5. CLASE ABSTRACTE

Adesea este util să creem clase de **bază** care să impună ca, clasele derivate din ele, să **conțină neapărat definiții** de anumite **metode**.

Clasa de **bază** conține doar **antete** de metode urmate de caracterul **punct şi virgulă** (**;**), iar aceste metode urmează să fie definite în clasele **derivate** dintr-o astfel de clasă de bază. Metodele menționate în acest fel într-o clasă de bază se numesc **metode abstracte**.

O **clasă abstractă** este o clasă care are **cel puțin o metodă membru abstractă**.

Antetul unei **metode abstracte** este de forma

... tip nume (lista declarațiilor parametrilor formali)

Exemplu:

```
class punct
{
   double x,y;
   ...
}
abstract class mpunct:punct
{
   double masura;
   ...
   public abstract double perimetru();
   public abstract double arie ();
}
```

La definirea claselor abstracte cuvântul **class** va fi precedat de cuvântul cheie **abstract**.

Clasa **mpunct** conține metodele abstracte **perimetru** şi **arie**. Aceste metode nu au sens pentru obiectele clasei **mpunct** care are ca date membru de instanță **coordonatele x** (**abscisă**) şi **y** (ordonată) moştenite de la clasa **punct** şi **masura** care este dată proprie. De aceea, metodele **perimetru** şi **arie** nu se pot defini în clasa **mpunct**. În schimb, ele vor fi definite în clase **derivate** din clasa **mpunct**. Clasa **mpunct** este o **clasă abstractă**, deoarece are două metode membru abstracte. **Metodele abstracte** se definesc prin **suprascriere** (**override**) ca şi cum ele ar fi metode **virtuale**.

Fie clasa **patrat** derivată din clasa **mpunct**. În clasa **patrat**, metodele **perimetru** şi **arie** se definesc în mod obişnuit:

```
class patrat:mpunct
{
   ...
   public override double perimetru()
   {
      return 4*masura;
   }
   public override double arie()
   {
      return masura*masura;
   }
}
```

Nu se pot instanția obiecte pentru clasele abstracte, deoarece ele nu implementează metodele membru abstracte pe care le conțin.

Deci o instanțiere de forma:

```
mpunct mp = new mpunct(...);
```

este eronată. Se pot declara referințe la clase abstracte la care să li se atribuie referințe la obiecte ale claselor derivate.

De exemplu, declarația:

```
mpunct mp;
```

este corectă, precum şi atribuirea:

```
mp = new patrat(...);
```

sau mai direct

```
mpunct mp = new patrat(...);
```

Exerciții

5.1. Să se definească o ierarhie de clase care are in vârful ei clasa **punct** din care derivă clasa abstractă **mpunct** care la rândul ei este o clasă de bază pentru clasele **patrat** şi **cerc**.

Clasa **mpunct** are trei metode abstracte:

- perimetru;
- arie;
- retNume.

Proiectul aplicației va avea numele **PatratCerc1**.

Clasa **punct** are următoarele **date membru de instanță**:

- **x** de tip **double** care defineşte absicisa **punctului** reprezentat de obiectul curent;
- **y** de tip **double** care defineşte **ordonata** punctului reprezentat de obiectul curent;
- **valid** de tip **bool** care are ca valoare **true** dacă datele membru **x** şi **y** au valori definite prin constructor sau prin citirea lor de la tastatură şi **false** în caz contrar.

Clasa **punct** are o **metodă statică** numită **citDouble** care citeşte de la tastatură un număr de tip **double** şi returnează valoarea acestuia. Celelalte metode ale clasei punct sunt **metode de instanță**. Astfel, clasa are doi constructori: **constructorul implicit** şi **constructorul** care are parametri pentru a iniţializa valorile datelor de instanță **x** şi **y**.
Constructorul implicit atribuie valoarea **false** datei membru **valid**, iar celălalt constructor atribuie valoarea **true** aceleiaşi date membru.
Clasa **punct** are o metodă de instanță cu numele **cit** prin intermediul căreia se citesc valori pentru datele membru **x** şi **y**. Totodată această metodă atribuie valoarea **true** datei membru **valid**.
În fine, o altă metodă de instanță a clasei **punct** este metoda întitulată **afis**. Aceasta afişează valorile datelor membru **x** şi **y** dacă **valid** are valoarea **true**. În caz contrar (**valid** are valoarea **false**), se afişează mesajul de eroare: **Punct invalid**.
Clasa **abstractă mpunct** derivă din clasa **punct**. Ea are următoarele date membru de **instanță** proprii:

masura de tip **double**;
mvalid de tip **bool**;

Clasa **mpunct** are doi constructori:
- constructorul **implicit** care atribuie lui **mvalid** valoarea **false**;
- constructorul cu parametri pentru iniţializarea datelor membru de instanță moştenite de la clasa punct, precum şi data membru de instanță proprie (masura). De asemenea, constructorul atribuie valoarea **true** datei membru **mvalid**. Pentru iniţializarea datelor membru **x**, **y** şi **valid**, constructorul apelează prin antetul său constructorul clasei de bază punct care iniţializează datele respective.

În afară de constructori, clasa **mpunct** conţine şi metodele **cit** şi **afis** pentru a citi de la tastatură valorile datelor de instanță **x**, **y** şi **masura**, respectiv pentru a afişa pe monitor datele respective.
Metoda **cit** atribuie datei membru **mvalid** valoarea **true**.
Metoda **afis** afişează textul de eroare:

Obiect mpunct invalid

dacă **mvalid** are valoarea **false**.
Clasa **mpunct** conţine **antetele metodelor abstracte**: **perimetru**, **arie** şi **retNume** care se definesc în clasele **patrat** şi **cerc** derivate din această clasă.
Clasa **patrat** are următoarele date membru de instanță:

nume de tip **string**;
pvalid de tip **bool**.

V-240

Limbajul C# pentru începători
clasa String, ierarhii de clase, indexare, interfețe, clase abstracte şi anonime, clasa Object

Liviu Negrescu, Lavinia Negrescu

1. CLASA *STRING*
1.1. Inițializarea obiectelor clasei *String*
1.2. Accesul la caracterele şirurilor de caractere ataşate obiectelor clasei *String*
1.3. Metode ale clasei *String*
1.3.1. Metode de instanță
1.3.2. Metode statice
1.4. Proprietatea *Length* a clasei *string*
1.5. Câmpul static al clasei *string*
1.6. Conversii
1.7. Operatori
1.7.1. Supraîncărcarea operatorilor relaționali
Rezumat

2. IERARHII DE CLASE
2.1. Clasă de bază şi clasă derivată
2.2. Protecția *protected*
2.3. Relația dintre constructorii claselor derivate şi cei ai claselor de bază
2.4. Metode *new*
2.5. Utilizarea lui *base* pentru accesarea numelor ascunse
2.6. Metode virtuale
Rezumat

3. INDEXARE
Rezumat

4. INTERFEȚE
Rezumat

5. CLASE ABSTRACTE
Rezumat

6. VARIABILE DE TIPURI IMPLICITE
Rezumat

7. CLASE ANONIME
Rezumat

8. CLASA *OBJECT*

Rezumat