6 Délégués

Exercice 31.
1. Déclarer un type délégué `IntAction` qui prend en argument un variable de type `int` et qui ne renvoie rien.
2. Ecrire une méthode statique `PrintInt` qui a un seul argument de type `int` et qui l'affiche sur la console.
3. Dans la méthode `Main`, déclarer une variable `act` de type `IntAction` et lui assigner la méthode `PrintInt`. Exécuter l'instruction `act(42)`.
4. Ecrire une méthode de signature
   `static void Perform(IntAction act, int[] arr)`
   qui applique le délégué `act` à tous les éléments du tableau `arr`. On utilisera une boucle `foreach`.
5. Créer un tableau `arr` et exécuter l'instruction `Perform(act, arr)`.
6. Ajouter la méthode `Console.WriteLine` au délégué `act` et exécuter de nouveau `Perform(act, arr)`.

Exercice 32. Le fichier ci-dessous déclare deux types de délégué :
`public delegate bool IntPredicate(int x);`
`public delegate void IntAction(int x);`
ainsi qu'une classe `IntList` qui dérive de `List<int>` et dont deux méthodes utilisent `IntPredicate` et `IntAction` :
- `Act(f)` qui applique le délégué `f` à tous les éléments de la liste.
- `Filter` qui crée une nouvelle liste contenant les éléments `x` de la liste tels que `p(x)` est vraie.

```
using System;
using System.Collections.Generic;

public delegate bool IntPredicate(int x);
public delegate void IntAction(int x);

// Une IntList contenant les éléments 7 9 13 peut-être construit avec
// new IntList(7, 9, 13) grâce au modificateur params.

class IntList : List<int> {

  public IntList(params int[] elements) : base(elements) { }

  public void Act(IntAction f) {
    foreach (int i in this)
      f(i);
  }
```

```
  public IntList Filter(IntPredicate p) {
    IntList res = new IntList();
    foreach (int i in this)
      if (p(i)) res.Add(i);
    return res;
  }
}

class Test {
  public static void Main(String[] args) {
    // code à ajouter ici
  }
}
```

1. Compléter la méthode `Main` afin de créer un liste `xs` de type `IntList` contenant les valeurs `12, 26, 33, 2`.
2. Expliquer ce que produisent les expressions

   ```
   xs.Act(Console.WriteLine);
   (xs.Filter(x => x%2==0)).Act(Console.WriteLine);
   ```

3. Utiliser une méthode anonyme pour écrire une expression qui affiche seulement les éléments de `xs` qui sont supérieurs à 25.
4. En utilisant la méthode `Act` et le fait q'une méthode anonyme peut faire référence à des variables extérieures, calculer la somme des éléments de `xs` sans explicitement écrire de boucle.
5. Adapter le programme en entier afin de pouvoir manipuler de la même manière une liste générique (en définissant une classe qui dérive de `List<T>` et en utilisant les types génériques de délégué, de la BCL, `Predicate<T>` et `Action<T>`).

Exercice 33. Ecrire une classe permettant le calcul de l'aire de sous-ensemble du carré $[0,1]^2$ par la méthode de Monte-Carlo. On doit pouvoir écrire (par exemple)
`CalculAire((x,y)=> x*x+y*y <1.0,1000);`
pour obtenir une valeur approchée de l'aire du quart de disque de rayon 1, obtenue à partir de 1000 tirages aléatoires d'un point dans le carré.