جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلـــوم
قـســـــــم علوم الذكاء الاصطناعي

Lecture 2

••••••••••••••••••••••••••••••

المادة : Object Oriented Programming
المرحلة : Second Stage
اسم الاستاذ: م.م مصطفى محمد شبر

Topic

- Introduction to Object-Oriented Programming (OOP)
- Fundamental concepts: Object, Class, Abstraction, Encapsulation, Inheritance, Polymorphism

Learning Objectives

By the end of this lecture, students will be able to:

1. Define the principles of Object-Oriented Programming.
2. Differentiate between procedural programming and OOP.
3. Understand the concepts of objects and classes.
4. Explain the role of abstraction and encapsulation in OOP.
5. Describe the importance of inheritance and polymorphism.

Theoretical Part

1. Introduction to Object Oriented Programming

- **Definition:** Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects, which encapsulate both data (attributes) and behavior (methods).
- Unlike procedural programming, OOP emphasizes modularity, reusability, and maintainability.

2. Object and Class

- Class: A blueprint or template from which objects are created. It defines attributes (data members) and methods (member functions).
- Object: An instance of a class that represents a real-world entity.

```
#include <iostream>

using namespace std;


class Student{

public:

   string name;

   int age;


   void display (){

      cout << "Name: " << name << ", Age: " << age << endl;

   }

}
```

```
int main (){
   Student s1;
   s1.name = "Ali;"
   s1.age = 20;
   s1.display() ;
}
```

3. Abstraction

- **Definition:** The process of hiding implementation details and exposing only essential features.
- Abstraction allows programmers to focus on what an object does rather than how it does it.

4. Encapsulation

Definition: The bundling of data and methods that operate on that data into a single unit (class).

Achieved in C++ by using access specifiers:

- public → accessible from outside the class.
- private → accessible only within the class.
- protected → accessible within the class and derived classes.

```
class BankAccount
{
private    :double balance;

public  :BankAccount(double first) {

      balance = first;

}

   void deposit(double amount) {

      balance += amount;

   }

   double getBalance () {

      return balance;

}

};
```

5. Inheritance

- **Definition:** Mechanism by which one class (derived class) can acquire the properties and behaviors of another class (base class).
- Inheritance promotes code reusability and hierarchical classification.

6. Polymorphism

- **Definition:** The ability of different objects to respond to the same message (function call) in different ways.

- **Types of Polymorphism:**
  - Compile-time (Static) polymorphism → function overloading, operator overloading.
  - Run-time (Dynamic) polymorphism → achieved using virtual functions.

- **Practical Illustrations**

Example 1 – Class and Object

```
#include <iostream>

using namespace std;


class Car  {

public:

   string brand;

   int year;


   void start (){

      cout << brand << " started in " << year << endl;
```

```
}
};

int main (){
   Car c1;
   c1.brand = "Toyota;"
   c1.year = 2022;
   c1.start;()
{
```

Example 2 – Encapsulation

```
#include <iostream>
using namespace std;
class Employee{
private:
   int salary;
public:
   void setSalary(int s) {
      salary = s;
{
```

```
    int getSalary} ()
        return salary;
{
};

int main} ()
    Employee e1;
    e1.setSalary(5000);
    cout << "Employee salary: " << e1.getSalary;()
{
```

- **Assignments**

- Define a class Book with attributes (title, author, price) and a method to display book information. Create two objects and display their data.
- Implement a class Rectangle that encapsulates length and width as private members, and provides public methods to calculate area and perimeter.
- Write a program that demonstrates abstraction by designing a class Shape with a pure virtual function calculateArea(), then implement it in subclasses Circle and Square.