Advanced Programming Tutorial
Worksheet 6: Object-Oriented Programming (Part 1)

Keywords: `class`, `enum`, constructors, access modifiers `public` and `private`, `const` methods, `friend`, `static` members

Prepare!

Before coming to the tutorial, and so that the time is enough to solve the exercise, revise the slides of the lecture unit "Object-Oriented Programming (pt I)" and answer the following questions:

- ☐ What is the difference between a class, a `struct`, and an object?
- ☐ What are some common member functions that you know? What do they do?
- ☐ What is the difference in purpose and syntax between a copy constructor and an assignment operator?
- ☐ What is a UML class diagram? What symbols do you know and what do they mean?
- ☐ What is the difference between a shallow and a deep copy? Give an example of a situation where a shallow copy can happen and what side-effects it may have.
- ☐ What is a const method? How do we specify it?
- ☐ Look into the `PanSimul` skeleton: What files are there? What does the `main` function do? What members does the `Person` class contain?
- ☐ Look into the `OnlineMUT` skeleton: What does the `main` function do? What classes are there?

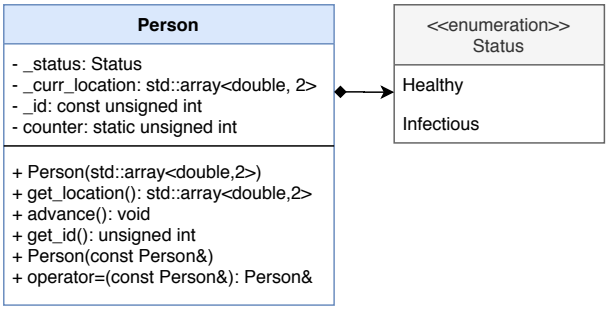
Exercise 1: Simulating infection spreads

Let's draft a simulation of the spread of an infection in a population. For this, you can imagine a 2D domain/space within which people can move around. Each `Person` has a status: they are either `Infectious` or `Healthy`. The simulation iterates over multiple time steps. In each step, a `Person` changes their location based on some rule (undefined at the moment).

The first step in this process is creating a `Person` class with the following properties:

- Since the `status` of each person only has two values (`Healthy` or `Infectious`), we can express this using an enumeration type or an `enum class` in `C++` (for details, see below).
- Each person has a unique `const` member variable `_id`. This member variable is initialized using the `static unsigned int counter` which is shared among all instances of the class (see below).

This `Person` class is visualized in the following UML class diagram:



Implement the following making sure that the tests in `main` succeed:

1. A constructor that initializes `_curr_location` and the unique `_id` of the object using the `counter`.
2. The copy constructor. Since each person has a unique `_id`, do not copy this – instead, use a new id.
3. The copy assignment operator.

You can use the included `CMakeLists.txt` to compile your code.

Some new concepts

`enum class`

An enumeration is a type whose values can be listed or "enumerated". For example, imagine we need a type to represent whether a `Person` is `Infectious` or `Healthy`. In `C++`, we can do this using an `enum class`:

```
enum class Status {Infectious, Healthy};
Status MyStatus = Status::Healthy;
```

`static`

A `static` variable can have different meanings when defined in different locations. Inside a class, a `static` variable is shared among all objects of that class. This means that every object of the `Person` class has access to the *same* `counter`.

We can access `static` members by using the `::` notation, i.e. `Person::counter`. This essentially means that we do not need to instantiate a `Person` object to access the `static` variable.

Exercise 2: Students & Courses

You are starting as a 1st semester Master's Student at a university, but as you wait on the line for two hours to apply for a student card and four more hours later in the semester to register for your exams, you decide to take action. You invent OnlineMUT (Online Management of University Tables), a system to manage student cards and exam registrations.

Use the provided code skeleton to implement the following, so that `main.cpp` works:

1. You identify students as the central component of your system. Therefore, you write a `Student` class. This class contains:
   - A name and a registration number (both strings): while a student can change their name, they can never change their registration number.
   - A list of courses that the student has registered for.
   - A constructor.
   - Methods to register for a course and print their list of courses.
2. A `StudentCard` should be created only for existing students, but it does not store a `Student` object: it only stores a `_student_name`. Due to data protection, only the `StudentCard` should be able to access names of students.
3. You then start drafting the concept of a `Course`. This class should contain:
   - A string representing the course id.
   - An `enum` representing the course type ("Lecture", "Seminar", "Practical").
   - A constructor that sets values to all of its members.
   - A `get_id()` method.
4. Sketch a UML Class Diagram to visualize the classes of your code (e.g. on diagrams.net or Apollon).
5. **Idea:** After completing your code, publish it as an open-source project on your favorite Git repository manager and pitch it to your colleagues. Who knows what happens next?

Epilogue

As you may have noticed already, we can (and in practice we do) use object-oriented programming both for "conventional", user-facing software with complex logic (e.g., `OnlineMUT`), but also for computation-intensive, simulation software (e.g., `PanSimul`). Popular examples are the PDE frameworks OpenFOAM, deal.II, SU2, and more, which contain hundreds of classes. A class can be a mesh, a cell, a numerical solver, and more. These are typically composed (each class uses other classes), and specialized (a general solver interface class will have concrete implementations to choose from at runtime). While older software often relies less or not at all on OOP (usually software written in C and Fortran), and people used to believe that OOP makes things slower, this is not a general truth. We will see in the Code Optimization lecture how we can organize our code to minimize memory transfers, which is usually the bottleneck, by storing next to each other data often used together.