كليـــة العلـــــوم  
قسم الأمن السيبراني

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

**Second Stage**

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

**Lecture (7)**

**Friend Functions and Friend Classes**

**Friend Functions and Friend Classes in OOP**

**1. Introduction to Encapsulation and Access Specifiers**

Before diving into **friend functions** and **friend classes**, it's important to understand a core principle of Object-Oriented Programming (OOP) known as **encapsulation**.

**Encapsulation:**

* Encapsulation is the concept of **bundling data (variables)** and the **methods (functions)** that operate on that data into a single unit, the class.
* It restricts direct access to some of the class's components, which ensures data integrity and hides the internal implementation details.

**Access Specifiers:**

There are three common access specifiers used to control access to class members:

* **public**: Members are accessible from outside the class.
* **private**: Members are only accessible within the class itself.
* **protected**: Members are accessible in the class and its derived classes.

In most cases, class data members are kept private to protect them from unauthorized access or modification.

However, there are situations where we may need to allow an external function or another class to access these private members. This is where **friend functions** and **friend classes** come into play.

**2. Friend Functions**

**What is a Friend Function?**

A **friend function** is a non-member function that is granted access to the private and protected members of a class. Although it is not a part of the class, it can access its private data as if it were.

**Why Use a Friend Function?**

1. **External Function Needs Access**: Sometimes, you need an external function to access the private members of a class, but you don't want to make the members public.
2. **Improving Flexibility**: A friend function can be useful when a function is logically connected to the class but doesn’t need to be a member function.

**Syntax of Friend Function**

The friend function is declared inside the class using the keyword friend, but it is defined outside the class.

class ClassName {

private:

int data;

public:

ClassName(int value) : data(value) {}

// Friend function declaration

friend void displayData(ClassName obj);

};

// Friend function definition

void displayData(ClassName obj) {

cout << "Data: " << obj.data << endl; // Accessing private member

}

**Code Example 1: Friend Function**

|  |
| --- |
| **Code no. 1. تنفيذ في المختبر** |
| #include <iostream>  using namespace std;  class Box {  private:  int length;  public:  Box(int len) : length(len) {} // Constructor to initialize length  // Friend function declaration  friend void printLength(Box box);  };  // Friend function definition  void printLength(Box box) {  // Can access the private member of Box class  cout << "Length of the box: " << box.length << endl;  }  int main() {  Box myBox(10);  printLength(myBox); // Calling the friend function  return 0;  } |

**Explanation:**

* The Box class has a private member length.
* printLength is a friend function of the Box class, so it can access the private member length.
* In the main function, an object myBox is created, and printLength(myBox) prints the value of the private member length of the myBox object.

**Code Example 2: Friend Function**

|  |
| --- |
| **Code no. 2. تنفيذ في المختبر** |
| #include <iostream>  using namespace std;  class Cylinder {  private:  double radius;  double height;  public:  // Constructor to initialize radius and height  Cylinder(double r, double h) : radius(r), height(h) {}  // Friend function declaration  friend double calculateVolume(Cylinder cyl);  };  // Friend function definition  double calculateVolume(Cylinder cyl) {  // Can access the private members of Cylinder class  const double PI = 3.14159265;  return PI \* cyl.radius \* cyl.radius \* cyl.height;  }  int main() {  Cylinder myCylinder(3.0, 5.0);  cout << "Volume of Cylinder: " << calculateVolume(myCylinder) << endl; // Calling the friend function  return 0;  } |

**Explanation:**

* The Cylinder class has two private members, radius and height.
* calculateVolume is a friend function of the Cylinder class, allowing it to access the private members radius and height.
* In the main function, a Cylinder object named myCylinder is created, and calculateVolume(myCylinder) calculates and prints the volume of the cylinder using the private data members of myCylinder

**3. Friend Classes**

**What is a Friend Class?**

A **friend class** is a class that is granted access to the private and protected members of another class. If a class is declared as a friend, all of its member functions can access the private members of the other class.

**Why Use a Friend Class?**

1. **Tightly Related Classes**: Sometimes two classes are closely related, and they need to share private members.
2. **Complex Relationships**: In scenarios where multiple classes interact heavily, declaring one class as a friend of another simplifies access control, avoiding the need for complex getter/setter methods.

**Syntax of Friend Class**

A class is made a friend of another class using the friend keyword in the class declaration.

class B; // Forward declaration of class B

class A {

private:

int privateData;

public:

A(int value) : privateData(value) {}

// Declare class B as a friend of class A

friend class B;

};

class B {

public:

void showAData(A obj) {

// Accessing the private member of class A

cout << "Private data of class A: " << obj.privateData << endl;

}

};

**Code Example 3: Friend Class**

|  |
| --- |
| #include <iostream>  using namespace std;  class Square; // Forward declaration of Square  class Rectangle {  private:  int width, height;  public:  Rectangle(int w, int h) : width(w), height(h) {}  // Friend class declaration  friend class Square;  };  class Square {  public:  int calculateArea(Rectangle rect) {  // Can access the private members of Rectangle  return rect.width \* rect.height;  }  };  int main() {  Rectangle rect(4, 5);  Square sq;  cout << "Area of Rectangle: " << sq.calculateArea(rect) << endl;  return 0;  } |

**Explanation:**

* The class Square is declared as a friend of the Rectangle class.
* Because of this friendship, the member function calculateArea of the Square class can access the private members width and height of the Rectangle class.
* This approach allows Square to calculate the area of a Rectangle object even though width and height are private in Rectangle.

**Code Example 4: Friend Class**

|  |
| --- |
| Code no. 4. تنفيذ في المختبر |
| #include <iostream>  #include <string>  using namespace std;  class Book {  private:  string title;  string author;  int pages;  public:  **// Constructor to initialize title and author**  Book(string t, string a, int p) : title(t), author(a), pages(p) {}  **// Declare Library as a friend class**  friend class Library;  };  class Library {  public:  **// Method to display book details, accessing private members of Book**  void displayBookDetails(const Book& book);  };  **// Friend class function definition**  void Library::displayBookDetails(const Book& book) {  **// Access private members of Book**  cout << "Book Title: " << book.title << endl;  cout << "Author: " << book.author << endl;  cout << "No. of Pages: " << book.pages << endl;  }  int main() {  Book myBook("Object oriented programming", "Abdulkadhem",350);  Library myLibrary;  myLibrary.displayBookDetails(myBook); // Using Library to display details of Book  return 0;  } |

**Explanation:**

* The Book class has two private members, title and author.
* Library is declared as a friend class of Book, granting it access to Book's private members.
* The displayBookDetails method in Library accesses the private members of Book to display its details.
* In the main function, a Book object myBook is created, and myLibrary.displayBookDetails(myBook) calls the friend class method to print the title and author of myBook.

**4. Key Points about Friend Functions and Classes**

**Friend Functions:**

* Declared inside the class but defined outside the class.
* Not a member function of the class but has access to its private and protected members.
* Can be useful when an external function needs to access private data of the class without being a member of the class.

**Friend Classes:**

* A class can grant another class access to its private and protected members by declaring it as a friend.
* All member functions of the friend class can access the private members of the original class.
* Useful when classes have a close relationship and need direct access to each other’s private data.

**Important Notes:**

* **Friendship is not mutual**: If class A declares class B as a friend, class B can access private members of class A, but class A cannot access private members of class B unless explicitly declared as a friend in class B.
* **Friendship is not inherited**: If class B is a friend of class A, the derived classes of B do not automatically become friends of class A.
* **Friendship breaks encapsulation to some extent**, so it should be used sparingly and only when necessary to maintain clarity and modularity in the code.

**5. Advantages and Disadvantages**

**Advantages:**

* **Controlled Access**: Friend functions and classes provide controlled access to private members without exposing them publicly.
* **Flexibility**: They offer a flexible way to allow certain external functions or classes to interact with a class’s private data, especially when these functions or classes are logically related but don’t belong to the class itself.
* **Better Design**: They can simplify code when classes have strong relationships, avoiding the need for complex getter and setter methods.

**Disadvantages:**

* **Violation of Encapsulation**: One of the core principles of OOP is encapsulation, and using friend functions or classes breaks this principle to some extent by allowing external entities to access private data.
* **Tightly Coupled Code**: Excessive use of friend functions or classes can lead to tightly coupled code, which can make maintenance and future extensions more difficult.
* **Misuse**: If overused, it can lead to poorly designed code that is difficult to debug and understand.