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 IV:

Classes and encapsulation in C++

You will learn:

How to define and use classes in C++ to model objects and organize program structure. You will practice creating attributes and methods.

You will also learn how encapsulation and how it helps protect data and control access using access specifiers such as private and public. Additionally, during the laboratory you will learn how to initialize objects using constructors and how destructors are used.

Finally, you will learn how to organize larger programs by splitting classes into separate header (.h) and implementation (.cpp) files, which is a common practice in real C++ projects.

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

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

1. Initial information

Class is a user-defined type that expands a concept of a structure. It combines variables and functions into a single unit. Classes are used to represent objects and organize programs using the principles of object-oriented programming.

Object is an instance of class.

Attributes are variables that store the state or properties of an object.

Methods are functions defined inside a class that operate on the object's data.

Encapsulation is the principle of restricting direct access to certain attributes of a class. It is typically implemented using access specifiers: **private**, **public**, and **protected**.

Constructor is a special method that is automatically called when an object of a class is created. It is used to initialize the object's attributes.

Destructor is automatically called when an object is destroyed. It is typically used to release resources or perform cleanup operations.

2. Theoretical content

A simple class definition:

```
class Motor {
public:
    int speed;

    void setSpeed(int s) {
        speed = s;
    }

    void displaySpeed() {
        std::cout << "Speed: " << speed << std::endl;
    }
};
```

Objects are instances of a class:

```
Motor m1;
m1.setSpeed(1500);
m1.displaySpeed();
```

Encapsulation allows data to be protected from direct modification by declaring attributes as private. Public methods can then be used to access or modify the data safely.

```
class Motor {
private:
    int speed;
public:
    void setSpeed(int s) {
        speed = s;
    }
    int getSpeed() {
        return speed;
    }
};
```

A constructor initializes an object when it is created. Example:

```
class Motor {
private:
    int speed;

public:
    Motor(int s) {
        speed = s;
    }
};
```

When an object is destroyed, destructor is automatically executed.

```
~Motor() {
    std::cout << "Motor object destroyed" << std::endl;
}
```

In larger programs, classes are usually separated into two files. Header file (.h) contains the class declaration and source file (.cpp) contains the implementation of the class methods.

Motor.h:

```
class Motor {
private:
    int speed;

public:
    Motor(int s);
    void setSpeed(int s);
    int getSpeed();
};
```

Motor.cpp:

```
#include "Motor.h"

Motor::Motor(int s) {
    speed = s;
}
void Motor::setSpeed(int s) {
    speed = s;
}
int Motor::getSpeed() {
    return speed;
}
```

3. Tasks

TASK 1.

Create a class of an environmental sensor. Think about what information such a device should store. Your class should allow storing the current and previous measurements, updating the measurements, displaying the current value.

Create several objects of this class and simulate measurements from different devices.

TASK 2.

Modify your class from Task 1 so that its attributes are private. Provide methods that allow setting and getting the sensor values, updating the measurements, displaying all current measurements in a readable format.

Additionally, add basic validation of wrong measurements inside the class methods. For example: temperature should be within a reasonable range (e.g. −50°C to 100°C), humidity should be between 0% and 100%, pressure should be within a realistic atmospheric range (e.g. 900–1100 hPa).

If a wrong value is provided, the program should display a warning and reject the update.

Extend the class with a method that simulates sensor measurements using random values. Create several sensor objects and simulate multiple measurement updates during program execution.

TASK 3.

Extend your class by adding constructors that initialize the object. Each device should have a device name and initial measurement values.

TASK 4.

Add a destructor to your class that prints a message when an object is destroyed. Create objects inside different scopes (for example inside functions or blocks of code) and observe when the destructor is executed.

TASK 5.

Create a class representing a LiDAR sensor used for distance measurements.

LiDAR sensor emits laser pulses and measures the distance to surrounding objects at different angles.

Create a simple 2D scanning LiDAR with a sensor name, minimum and maximum measurement range, number of scan angles, distance measurements.

Your class should generate random distance measurements within the sensor range, display scan results, detect the closest detected obstacle, check safety based on the closest distance.

TASK 6.

Change your program so that the class definition and implementation are placed in separate files. Your project should contain files such as main.cpp, Lidar.cpp, Lidar.h.

Make sure the program compiles and runs correctly.