Workshop 6 - Exercises

This week, we will practise some basic object-oriented modelling. If you are happy with your solution for the first question and ready to dig a little deeper, consider the two extension exercises.

Don't worry if you do not manage to complete all tasks within the workshop time. You can finish the sheet on your own time and sample solutions will be made available on Canvas before the next workshop.

1) Objects and Classes: Object-oriented modelling

Imagine you are a software engineer who works for a film production company. The boss has asked you to sketch out a computer program that can systematically reproduce the relevant information contained in a film script. This could be very useful for automatically creating digital storyboards in the future.

For now, your task is simply to devise an object-oriented model of a human character that describes its basic properties and actions. For example, every character in a film has a name and must be able to do certain things as described in a script – for instance walk, eat, drink, talk, etc.

Think about how a "Character" class might be structured here – what would be its variables, and what its member functions? Then go and create the class in a C++ program, where each action should trigger a simple text output (e.g., john.walk() -> "John is walking").

In your main class, you can now call the constructor to create a named character. Then, call the member functions of the character to assign important properties and make it do different things – like in a simple movie script. For example, make it introduce itself (e.g., it says its name and age).

2) Interacting with other objects (Extension)

For now, our human character is alone in the world and only able to do things in the abstract. You need to create more classes in your program to represent things in the scene that a human character can interact with. Note that the classes don't really need any properties of their own right now, but each should have a "char* getDescription()" member function to return a string for the script output (e.g., "a chair").

Adapt the program and the main function to create the following objects: A chair, a sandwich, and a refrigerator.

Now add additional member functions to your "Character" class to facilitate interaction with the objects in the scene (e.g., make John open the refrigerator, or eat the sandwich). How do the parameter lists of these member functions differ to the more generic functions we had before?

Tip: Within a class, you may specify multiple member functions of the same name if (and only if) their parameter lists are different (with respect to the number, type, and/or order of variables in the list). So, for instance, it is allowed to have a function eat() next to a function eat(char* food).

3) Interacting with another object of the same type (Extension)

For our computer-simulated scene, let's say we want to create a number of characters to interact with each other and the objects around them.

Let's create two characters for our scene: John and his daughter Holly. First, let them introduce themselves. Then, let them talk to each other ("John talks to Holly"). Then, John opens the refrigerator, takes out a sandwich, and gives this to Holly. Holly then eats the sandwich.

How do you need to extend your existing model to facilitate this? Adapt the program and change the main function to illustrate the interaction described above.

4) Calculating area of rectangle and triangle

Write a program that defines a shape class with a constructor that gives value to width and height. Then define two sub-classes triangle and rectangle, that calculate the area of the shape area (). In the main calculate area of triangle and rectangle separately.