Választott feladat

```
Vonatjegy

Tervezze meg egy vonatjegy eladó rendszer egyszerűsített objektummodelljét, majd valósítsa azt meg! A vonatjegy a feladatban mindig jegyet és helyjegyet jelent együtt. Így egy jegyen minimum a következőket kell feltüntetni:      vonatszám, kocsiszám, hely     indulási állomás, indulási idő     érkezési állomás, érkezési idő

A rendszerrel minimum a következő műveleteket kívánjuk

elvégezni:     vonatok felvétele     jegy kiadása

A rendszer később lehet bővebb funkcionalitású (pl. késések kezelése, vonat törlése, menetrend stb.), ezért nagyon fontos, hogy jól határozza meg az objektumokat és azok felelősségét.

Valósítsa meg a jeggyel végezhető összes értelmes műveletet operátor átdefiniálással (overload), de nem kell ragaszkodni az összes operátor átdefiniálásához! A megoldáshoz ne használjon STL tárolót!
```

Feladatspecifikáció

A programban lehetőség kell legyen új vonat hozzáadására. *Ennek formátuma:*

[Vonatszám - azonosító]

[Indulási állomás - szöveg] [Indulási idő - időpont]

[Érkezési állomás - szöveg] [Érkezési idő - időpont]

A programban lehetőség kell legyen jegy kiadására. Azonban többféle jegyet is lehet vásárolni:

- Teljesárú jegy / 1-es azonosítójú
- Diákjegy – kedvezmény / 2-es azonosítójú
- Gyűjtőjegy – többször érvényesíthető / 3-as azonosítójú *Ennek formátuma:*

[Jegy típusa – a fenti azonosítók valamelyike]

[Vonatszám – a vonat azonosítója, ahova a jegy szól]

[Kocsiszám]

[Helyszám]

[Jegytípus specifikus paraméterek – pl. gyűjtőjegynél a darabszám] Ha a megadott vonaton foglalt a hely, akkor kivétel keletkezik.

A programban lehetőség kell legyen a jegyek érvényesítésére. Különböző jegyeknél különböző kritériumoknak kell teljesülniük:

- Teljesárú jegy – csak személyigazolvány kell
- Diákjegy – az érvényesítő utasnak diáknak kell lennie
- Gyűjtőjegy – még nem fogyott el az összes

A jegy érvényesítés úgy történik, hogy a vonatszám, kocsiszám és a helyszám alapján megkeressük a jegyet, majd az utas paraméterrel lefuttatjuk az érvényesítő függvényt. Amennyiben valamelyik kritérium nem teljesült, úgy kivétel keletkezik.

1. A PROGRAM BEMENETEI

A *vonatok.txt* és a *jegyek.txt* fájlok a fent definiált formátumokban. A program induláskor ezekből a fájlokból tölti be az aktuális állapotot.

2. A PROGRAM KIMENETE

A futás során a menürendszer segítségével változhat az állapot (tudunk vonatot, jegyet hozzáadni). Ezeket a változásokat a program elmenti a kilépésnél. Az érvényesített jegyek nem mentődnek el újra.

3. MENÜRENDSZER

- *Vonat hozzáadása*: a fent definiált formátumban
- *Jegy kiadása*: a fent definiált formátumban
- *Jegy érvényesítése*: itt a program kéri a vonatszámot, kocsiszámot és a helyszámot, valamint az utasnak az adatait (van-e személyigazolványa, diákigazolványa).
- *Kilépés / mentés*

Pontosított feladatspecifikáció

A feladat egy vonatjegy eladó rendszer megvalósítása.

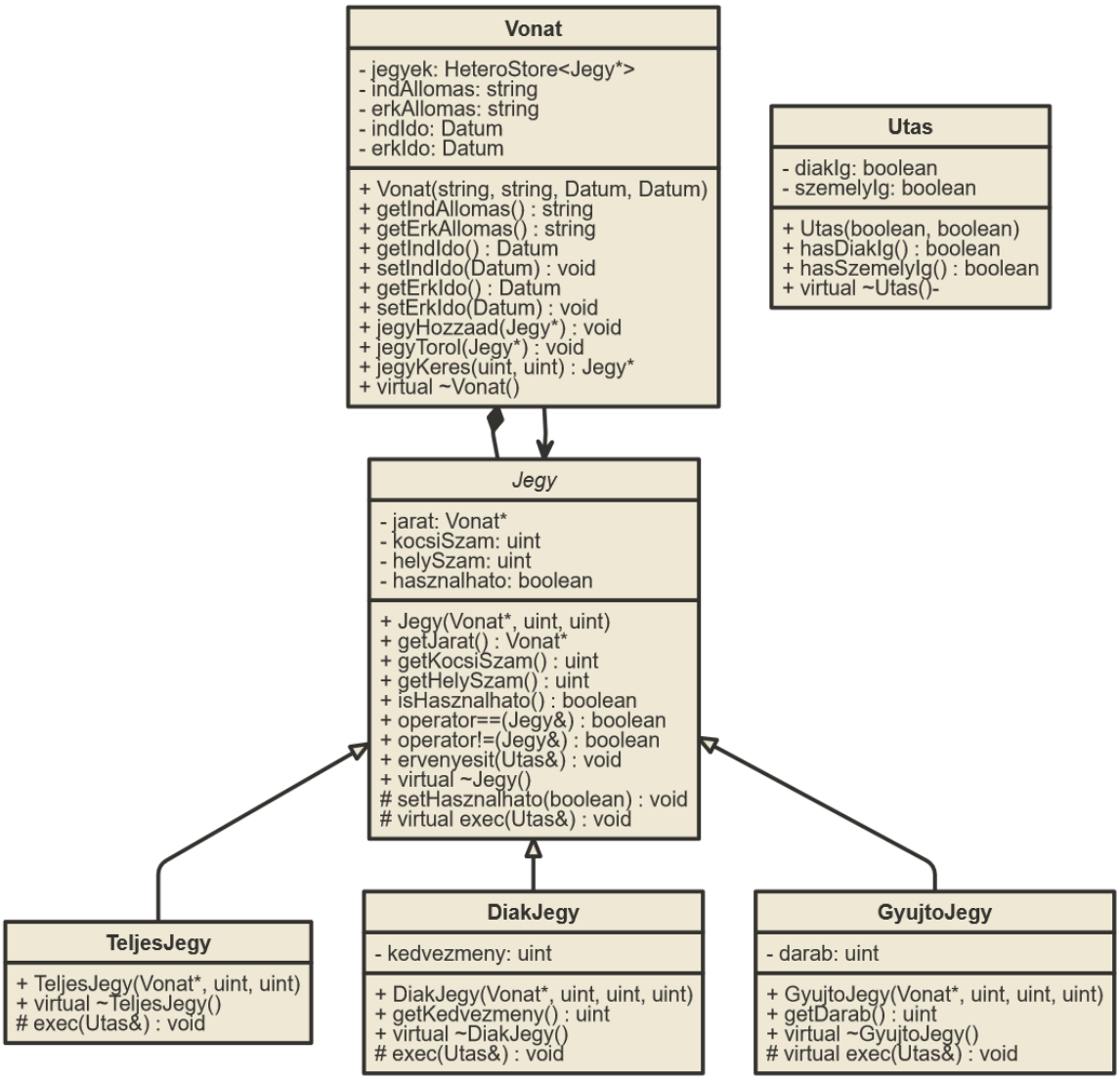
A program a vonatok és a jegyek adatait dinamikusan allokált memóriaterületen tárolja. A jegyeket egy heterogén kollekcióban fogom tárolni az adott vonat egy adattagjaként. A program főbb osztályai a Jegy osztály (amely egy absztrakt osztály, belőle lehet származtatni a konkrét jegytípusokat), valamint a Vonat osztály.

A program nem ellenőrzi a fájlokból beolvasott adatokat, ezeket helyesnek feltételezi. Az adatokat a menürendszerrel lehet módosítani, ebben az esetben lesz hitelesítés (pl. hogy egy helyen ne üljenek 2-en), így csak helyes adatok kerülnek be a fájlokba.

Terv

A feladat az objektumok és a tesztprogram megtervezését igényli.

Objektum terv



Algoritmusok

- **Jegyek érvényesítése:**
  if !isHasznalhato() then
  exception NEM_HASZNALHATO else
  kritériumok ellenőrzése (exec függvény)
- **Kritériumok ellenőrzésére példa: (diákjegy esetén)** if !utas.hasSzemelyIg() or !utas.hasDiakIg() then
  exception NEM_AZONOSITHATO else
- **Kritériumok ellenőrzése másik példa: (gyűjtőjegy esetén)**
  if !utas.hasSzemelyIg() then
  exception NEM_AZONOSITHATO else

```
        darab = darab – 1;  if(darab
== 0) then
                setHasznalhato(false)
```

- **A menüt egy állapotgép fogja kezelni!**

Megvalósítás

A feladat megoldása a tervezésben előírt osztályokon kívül a HeteroStore, a Datum és a Menu osztályok megvalósítását igényelte. Azonban a tervezéshez képest annyi változás volt, hogy minden adattaghoz írtam setter metódusokat, valamint megírtam a beolvasó és kiíró operátorokat. Továbbá írtam a vonathoz kiíró és beolvasó metódusokat is.

A program több állományra van bontva az osztályok szerint. A NHF3-hoz képest több tesztesetet írtam, hogy a program összes függvénye le legyen tesztelve.

A továbbiakban bemutatom a fontosabb interfészeket és algoritmusokat a program forrása alapján generált dokumentáció alapján.

Jegy osztályok

Az absztrakt jegy osztály a konkrét jegyek közös tulajdonságait, metódusait tartalmazza. Nem lehet példányosítani. Az exec metódus a kritériumokat teszteli.

Publikus tagfüggvények

	Jegy (**Vonat** *j=NULL, uint ksz=0, uint hsz=0)
Vonat *	**getJarat** () const
void	**setJarat** (**Vonat** *val)
uint	**getKocsiSzam** () const
void	**setKocsiSzam** (uint val)
uint	**getHelySzam** () const
void	**setHelySzam** (uint val)
bool	**isHasznalhato** () const
void	**ervenyesit** (const **Utas** &utas)
bool	**operator==** (const **Jegy** &masik) const
virtual void	**kiir** (std::ostream &os) const =0
virtual	**~Jegy** () virtuális destruktor

Védett tagfüggvények

void	**setHasznalhato** (bool val)
virtual void	**exec** (const **Utas** &utas)=0

A TeljesJegy osztály a teljesárú jegy megvalósítását tartalmazza. Az érvényesítéshez csak személyigazolvány szükséges.

Publikus tagfüggvények

	TeljesJegy (**Vonat** *j=NULL, uint ksz=0, uint hsz=0)
void	**kiir** (std::ostream &os) const

▸ Publikus tagfüggvények a(z) **Jegy** osztályból származnak

Védett tagfüggvények

void	**exec** (const **Utas** &utas)

▸ Védett tagfüggvények a(z) **Jegy** osztályból származnak

A DiakJegy osztály a diákjegy megvalósítását tartalmazza. Az érvényesítéshez személyigazolvány és diákigazolvány is szükséges.

Publikus tagfüggvények

	DiakJegy (**Vonat** *j=NULL, uint ksz=0, uint hsz=0, uint kedvezmeny=0)
uint	**getKedvezmeny** () const
void	**setKedvezmeny** (uint val)
void	**kiir** (std::ostream &os) const

▸ Publikus tagfüggvények a(z) **Jegy** osztályból származnak

Védett tagfüggvények

void	**exec** (const **Utas** &utas)

▸ Védett tagfüggvények a(z) **Jegy** osztályból származnak

A GyujtoJegy osztály a gyűjtőjegy megvalósítását tartalmazza. Érvényesítéshez személyigazolvány szükséges, többször is érvényesíthető, de ugyanarra a helyre szól mindig.

Publikus tagfüggvények

	GyujtoJegy (**Vonat** *j=NULL, uint ksz=0, uint hsz=0, uint darab=0) darab - ahányszor még felhasználható a jegy.
uint	**getDarab** () const
void	**setDarab** (uint val)
void	**kiir** (std::ostream &os) const

▸ Publikus tagfüggvények a(z) **Jegy** osztályból származnak

Amennyiben nem sikerült az érvényesítés úgy ErvenyesitesiHiba kivétel keletkezik.

Vonat osztály

A vonat osztály egy, a program futása során használatos konkrét vonat adatainak tárolására alkalmas. A vonathoz tartozik paraméter nélküli konstruktor a beolvasás miatt. A vonat tudja kezelni a hozzá tartozó jegyeket a jegyHozzaad, jegyTorol és a jegyKeres metódusaival.

Publikus tagfüggvények

	Vonat (uint az=0, string ia="", **Datum** ii={0, 0, 0, 0, 0}, string ea="", **Datum** ei={0, 0, 0, 0, 0})
uint	**getAzonosito** () const
void	**setAzonosito** (uint val)
string	**getIndAllomas** () const
void	**setIndAllomas** (string val)
string	**getErkAllomas** () const
void	**setErkAllomas** (string val)
Datum	**getIndIdo** () const
void	**setIndIdo** (**Datum** val)
Datum	**getErkIdo** () const
void	**setErkIdo** (**Datum** val)
void	**jegyHozzaad** (**Jegy** *jegy)
void	**jegyTorol** (**Jegy** *jegy)
void	**jegyekBeolvas** (std::istream &is)
void	**jegyekKiir** (std::ostream &os) const
Jegy *	**jegyKeres** (uint ksz, uint hsz) const
virtual	**~Vonat** () virtuális destruktor

Amennyiben a hely foglalt, ahova a jegyet hozzá szeretnénk adni úgy FoglaltHiba kivétel keletkezik. Ha a jegyhez tartozó vonatazonosító nem megfelelő, akkor RosszVonatHiba kivétel keletkezik.

Utas osztály

Az utas osztály egy utas azon adatait tárolja, amelyek szükségesek az érvényesítéshez. Jelen esetben azt tárolja, hogy az adott utasnál van-e diákigazolvány, valamint személyigazolvány.

Publikus tagfüggvények

	Utas (bool szlg, bool dlg)
bool	**hasSzemelyIg** () const
void	**setSzemelyIg** (bool val)
bool	**hasDiakIg** () const
void	**setDiakIg** (bool val)
virtual	**~Utas** () virtuális destruktor

HeteroStore osztály

A HeteroStore osztály egy heterogén kollekciót valósít meg template-vel. Hozzálehet adni elemet, valamint predikátumok alapján lehet törölni, keresni és bejárni a kollekcióban.

Publikus tagfüggvények

	HeteroStore ()
size_t	**size** () const
void	**hozzaad** (T *ptr)
template<typename Pred >	
void	**torol** (Pred unpred)
template<typename Pred >	
T *	**keres** (Pred unpred) const
template<typename Pred >	
void	**bejar** (Pred unpred) const
virtual	**~HeteroStore** () virtuális destruktor

Dátum osztály

A dátum osztály egy (év, hónap, nap, óra, perc) formátumú dátumot képes eltárolni. Megírtam a hozzá tartozó egyenlőség vizsgáló operátort, valamint a beolvasó és kiíró operátorokat.

Publikus tagfüggvények

	Datum (uint ev=0, uint honap=0, uint nap=0, uint ora=0, uint perc=0)
uint	**getEv** () const
void	**setEv** (uint val)
uint	**getHonap** () const
void	**setHonap** (uint val)
uint	**getNap** () const
void	**setNap** (uint val)
uint	**getOra** () const
void	**setOra** (uint val)
uint	**getPerc** () const
void	**setPerc** (uint val)
bool	**operator==** (const **Datum** &masik) const
virtual	**~Datum** () virtuális destruktor

Fontosabb algoritmusok megvalósítása

Az alábbi algoritmusokhoz fontos megismerkedni az ErvenyesitesiHiba interfészével:

- *Első paraméter:* ha részben azért keletkezett a hiba, mert nem használható a jegy.
- *Második paraméter:* ha részben azért keletkezett a hiba, mert nem volt az utasnál személyigazolvány.
- *Harmadik paraméter:* ha részben azért keletkezett a hiba, mert nem volt az utasnál diákigazolvány.

Az ervenyesit metódus (csak a használhatóságot teszteli)

```
void Jegy::ervenyesit(const Utas& utas) {
   if(!hasznalhato)
      throw ErvenyesitesiHiba(true, false, false);
   exec(utas);
}
```

A teljesjegy kritériumainak tesztelése (exec metódus)

```
void TeljesJegy::exec(const Utas& utas) {
    if(!utas.hasSzemelyIg())
        throw ErvenyesitesiHiba(false, true, false);
    setHasznalhato(false);
}
```

A diákjegy kritériumainak tesztelése (exec metódus)

```
void DiakJegy::exec(const Utas& utas) {
    if(!utas.hasSzemelyIg() || !utas.hasDiakIg())
        throw ErvenyesitesiHiba(false, !utas.hasSzemelyIg(), !utas.hasDiakIg());
    setHasznalhato(false);
}
```

A gyűjtőjegy kritériumainak tesztelése (exec metódus)

```
void GyujtoJegy::exec(const Utas& utas) {
    if(!utas.hasSzemelyIg())
        throw ErvenyesitesiHiba(false, true, false);
    if(--darab == 0)
        setHasznalhato(false);
}
```

Tesztelés

A tesztelés kétféleképpen zajlik. Először a menüpontokat tesztelem, a megadott bemeneti adatokkal (standard_input.txt) az összes lehetséges menüponton végigmegy a program, majd bezárja a menüt. Ezután lefutnak a tesztesetek, amelyek a gtest_lite segítségével vannak implementálva. Feladatuk, hogy külön-külön teszteljék a program függvényeit, metódusait és operátorait.

Memóriakezelés teszt

A memóriakezelés tesztelését a MEMTRACE modullal végeztem. A futások során nem tapasztaltam memóriakezelési hibát, és a JPORTA előzetes feltöltésnél se volt semmi probléma.

Lefedettségi teszt

A szabványos kimeneti és a funkcionális tesztek a program minden részét lefedték. Azonban a programban volt olyan rész, amely egy fájl megnyitásának sikertelenségét vizsgálta. Azonban ilyen állapotba a program normál futási körülmények között soha nem kerül, így ezeket a részeket kikommenteltem. Így a jporta 100%-os *Overall Code coveraget* adott vissza.

Tartalomjegyzék