AGH University of Krakow | Faculty of Mechanical Engineering and Robotics
| Department of Robotics and Mechatronics

MECHATRONIC ENGINEERING | I cycle | semester VI | 2025/2026

Object oriented programming and software engineering

Instruction VII:

Abstract and nested classes, friend classes and friend functions

You will learn:

How to design systems using abstract classes and how they define a common interface for different types of objects. You will understand how derived classes implement specific behavior while sharing a common structure.

You will also learn how nested classes can be used to logically group components inside a larger system, improving code organization.

Additionally, you will explore friend classes and friend functions, which allow controlled access to private data when cooperation between classes is required.

By the end of the laboratory you will be able to design structured systems where different components interact while maintaining encapsulation.

Course supervisor:
dr inż. Lucjan Miękina, miekina@agh.edu.pl

Instruction author:
mgr inż. Joanna Koszyk, jkoszyk@agh.edu.pl

1. Initial information

Abstract class is a class that contains at least one pure virtual function and cannot be instantiated. It defines a common interface for derived classes.

Nested class is a class defined inside another class. It is typically used when one class is closely related to another and should not exist independently.

Friend function is a function that has access to the private members of a class.

Friend class has access to the private members of another class.

2. Theoretical content

Example of an abstract class:

```
#include <iostream>

class Sensor {
public:
    virtual void measure() = 0; // pure virtual function

    void displayInfo() {
        std::cout << "Generic sensor" << std::endl;
    }
};

class TemperatureSensor : public Sensor {
public:
    void measure() override {
        std::cout << "Measuring temperature" << std::endl;
    }
};
```

In this example, the class Sensor is abstract and cannot be used to create objects. The class TemperatureSensor implements the required function and can create objects.

Example of a nested class:

```
#include <iostream>

class SensorModule {
public:
    class DataBuffer {
    private:
        int value;

    public:
        void set(int v) { value = v; }
        int get() { return value; }
    };

    void processData() {
        DataBuffer buffer;
        buffer.set(10);
        std::cout << "Buffer value: " << buffer.get() << std::endl;
    }
};
```

Example of a friend function:

```cpp
#include <iostream>

class Sensor {
private:
    double measurement;

public:
    Sensor(double m){
        measurement = m;
    }

    friend void printMeasurement(const Sensor& s);
};

void printMeasurement(const Sensor& s) {
    std::cout << "Measurement: " << s.measurement << std::endl;
}
```

Example of a friend class:

```cpp
#include <iostream>

class MotorController {
private:
    int speed;

    void internalCheck() {
        std::cout << "Internal check" << std::endl;
    }

public:
    MotorController(int s) : speed(s) {}

    friend class DiagnosticModule;
};

class DiagnosticModule {
public:
    void analyze(MotorController& m) {
        std::cout << "Speed: " << m.speed << std::endl;  // access
private variable
        m.internalCheck();                               // access
private function
    }
};
```

3. Tasks

TASK 1.

Design an abstract class representing a generic sensor in an engineering system. The class should define a common interface for performing measurements and printing results. Create several derived classes representing different types of sensors, such as temperature, pressure, or distance sensors. Each class should implement its own measurement logic. Simulate measurements and display results for multiple sensor types.

TASK 2.

Design a class representing a robot. Inside this class, define a nested class representing a components such as motor, sensor. The nested class should handle a specific responsibility, for example storing recent measurements or changing speed. Use the outer class to manage the sensor and motor and interact with the nested component.

TASK 3.

Design a class representing a measurement device where certain internal data (e.g., raw measurement values or calibration parameters) are private. Create a separate function responsible for analyzing or calibrating the device. This function should be declared as a friend so that it can access private members. Use this function to modify or evaluate internal data.

In the next step, add a friend class representing data analyser.