change its behavior when its internal state changes. The Strategy pattern is used to define a family of algorithms, encapsulating each one,