AAAAAAAAAAAAAAAAAAAAAAAAAAAA

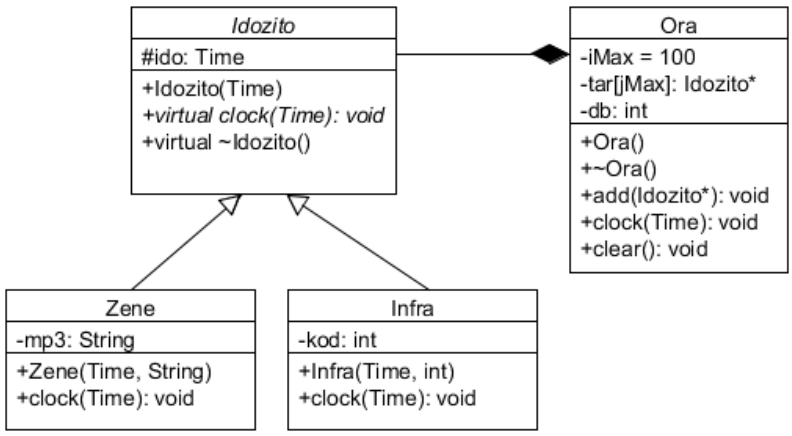
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G2/	**20 perc**
Név: **Neptun:**	Összpont:

1. feladat 3 pont

Egy okostelefon ébresztőóra alkalmazását objektumorientált módon szeretnénk kialakítani. Ehhez olyan időzítő objektumokat készítünk, amelyek tárolják az ébresztés időpontját (Time), és a megfelelő időpontban elvégzik a hozzájuk rendelt tevékenységet. Jelenleg 2 különböző tevékenységet definiáltunk: *Zene* objektum lejátssza kiválasztott mp3 fájlt, az *Infra* objektum pedig bekapcsolja a TV-t. A *Zene* objektum szövegesen (String) tárolja a fájl nevét, ezzel szemben az *Infra* objektum egy egész számot tárol, ami a TV kódja. Az objektumok *clock()* metódusa paraméterként megkapja a pillanatnyi időt (Time), és ha szükséges elvégzi a megfelelő tevékenységet. Egyszerre legfeljebb 40 ilyen időzítő objektumunk lehet. A rendszerben megvalósítandó műveleteket egy kódrészlettel mutatjuk be:

```
Ora vekker;              // Ez lesz az óra, ami tárolja az időzítő objektumokat
vekker.add(new Zene(1615, "boci-boci.mp3")); // 16:15-kor boci-boci.mp3
vekker.add(new Infra(2015, 523));// 20:15-kor bekapcsolja az 523-as infa-kódú TV-t
vekker.clock(1600);      // meghívja minden időzítő clock() függvényét az idővel
...
vekker.clock(2015);      // ismét meghívja minden időzítő clock() függvényét
vekker.clear();          // minden időzítést megszüntet
```

Feltételezheti, hogy az *Ora* 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 időzítőkkel. Az osztálydiagramban jelölje az adattagok és metódusok láthatóságát is! Az osztályok adattagjai kívülről ne legyenek elérhetőek! A *String* és a *Time* osztályra típusként hivatkozzon! A String osztály dinamikus memóriaterületet használ, a Time osztály pedig összehasonlítható. A példában megadott időformátum a feladat szempontjából lényegtelen. Azzal ne foglalkozzon!



A virtuális függvényeket a virtual kulcsszóval külön kiemeltük.

---

2. feladat 3 pont

Deklarálja az osztályokat! **Valósítsa** meg az *Ora* osztály *konstruktorát*, *destruktorát*, valamint az *add() és clock()* metódusát!

```
class Idozito {
protected:
    Time ido;
public:
    Idozito(Time);
    virtual void clock(Time) = 0;
    virtual ~Idozito() {}
};

class Zene : public Idozito {
    String mp3;
public:
    Zene(Time, String);
    void clock(Time);
};

class Infra : public Idozito {
    int kod;
public:
    Infra(Time, int);
    void clock(Time);
};
```

```
class Ora {
    enum { iMax = 40 };
    Idozito *tar[iMax];
    int db;
public:
    Ora() : db(0) {}
    void add(Idozito *i) {
        tar[db++] = i;
    }
    void clock(Time tim) {
        for (int i = 0; i < db; i++)
            tar[i]->clock(tim);
    }
    void clear();
    ~Ora() { clear(); }
};
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

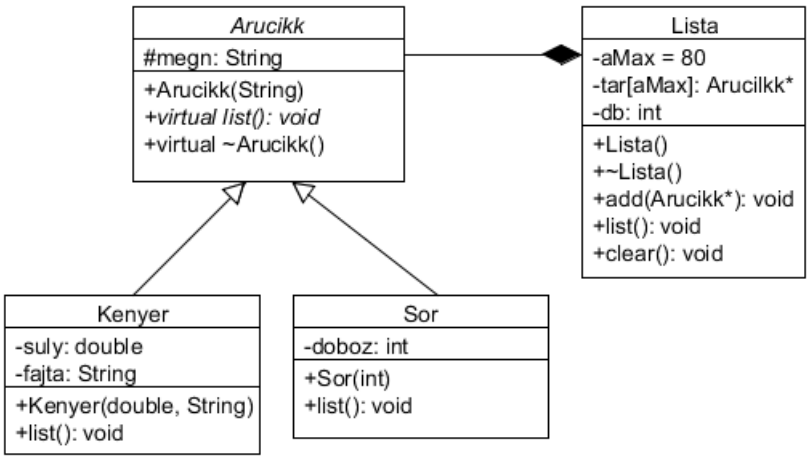
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G2/	**20 perc**
Név: **Neptun:**	Összpont:

1. feladat 3 pont

Egy okostelefon bevásárló lista alkalmazását objektumorientált módon szeretnénk kialakítani. A listában a különböző árucikkeket különböző objektumok valósítják meg. Egyetlen közös tulajdonságuk a megnevezés (String). Jelenleg 2 árucikket definiáltunk: Kenyér *(Kenyer)*, aminek a súlyát (double) és a fajtáját (String) is tárolni akarjuk, valamint sör *(Sor)* amiből mindig ugyanazt a dobozos fajtát vesszük, így csak a darabszámot akarjuk eltárolni a listánkban. Az objektumok *list()* metódusa írja ki a megnevezést és a mennyiséget! Egyszerre legfeljebb 80 ilyen objektumunk lehet. A rendszerben megvalósítandó műveleteket egy kódrészlettel mutatjuk be:

```
Lista cetli;              // Ez a bevásárló listánk
cetli.add(new Kenyer(0.5, "Barna")); // fél kiló barna kenyér kell
cetli.add(new Sor(10)); // 10 doboz sör is kell
cetli.list();             // kiírjuk a listát, hogy mit kell megvenni
cetli.clear();            // mindent megvettünk töröljük a listát
```

Feltételezheti, hogy az *Lista* 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 árucikkel. 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 *Lista* osztály *konstruktorát*, *destruktorát*, valamint az *add() és list()* metódusát!

```
class Arucikk {
protected:
    String megn;
public:
    Arucikk(String);
    virtual void list() = 0;
    virtual ~Arucikk() {}
};

class Kenyer : public Arucikk {
    double suly;
    String fajta;
public:
    Kenyer(double, String);
    void list();
};

class Sor : public Arucikk {
    int doboz;
public:
    Sor(int);
    void list();
};
```

```
class Lista {
    enum { aMax = 80 };
    Arucikk *tar[aMax];
    int db;
public:
    Lista() : db(0) {}
    void add(Arucikk *a) {
        tar[db++] = a;
    }
    void list() {
        for (int i = 0; i < db; i++)
            tar[i]->list();
    }
    void clear();
    ~Lista() { clear(); }
};
```