10. LABOR

TÖBBSZÖRÖS ÖRÖKLÉS

Hallgatónak: általános információk

Az összes feladat hibátlan megoldására 1 iMSc pont jár.

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:

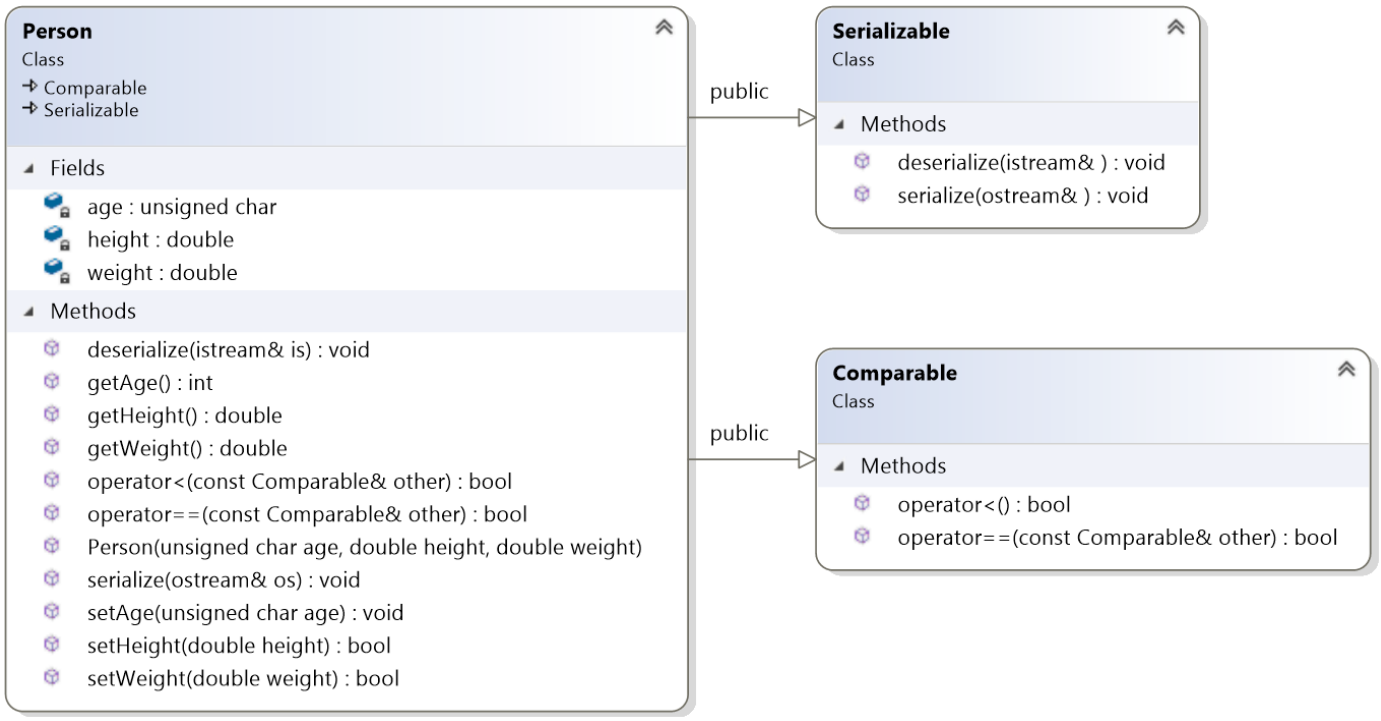
comparable.h

```
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

Vizsgáld meg a kapott *InterfacePractice* kiindulási csomagot, és vesd össze az alábbi UML diagrammal.



1. ábra UML diagramm: Person, Serializable, Comparable

1.a: Serializable

A *Person* osztály azáltal, hogy öröklődik a *Serializable* absztrakt osztályból, rendelkezni fog a perzisztencia tulajdonsággal:

class Person : public Serializable // fontos a sorrend

- A megoldáshoz felhasználjuk az előző feladat interfészét.
- A Person osztályt úgy szerializáljuk, hogy tab karakterekkel elválasztva írjuk ki a streamre a paramétereit. Vizsgáld meg és értsd meg a serialize és deserialize függvényelet – ezek a Serializable ősosztályban tisztán virtuális függvényekként vannak definiálva.
- Csak a Serializable interfészből örökölt függvényeket felhasználva készítsd el az `operator<<` implementációját! Változtasd meg úgy az operátor paramétereit, hogy mostantól bármilyen, Serializable leszármazott osztályt ki tudjon írni!
- Teszteléshez használd fel a mellékelt *interfaceTest.cpp* állományt!

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

1.b: Comparable

A *Person* osztály azáltal, hogy öröklődik a *Comparable* absztrakt osztályból is, rendelkezni fog az összehasonlíthatósági tulajdonságokkal:

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

- Írd meg a leszármazottban a Comparable absztrakt ősosztályból örökölt tisztán virtuális függvények felüldefiniált változatát, hogy össze lehessen hasonlítani két Persont. Amíg ezt nem írod meg, a Person is absztrakt osztály marad.
    - Az operator< implementálásakor az életkort vedd alapul.
- A megoldáshoz használd fel az előző feladat interfészeit.
- Teszteléshez használd fel a mellékelt *interfaceTest.cpp* állományt!
    - Az összehasonlítás teszteléséhez kommentezd ki a megfelelő sorokat

2. 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!

- Az interfaceTest osztályban írd ki a people tömb elemeit a vásárolt osztálykönyvtárban a "people.txt" fájlba.
- Értsd meg, hogy a Saver/Loader osztálykönyvtár nem ismeri az általunk létrehozott Person típust és tulajdonságati: csupán az interfészből való leszármazást feltételezi.
- iMSC: Próbáld ki a visszatöltést is a Loader osztályon keresztül egy új tömbbe.

3. Dog osztály

A fentiekre támaszkodva, **önállóan** készíts el egy Dog osztályt: a kutyának neve (string) és életkora van.

- A kutyákat is lehessen szerializálni és deszerializálni. Ezt a Serializable interfész megvalósításával elkészítve, a Saver/Loader páros, illetve az operator<< is automatikusan működni fog.
- Kutya példányokat is perzisztálj
- A kutyákat is lehessen összehasonlítani, szintén életkor alapján.

4. iMSC feladat: Rendező algoritmus

Készíts egy függvényt (lehet pl. egy Sorter osztály statikus tagfüggvénye), amely rendezni tud egy paraméterül kapott, tetszőleges osztályból álló tömböt, feltéve, hogy az adott osztály Comparable leszármazott. Bármilyen rendezési algoritmust használhatsz (pl. bubblesort).

Teszteld a megoldásodat a Person-ok illetve Dog-ok tömbjén!

Gyakorlófeladatok

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