BOURGEOIS Charles SIO2
StJoSup, Le Havre

Compte-rendu
Interfaces

2022

Table des matières

1) Qu’est-ce qu’une interface ?

Une interface est un ensemble de signatures de méthodes publiques d’un objet, elle possède un niveau d’abstraction supérieur à celui des classes abstraites.

Les interfaces ont des méthodes « sans corps », donc des méthodes implicitement abstraites. La particularité des interfaces est qu'elles ne peuvent pas être instanciées.

Les interfaces peuvent être implémentées dans une classe, l’un des intérêts principaux liés à la définition de celle-ci est que l’utilisateur vas se retrouver obligé de suivre le plan donné dans cette interface. En effet, quand une classe implémente une interface, elle s’engage à se conformer à celle-ci et à définir/utiliser toutes les méthodes de l’interface, c’est une sorte de contrat à respecter.

Il est important de distinguer les classes abstraites aux interfaces, les interfaces ne peuvent contenir que les signatures des méthodes, contrairement aux classes abstraites.

En programmation, nous pouvons créer notre propre interface ou utiliser une interface déjà existante comme Icloneable, une interface contenant des méthodes permettant de « cloner ». Pour définir une interface, il faut ajouter le mot clef « *interface* » à la place de « *class* ».

2) Définition des méthodes sous forme algorithmique

Premièrement, intéressons-nous à l'interface IStats, créé dans le cadre d'un TP.

Nous déclarons l'interface suivante :

```
namespace Interface
{
    5 références
    interface IStats
    {
        2 références
        double Moyenne();
        2 références
        double EcartType(List<double>lesNombres);

        4 références
        void AjouteNote(double note);
    }
}
```

Qui contient donc une méthode Moyenne (), EcartType(lesNombres : collection de réel) et AjouteNote(réel : note).

Définissons maintenant la classe NoteEleve, qui utilisera l'interface IStats :

```
1 référence
class NoteEleve : IStats
{
    private List<double> lesNombres = new List<double>();
    private double resultat;
    private double cumul;
```

La particularité est que nous ajoutons, avant nos méthodes, « *NomDeL'interface.NomDeLaMethode* »

```
2 références
double IStats.EcartType(List<double> sequence)
{
    double result = 0;

    if (sequence.Any())
    {
        double average = sequence.Average();
        double sum = sequence.Sum(d => Math.Pow(d - average, 2));
        result = Math.Sqrt((sum) / sequence.Count());
    }
    return result;
}
```

Nous pourrons donc utiliser cette interface pour n'importe quelles données, où nous souhaitons calculer la moyenne ou l'écart type, des notes scolaires, des températures…

Dans mon cas personnel, j'ai créé une interface de gestion de boîte de pilule.

```
4 références
interface IActions
{
    2 références
    void Ajouter(Pilule objet);
    1 référence
    void Retirer(Pilule objet);


}
```

Dans ma classe Pilule, l'interface sera utilisé de la sorte :

```
class Pilule :IActions
{
    private List<Pilule> lesPilules = new List<Pilule>();
    string couleur;
    string nom;

    2 références
    public Pilule(string couleurSai, string nomSai)
    {
        this.couleur = couleurSai;
        this.nom = nomSai;
    }
    2 références
    void IActions.Ajouter(Pilule pilule)
    {
        lesPilules.Add(pilule);
    }
```

Passons maintenant à l'utilisation d'une interface déjà existante, avec l'interface Icloneable.

Créeons une classe voiture, qui utilise l'interface ICloneable

```
6 références
class  voiture : ICloneable
{
    int width;

    2 références
    public voiture(int widht)
    {
        this.width = widht;
    }

    1 référence
    public object Clone()
    {
        return new voiture(this.width);
    }

    0 références
    public override string ToString()
    {
        return string.Format("Width of car = " + this.width);
    }
}
```

Dans notre programme de test nous pouvons remarquer que notre voiture2 sera cloné de la voiture1 grâce à la méthode Clone() :

```
0 références
static void Main(string[] args)
{
    voiture carOne = new voiture(1695);
    voiture carTwo = carOne.Clone() as voiture;

    Console.WriteLine("{0}mm", carOne);
    Console.WriteLine("{0}mm", carTwo);
}
```

3) Conclusion

Après avoir vu précédemment les classes abstraites, nous nous sommes intéressés aux interfaces, qui à première vue se ressemblent mais qui se trouvent être totalement différents, notamment par le fait qu'une interface ne peut pas être instancié.

Les interfaces sont donc utiles pour forcer un développeur à utiliser toutes les méthodes, présentes dans l'interface, dans sa classe personnelle.