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 VI:

Encapsulation, inheritance and polymorphism in C++

You will learn:

How encapsulation and inheritance are used in object-oriented programming to organize code and model relationships between objects.

You will learn how a base class can define common attributes and methods that are shared by multiple derived classes. During this laboratory you will also learn how inheritance allows new classes to reuse and extend existing functionality.

You will then learn polymorphism, which allows objects of different classes to be treated in a uniform way while still executing their own specific behavior.

By the end of the laboratory, you will be able to design class hierarchies and implement programs where multiple object types interact through a common interface.

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

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

1. Initial information

Encapsulation is the principle of restricting direct access to the internal data of a class. Data members are typically declared as private, and access is provided through public methods. This protects the internal state of objects and ensures that data can only be modified in controlled ways.

Inheritance allows a class to derive from another class. Attributes and methods can be inherited. The original class is called the base class, and the new class is called the derived class. Inheritance allows programmers to reuse existing code and represent hierarchical relationships between objects.

Base Class defines common attributes and methods that can be shared by other classes.

Derived Class inherits the properties of the base class and may introduce additional attributes or methods.

Polymorphism allows objects of different classes to be treated as objects of a common base class. The specific behavior of an object is determined at runtime.

2. Theoretical content

Example of a base class:

```
class Sensor {
protected:
    std::string name;

public:
    void setName(std::string n) {
        name = n;
    }

    void displayName() {
        std::cout << "Sensor: " << name << std::endl;
    }
};
```

A derived class can extend the base class:

```
class TemperatureSensor : public Sensor {
private:
    double temperature;

public:
    void setTemperature(double t) {
        temperature = t;
    }

    void displayMeasurement() {
        std::cout << "Temperature: " << temperature << std::endl;
    }
};
```

The derived class automatically inherits the methods of the base class. Usage:

```
TemperatureSensor sensor;
sensor.setName("Outdoor Sensor");
sensor.setTemperature(21.5);
sensor.displayName();
sensor.displayMeasurement();
```

Polymorphism allows the same method name to perform different actions depending on the object type.

```
class Sensor {
public:
    virtual void measure() {
        std::cout << "Measurement" << std::endl;
    }
};

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

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

Objects can be accessed through a base class pointer:

```
Sensor* s;

TemperatureSensor t;
PressureSensor p;

s = &t;
s->measure();

s = &p;
s->measure();
```

3. Tasks

TASK 1.

Create a base class representing a measurement device used in a mechatronic system. Think about what information all measurement devices should share. Additionally, implement methods that allow modifying device parameters, enabling or disabling the device, displaying basic device information. Create several objects and verify that the class works correctly.

TASK 2.

Extend your design from Task 1 by creating several derived classes representing different types of sensors, for example temperature sensor, pressure sensor, LiDAR sensor. Each derived class should introduce additional attributes and functionality specific to that sensor. Design methods that allow each sensor to generate or update measurements. Test the program.

TASK 3.

In the next task, extend your classes from Task 2 so that they analyze the collected measurements. For example temperature sensor should calculate average temperature from several readings, pressure sensor should determine if pressure is within a normal atmospheric range, LiDAR sensor should find the closest detected obstacle. Each sensor should implement its own analysis method.

TASK 4.

Modify the base class so that it declares a virtual method responsible for performing a measurement. Each derived sensor class should implement its own version of this method. Then create a system that stores different sensor objects using pointers to the base class. Use a loop to trigger measurements for all sensors in the system.