AAAAAAAAAAAAAAAAAAAAAAAAAAAA

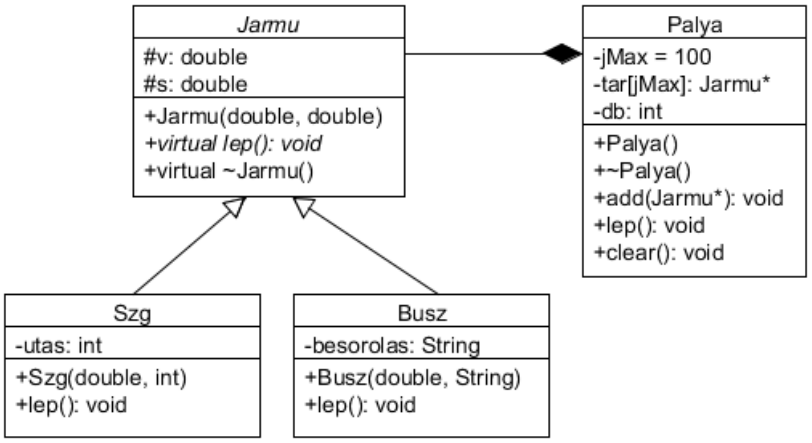
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G1/	**20 perc**
Név: **Neptun:**	Összpont:

1. feladat 3 pont

Autópálya forgalmát szeretnénk modellezni. A modellben a járműveket olyan objektumok valósítják meg, amelyek tárolják az átlagsebességet (m/s) és az indulástól megtett utat (m). Jelenleg 2 különböző járműtípusunk van, de később szeretnénk a modellt bővíteni. Az egyes járműtípusok a közös adatokon kívül további adatokkal is rendelkeznek: A személygépkocsi (*Szg*) tárolja az utasok számát (int), a busz (*Busz*) tárolja a közlekedésbiztonsági besorolásának kategóriáját (String). Az objektumok *lep()* metódusa a standard kimenetre írja ki a járműtípus nevét (Személygépkocsi/ Busz) és az addig megtett út hosszát! A metódust másodpercenként egyszer hívjuk meg, és tudjuk azt is, hogy legfeljebb 100 járművünk lehet a modellben. A rendszerben megvalósítandó műveleteket egy kódrészlettel mutatjuk be:

```
Palya M7;                       // Ez lesz az autópálya
M7.add(new Szg(35.5, 3));       // Személygépkocsi: 35,5 m/s, 3 fő
M7.add(new Busz(25, "E6791"));// Busz: 25 m/s, besorolás: E6791
M7.lep();                       // eltelt 1 sec, minden járművet léptetünk
M7.clear();                     // minden járművet töröl
```

Feltételezheti, hogy a *Palya* osztályból példányosított objektumot nem akarjuk paraméterként átadni, és értékadás jobb, ill. bal oldalán sem szerepel. **Tervezzen** meg és **rajzoljon** fel egy olyan osztályhierarchiát, amivel a fenti egyszerű modell megvalósítható és könnyen bővíthető új járműtípusokkal. Az osztálydiagramban jelölje az adattagok és metódusok láthatóságát is! Az osztályok adattagjai kívülről ne legyenek elérhetőek! A *String* osztályra típusként hivatkozzon! (A String osztály dinamikus memóriaterületet használ!)



A virtuális függvényeket a virtual kulcsszóval külön kiemeltük.

---

2. feladat 3 pont

Deklarálja az osztályokat! **Valósítsa** meg a *Palya* osztály *konstruktorát*, *destruktorát*, valamint az *add() és lep()* metódusát!

```
class Jarmu {
protected:
    double v;
    double s;
public:
    Jarmu(double, double);
    virtual void lep() = 0;
    virtual ~Jarmu() {}
};

class Szg : public Jarmu {
    int utas;
public:
    Szg(double, int);
    void lep();
};

class Busz : public Jarmu {
    String besorolas;
public:
    Busz(double, String);
    void lep();
};
```

```
class Palya {
    enum { jMax = 100 };
    Jarmu *tar[jMax];
    int db;
public:
    Palya() : db(0) {}
    void add(Jarmu *j) {
        tar[db++] = j;
    }
    void lep() {
        for (int i = 0; i < db; i++)
            tar[i]->lep();
    }
    void clear();
    ~Palya() { clear(); }
};
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

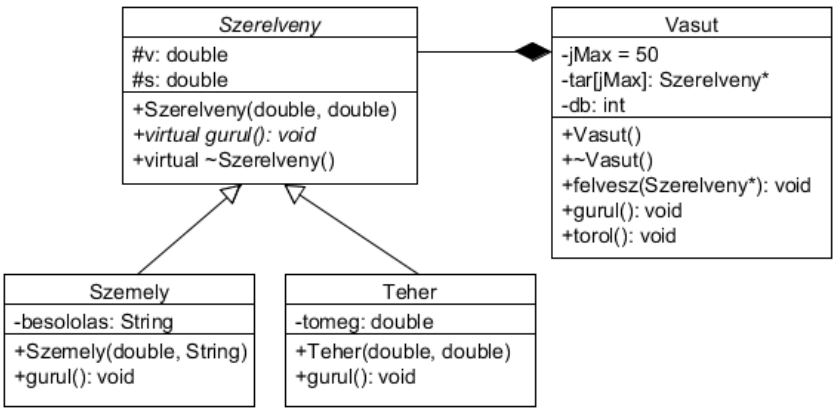
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G1/	**20 perc**
Név: **Neptun:**	Összpont:

1. feladat 3 pont

Vasúti forgalmat szeretnénk modellezni. A vasúti szerelvényeket olyan objektumok valósítják meg, amelyek tárolják a szerelvény átlagsebességét (m/s) és az indulástól megtett utat (m). Jelenleg 2 különböző szerelvénytípusunk van, de később szeretnénk a modellt bővíteni. Az egyes típusok a közös adatokon kívül további adatokkal is rendelkeznek: A személyvonat (*Szemely*) tárolja a vonat besorolását (String), a tehervonat (*Teher*) tárolja a szerelvény teljes tömegét (double). Az objektumok *gurul()* tagfüggvénye frissítse az átlagsebességnek megfelelően a megtett utat, és írja ki a standard kimenetre a szerelvény típusának feltüntetésével (Személy/Teher)! A függvényt másodpercenként egyszer hívjuk meg, és tudjuk azt is, hogy legfeljebb 50 vonatunk lehet a modellben. A rendszerben megvalósítandó műveleteket egy kódrészlettel mutatjuk be:

```
Vasut BB;                  // Ez lesz a modellezett vasútvonalunk
BB.felvesz(new Szemely(15.5, "Sebes"));  // Személy: 15.5 m/s, Sebes vonat
BB.felvesz(new Teher(25, 1000));// Teher: 25 m/s, 1000t
BB.gurul();                // eltelt 1 sec, minden szerelvényt „továbbgurítunk”
BB.torol();                // minden szerelvényt törlünk
```

Feltételezheti, hogy a *Vasut* osztályból példányosított objektumot nem akarjuk paraméterként átadni, és értékadás jobb, ill. bal oldalán sem szerepel. **Tervezzen** meg és **rajzoljon** fel egy olyan osztályhierarchiát, amivel a fenti egyszerű modell megvalósítható és könnyen bővíthető új vonattípusokkal. Az osztálydiagramban jelölje az adattagok és metódusok láthatóságát is! Az osztályok adattagjai kívülről ne legyenek elérhetőek! A *String* osztályra típusként hivatkozzon! (A String osztály dinamikus memóriaterületet használ!)



A virtuális függvényeket a virtual kulcsszóval külön kiemeltük.

---

2. feladat 3 pont

Deklarálja az osztályokat! **Valósítsa** meg a *Vasut* osztály *konstruktorát*, *destruktorát*, valamint a *felvesz() és gurul()* metódusát!

```
class Szerelveny {
protected:
    double v;
    double s;
public:
    Szerelveny(double, double);
    virtual void gurul() = 0;
    virtual ~Szerelveny() {}
};

class Szemely : public Szerelveny {
    String besorolas;
public:
    Szemely(double, String);
    void gurul();
};

class Teher : public Szerelveny {
    double tomeg;
public:
    Teher(double, double);
    void gurul();
};
```

```
class Vasut{
    enum { jMax = 50 };
    Szerelveny *tar[jMax];
    int db;
public:
    Vasut() : db(0) {}
    void felvesz(Szerelveny *j) {
        tar[db++] = j;
    }
    void gurul() {
        for (int i = 0; i < db; i++)
            tar[i]->gurul();
    }
    void torol();
    ~Vasut() { torol(); }
};
```