ENGLISH INTERVIEW

OOP

Encapsulation:

Description: Encapsulation involves packaging data and the methods that operate on that data into a single object. Other parts of the program should not need to know how the internal object performs its functions.

Advantages: Provides data protection and allows control over access to data. Promotes modularity and facilitates changes in the internal implementation of an object without affecting external parts of the program.

Inheritance:

Description: Inheritance allows creating a new class using the properties and methods of another class. The new class is called a subclass, and the class from which it inherits properties is the parent class or superclass.

Advantages: Enables code reuse, simplifies the extension of functionality, and eases code maintenance.

Polymorphism:

Description: Polymorphism allows objects of one class to use methods or properties of other classes. This can manifest in the use of methods with the same name but different implementations or in using methods or properties through a common interface.

Advantages: Provides flexibility and universality in code. Allows working with objects without needing to know their specific implementation.

Abstraction:

Description: Abstraction involves hiding unnecessary details and using only those characteristics of an object that are essential for achieving a specific goal.

It allows defining and using only the features of an object that are relevant in a given context.

Advantages: Simplifies and generalizes the use of objects, facilitates understanding and interaction between components of the system.

SOLID

SOLID is a set of principles in object-oriented programming introduced by Robert Martin (Uncle Bob) in 1995. The idea behind these principles is to avoid dependencies between code components. If there are too many dependencies, the code becomes hard to maintain (spaghetti code). Its main problems include:

- **Rigidity**: Every change causes a cascade of changes.
- **Fragility**: Changes in one part break other parts.
- **Immobility**: Code cannot be easily reused outside its context.

1) Single Responsibility Principle

Every class should have only one responsibility. This doesn't mean a class should have only one method; rather, all methods of a class should be focused on achieving a common goal. If there are methods that do not align with the class's purpose, they should be moved out.

2) Open/Closed Principle

Classes should be open for extension but closed for modification. If a class's functionality involves many branches or sequential steps, and there's a potential for these to increase, design the class so that new branches or steps don't require modification.

3) Liskov Substitution Principle

If an object of a base class can be replaced with an object of its derived class without affecting the program's correctness, then it adheres to the Liskov Substitution Principle.

4) Interface Segregation Principle

It's better to have many specialized interfaces than one general interface. A single general interface may force derived classes to inherit methods they don't logically need.

5) Dependency Inversion Principle

Consists of two statements:

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on implementation details. Implementations should depend on abstractions.

Conclusion

KISS

Description: The KISS principle suggests that solutions to problems should be as simple as possible. Programmers should avoid unnecessary complexity and opt for straightforward and understandable solutions.

Application:

- Choose simple and understandable constructs when designing and writing code.
- Avoid excessive use of complex or convoluted patterns.