Laborator – Operatori

Obiective

După completarea acestui laborator veți dobândi următoarele cunoștințe:

- Crearea de operatori pentru adunare, scădere, testare de egalitate, multiplicare, împărțire și cast
- Suprascrierea metodelor **Equals, ToString** și **GetHashCode**

Condiţii prealabile

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

- Folosirea moștenirii în C#
- Definirea constructorilor
- Compilarea
- Operatori C# de bază

➢ Exerciţiul 1- Definirea de operatori pentru clasa BankAccount

În laboratorul anterior, ați creat diverse clase pentru sistemul bancar. Clasa **BankAccount** reține detalii privind conturile clienților, incluzând numărul contului și soldul. Ați creat de asemenea o clasă **Bank** ce acționează ca o fabrică pentru crearea și managementul obiectelor de tip **BankAccount**.

În acest exercițiu, veți definii operatorii == și != pentru clasa **BankAccount**. Implementarea implicită pentru acești operatori, moștenită de la clasa Object, testează dacă referințele sunt identice. Vor fi redefiniți pentru a testa informațiile legate de conturi.

Veți rescrie apoi metodele **Equals** și **ToString**. Metoda **Equals** este des folosită la runtime și ar trebui să aibe același comportament cu operatorul de egalitate. Multe clase din platforma .NET folosesc metoda **ToString** pentru a se obține o reprezentare printabilă a unui obiect.

Pentru defini operatorii == și !=

- Deschideți proiectul de startup
- Adăugați o metodă ce supraîncarcă operatorul == aplicat pe două obiecte de tipul **BankAccount** ce întoarce o valoare booleană
- În corpul metodei, scrieți instrucțiuni ce verifică dacă numărul, tipul și soldul celor două conturi sunt egale. Dacă da, se întoarce **true**, altfel **false.**
- Dacă compilați proiectul în acest moment, veți primi o eroare. De ce?
- Adăugați o metodă ce supraîncarcă operatorul != aplicat pe două obiecte de tipul **BankAccount** ce întoarce o valoare booleană
- În corpul metodei, scrieți instrucțiuni ce verifică dacă numărul, tipul și soldul celor două conturi sunt egale. Dacă da, se întoarce **false**, altfel **true.**
- Salvați și compilați proiectul. Rezolvați eventualele erori de apar.

Pentru a testa operatorii

- Adăugați o nouă clasă proiectului, **TestOperatori**.
- Scrieți în această clasă o metodă **Main** și setați-o ca metodă de început a aplicației
- Creați două obiecte de tipul **BankAccount** în metoda **Main.** Pentru a face acest lucru:
  - Folosiți **Bank.CreateAccount()** și inițializați conturile la același sold și cu același tip
  - Salvați numerele conturilor generate în două variabile **long** numite accNumber1 și accNumber2
- Definiți două variabile de tipul **BankAccount** numite account1 și account2. Apelați metoda **Bank.GetAccount()** pentru a obține în aceste variabile referințe către cele 2 conturi.
- Comparați account1 cu account2 folosind operatorul ==. Testul ar trebui să întoarcă **false**, deoarece cele două conturi au numere diferite.
- Testați rezultatul întors de operatorul != pentru aceleași conturi. Rezultatul ar trebui să fie **true**

Suprascrierea metodelor Equals, ToString și GetHashCode

- Adăugați o metodă **Equals** clasei **BankAccount**. Această metodă trebuie să suprascrie metoda clasei Object, primind un parametru de tip **object** și întorcând o valoare booleană. Corpul metodei trebuie să execute același cod ca operatorul ==, dar ca metodă a instanței, nu a clasei. Folosiți operatorul == între **this** și parametrul primit.
- Adăugați o metodă **ToString** clasei **BankAccount**. Această metodă trebuie să suprascrie metoda clasei Object, neprimind parametri și întorcând o valoare de tip string. Corpul metodei trebuie să întoarcă o reprezentare a obiectului sub forma unui string.
- Adăugați o metodă **GetHashCode** clasei **BankAccount**. Această metodă trebuie să suprascrie metoda clasei Object, neprimind parametri și întorcând o valoare de tip **int**. Cel mai la îndemână este întoarcerea numărului contului. Va trebui să faceți cast la **int** întâi.

Pentru a testa metodele Eqals și ToString

- În **Main**-ul definit pentru a testa operatorii == și !=, înlocuiți acești operatori apelând metoda **Equals**.
  - Înlocuiți (account1 == account2) prin (account1.Equals(account2))
  - Înlocuiți (account1 != account2) prin (!account1.Equals(account2))
- Folosiți **Console.WriteLine** pentru a afișa conținutul celor două conturi. Metoda **WriteLine** va apela automat metoda **ToString** a conturilor pentru a putea face afișarea.

➢ Exerciţiul 2- Tratarea numerelor raționale

În acest exercițiu, veți defini o clasă complet nouă ce se ocupă de numerele irațională. Un număr irațional este un număr ce se poate scrie ca raport a două numere întregi. Veți defini în continuare o clasă **Rational** care constă dintr-o pereche de variabile întregi, numite dividend și divisor, și calculele ce se pot realiza asupra lor. Următorii operatori și metode vor fi definite:

- Rational(int dividend) – constructor ce setează deîmpărţitul la valoarea oferită și divizorul la valoarea 1
- Rational(int dividend, int divisor) – ambele valori oferite

- **==** și **!=** - acești operatori vor trebui să facă comparația bazându-se pe valoarea numerică a celor două obiecte (ex: Rational(6, 8) == Rational(3, 4)). Va trebui de asemenea suprascrisă metoda **Equals**
- **<**, **>**, **<=**, **>=** - vor efectua comparații între numere raționale (ex: Rational(6, 8)>Rational(1, 4))

Crearea constructorilor, a metodei ToString și a metodei GetDecimalValue

- Deschideți un proiect nou, de tipul **ConsoleApplication**, numit **Rational**
- Adăugați o clasă **Rational** în acest proiect; definiți două variabile private ale clasei, dividend și divisor. Inițializați-le cu 0 și respectiv 1. Adăugați cei doi constructori specificați. Pentru constructorul ce setează valoarea lui divisor, verificați ca aceasta să fie diferită de 0; aruncați **ArgumentOutOfRangeException** dacă este cazul.
- Suprascrieți metoda **ToString** pentru a obține o variantă printabilă a unui obiect **Rational**; folosiți un format ales de Dvs.
- Metoda **GetDecimalValue** va întoarce o valoare de tip decimal, valoarea împărțirii lui dividend la divisor. **Obs:** în momentul executării împărțirii, va trebui făcut cast la decimal. Împărțirea se va face implicit pe **int**, pierzându-se valoarea inițială.

```
public decimal GetDecimalValue()
```

Definirea operatorilor relaționali.

- Supraîncărcați operatorul **==** pentru clasa **Rational**. Corpul metodei de supraîncărcare va compara valorile decimal a celor două obiecte **Rational** și va întoarce **true** dacă acestea sunt egale și **false** altfel.
- Supraîncărcați operatorul **!=** pentru clasa **Rational**. Corpul metodei de supraîncărcare va compara valorile decimal a celor două obiecte **Rational** și va întoarce **false** dacă sunt egale și **true** altfel.
- Suprascrieți metoda **Equals**, folosindu-vă de operatorul == în implementarea ei.
- Definiți perechea de operatori **< și >**. Folosiți cod asemănător celui folosit la implementarea lui ==
- Definiți perechea de operatori **<= și >=**. Hint: o variantă elegantă este folosirea operatorilor < și >
- Salvați și compilați proiectul. Salvați eventualele erori.

Testarea constructorilor, a metodei ToString și a operatorilor relaționali

- Adăugați o nouă clasă proiectului, **TestOperatori**.
- Scrieți în această clasă o metodă **Main** și setați-o ca metodă de început a aplicației
- Creați două variabile de tip **Rational**, r1 și r2. Instanțiați-le folosind perechile de valori (1, 2) și (1, 3)
- Afișați-le folosind **Console.WriteLine** pentru a testa metoda **ToString**
- Efectuați următoarele comparații, afișând rezultatul:
  - Este r1 > r2?
  - Este r1 <= r2?
  - Este r1 != r2?
- Compilați și rulați programul. Verificați rezultatele
- Modificați r2, instanțiind-o la (2, 4).
- Compilați și rulați programul din nou. Verificați rezultatele.