ISIMA ZZ2F2 (promo 2011) - Examen de .NET
Durée 2H – Documents de TP autorisés uniquement

Concepts .Net

1 – Expliquez par quels mécanismes la plate-forme .Net permet l'interopérabilité entre des langages différents (C#, J#, VB.Net, etc.). (2 pts)

2 – Expliquez le principe de compilation JIT. (2 pts)

4 – Qu'est-ce qu'un *assembly* et quelle est son utilité ? (1 pt)

Framework .Net

1 – Décrivez l'architecture retenue dans le framework .Net pour la manipulation de flux d'entrées/sorties. (2.5 pts)

2 – Quel est l'intérêt d'utiliser les génériques plutôt que la classe de base universelle *object* ? (1 pt)

3 – Quels sont les avantages et les inconvénients d'utiliser des exceptions plutôt que des codes d'erreurs ? (2 pts)

Implémentation

1 – Quelle sont les différences entre un type valeur et un type référence ? (1 pt)

2 – En C#, pour indiquer qu'un type est comparable, il doit implémenter l'interface `IComparable` :

```
using System;

public interface IComparable<T> {
    int CompareTo(T comparee);
}
```

La méthode `CompareTo` doit renvoyer un entier < 0 si le comparant est inférieur au comparé, == 0 s'ils sont égaux et > 0 si le comparant est supérieur au comparé.
Cependant, utiliser la méthode `CompareTo` peut parfois être fastidieux ; il serait pratique d'obtenir automatiquement des méthodes `GreaterThan`, `LowerThan`, `Equals`, etc.

a) **Définissez une classe MyComparable, qui fournisse directement aux classes dérivées ce type de méthode.** (3 pts) Voici un exemple d'utilisation de cette classe :

```
public class Foo : MyComparable<Foo> {
    public int i;
    public override int CompareTo(Foo comparee) {
        return this.i - comparee.i;
    }
}
```

```
...
Foo f1 = new Foo();
Foo f2 = new Foo();

f1.i = 1;
f2.i = 9;


// En étendant MyComparable, Foo obtient directement les fonctions
// LowerThan, LowerThanOrEquals, Equals, GreaterThanOrEquals, GreaterThan
Console.WriteLine(f1.LowerThan(f2)); // true
Console.WriteLine(f1.Equals(f2));    // false
...
```

b) En utilisant MyComparable, écrivez une fonction (méthode statique) générique qui accepte deux paramètres et renvoie le plus grand des deux. (1.5 pts)

3 – Le code suivant présente une classe `Simulator`, qui possède un état (un entier) et fait évoluer cet état pendant un certain temps (cf. méthode `Run`). La classe `Controller` sert à contrôler le simulateur : dans la réalité, elle ferait le lien entre l'interface graphique et le simulateur. Elle possède des propriétés pour spécifier l'état initial du simulateur, la durée de simulation souhaitée, et pour accéder à l'état du simulateur. La classe `Program` initialise et lance une simulation (encore une fois, dans la réalité, la classe `Program` lancerait une interface graphique qui elle interagirait avec le contrôleur).

```
using System;

namespace Simulateur
{
    class Simulator
    {
        public Simulator(int initialState, int endTime)
        {
            Time  = 0;
            State = initialState;
            end   = endTime;
        }

        public void Run()
        {
            while (Time < end)
            {
                computeNextState();
                ++Time;
            }
        }

        private void computeNextState()
        {
            ++State;
        }

        public int Time  { get; private set; }
        public int State { get; private set; }

        private int end;
    }
```

```
public class Controller
{
    public int InitialState { get; set; }
    public int EndTime      { get; set; }

    public int SimState
    {
        get { return sim.State; }
    }

    public void LaunchSimulation()
    {
        sim = new Simulator(InitialState, EndTime);
        sim.Run();
    }

    private Simulator sim;
}

public class Program
{
    static void Main(string[] args)
    {
        Controller con   = new Controller();
        con.InitialState = 0;
        con.EndTime      = 1000;

        con.LaunchSimulation();

        Console.WriteLine(con.SimState);
    }
}
}
```

On souhaite ajouter à ce code des fonctionnalités de logging : à chaque pas de simulation, on aimerait qu'un message soit enregistré dans un fichier, indiquant le temps de simulation et la valeur courante de l'état. Ce logging doit pouvoir être activé ou désactivé, et le nom du fichier doit être paramétrable. D'autre part, l'utilisateur a également besoin de pouvoir sauvegarder l'état de la simulation à un instant t paramétrable (pour pouvoir redémarrer une simulation à partir d'un état stable avec des paramètres différents).

Utilisez la programmation événementielle pour implémenter ces fonctionnalités. Vous pouvez créer autant de classes que vous le souhaitez, et modifier le contrôleur et le simulateur à votre guise. (4 pts)