Advanced Programming Tutorial
Worksheet 6: OOP

Firas Drass

Object-Oriented Programming (OOP)

- Object Oriented Programming (OOP) is one of the most widely used programming paradigm

- Why is it extensively used?
  - Well suited for building trivial and complex applications
  - Allows re-use of code thereby increasing productivity
  - New features can be easily built into the existing code
  - Reduced production cost and maintenance cost

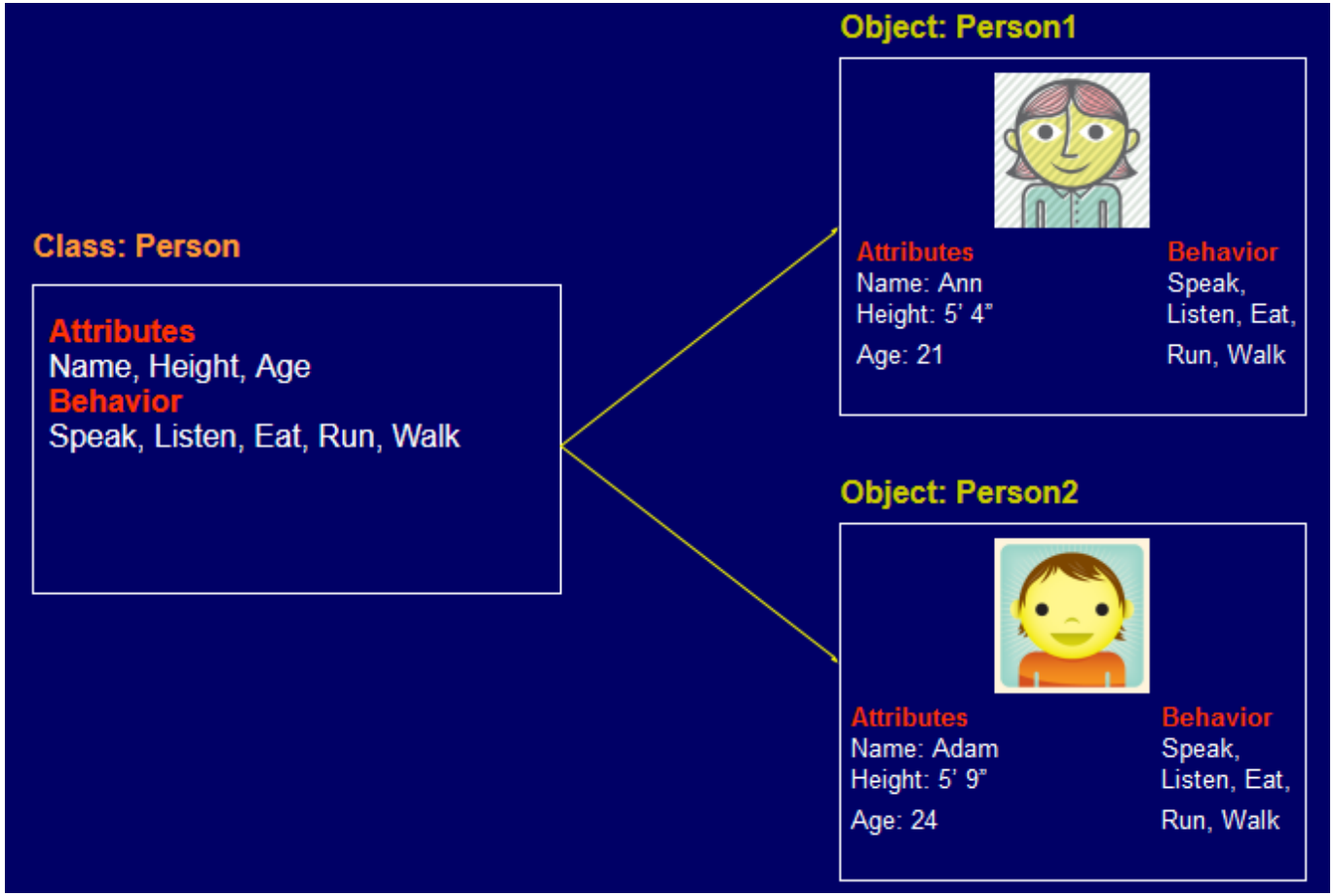
- Common programming languages used for OOP include C++, Java, and C#

Building Blocks of OOP: Objects & Classes

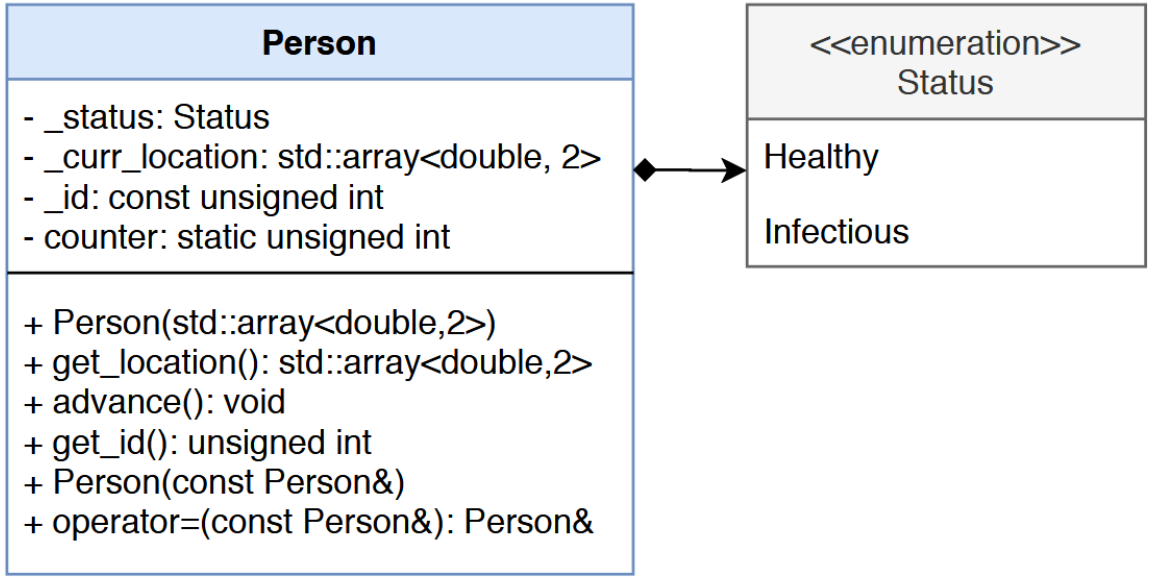
- Class has
  - Set of attributes or properties that describes every object
  - Set of behavior or actions that every object can perform

- Object has
  - Set of data (value for each of its attribute)
  - Set of actions that it can perform

Real World Example of Objects & Classes



Exercise 1: Simulating infection spreads



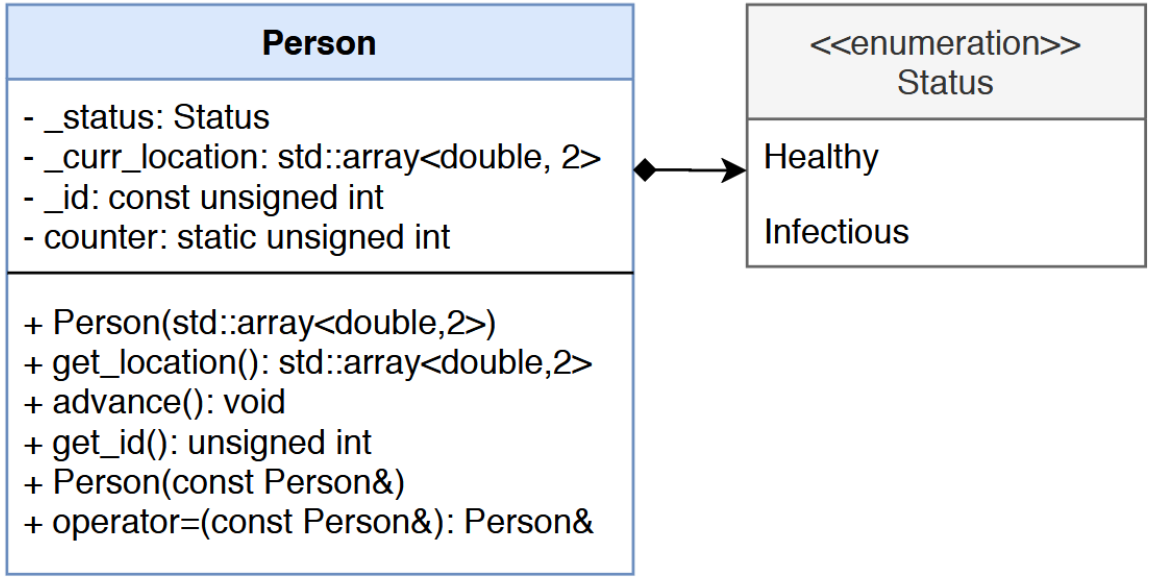
Person UML diagram

Let us draft a simulation of the spread of an infection in a population. 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 and Infectious), we can express this using an enumeration type or an enum class in C++
- 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

Exercise 1: Simulating infection spreads

1. Download **main.cpp, person.cpp, person.h, CMakeLists.txt** from the moodle.
2. Implement a constructor that initializes **_curr_location** and the unique **_id** of the object using the counter.
3. Implement the copy constructor. Since each person has a unique **_id**, do not copy a new id.
4. Implement the copy assignment operator



Exercise 2: Students and 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 ONLINE MUT (Online Management of University Tables), a system to manage student cards and exam registrations.

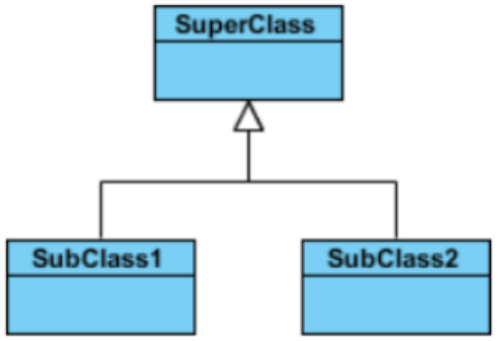
Exercise 2: Students and Courses

1. Download **main.cpp, Student.cpp, Student.h, StudentCard.cpp, StudentCard.h, Course.cpp, Course.h, CMakeLists.txt** from the moodle.
2. 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.
3. Implement **StudentCard** class. 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.
4. 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 it members.
   - A get_id() method.

Exercise 2: Students and Courses

4. Sketch a UML Class Diagram to visualize the classes of your code (e.g on https://apollon.ase.in.tum.de/ or https://www.drawio.com/)

Inheritance

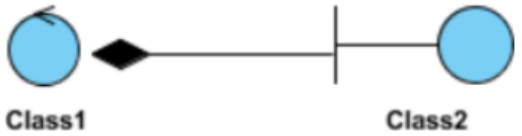


- Represents an “is-a” relationship
- An abstract class name is shown in italics

Aggregation

- Represents a “part of” relationship
- Class2 is part of Class1
- Objects of Class1 and Class2 have separate lifetimes

Composition



- A special type of aggregation where parts are destroyed when the whole is destroyed
- Object of Class2 live and die with Class1
- Class2 cannot stand by itself