# Task #5

1.clean code principles :

There are 13 principles to achieve the clean code :

1.Single Responsibility Principle (SRP)

It means that each class should have one responsibility or goal to achieve

2.Open/Closed Principle (OCP)

Software entities should be open for extension but closed for modification

3.Liskov Substitution Principle (LSP)

4.Interface Segregation Principle

5.Dependency Inversion Principle

6.Composition Over Inheritance

7.Law of Demeter

8.Keep It Simple , Stupid

9.Don’t Repeat yourself

10.Single Level of Abstraction

11.Law of Least Astonishment

12.Separation of Concerns

13.YAGNI (You Ain’t Gonna Need It)

These principles when followed can lead to readable and maintainable code

2.types of problems that design patterns can solve

Design patterns are reusable solutions for common problems that developers face during software development. Here are some examples for problems types that design patterns can solve :

1.creational problems

2.structural problems

3.behavioral problems

4.concurrency problems

5.architectural problems

3.when to use SOLID principles

SOLID principles are 5 principles from clean code’s 13 principles :

S : represents Single Responsibility Principle (SRP)

We use it when we want to ensure that each class or module has a single responsibility

O : represents Open/Closed Principle (OCP)

We use it when we want to design modules or classes so they are open for extension but closed for modification

L : represents Liskov Substitution Principle (LSP)

We use it when we want to ensure that subtypes can be substituted for their base types with out affecting the correctness of the program

I : represents Interface segregation principle (ISP)

We use it when we want to design interfaces that are cohesive and tailored to specific needs of clients

D : represents Dependency Inversion Principle (DIP)

We use it when we want to decouple high level modules from low level modules by depending on abstraction rather than concrete implementation

4.design pattern vs architecture pattern

Design patterns and architecture patterns are important concepts in software development but they serve different purposes and operate at different levels of abstraction

Design patterns:

design patterns are reusable solutions to common problems that occur within the context of software design. They provide guidelines for designing classes , objects and relationships between them. Design patterns focus on the structure and behavior of individual components within a system. Examples of design patterns include the singleton pattern, factory pattern, observer pattern, and many others. Design patterns help improve code organization , maintainability, and extensibility. By providing proven solutions to design problems.

On the other hand, architecture patterns concern high level concerns related to the overall organization and structure of the software system. They define the framework of the system and establish the relationships between its major components. Architecture patterns provide a blueprint for the system organization, communication, and coordination among different modules and subsystems. Examples of architecture patterns include The Model View Controller (MVC) pattern, layered architecture, microservice architecture, and client service architecture. Architecture patterns help tackle issues such as scalability, performance, security, and maintainability at a system wide level.

In summary, design pattern focus on the design of individual components of a system, while Architecture patterns deal with overall structure or organization.

5.Two basics computer architecture and which is better

1.Von Neumann architecture

2.Harvared architecture

Determining which is better depends on specific requirements. Each architecture has its own weaknesses and strengths. Let’s see which is better in several conditions.

Performance: the Harvard architecture can potentially provide petter performance in certain scenarios.

Flexibility: Von Neumann architecture is more flexible than the Harvard architecture.

Cost and complexity: the Harvard architecture requires additional hardware components.

6.RISC vs CISC

RISC: Reduce Instruction Set Computer

CISC: Complex Instruction Set Computer

RISC:

RISC processors have simplified instruction set, with a small number of simple and atomic instructions.

Instructions in RISC architecture typically execute at one clock cycle, leading to simpler and faster execution pipeline.

RISC processors rely on a load/store architecture, where data must be loaded from memory into registers before it can be operated on.

CISC:

CISC processors have more extensive and complex instructions set, with instructions can perform multiple operations in a single instruction.

Instructions in CISC architecture can have variable lengths and execute in multiple clock cycles, leading to more complex and slower execution pipeline.

CISC architecture often support memory to memory operations, where instructions can directly operate on data stored in memory, reducing the need for explicit data movement.

7.Difference between scheduling algorithms and when to use them

Scheduling algorithms are used in operating systems to determine the order and priority of executing tasks or processes. The choice of algorithm depends on the specific requirements of the system.

1.First Come First Served (FCFS)

Non preemptive algorithm that schedules processes in the order they arrive.

We use FCFS when fairness and simplicity are more important than efficiency.

2.Shortest Job First (SJF)

SJF is a non preemptive algorithm that schedules the process with the smallest total execution time.

We use it when we want to minimize waiting time.

3.Round Robbin (RR)

RR is a preemptive algorithm where each process is assigned a fixed time slice or quantum for execution, and then it is moved to the end of the queue.

We us RR when fairness and responsiveness are important.

4.Priorety Scheduling

Priority scheduling assigns a priority value to each process and schedules the highest priority process first.

We use priority scheduling when different processes have different levels of importance.

5.Multilevel Queue Scheduling

Multilevel Queue Scheduling assigns processes to different queues based on priority or other criteria.

We use it when processes have different characteristics or service requirements.

6.Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling allows processes to move between different queues based on their behavior and history.

We use it when requirements of processes change with time.

8.what is fragmentation and when it occurs

There are two types of fragmentation

External fragmentation: it occurs in memory or storage systems when free space becomes scattered through out the available memory or disk, making it difficult to allocate larger continues blocks or memory or storage.

Internal fragmentation: it occurs when allocated memory or storage is not fully utilized, resulting in wastage of memory or storage within allocated blocks. It happens when the allocated block is larger than the required size, leading to unused or wasted space within the block.

9.Semi Structured Database

Semi structured database is a type of database that allows flexible and varying data structures within the database.

Unlike traditional relational databases that enforce a fixed schema with predefined tables and columns, semi structured databases accommodate data with irregular or dynamic structures.

Semi structured databases are commonly used in scenarios when data has varying or unknown structures, such as web data, social media data, sensor data, or log files. They provide a more agile and adaptable approach to data storage, making it easier to handle diverse and changing data format. Examples of semi structured databases include MongoDB (document oriented),

Appache Cassandra (columnar), amd apache HBase (columnar).