10. LABOR

TÖBBSZÖRÖS ÖRÖKLÉS

Általános információk

Az összes feladat hibátlan megoldására 1 iMSc pont jár. (Beadni mindenkinek kötelező mindet).

Kötelező feladatok

0. Interfész áttekintés: Serializable, Comparable

Az első két feladatban az interfészek egy hasznos, gyakorlati alkalmazásával ismerkedhetsz meg. A cél, hogy bármilyen nemabsztrakt osztályt felruházzunk két általános tulajdonsággal: **perzisztenciával** (*persistence*) (program terminálása után is megmarad az objektumok értéke) és **összehasonlíthatósággal**.

A perzisztenciát ebben az esetben szerializálással (*serialization*) szeretnénk megvalósítani (de lehetne akár adatbázisba mentés/betöltés is), azaz ki szeretnénk menteni az objektumaink állapotát egy szövegfájlba, melyből később visszatölthetjük értékeiket. Egy osztály akkor lesz szerializálható, ha megvalósítja a *Serializable* interfészt (azaz örököltet belőle):

serializable.h

```
class Serializable
{
public:
	// Beleírja az os-be a mentendő részeit
	virtual void serialize(std::ostream& os) const = 0;

	// Visszaállítja magát az is-ből
	virtual void deserialize(std::istream& is) = 0;
};
```

Az összehasonlíthatóságot pedig a *Comparable* interfész megvalósítása fogja biztosítani:

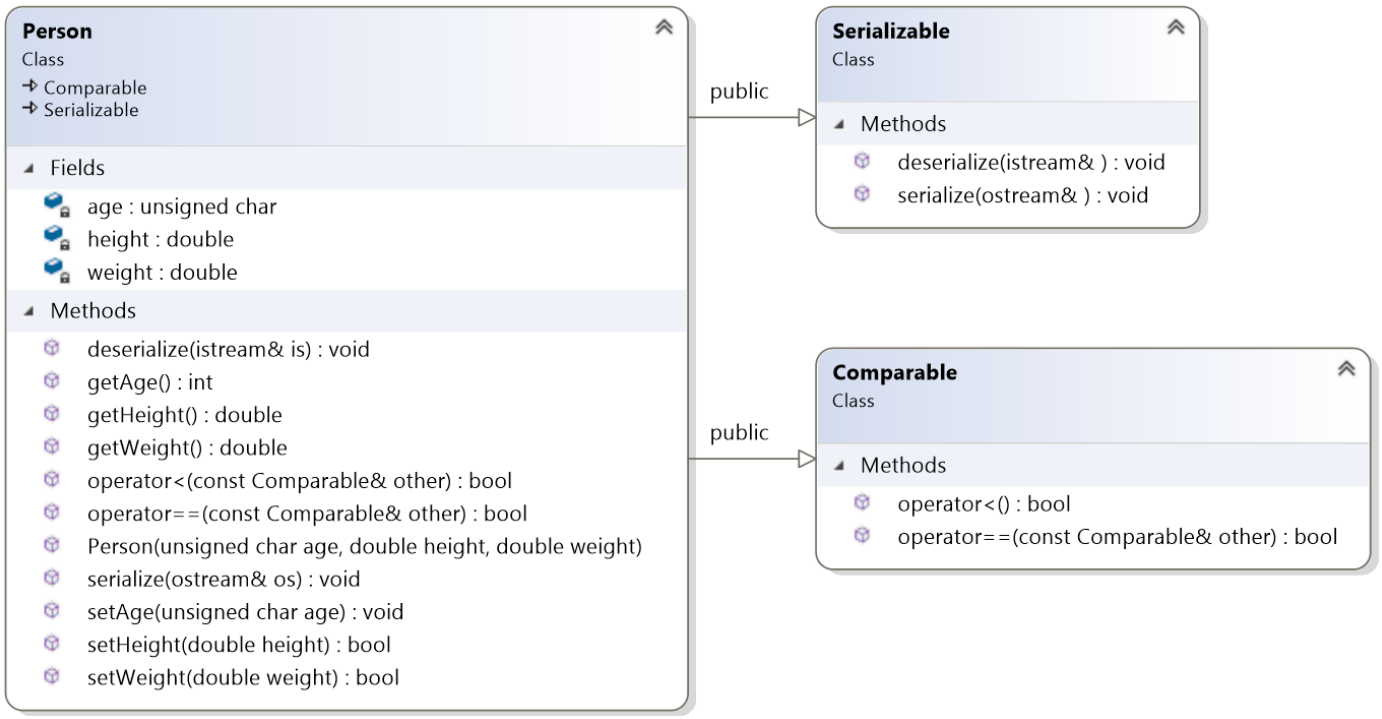
comparable.h

```
class Comparable
{
public:
	// Igazzal tér vissza, ha a két Comparable egyenlő (implementáció függő),
	// minden más esetben hamissal.
	virtual bool operator==(const Comparable& other) const = 0;

	// Igazzal tér vissza, ha a bal oldali Comparable kisebb (szintén
	// implementáció függő),
	// mint a jobb oldali (other), minden más esetben hamissal.
	virtual bool operator<(const Comparable& other) const = 0;
};
```

1. Interfész implementálás többszörös örökléssel: Person osztály UML alapján

Készíts egy *InterfacePractice* nevű programot, majd valósítsd meg benne a következő UML diagrammot:



1. ábra UML diagramm: Person, Serializable, Comparable

A *Person* osztály azáltal, hogy öröklődik a *Serializable* és *Comparable* absztrakt osztályokból, rendelkezni fog a perzisztencia és összehasonlíthatósági tulajdonságokkal:

class Person : public Comparable, public Serializable // fontos a sorrend

- A megoldáshoz használd fel az előző feladat interfészeit.
- Teszteléshez használd fel a mellékelt *interfaceTest.cpp* állományt!
- Az operator< implementálásakor az életkort vedd alapul.

Megjegyzés: A VS a generált UML diagrammon nem tünteti fel, hogy melyik függvény konstans tagfüggvény.

2. Rendező osztálykönyvtár: implementáció

Írj egy kb. 15 rendezőalgoritmusból álló statikus osztálykönyvtárt (*SorterLib solution*), amely tetszőleges típusú objektumokból álló tömböt képes rendezni. *(Ügyelj a projekt helyes létrehozására: Win32 Console Application -> Static library, de Precompiled header nélkül!)*

- Az egyes rendezőfüggvényeket a *Sorter* osztályban írd meg! Legyenek publikus láthatóságúak és statikusak.
- Az idő szűke miatt, elegendő csak egy rendező függvényt megvalósítani (legyen beszédes neve: derüljön ki, hogy melyik algoritmusról van szó, például:
  - `void SorterLibNS::Sorter::bubbleSort(Comparable** pItems, unsigned itemCount)`
- Az rendezendő tömböt vedd át *Comparable***-ként.
  - Ehhez a *SorterLib solution*-ben is szükséged van a *Comparable* osztályra, így át kell másolnod oda a *comparable.h*-t
  - Annak érdekében, hogy egyértelmű legyen, hogy melyik *Compare* implementációt használod majd a következő feladatban (az *InterfacePractice* folytatása), érdemes névtérbe szervezned a statikus osztálykönyvtárad összes osztályát. Ennek a névtérnek a neve legyen *SorterLibNS*.
- A *SorterLib* projektben is törekedj a deklarációk és definíciók dekompozíciójára: *comparable.h, sorter.h, sorter.cpp*

3. Rendező osztálykönyvtár: teszt

Az előbb megírt statikus osztálykönyvtár rendezőfüggvényének segítségével rendezd az első feladatban megírt *Person** típusú objektumok tömbjét (`Person* people[PEOPLE_COUNT];`)

Emlékeztető: külső, statikus programkönyvtár használata

- Jobb klikk arra a **projektre**, amiben használni szeretnéd a könyvtárat (most *InterfacePractice*)
- Lenyíló menüben válaszd a *Properties*-t (legalul)
- Előugró ablakban bal oldalt
  - *Configuration Properties ->*
    - *C/C++-> General -> Additional Include Directories*
      - Add hozzá azt a *SorterLib* solution *mappájának elérési útvonalát*
    - *Linker -> Input -> Additional Dependencies*
      - A *SorterLib* buildelése után meg tudod adni az előállt *SorterLib.lib elérési útvonalát* (Debug vagy Release mappában lesz, attól függ hogyan fordítottad)
- Ezek után az *InterfacePractice*-ben tudsz *SorterLib*-beli headert includeolni a következőképp: `#include "SorterLib/comparable.h"`
  - Itt a *SorterLib* már a solution-ön belüli *SorterLib* projekt mappája!

További teendők

- A teszteléshez át kell írnod a *Person* osztályt, hogy a *SorterLibNS::Comparable*-ből öröklődjön
  - `#include "SorterLib/comparable.h"`
  - `class Person : public SorterLibNS::Comparable, public Serializable`
- Mikor átadod a rendező függvénynek a people tömböt szerializálás előtt, castold át SorterLibNS::Comparable-re:
  - ```
    using namespace SorterLibNS;
    //...
    Sorter::bubbleSort((Comparable**)people, PEOPLE_COUNT);
    ```
  - (cast: nagy vonalakban Person**-ból Comparable**-ként kezelhető változót készít)

Ha mindent jól csináltál, listázáskor a people elemei életkor szerint rendezve jelennek meg.

4. Saver, Loader

Vásároltál egy *PersistenceAPI* nevű osztálykönyvtárat, hogy leegyszerűsítsd a meglévő programod. Ezt a *PersistenceAPI* mappában találod. Írd át a programod úgy, hogy a szerializálást és deszerializálást azon keresztül végezze! (Ha a libet nem tudná a linker használni (eltérő fordítóprogram vagy verzió miatt), csak abban az esetben fordítsd újra!)

Gyakorlófeladatok

1. Konstruktorból hívott virtuális függvény
2. Adózó és beteg alkalmazott
3. Hibák a többszörös öröklésben
    ```