**SOFTWARE PRINCIPLE OF GRASP:**

**INTRODUCTION:**

The Information Expert Principle is one of the key principles of GRASP, and it helps developers assign responsibilities to objects in a way that is consistent with the principles of object-oriented design.

The Information Expert Principle is a fundamental principle in object-oriented programming (OOP) that states that an object should be responsible for its own behavior and data. In other words, an object should be the expert on its own internal state and behavior.

**IMPORTANCE:**

This principle is essential in designing robust, maintainable, and scalable software systems. By following the Information Expert Principle, you can:

* **Encapsulate data and behavior**: Each object should hide its internal state and behavior from the outside world, and only expose a public interface through which other objects can interact with it.
* **Reduce coupling**: By minimizing the dependencies between objects, you can reduce the complexity of your system and make it easier to modify or extend individual components without affecting the entire system.
* **Increase cohesion**: By grouping related data and behavior within a single object, you can increase the cohesion of your system, making it more intuitive and easier to understand.

In object-oriented programming, an object represents a real-world entity or concept, such as a bank account, a customer, or a product. The Information Expert Principle states that each object should be responsible for managing its own internal state and behavior. This means that the object should:

* **Know its own internal state**: The object should have access to its own data and be able to modify it as needed.
* **Control its own behavior**: The object should be able to perform actions or operations on its own data without relying on external entities.

**EXAMPLE**;

CODING IN PYTHON:

**Let's consider an example of a BankAccount class to illustrate the Information Expert Principle.**

**Incorrect Example (Violating the Information Expert Principle)**

**class BankAccount:**

**def \_\_init\_\_(self, account\_number, balance):**

**self.account\_number = account\_number**

**self.balance = balance**

**class Bank:**

**def deposit(account, amount):**

**account.balance += amount**

**return account.balance**

**def withdraw(account, amount):**

**if amount > account.balance:**

**raise ValueError("Insufficient funds")**

**account.balance -= amount**

**return account.balance**

**account = BankAccount("123456", 1000)**

**print(Bank.deposit(account, 500)) # Output: 1500**

**print(Bank.withdraw(account, 200)) # Output: 1300**

In this example, the Bank class is responsible for modifying the internal state of the BankAccount object. This violates the Information Expert Principle because the BankAccount object is not responsible for its own behavior and data.

CODING IN PYTHON:

**Corrected Example (Following the Information Expert Principle)**

**class BankAccount:**

**def \_\_init\_\_(self, account\_number, balance):**

**self.account\_number = account\_number**

**self.balance = balance**

**def deposit(self, amount):**

**self.balance += amount**

**return self.balance**

**def withdraw(self, amount):**

**if amount > self.balance:**

**raise ValueError("Insufficient funds")**

**self.balance -= amount**

**return self.balance**

**account = BankAccount("123456", 1000)**

**print(account.deposit(500)) # Output: 1500**

**print(account.withdraw(200)) # Output: 1300**

**EXPLANATION:**

In this corrected example, the BankAccount class is responsible for its own internal state and behavior. The deposit and withdraw methods modify the internal state of the object, and the object is the expert on its own internal state and behavior.

By following the Information Expert Principle, we've encapsulated the data and behavior of the BankAccount class, reducing coupling and increasing cohesion. This makes the system more modular, flexible, and easier to maintain.

**THE END**