5 TP 5

Exercice 26. Modifier la classe représentant des fractions de l'exercice 25 (TP 4) de sorte qu'elle "implémente" l'interface `IComparable`. Pour faire un test, créer une liste d'objets (un `ArrayList`). Ajouter quelques fractions à cette liste et trier la liste en invoquant la méthode `sort`. Afficher la liste triée en utilisant une boucle `foreach`.

Exercice 27. Que produit la méthode `Affiche` ci-dessous ? Construire un objet qui fasse que cette méthode `Affiche` ne se termine jamais quand il est passé en argument.

```
using System;
using System.Collections;

public static void Affiche(object ob)
{
  IEnumerable obEnu = ob as IEnumerable;
  if (obEnu != null)
  {
    foreach (object x in obEnu)
    {
      Affiche (x);
    }
  }
  else Console.Write(ob+ " ");
}
```

Exercice 28. On veut que la classe `ListeFacteursPremiers` ci-dessous permette d'obtenir l'ensemble des facteurs premiers d'un entier n (représenté par un `uint`) dans une boucle `foreach`. Pour cela la classe `ListeFacteursPremiers` "implémente" l'interface `IEnumerable` de l'espace de noms `System.Collections`. Le code est volontairement incomplet. On demande de terminer l'écriture de cette classe.

```
public class ListeFacteursPremiers : IEnumerable
{
  uint n;

  class EnumFacteursPremiers : IEnumerator
  {
    // attributs
    ...

    public EnumFacteursPremiers(uint n){...}

    public bool MoveNext (){...}

    public void Reset (){...}

    public object Current
    {
     get {...}
    }
  }

  public ListeFacteursPremiers (uint n){...}

  public IEnumerator GetEnumerator ()
  {
    return new EnumFacteursPremiers (n);
  }
}
```

Exercice 29. Une structure générique de type `Pair <T, U>` peut être définie par

```
public struct Pair<T, U>
{
public T Fst { get; private set; }
public U Snd { get; private set; }
public Pair(T fst, U snd) : this() { Fst=fst; Snd=snd; }
public override string ToString() { return "[" + Fst + "," + Snd + "]";}
}
```

1. Créer un projet, implémenter ce code, vérifier qu'il compile.
2. Déclarer une variable de type `Pair < string, int>` et lui affecter des valeurs.
3. Déclarer une variable de type `Pair <string, double>` et lui affecter des valeurs.
4. Pouvez-vous assigner la variable déclarée en 2 dans celle déclarée en 3 ?

5. Déclarer une variable grades de type `Pair<string, int>[]`. Créer un tableau de 5 éléments et affecter les lignes 0,1 et 2.
6. Utiliser un foreach pour itérer sur le tableau grades et afficher tous ces éléments. Quelles sont les valeurs que vous n'avez pas assignées ?
7. Déclarer une variable `appointment` de type `Pair< Pair<int, int>, string>` et créer une valeur de ce type, puis l'affecter à la variable.
   Quel est le type de `appointment.Fst.Snd` ?
8. Déclarer et implémenter une méthode Swap permettant dans `Pair<T,U>` de retourner une nouvelle structure `Pair<U,T>` en échangeant les deux éléments.

Exercice 30 (Jeu à deux joueurs)**.** On considère une énumération `Resultat` et trois classes `Position`, `Partie` et `Joueur` pour modéliser un jeu à deux joueurs (cf le code ci-dessous).

Description des classes Les trois premières valeurs de l'énumération `Resultat` correspond aux trois résultats possibles d'une partie : le joueur 1 gagne, le joueur 0 gagne, partie nulle. La dernière valeur `Resultat.indertermine` correspond à une partie qui n'est pas terminée et dont l'issue n'est pas déterminée.

La classe abstraite `Position` définit un type d'objet représentant l'état du jeu à un moment donné de la partie. En plus d'un constructeur, la classe `Position` a 3 membres :

— Un attribut `bool j1aletrait` qui détermine quel joueur a le trait (`true` si le joueur 1 a le trait)
— Un constructeur `Position(bool j1aletrait)` permattant d'initialiser l'attribut précédent.
— Une méthode abstraite `Resultat Eval()` qui renvoie le résultat de la partie (une des 3 valeurs `Resultat.j1gagne`, `Resultat.j0gagne`, `Resultat.partieNulle`) si la position courante est une position terminale, c'est-à-dire pour laquelle l'issue du jeu est connue, ou sinon la valeur `Resultat.indetermine`.

La classe abstraite `Joueur` définit un type d'objet représentant un joueur et a un seul membre, la méthode `public abstract Position Jouer(Position p)`, qui doit renvoyer la nouvelle position obtenue suite au coup joué à partir de la position `p`.

La classe `Partie` fait jouer chaque joueur à son tour et afficher les positions jusqu'à la fin de la partie.

Code

```
public enum Resultat{ j1gagne, j0gagne, partieNulle, indetermine }

public abstract class Position{
 public bool j1aletrait;
 public Position(bool j1aletrait){ this.j1aletrait=j1aletrait; }
 public abstract Resultat Eval();
}
```

```
public abstract class Joueur{
 public abstract Position Jouer(Position p);
}

public class Partie{
 Position pCourante;
 Joueur j1,j0;
 public Resultat Re;

 public Partie(Joueur j1, Joueur j0, Position pInitiale){
  this.j1 = j1; this.j0 = j0; pCourante = pInitiale;
 }

 public void Commencer(){
  Resultat r;
  Affiche ();
  do {
   if (pCourante.j1aletrait) {pCourante = j1.Jouer(pCourante);}
   else {pCourante = j0.Jouer(pCourante);}
   Affiche ();
   r = pCourante.Eval ();
  }
  while(r == Resultat.indetermine);
  Re = r;
  switch (r){
   case Resultat.j1gagne : Console.WriteLine("j1 a gagné."); break;
   case Resultat.j0gagne : Console.WriteLine("j0 a gagné."); break;
   case Resultat.partieNulle : Console.WriteLine("Partie nulle."); break;
   }
  }
 void Affiche(){Console.WriteLine(pCourante);}
}
```

Soit le jeu suivant. On fixe un nombre réel s strictement positif qui représente un seuil à franchir. Chacun des deux joueurs commence la partie avec un score de 0. A chaque tour, chaque joueur fabrique un nombre aléatoire dont l'espérance doit être égal à 1 et l'ajoute à son score. On considère que les joueurs jouent simultanément. Le premier joueur dont le score dépasse strictement s a gagné. Si les deux scores franchissent simultanément le seuil s, la partie est nulle.

1. Concevoir une classe `PositionTA` dérivée de `Position` pour représenter un état possible du jeu. On peut utiliser trois attributs de type `Double` (le seuil, le score du joueur 1 et le score du joueur 0). Comme on considère que les joueurs jouent simultanément, on convient que le joueur 1 commence toujours et on fait en sorte que la

méthode `Eval()` renvoie la valeur `Resultat.indetermine` si le joueur 1 n'a pas le trait (ce qui signifie qu'il vient de jouer).

2. Concevoir une classe `JoueurUnif` dérivée de `Joueur` correspondant à un joueur utilisant des nombres aléatoires de loi uniforme sur l'intervalle $[0,2]$, et une classe `JoueurExp` correspondant à la loi exponentielle (utiliser que $-\log(U)$ suit une loi exponentielle de paramètre 1 si U suit une loi uniforme).
3. Dans la méthode `Main`, simuler 1000 parties entre `JoueurUnif` et `JoueurExp`. On comptera le nombre de victoires de chacun et le nombre de parties nulles.
4. Trouver une façon de jouer (une classe dérivée de `Joueur`) pour laquelle la probabilité de gagner contre `JoueurUnif` sachant que la partie n'est pas nulle soit strictement plus grande que $1/2$.