AAAAAAAAAAAAAAAAAAAAAAAAAAAA

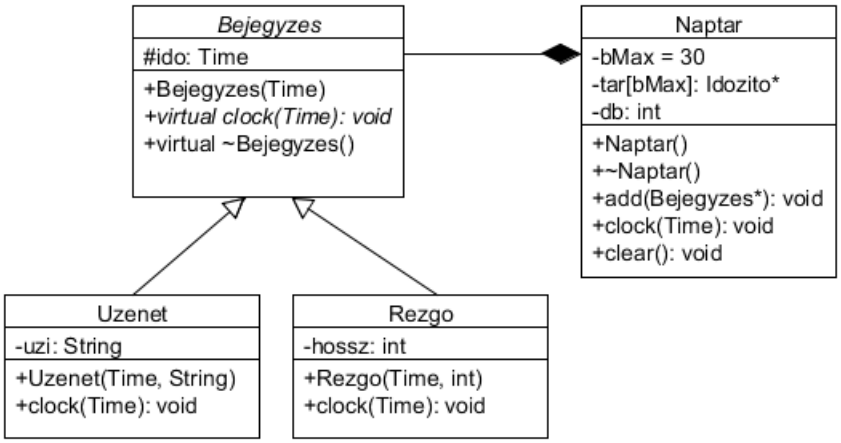
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G3/	**20 perc**
Név: **Neptun:**	Összpont:

1. feladat 3 pont

Egy telefon naptár alkalmazását objektumorientált módon szeretnénk kialakítani. Ehhez olyan objektumokat készítünk, amelyek tárolják az naptárbejegyzés időpontját (Time), és a megfelelő időpontban elvégzik a hozzájuk rendelt tevékenységet. Jelenleg 2 különböző tevékenységet definiáltunk: *Uzenet* objektum kiírja benne tárolt szöveget (String), a *Rezgo* objektum pedig rezgő jelzést ad. A *Rezgo* objektum egy egész számot tárol, ami rezgetés hossza (sec). Az objektumok *clock()* metódusa paraméterként megkapja a pillanatnyi időt (Time), és ha szükséges elvégzi a megfelelő tevékenységet. Egyszerre legfeljebb 30 ilyen objektumunk lehet. A rendszerben megvalósítandó műveleteket egy kódrészlettel mutatjuk be:

```
Naptar cal;       // Ez lesz az naptár, ami tárolja a objektumokat
cal.add(new Uzenet(1615, "Prog2 ZH")); // 16:15-kor prog2 ZH
cal.add(new Rezgo(2015, 3));           // 20:15-kor 3 sec rezgés
cal.clock(1500);  // meghívja minden objektum clock() függvényét az idővel
...
cal.clock(1516);  // ismét meghívja minden objektum clock() függvényét
cal.clear();      // minden bejegyzést megszüntet
```

Feltételezheti, hogy a *Naptar* 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 naptárbejegyzésekkel. 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

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

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

class Uzenet : public Bejegyzes {
    String uzi;
public:
    Uzenet(Time, String);
    void clock(Time);
};

class Rezgo : public Bejegyzes {
    int hossz;
public:
    Rezgo(Time, int);
    void clock(Time);
};
```

```
class Naptar {
    enum { bMax = 30 };
    Bejegyzes *tar[bMax];
    int db;
public:
    Naptar() : db(0) {}
    void add(Bejegyzes *b) {
        tar[db++] = b;
    }
    void clock(Time tim) {
        for (int i = 0; i < db; i++)
            tar[i]->clock(tim);
    }
    void clear();
    ~Naptar() { clear(); }
};
```

BBBBBBBBBBBBBBBBBBBBBBBBBBB

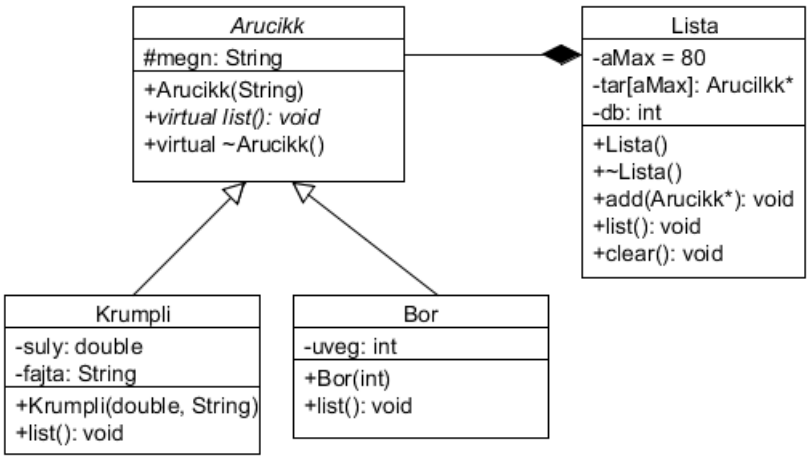
Programozás alapjai II. 3. ellenőrző dolgozat. 2013.04.04. Kurz/**Terem**: G3/	**20 perc**
Név: **Neptun:**	Összpont:

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: Krumpli *(Krumpli)*, aminek a súlyát (int) és a fajtáját (String) is tárolni akarjuk, valamint bor *(Bor)* amiből mindig ugyanazt a fajtát vesszük, így csak az üvegek akarjuk eltárolni a listánkban. Az objektumok *list()* metódusa írja ki 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 Krumpli(3, "édes")); // 3 kiló édesburgonya
cetli.add(new Bor(4));    // 4 üveg bor is kell
cetli.list();             // kiírjuk a listában levő adatokat
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 clear()* metódusát!

```
class Arucikk {
protected:
    String megn;
public:
    Arucikk(String);
    virtual void list() = 0;
    virtual ~Arucikk() {}
};
class Krumpli : public Arucikk {
    double suly;
    String fajta;
public:
    Krumpli(double, String);
    void list();
};
class Bor : public Arucikk {
    int uveg;
public:
    Bor(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(); }
}
```