SCS1310: Object-Oriented Modelling and Programming

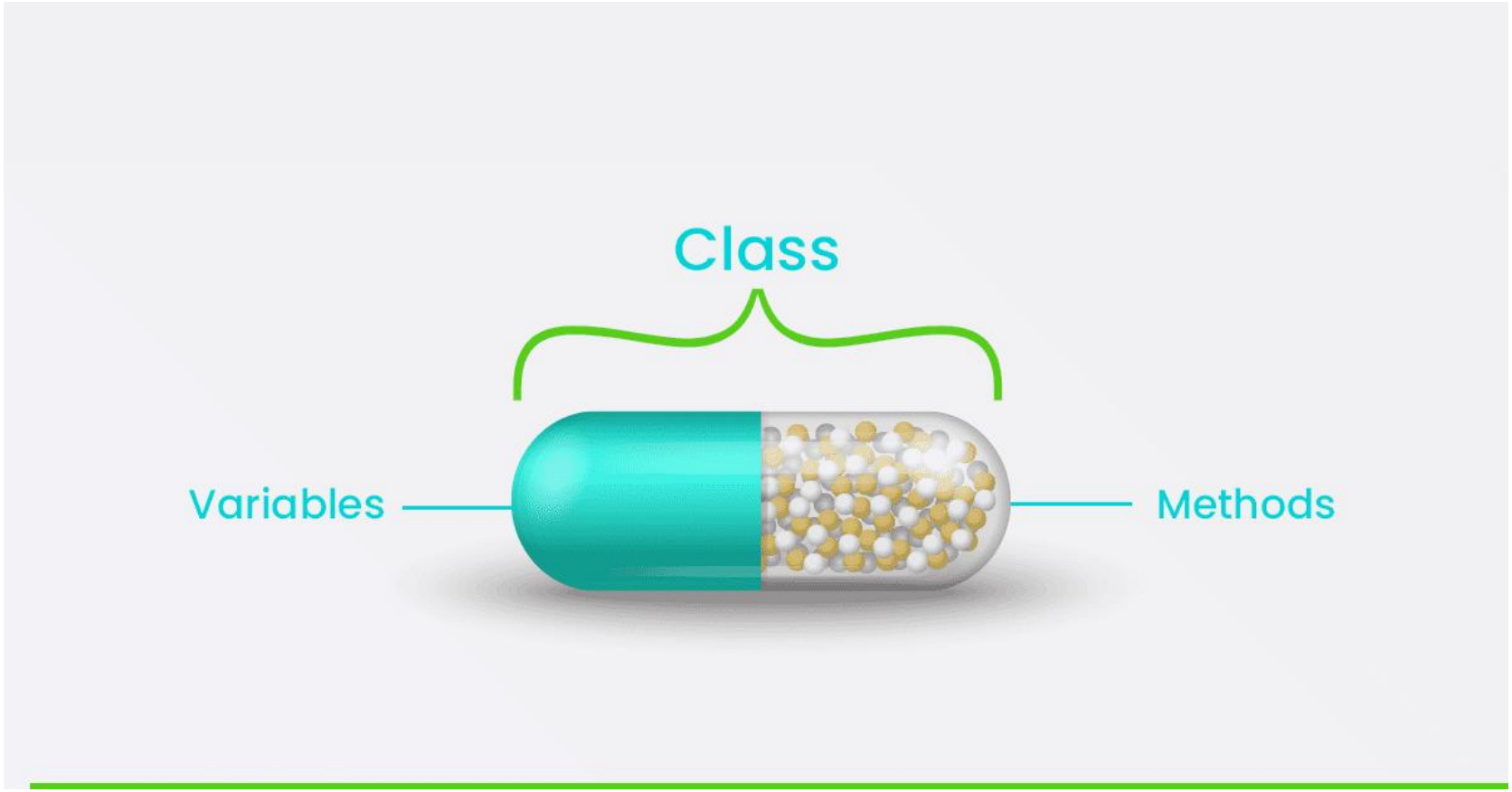
Encapsulation

Viraj Welgama

Encapsulation

- one of the fundamental principles of Object-Oriented Programming (OOP).
- It refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit, typically a **class**.
- Encapsulation also involves restricting direct access to some of an object's components, which is achieved using access modifiers like **private**, **protected**, and **public**.
- This ensures data **security** and **modularity**.

Encapsulation



Encapsulation allows to;

- Data Hiding:
  - Encapsulation hides the internal state of an object from the outside world.
  - other parts of the program don't need to know the inner workings of a class to use its objects. They just need to know how to interact with it.

Encapsulation allows to;

- Controlled Access:
  - Encapsulation allows us to control how data is accessed and modified.
  - By using access modifiers like public, private, and protected, we can specify who can access the data and methods.
    - For example, some methods may be public (accessible to everyone), while others may be private (only accessible within the class itself).

Encapsulation allows to;

- Modularity and Flexibility
  - Encapsulation promotes modularity by organizing code into self-contained units (classes).
  - This makes it easier to understand and maintain the code.
  - Additionally, encapsulation allows us to change the internal implementation of a class without affecting other parts of the program, as long as the external interface remains the same.

Encapsulation: Implementation

```cpp
class BankAccount {
private:
    std::string ownerName;
    double balance;

public:
    // Constructor
    BankAccount(std::string name, double initialBalance)
        : ownerName(name), balance(initialBalance) {}
    // Public method to deposit money
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            std::cout << "Deposited: $" << amount << std::endl;
        } else {
            std::cout << "Invalid deposit amount!" << std::endl;
        }
    }
    // Public method to withdraw money
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            std::cout << "Withdrawn: $" << amount << std::endl;
        } else {
            std::cout << "Invalid withdrawal amount or insufficient balance!" << std::endl
        }
    }
    // Public method to check balance
    double getBalance() const {
        return balance;
    }
    // Public method to display account details
    void display() const {
        std::cout << "Owner: " << ownerName << ", Balance: $" << balance << std::endl;
    }
};
```

Encapsulation: Implementation

```
class Car {
private:
    std::string brand; // Private member variable
    int year; // Private member variable

public:
    // Public member functions to set and get private member variables
    void setBrand(const std::string &b) {
        brand = b;
    }

    std::string getBrand() const {
        return brand;
    }

    void setYear(int y) {
        year = y;
    }

    int getYear() const {
        return year;
    }

    // Public member function to display car information
    void displayInfo() const {
        std::cout << "Car brand: " << brand << ", Year: " << year << std::endl;
    }
};
```