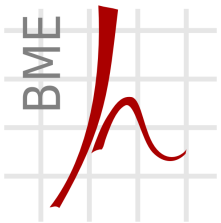
STL gyakorlat

C++



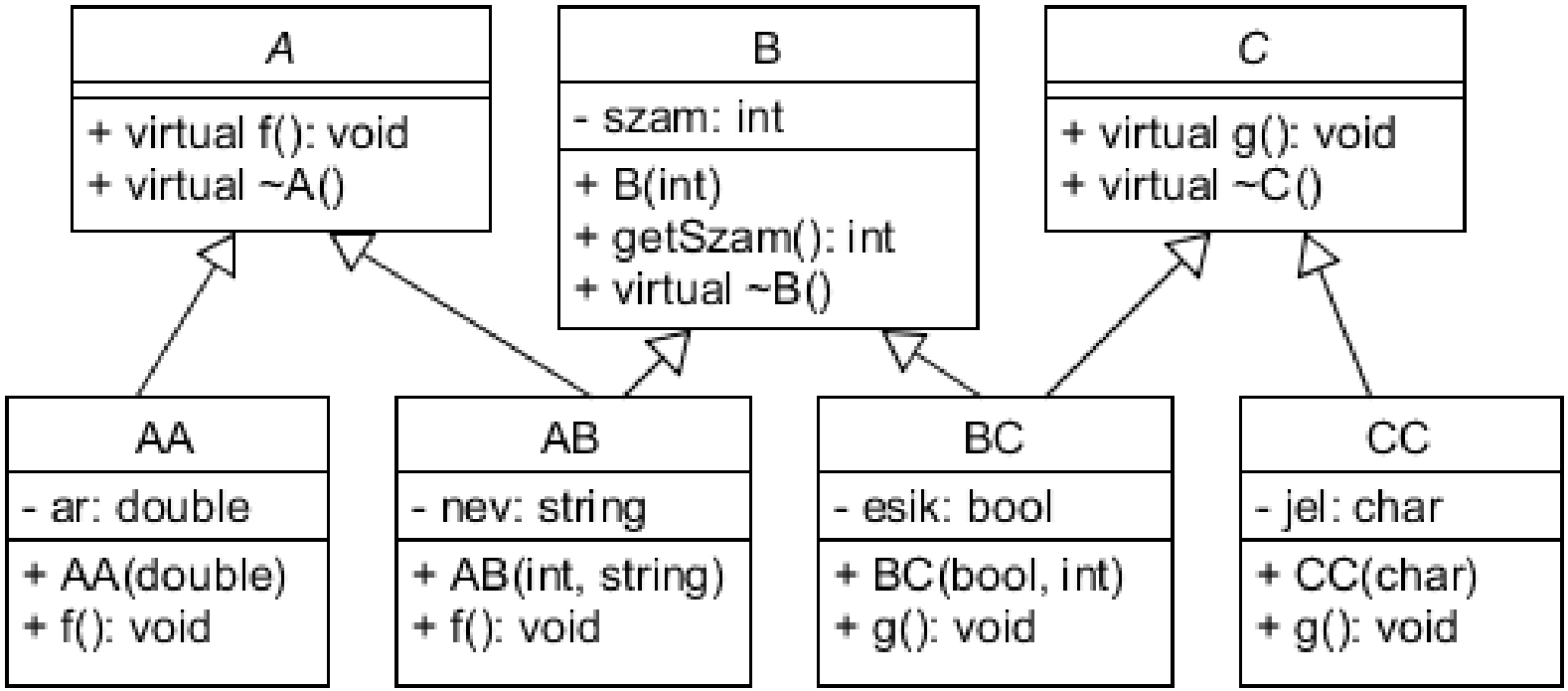
Híradástechnikai Tanszék

Izsó Tamás

2016. május 9.

Komponensek kapcsolata

Deklarálja az alábbi osztálydiagramon szereplő osztályok közül az A, AA és AB osztályokat! A konstruktorokat valósítsa is meg! A tagváltozók kezdeti értékét a konstruktorparaméterek adják. (3p)



KZH 4

```
struct A {
    virtual void f() = 0;
    virtual ~A();
};
```

KZH 4

```
struct A {
    virtual void f() = 0;
    virtual ~A();
};

class AA : public A {
    double ar;
public:
    AA(double d) :ar(d) {}
    void f();
};
```

KZH 4

```
struct A {
    virtual void f() = 0;
    virtual ~A();
};

class AA : public A {
    double ar;
public:
    AA(double d) :ar(d) {}
    void f();
};

class AB : public A, public B {
    string nev;
public:
    AB(int i, string s) :B(i), nev(s) {}
    void f();
}
```

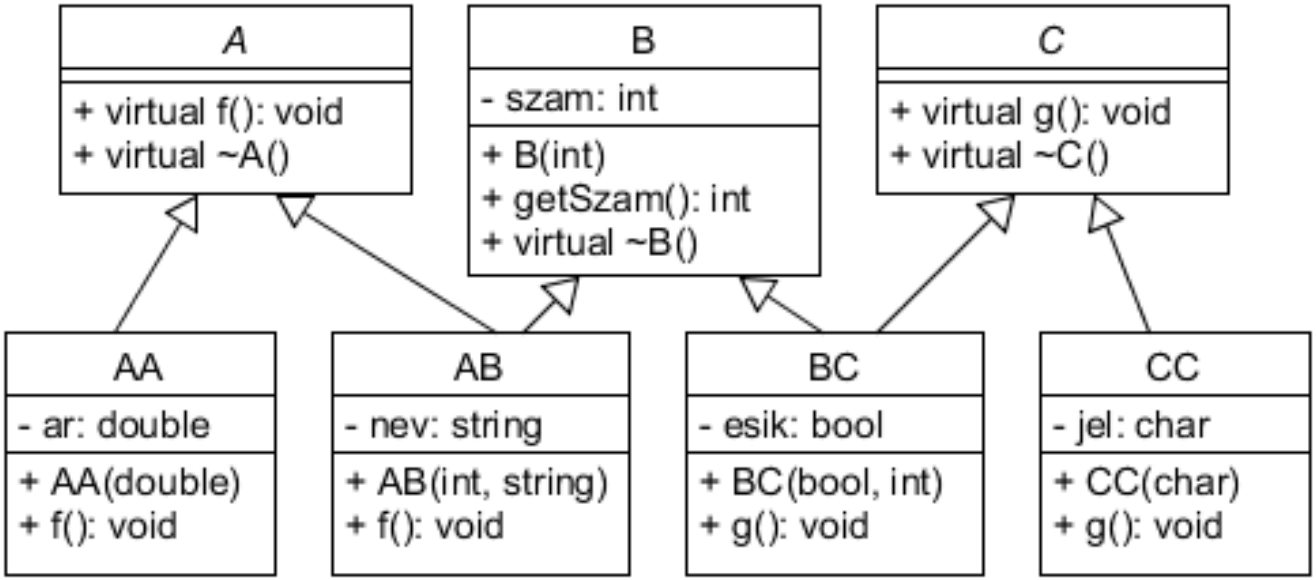
1. b

Adjon példát egy adatfolyam extractor deklarációjára! (1p)

```
std::istream& operator>>(std::istream&, int&);
```

1. c

Az a) részfeladat összes osztálya rendelkezésére áll. Az std::list sablon felhasználásával hozzon létre egy olyan objektumot (data), amivel "tárolni" tud AB és BC típusú objektumpéldányokat is. Ezután a dinamikus memóriában hozzon létre 500 db AB példányt és egy BC példányt úgy, hogy később elérje azokat! Ügyeljen a konstruktorok paraméterezésére! (2p)



1. c

```
std::list<B*> data;
```

1. c

```
std::list<B*> data;

for (int i = 0; i < 500; i++)
    data.push_back(new AB(i, "C++11"));
data.push_back(new BC(false, 123));
```

1. d

Mit ír ki az alábbi programrészlet a kimenetre? (1p)

```
struct AA {
    virtual void g() { std::cout << "AA::g";}
};
struct BB : public AA {
    void g() { std::cout << "BB::g";}
};
int main { BB b; b.g(); return 0; }
```

1. d

Mit ír ki az alábbi programrészlet a kimenetre? (1p)

```
struct AA {
    virtual void g() { std::cout << "AA::g";}
};
struct BB : public AA {
    void g() { std::cout << "BB::g";}
};
int main { BB b; b.g(); return 0; }
```

Kiírás: BB::g

1. e

Mi a hiba az alábbi programrészletben? (1p)

```
class A { A() {} ~A() {} };
{ A a; }
```

1. e

Mi a hiba az alábbi programrészletben? (1p)

```
class A { A() {} ~A() {} };
{ A a; }
```

A összes tagfüggvénye privát.

Singleton osztály (érdekesség)

Van-e értelme olyan osztálynak, aminek minden konstruktora privát?

```
class Singleton {
public:
    static Singleton* getInstance( ) {
       if( instance == 0 ) instance = new Singleton;
       return instance;
     }
    ~Singleton( ) {...};
private:
    Singleton( ) {...};
    static Singleton* instance;
};

Singleton* Singleton::instance=0;
```

1. f

Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

A `throw` utasítás értéket dob, ezért másoló konstruktort hívhat.	
A konstruktor előbb hajtja vére a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	
Pointerek kasztolásánál a cím sohasem változik meg.	
A `dynamic_cast` mindig sikerül.	

1. f

Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

A `throw` utasítás értéket dob, ezért másoló konstruktort hívhat.	igaz
A konstruktor előbb hajtja vére a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	
Pointerek kasztolásánál a cím sohasem változik meg.	
A `dynamic_cast` mindig sikerül.	

1. f

Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

A `throw` utasítás értéket dob, ezért másoló konstruktort hívhat.	igaz
A konstruktor előbb hajtja vére a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	hamis
Pointerek kasztolásánál a cím sohasem változik meg.	
A `dynamic_cast` mindig sikerül.	

1. f

Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

A `throw` utasítás értéket dob, ezért másoló konstruktort hívhat.	igaz
A konstruktor előbb hajtja vére a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	hamis
Pointerek kasztolásánál a cím sohasem változik meg.	hamis
A `dynamic_cast` mindig sikerül.	

1. f

Jelölje, hogy igaz (I) vagy hamis (H)! (2p)

A `throw` utasítás értéket dob, ezért másoló konstruktort hívhat.	igaz
A konstruktor előbb hajtja vére a programozott törzset, és csak ezután hívja a tartalmazott objektumok konstruktorait.	hamis
Pointerek kasztolásánál a cím sohasem változik meg.	hamis
A `dynamic_cast` mindig sikerül.	hamis

2. a

Készítsen **adapter sablont** (`PhyTomb`), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van at() tagfüggvénye. Egy N elemű `PhyTomb` elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a $-N$ pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként $N = 3$, a tároló pedig az std::vector legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az operator[]! Példa a használatra: (4p)

2. a

Készítsen adapter sablont (`PhyTomb`), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van at() tagfüggvénye. Egy N elemű `PhyTomb` elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a $-N$ pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként $N = 3$, a tároló pedig az std::vector legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! **A sorozattároló minden tagfüggvénye legyen elérhető**, kivéve az operator[]! Példa a használatra: (4p)

2. a

Készítsen adapter sablont (`PhyTomb`), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van at() tagfüggvénye. Egy N elemű `PhyTomb` elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a $-N$ pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként $N = 3$, a tároló pedig az std::vector legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! A sorozattároló minden tagfüggvénye legyen elérhető, **kivéve az operator[]**! Példa a használatra: (4p)

2. a

Készítsen adapter sablont (`PhyTomb`), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van at() tagfüggvénye. Egy N elemű `PhyTomb` elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a $-N$ pedig az első elemet. Úgy alakítsa ki a sablont, hogy **alapértelmezésként** $N = 3$, a tároló pedig az std::vector legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az operator[]! Példa a használatra: (4p)

2. a

Készítsen adapter sablont (`PhyTomb`), ami minden olyan szabványos sorozattárolóra alkalmazható, melynek van at() tagfüggvénye. Egy N elemű `PhyTomb` elemei pozitív és negatív indexértékkel is elérhetők. Míg a pozitív indexértékek a szokásos elérést eredményezik, addig a negatív indexek a tömb végétől haladnak visszafelé. A -1 az utolsó elemet adja, a $-N$ pedig az első elemet. Úgy alakítsa ki a sablont, hogy alapértelmezésként $N = 3$, **a tároló pedig az std::vector** legyen! Ügyeljen arra, hogy a sorozattárolókra jellemző konstruktorok elérhetőek legyenek! A sorozattároló minden tagfüggvénye legyen elérhető, kivéve az operator[]! Példa a használatra: (4p)

2. a

Használata:

```
PhyTomb<int> t3(3);     // 3 elemű int tömb
t3.at(1) = 1;           // első eleme
std::cout << t3.at(-3); // ez is az első
t3.at(2) = 3;           // utolsó eleme
std::cout <<< t3.at(-1); // utolsó (3.) elem
```

2. a

```
template <typename T, int S = 3, class C = std::vector<T> >
class PhyTomb : public C {
    T& operator[](int ix);
    T operator[](int ix) const;
 public:
    PhyTomb(size_t n = S, const T& value = T()) : C(n, value) {}
    template<typename Iter>
    PhyTomb(Iter first, Iter last) : C(first, last) {}
    T& at(int ix) { if (ix < 0) ix += S; return C::at(ix); }
    T at(int ix) const { if (ix < 0) ix += S; return C::at(ix); }
};
```

2. b

Hozzon létre az elkészített adapter és az std::deque felhasználásával egy 40 elemű long tömböt!

```
PhyTomb<long, 40, std::deque<long> > p40;
```

2. c

Írjon C++ függvénysablont (count_if), ami iterátorokkal megadott adatsorozatban megszámolja azokat az elemeket, amire a paraméterként átadott predikátum igaz értéket ad! A függvény első két paramétere két iterátor, amivel a szokásos módon megadjuk a jobbról nyílt intervallumot. A függvény 3. paramétere pedig egy predikátum, ami egy egyparaméteres függvény vagy függvényobjektum. Ha jól oldja meg a feladatot, akkor az alábbi kódrészlet lefutása után az eredmény 2.

```
bool negativ(int a) { return a < 0; }
int sorozat[] = { 1, 4, 9, -16, 25, 3, -72, 100, 3};
// a sorozat
int eredmeny = count_if (sorozat, sorozat+9, negativ);
```

2. c megoldás

```
template <class I, class P>
int count_if(I first, I last, P pred) {
    int cnt=0;
    while (first != last)
        if (pred(*first++)) cnt++;
    return cnt;
}
```

2. d megoldás

Készítsen olyan függvényobjektum sablont, ami a count_if sablonnal felhasználva alkalmas az olyan elemek megszámolására, amelyek kisebbek a függvényobjektum konstruktorában megadott értéknél!

```
template <class T>
class KisebbMint {
    T ref;
public:
    KisebbMint(const T& a) : ref(a) {}
    bool operator()(const T& a) const { return a < ref; }
};
```

2. e megoldás

A részfeladatok eredményeit felhasználva írjon kódrészletet, ami a b) részfeladatban létrehozott tömbben megszámolja a negatív elemeket!

```
std::cout << count_if(p40.begin(), p40.end(), KisebbMint<long>(0));
```

3. feladat

A Film osztályban házimozinkhoz tárolunk adatokat.

```
class Film {
    std::string cim;
    int polc;
public:
    Film(const std::string& n = "", int p = 0);
    int getPolc() const;
    std::string getCim() const;
    void setPolc(int);
    void setCim(const std::string&);
    virtual ~Film();
};
```

Adott továbbá a Serializable osztály.

```
struct Serializable {
    virtual void write(std::ostream& os) const = 0;
    virtual void read(std::istream& is) = 0;
    virtual ~Serializable() {}
};
```

3a. feladat

A fenti osztályok felhasználásával, de azok módosítása nélkül hozzon létre a Film osztállyal kompatibilis, perzisztens PFilm osztályt! Megoldásában vegye figyelembe, hogy a címben szóköz is lehet! Az elmentett állapot visszatöltésekor az osztály végezzen ellenőrzést, hogy jó adatokat kap-e, azonban nem kell bombabiztos megoldás! Hiba esetén dobjon std::out_of_range kivételt!

3a. feladat megoldás

```
struct PFilm : public Film, public Serializable {
    PFilm(const std::string& n = "", int p = 0) : Film(n, p) {}
    void write(std::ostream& os) const {
        os << "FILM" << std::endl;
        os << getCim() << std::endl;
        os << getPolc() << std::endl;
    }
    void read(std::istream& is) {
        std::string line;
        (is >> line).ignore(1);
        if (line != "FILM")
            throw std::out_of_range("Film::Read?");
        std::getline(is, line);.
        setCim(line);
        int a;
        (is >> a).ignore(1);
        setPolc(a);
    }
};
```

3b. feladat

Egy rövid kódrészletben hozzon létre egy PFilm példányt a kedvenc filmcímével. Mentse ki az objektum adatait a szabványos kimenetre! Kommentben adja meg, hogy mit írt ki! Jelölje a nem látható karaktereket is!

3b. feladat

Egy rövid kódrészletben hozzon létre egy PFilm példányt a kedvenc filmcímével. Mentse ki az objektum adatait a szabványos kimenetre! Kommentben adja meg, hogy mit írt ki! Jelölje a nem látható karaktereket is!

```
PFilm f3("A destruktor bosszuja", 2)
f3.write(std::cout);
// FILM\nA destruktor bosszuja\n2\n
```

3c. feladat

Tételezze fel, hogy rendelkezésére áll a gyakorlaton elkészített PKomplex osztály is, ami szintén a Serializable osztály segítségével valósítja meg a perzisztens viselkedést! Egészítse ki megoldását, hogy az alábbi kódrészlet az elvárásoknak megfelelően működjön!

```
PFilm f1("Vak asszony visszanez", 1), f2("A destruktor bosszuja", 2);
PKomplex k1(2, 4);
stringstream ss;
f1.write(ss);
k1.write(ss);
f2.write(ss);
PFilm nf1, nf2;
PKomplex nk1;
ss >> nf1 >> nk1 >> nf2;
```

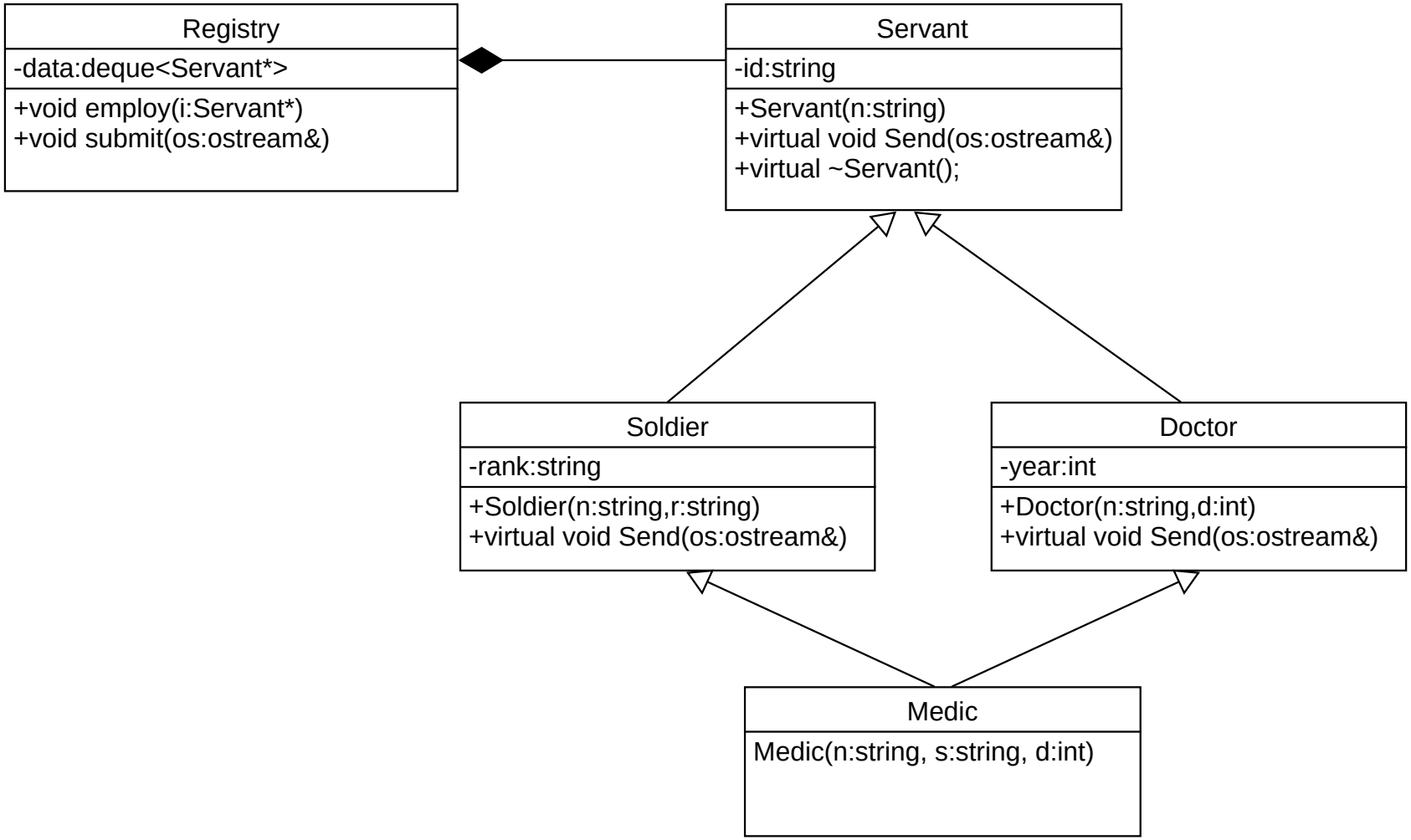
3c. feladat megoldása

```
istream& operator>>(istream& is, Serializable& s) {
        s.read(is);
        return is;
}
```

4. feladat

A burkus király egységesíteni akarja a közalkalmazotti nyilvántartásokat. Egységes rendszerben (Registry) kell tárolni a közalkalmazottakat (Servant), akik az első verzióban katonák (Soldier) és az orvosok (Doctor) lehetnek, de később további szakmák dolgozóit (bírók, tanárok, stb) is kezelni kell tudni. A modell adjon támogatást arra, hogy vannak orvosok, akik egyben katonák is (Medic). A közalkalmazottaknak kiíratható (send) az azonosítója (id, string), a katonáknak a rendfokozata (rank, string), az orvosoknak pedig a diploma megszerzésének éve (year, int). A Medic-nek azonosítója, rendfokozata és diplomaszerzési éve is van, és kezelhető katonaként és orvosként is. Kiíratáskor a Medic minden adata kiíródik, nem baj, ha többször is. A kiíratást minden tárolt típusnál a függvényparaméterként megadott ostream-re lehet kérni. A Registry rendszerbe fel lehessen venni új közalkalmazottat (employ), illetve paraméterben megadott ostream-re ki lehessen írni a meglevő közalkalmazottak minden tárolt adatát (submit). Ha a Registry rendszer megsemmisül, a benne tárolt adatok is elvesznek.

4a UML ábra


Registry
-data:deque<Servant*>
+void employ(i:Servant*)
+void submit(os:ostream&)
Servant
-id:string
+Servant(n:string)
+virtual void Send(os:ostream&)
+virtual ~Servant();
Soldier
-rank:string
+Soldier(n:string,r:string)
+virtual void Send(os:ostream&)
Doctor
-year:int
+Doctor(n:string,d:int)
+virtual void Send(os:ostream&)
Medic
Medic(n:string, s:string, d:int)

4a. feladat

```
class Servant {
  string id;
public:
  Servant(string n) : id(n) {}
  virtual void send(ostream& os) const;
  virtual ~Servant();
};

class Doctor: virtual public Servant {
  int year;
public:
  Doctor(string n, int d) : Servant(n), year(d) {}
  virtual void send(ostream& os) const;
};

class Soldier: virtual public Servant {
  string rank;
public:
  Doctor(string n, string r) : Servant(n), rank(r) {}
  virtual void send(ostream& os) const;
};

class Medic: public Soldier, public Doctor {
public:
  Medic(string n, string s, int d) : Servant(n), Soldier(n,s), Doctor(n,d) {}
  virtual void send(ostream& os) const;
};
```

4a. feladat

```
class Registry{
    vector<Servant*> data;
public:
    void employ(Servant* i);
    void submit(ostream& os) const;
};
```

4a. feladat

```
void Registry::employ(Servant* s) {
  data.push_back(s);
}
void Registry::submit(ostream& os) {
  for (int i = 0; i < data.size(); i++) data[i]->send(os);
}
void Servant::send(ostream& os) {
  os << id;
}
void Doctor::send(ostream& os) {
  Servant::send(os);
  os << "␣" << year;
}
void Medic::send(ostream& os) {
  Soldier::send(os);
  os << "␣";
  Doctor::send(os);
}
```