2. Übung, Thema: Operator Overloading

Einfügung

Gerade wenn es ums Geld mit hohen Beträgen geht, die Ungenauigkeiten durch Rundungen können teuer werden. Deshalb implementiere eine Klasse welche komplett ohne Kommazahlen aus kommt. Dabei wollen wir mit Hilfe von Operator Overloading und Properties noch eine besonderen komfortablen Umgang mit dieser Klasse ermöglichen.

---

1. Klasse Euro mit Properties und der Methode ToString

Eine Instance der Klasse Euro soll mit folgenden Konstruktor erstellt werden:

```
public Euro(long euro,long cents)
```

```
public Euro(long totalCents)
```

- Parameter "*euro*" ist hierbei die Anzahl von euro.
- Parameter "*cents*" ist hierbei die Anzahl von cents.
- Parameter "*totalCents*" ist hierbei die Anzahl von cents wenn auch der Euro Betrag in cents umgerechnet wird.

> ***Beispiel:*** *Ein Geldbetrag 2,50 euro = new Euro(2L, 50L);*

Property TotalCents soll folgendermassen deklariert werden.

```
public long TotalCents { get; private set; }
```

Dieses Property soll den Geld Betrag, komplet umgerechnet in Cents, zurück gegeben.

Property Cents: soll ausserhalb abrufbar sein aber nicht veränderbar sein. Der Wert soll vom Typ **long** sein. Der Wert entspricht der Anzahl von Cents nach dem Komma des Euro Betrags.

Property EuroAmount: soll ausserhalb abrufbar sein aber nicht veränderbar sein. Der Wert soll vom Typ **long** sein. Der Wert entspricht der Anzahl von Euros ohne der restlichen Cents.

The Method ToString gibt eine textuelle Representation von Geldbetrag zurück

> ***Beispiel:*** *new Euro(2L, 5L).ToString() = "Euro: 2, Cents: 5";*

Nutze folgendes Program zum Testen:

```
Euro euroMoney = new Euro(5L, 30L);
Console.WriteLine($"euroMoney.TotalCents = {euroMoney.TotalCents}");
Console.WriteLine($"euroMoney.Cents = {euroMoney.Cents}");
Console.WriteLine($"euroMoney.EuroAmount = {euroMoney.EuroAmount}");
Console.WriteLine(euroMoney);
euroMoney = new Euro(-2L, 10L);
Console.WriteLine($"euroMoney.TotalCents = {euroMoney.TotalCents}");
Console.WriteLine($"euroMoney.Cents = {euroMoney.Cents}");
Console.WriteLine($"euroMoney.EuroAmount = {euroMoney.EuroAmount}");
Console.WriteLine(euroMoney);
euroMoney = new Euro(510L);
Console.WriteLine($"euroMoney.TotalCents = {euroMoney.TotalCents}");
Console.WriteLine($"euroMoney.Cents = {euroMoney.Cents}");
```

```
Console.WriteLine($"euroMoney.EuroAmount = {euroMoney.EuroAmount}");
Console.WriteLine(euroMoney);
```

Ausgabe sollte folgendes sein:

---

euroMoney.TotalCents = 530 euroMoney.Cents = 30
euroMoney.EuroAmount = 5
Euro: 5, Cents: 30
euroMoney.TotalCents = -190
euroMoney.Cents = -90
euroMoney.EuroAmount = -1
Euro: -1, Cents: -90
euroMoney.TotalCents = 510
euroMoney.Cents = 10
euroMoney.EuroAmount = 5
Euro: 5, Cents: 10

---

5. Implementierung von + und - operator

Es soll nun möglich sein, Instanzen der Klasse mit einander zu addieren und subtrahieren. Dazu müssen die Operatoren + und - überladen werden. Dabei soll ein neue Instance entstehen welche das Ergebnis aus der Addition/Subtraktion darstellt.

Hier ein Link zum Thema als Overloading:

https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/operators/operator-overloading

Test Programm:

```
Euro euroMoney = new Euro(2L, 80L);
Euro summand = new Euro(1L, 50L);

Console.WriteLine($"euroMoney + summand = {euroMoney + summand}");
Console.WriteLine($"euroMoney - summand = {euroMoney - summand}");
Console.WriteLine($"summand - euroMoney = {summand - euroMoney}");

euroMoney += summand;
Console.WriteLine($"euroMoney += summand; euroMoney = {euroMoney}");
euroMoney -= summand;
Console.WriteLine($"euroMoney -= summand; euroMoney = {euroMoney}");
```

Ausgabe:

---

euroMoney + summand = Euro: 4, Cents: 30
euroMoney - summand = Euro: 1, Cents: 30
summand - euroMoney = Euro: -1, Cents: -30
euroMoney += summand; euroMoney = Euro: 4, Cents: 30
euroMoney -= summand; euroMoney = Euro: 2, Cents: 80

5.1 Implementierung von + und - Operatoren mit einen anderen Datentypen

Nun soll die Operatoren + und - auch überladen werden mit Berechnung von Integer Werten und Euro Instanzen.

Test Programm:

```
Euro euro = new Euro(0L);

Console.WriteLine($"0 + 60 = {euro + 60}");
Console.WriteLine($"120 + 0 = {120 + euro}");
Console.WriteLine($"0 + 240 = {euro - 240}");
euro += 200;
Console.WriteLine($"120 - 200 = {120 - euro}");
```

Ausgabe:

---

0 + 60 = Euro: 0, Cents: 60
120 + 0 = Euro: 1, Cents: 20
0 + 240 = Euro: -2, Cents: -40
120 - 200 = Euro: 0, Cents: -80

---

6. Implementierung von ++ und -- operator

Nun soll der Operator ++ und -- noch implementiert werden.

```
Euro euroMoney = new Euro(0L);

euroMoney++;
Console.WriteLine($"euroMoney++; euroMoney = {euroMoney}");
euroMoney--;
euroMoney--;
Console.WriteLine($"euroMoney++; euroMoney = {euroMoney}");
```

Ausgabe:

---

euroMoney++; euroMoney = Euro: 0, Cents: 1
euroMoney++; euroMoney = Euro: 0, Cents: -1

---

7. Implementierung von implicit und explicit Cast

- Implementiere nun eine implizite Conversion so dass eine Euro Instanze zu einem long Wert konvertiert werden kann. Dabei soll der Cast nicht explizit aufgerufen werden müssen.
- Implementiere nun eine explizite Conversion so dass eine Euro Instanze in einen bool Wert konvertiert werden kann. Der boolean ist false falls der Geldbetrag negative ist, ansonsten true. Dabei muss dann der Cast explizit aufgerufen werden.

Hier ein Link wie das mit den Casting funktioniert:

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/user-defined-conversion-operators

Test Program:

```
Euro positiveEuro = new Euro(2L, 0L);
long totalCents = positiveEuro;
Console.WriteLine($"totalCents = {totalCents}");
```

```
if ((bool)positiveEuro)
{
    Console.WriteLine("Euro is positive");
}

Euro negativeEuro = new Euro(-1L, 0L);
if (!(bool)negativeEuro)
{
    Console.WriteLine("Euro is negative");
}
```

Ausgabe:

---

totalCents = 200
Euro is positive
Euro is negative
