كليـــة العلـــــوم  
قسم الأمن السيبراني

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

**Second Stage**

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

**Lecture (5)**

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

**2.4. Polymorphism:**

Polymorphism allows objects of different types to be treated as objects of a common superclass. It enables a single function or method to behave differently based on the object that calls it.

|  |
| --- |
| خلفيات 4K الحرباء جديدة , اجمل صور الزواحف , 2024 Chameleon Wallpapers |  صقور الإبدآع#include <iostream>  using namespace std;  class Animal {  public:  virtual void makeSound() {  cout << "Some generic animal sound" << endl;  }  };  class Dog : public Animal {  public:  void makeSound() override {  cout << "Woof!" << endl;  }  };  class Cat : public Animal {  public:  void makeSound() override {  cout << "Meow!" << endl;  }  };  int main() {  Animal\* myAnimal;  Dog myDog;  Cat myCat;  myAnimal = &myDog;  myAnimal->makeSound(); // Outputs: Woof!  myAnimal = &myCat;  myAnimal->makeSound(); // Outputs: Meow!  return 0;  } |

**Explanation:**

* Here, we use polymorphism to treat Dog and Cat objects as Animal objects. The method makeSound() behaves differently depending on whether it's called by a Dog or a Cat.
* Polymorphism enables flexibility and the ability to handle multiple object types through a common interface (in this case, Animal).

**Example**: Let’s consider a simple example where we have different geometric shapes like circles, rectangles, and triangles. We want to write a method that can draw any shape, but the way we draw each shape is different.

To achieve this, we can create a base class Shape that has a virtual method draw(). Each derived class will override the draw() method to provide its specific implementation.

|  |
| --- |
| #include <iostream>  using namespace std;  class Shape { // Base class (superclass)  public:  virtual void draw() { // Virtual function to allow overriding in derived classes  cout << "Drawing a generic shape" << endl;  }  };  class Circle : public Shape { // Derived class representing a circle  public:  void draw() override { // Overriding the draw() function for Circle  cout << "Drawing a circle" << endl;  }  };  class Rectangle : public Shape { // Derived class representing a rectangle  public:  void draw() override { // Overriding the draw() function for Rectangle  cout << "Drawing a rectangle" << endl;  }  };  class Triangle : public Shape { // Derived class representing a triangle  public:  void draw() override { // Overriding the draw() function for Triangle  cout << "Drawing a triangle" << endl;  }  };  int main() {  Shape\* shape; // Pointer to base class (Shape)  Circle circle;  Rectangle rectangle;  Triangle triangle;  shape = &circle; // Assign pointer to Circle object and call draw()  shape->draw(); // Output: Drawing a circle  shape = &rectangle; // Assign pointer to Rectangle object and call draw()  shape->draw(); // Output: Drawing a rectangle  shape = &triangle; // Assign pointer to Triangle object and call draw()  shape->draw(); // Output: Drawing a triangle  return 0;  } |

**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.
* **Polymorphism** allows for flexibility in method behavior.