Laborator 8.1 – Crearea obiectelor

Obiective

După completarea acestui laborator veți dobândi următoarele cunoștințe:

- Suprascrierea constructorului default
- Supraîncărcarea constructorului default
- Inițializarea datelor **read-only**

Condiţii prealabile

Înainte de a realiza acest laborator trebuie să fiţi familiarizaţi cu următoarele concepte:

- Crearea claselor și instanțierea obiectelor
- Definirea și apelarea metodelor

➢ Exerciţiul 1 – Crearea și utilizarea claselor

În acest exerciţiu veţi schimba clasa **BankAccount** pe care ați creat-o în ședința trecută. Veți șterge metodele care populau cu valori câmpurile clasei și le veți înlocui cu o serie de constructori care vor fi utilizați la instanțiarea unui obiect din această clasă.

Crearea constructorului default

- Deschideți suportul laboratorului cu numărul nouă și ștergeți metoda **Populate()**.
- Scrieți un constructor default cu următoarele caracteristici:
  - Numele constructorului este **BankAccount**;
  - Constructorul este **public**;
  - Nu primește niciun parametru și nu are tip de returnare;
- În corpul constructorului se vor realiza următoarele operații:
  - Asignarea unei valori câmpului **accNumber** utilizând metoda **NextNumber()**;
  - Câmpul **amount** va fi iniţializat cu valoarea **0**;
  - Câmpul **accType** va fi iniţializat cu valoarea **AccountType.Cec**;

Supraîncărcarea constructorului default

- Creați un alt constructor, **BankAccount**(**public** și fără tip), care primește un parametru de tip **AccountType** numit **tCont**. Operațiile care se vor face în interiorul acestui constructor sunt:

  - Asignarea unei valori câmpului **accNumber** utilizând metoda **NextNumber()**;
  - Câmpul **amount** va fi inițializat cu valoarea **0**;
  - Câmpul **accType** va fi inițializat cu valoarea parametrului **tCont**;

- Creați un alt constructor, **BankAccount**(**public** și fără tip), care primește un parametru de tip **decimal** numit **sCont**. Operațiile care se vor face în interiorul acestui constructor sunt:

  - Asignarea unei valori câmpului **accNumber** utilizând metoda **NextNumber()**;
  - Câmpul **amount** va fi inițializat cu valoarea parametrului **sCont**;
  - Câmpul **accType** va fi inițializat cu valoarea **AccountType.Cec**;

- Definiți un constructor final, **BankAccount**(**public** și fără tip), care primește doi parametrii: **tCont** de tip **AccountType** și **sCont** de tip **decimal**. Operațiile realizate în corpul constructorului vor fi:

  - Asignarea unei valori câmpului **accNumber** utilizând metoda **NextNumber()**;
  - Câmpul **amount** va fi inițializat cu valoarea parametrului **sCont**;
  - Câmpul **accType** va fi inițializat cu valoarea parametrului **tCont**;

Testarea constructorilor

- În metoda **Main()** din clasa **Program** declarați patru variabile de tip **BankAccount**: **account1**, **account2**, **account3** și **account4**.
- Instanțiați **account1** utilizând constructorul default.
- Instanțiați **account2** utilizând constructorul ce primește un singur parametru de tip **AccountType** și setați valoarea acestui parametru la **AccountType.Deposit**.
- Instanțiați **account3** utilizând constructorul ce primește un singur parametru de tip **decimal** și setați valoarea acestui parametru la **100**.
- Instanțiați **account4** utilizând constructorul ce primește doi parametrii și setați valoarile acestora la **AccountType.Deposit** respectiv **500**.
- În clasa **BankAccount** declarați o metodă care afișează valorile unui cont bancar primit ca parametru:

  - Numele metodei va fi **Write**;

- Metoda va fi de tip **void** (nu va returna nimic);
- Metoda va fi publică, **statică**;
- Metoda primește un parametru de tip **BankAccount**;

- Utilizați metoda **BankAccount.Write()** pentru a afișa cele patru conturi create.
- Salvați și compilați programul.

➢ Exercițiul 2 – Inițializarea datelor read-only

În acest exercițiu veți crea o clasă numită **BankTransaction** care va păstra informații despre tranzacțiile legate de un anumit cont (retragere sau depozit).

De fiecare dată, când soldul unui cont va fi modificat printr-un depozit sau o retragere va fi creat un nou obiect de tip **BankTransaction**. Un obiect de tip **BankTransaction** va conține data tranzacției (**System.DateTime**) și suma de bani (pozitivă în caz de depozit, negativă altfel). Deoarece data unei tranzacții bancare nu poate fi modificată, va fi declarată ca un câmp **read-only**.

Constructorul clasei **BankTransaction** va primi un parametru de tip **decimal**, cu a cărui valoare va fi populat câmpul **amount** aparținând clasei. Câmpul **date** va fi populat utilizând proprietatea **DateTime.Now** care returnează data şi ora curentă.

În clasa **BankAccount** veți modifica metodele **Deposit()** și **Withdraw()**. Veți păstra tranzacția realizată într-o variabilă de tip **System.Collection.Queue**. O coadă (**queue**) este o structură de date care păstrează o listă de obiecte. Există metode deja implementate pentru această clasă, pe care le vom folosi în adăugarea unui nou element și iterarea prin coadă (Utilizarea unei cozi este preferabilă față de un **array** pentru că nu are o dimensiune fixă, își poate mări dimensiunea pe măsură ce alte tranzacții sunt efectuate).

Crearea clasei BankTransaction

- Adăugați o nouă clasă numită **BankTransaction** proiectului deja existent. Puteți să faceți acest lucru în același fișier sau în altul. Dacă alegeți să creați un fișier separat trebuie să vă asigurați că spațiul de nume este **SuportLaborator8**.
- Clasa va avea două câmpuri **private** și **readonly**:
  - **amount** de tip **decimal**;
  - **date** de tip **DateTime**;
- Adăugați două metode publice care vor returna valorile celor două câmpuri.

```
public decimal Amount( )
public DateTime Date( )
```

Crearea constructorului

- Creați un constructor public pentru această clasă care primește un singur parametru de tip **decimal** ce reprezintă valoarea de inițializare a câmpului **amount**.
- Câmpul **date** va fi inițializat tot în constructor utilizând proprietatea **DateTime.Now** ce returnează ora și data curentă.
- Compilați proiectul și corectați eventualele erori.

Crearea tranzacțiilor

- Cum s-a spus și în introducere, tranzacțiile vor fi create în clasa **BankAccount** și păstrate într-o coadă de fiecare dată când metodele **Deposit()** și **Withdraw()** vor fi apelate.
- Înaintea de toate nu uitați să adăugați următorul *namespace*:

```
Using System.Collections;
```

- Adăugați o variabila privată, de tip **Queue**, numită **transactionQueue** în clasa **BankAccount**.

```
private Queue transactionQueue = new Queue( );
```

- În metoda **Deposit()** înainte de instrucțiunea return, creați un obiect de tip **BankTransaction** utilizând valoarea lui **amount** ca parametru. Adăugați obiectul nou creat în coadă utilizând metoda **Enqueue** a clasei **Queue**.

```
BankTransaction tran = new BankTransaction(amount);
transactionQueue.Enqueue(tran);
```

- În metoda **Withdraw()**, dacă sunt suficiente fonduri, realizați același lucru ca la **Deposit()**. Nu uitați ca **amount** trebuie dată cu semnul schimbat daca este vorba de o retragere.

Testarea tranzacțiilor

- Adăugați o nouă metodă numită **Tranzactions** la clasa **BankAccount**. Metoda va fi publică și va returna coada de tranzacții.
- Modificați metoda **Write()** pentru a afișa coada de tranzacții specifice contului primit ca parametru. Clasa **Queue** implementează interfața **IEnumerable** ceea ce ne va permite să utilizăm instrucțiunea **foreach** pentru a itera prin coadă:

```
foreach (BankTransaction tran in cb.Tranzactions( ))
{
      Console.WriteLine("Data/Ora: {0}\tSuma: {1}",tran.Date( ),
            tran.Amount( ));
}
```

- Utilizați metodele **TestWithdraw()** și **TestDeposit()** pentru a realiza tranzacții.
- Afișați fiecare cont bancar înainte și după tranzacții pentru a observa modificările.
- Compilați și executați programul.