كليـــة العلـــــوم
قسم الأمن السيبراني

**Subject: Object Oriented Programming (OOP)**

**Second Stage**

**Lecturer: Dr. Abdulkadhem A. Abdulkadhem**

**Lecture (4)**

**Introduction to Object-Oriented Programming (OOP)**

**1. Introduction to Object-Oriented Programming:**

Object-Oriented Programming (OOP) is a programming paradigm that uses 'objects' to represent data and methods that operate on that data. This approach allows programmers to model real-world entities, making code more modular, reusable, and easier to maintain.
Core Concepts of OOP:

* **Object**: An object is an instance of a class that contains attributes (data) and methods (functions) that operate on the data. Think of it as a real-world entity, like a car or a person.
- **Class**: A class is a blueprint for creating objects. It defines the attributes and methods that its objects will have.

**2. Key Concepts of OOP:**

****

**2.1. Abstraction:**

Abstraction is the concept of hiding complex implementation details and exposing only the necessary parts. This allows users to interact with objects without worrying about their internal workings.

**Example**: In a car, you interact with the steering wheel without needing to know how the engine works.

|  |
| --- |
| #include <iostream>using namespace std;class Car {public: void startEngine() { cout << "Engine started!" << endl; } void drive() { cout << "Car is driving!" << endl; }};int main() { Car myCar; myCar.startEngine(); myCar.drive(); return 0;} |

**Explanation:**

* The Car class abstracts the complexity of starting and driving a car. The user doesn't need to know how the engine starts internally, just that they need to call the startEngine() and drive() methods.
* When we create an object myCar, we can use these functions to simulate starting and driving the car, without needing to understand the inner workings of the engine.

**2.2. Encapsulation:**

Encapsulation is the bundling (تجميع) of data (attributes) and methods (functions) that operate on the data into a single unit or class. It also restricts direct access to some components, which is essential for protecting data integrity.

|  |
| --- |
| #include <iostream>using namespace std;class BankAccount {private: int balance;public: BankAccount(int initialBalance) { balance = initialBalance; } void deposit(int amount) { balance += amount; } void withdraw(int amount) { if (amount <= balance) { balance -= amount; } else { cout << "Insufficient funds!" << endl; } } int getBalance() { return balance; }};int main() { BankAccount account(500); account.deposit(200); account.withdraw(100); cout << "Current balance: " << account.getBalance() << endl; return 0;} |

**Explanation:**

* The BankAccount class encapsulates the balance attribute as a private variable. Users cannot modify it directly but can interact with it via public methods like deposit(), withdraw(), and getBalance().
* This protects the balance from being accidentally modified, ensuring that all changes occur through the proper methods.

**2.3. Inheritance:**

Inheritance allows one class to inherit (يرث) the attributes and methods of another class, promoting (يتميز) code reuse. The new class, known as the 'derived' or 'child' class, can also have its own additional attributes and methods.

|  |
| --- |
| #include <iostream>using namespace std;class Animal {public: void eat() { cout << "This animal is eating!" << endl; }};class Dog : public Animal {public: void bark() { cout << "The dog is barking!" << endl; }};int main() { Dog myDog; myDog.eat(); // Inherited method myDog.bark(); // Dog-specific method return 0;} |

Explanation:

* The Dog class inherits from the Animal class. This means that all Dog objects can use the eat() method defined in Animal.
* Inheritance allows us to reuse the functionality of the Animal class while adding more specific methods, like bark(), to the Dog class.

**3. Conclusion:**

Object-Oriented Programming makes software design more organized and intuitive by breaking down a problem into objects. Through abstraction, encapsulation, inheritance, and polymorphism, OOP encourages code reuse, modularity, and ease of maintenance.
Key Takeaways:

* **Abstraction** simplifies complex systems.
* **Encapsulation** protects data and ensures controlled access.
* **Inheritance** promotes code reuse.