Object Oriented programming and software engineering – Lab 7

Adam Korytowski – 2025

1. Friend function

A friend function in C++ is a function that is not a member of a class but has access to the class's private and protected members. It is declared inside the class with the *friend* keyword.

Why Use Friend Functions?

- Allows external functions to access private members of a class.
- Helps in cases where an external function needs to operate on multiple classes.

Key Features of Friend Functions

1. Not a Member Function → It is declared inside the class but defined outside.
2. Can Access Private Members → It can access private and protected members of the class.
3. Requires Explicit Declaration → It must be declared inside the class using *friend*.
4. Not Called Using Object → It is called like a normal function, not using object.function().

When to Use Friend Functions?

- When you need **external access** to private members.
- When **operator overloading** requires access to private data.
- When functions need to operate on **multiple objects**.

When to Avoid?

- If you can achieve the same functionality with **public member functions**.
- If excessive use makes the class less **encapsulated**.

Example:

```cpp
class Character
{
    string name;

public:

    friend void interact(Character character, Enemy enemy);
    Character(string in_name) : name(in_name) {}

};
```

```cpp
void interact(Character character, Enemy enemy)
{
    cout << "Interacting: " << character.name << " and " << enemy.getName() << endl;
}

class Enemy
{
    string name;

public:
    Enemy(string in_name) : name(in_name) {}

    string getName() { return name; }
};
```

```cpp
int main()
{
    Character* c = new Character("Warrior");
    Enemy* e = new Enemy("Monster");
    interact(*c, *e);
}
```

2. Friend Classes

A friend class in C++ allows one class to access the private and protected members of another class. This is useful when two or more classes need to work closely together.

Why Use Friend Classes?

- Tightly coupled classes → When one class needs direct access to another's private data.
- Improves performance → Reduces the need for getter/setter methods.
- Useful in operator overloading → When overloading operators between two classes.

```cpp
class Character
{
    friend class Inventory;

    int attackPower;

    string name;

public:
    Character(string in_name) : name(in_name) {}

};
```

```cpp
class Inventory
{
public:
    void equipWeapon(Character& c, int extraAttack)
    {
        cout << c.name << " equipped a weapon! Attack increased by " << extraAttack << "!\n";
        c.attackPower += extraAttack;
    }
```

```cpp
Character* c = new Character("Warrior");
Inventory inventory;
inventory.equipWeapon(*c, 10);
```

Why Use a Friend Class Here?

- Direct access to private stats without needing setters.
- Encapsulation is still maintained (only Inventory can modify Character's stats).

3. Tasks

- Come up with reasons and create 3 friend functions for your existing classes from previous laboratories, use them in code (3 pts)
- Come up with reasons and create 2 friend classes in your code from previous laboratories, use them in code (3 pts)