Computer Lab #9.

Multiple Inheritance in C++

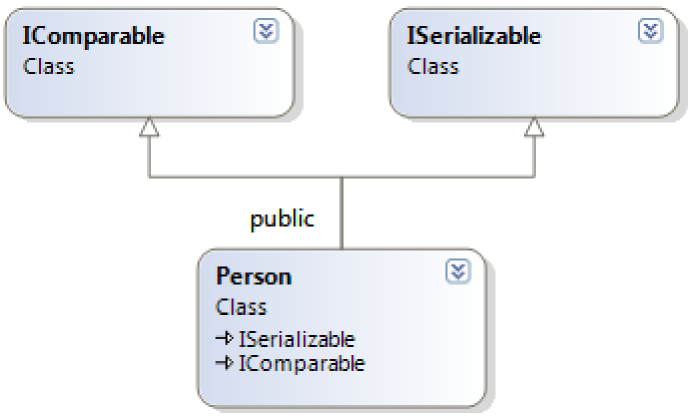
Multiple inheritance refers to a feature of some object-oriented programming languages in which a class can inherit behaviors and features from more than one superclass. Languages that support multiple inheritance include C++, Curl, Dylan, Eiffel, Logtalk, OCaml, Perl, Python, etc.

Multiple inheritance in C++ has two flavors, namely interface inheritance and implementation inheritance. Implementation inheritance is rarely used and can be rather problematic. At this computer lab you will deal with multiple interface inheritance, which is widely used. The use of pure virtual functions will enable you to specify only an interface, without implementation. As a rule of thumb, an (abstract) interface class should be compact, containing no more than several pure virtual functions. This way, you may define multiple specialized interfaces that a derived class can combine as needed.

Task 1

Download the source code archive from the Course Webpage. Add the downloaded files to your C++ project. Create the missing source files and add them to the project.

IComparable and *ISerializable* are abstract interface classes, the *Person* class is derived from them both (multiple interface inheritance is used).



Write implementation for missing member functions in ***Person.cpp***. The *Person* class should be able to save/load a *Person* object to file, and to compare itself with other *Person* objects.

a) Save, load and comparison functions are inherited from two abstract base classes: *ISerializable* and *IComparable*.

b) Loading and saving must be implemented using the built-in *ostream* and *istream* classes. Full code is given out; check declarations and implementations of the respective functions.

c) The *operator>* must be implemented based on the attribute *age*, that is the older person is ‘greater’ than the younger one. It is your task to write the implementation (see operator overloading).

Run the program and make sure that the first task is completed without errors (locate the place for the first breakpoint in *main.cpp*).

<u>Decomposition of the program:</u>

- *IComparable.h, ISerializable.h, Person.h*: class declaration headers;
- *Person.cpp*: implementation of class members declared in the header;
- *main.cpp*: the main program with testing code.

Task 2 (optional)

Assume that you are going to write a class with 15 sorting methods (class *Sorter*), and you want to publish only the compiled binary and not the source code.

a) Download the ***Sorter.h***, and create a ***Sorter.cpp*** file. Add both files to the project.

b) Write one-two sorting methods in the class *Sorter*. You can find declarations in ***Sorter.h***; your task is to write implementations in ***Sorter.cpp***. Chose the sorting algorithms at your option.

Run the program and make sure that the both tasks are completed, that is all instructions in ***main.cpp*** are executed successfully.

Decomposition of the program:

- *IComparable.h, ISerializable.h, Person.h, Sorter.h*: class declaration headers;
- *Person.cpp, Sorter.cpp*: implementation of class members declared in the header;
- *main.cpp*: the main program with testing code.