

Les patrons de conceptions I



Objectifs

- Notion de patrons de conception
- Patron Singleton
- Patron Strategie

Patrons de conception

- Origines tirées des travaux de l'architecte Christopher Alexander dans les années 70
- Propose des solutions basées sur l'expérience à des problèmes classiques
- Propose un vocabulaire commun entre les différents intervenants
- Chaque patron est organisé en trois parties :
 - Contexte
 - Problème
 - Solution
- Donc une solution pour un problème dans un contexte donné

Patrons de conception

- Principaux « *Design Patterns: Elements of Reusable Software* » (Gamma, Helm, Johnson et Vlissides – Gang of Four GoF)
- 23 patrons du GoG répartis en 3 familles de problèmes :
 - Créateurs :
 - Instanciation et paramétrisation des classes et objets => **Création d'objets**
 - Ces patrons aident à créer des objets pour vous au lieu de le faire directement (new)
 - Structuraux :
 - Définissent comment organiser les classes d'une application => **Composition d'objets**
 - Ces patrons aident à composer des groupes d'objets en structures plus grandes
 - Comportementaux :
 - Organisent les objets pour qu'ils collaborent => **interaction entre les objets**
 - Ces patrons aident à définir la communication entre les objets

Singleton (Créateur)

Singleton

Type: Creational

What it is:

Ensure a class only has one instance and provide a global point of access to it.

Singleton
-static uniqueInstance
-singletonData
+static instance()
+SingletonOperation()

!



Singleton – C# version 1

```
public class SingletonV1_MethodeStatique
{
    private static SingletonV1_MethodeStatique _instance;

    public static SingletonV1_MethodeStatique ObtenirInstance()
    {
        if (_instance is null)
        {
            _instance = new SingletonV1_MethodeStatique();
        }

        return _instance;
    }

    public string ExempleMehodeDeVotreInstanceUnique()
    {
        return "Je suis une instance unique !";
    }
}
```

Singleton – C# version 2

```
public class SingletonV2_ProprieteStatique
{
    private static SingletonV2_ProprieteStatique _instance;

    public static SingletonV2_ProprieteStatique Instance {
        get
        {
            if (_instance is null)
            {
                _instance = new SingletonV2_ProprieteStatique();
            }

            return _instance;
        }
    }

    public string ExempleMehodeDeVotreInstanceUnique() {
        return "Je suis une instance unique !";
    }
}
```

Singleton – C# version 3



```
public class SingletonV3_ThreadSafe {
    private static SingletonV3_ThreadSafe _instance;
    private static object _lock = new object();

    public static SingletonV3_ThreadSafe Instance {
        get {
            if (_instance is null) {
                lock (_lock) {
                    if (_instance is null) {
                        _instance = new SingletonV3_ThreadSafe();
                    }
                }
            }
            return _instance;
        }
    }

    public string ExempleMehodeDeVotreInstanceUnique() {
        return "Je suis une instance unique !";
    }
}
```

Singleton C# – Utilisation

```
SingletonV3_ThreadSafe s = SingletonV3_ThreadSafe.Instance;  
s.ExempleMehodeDeVotreInstanceUnique();  
  
// Ou  
  
SingletonV3_ThreadSafe.Instance.ExempleMehodeDeVotreInstanceUnique();
```

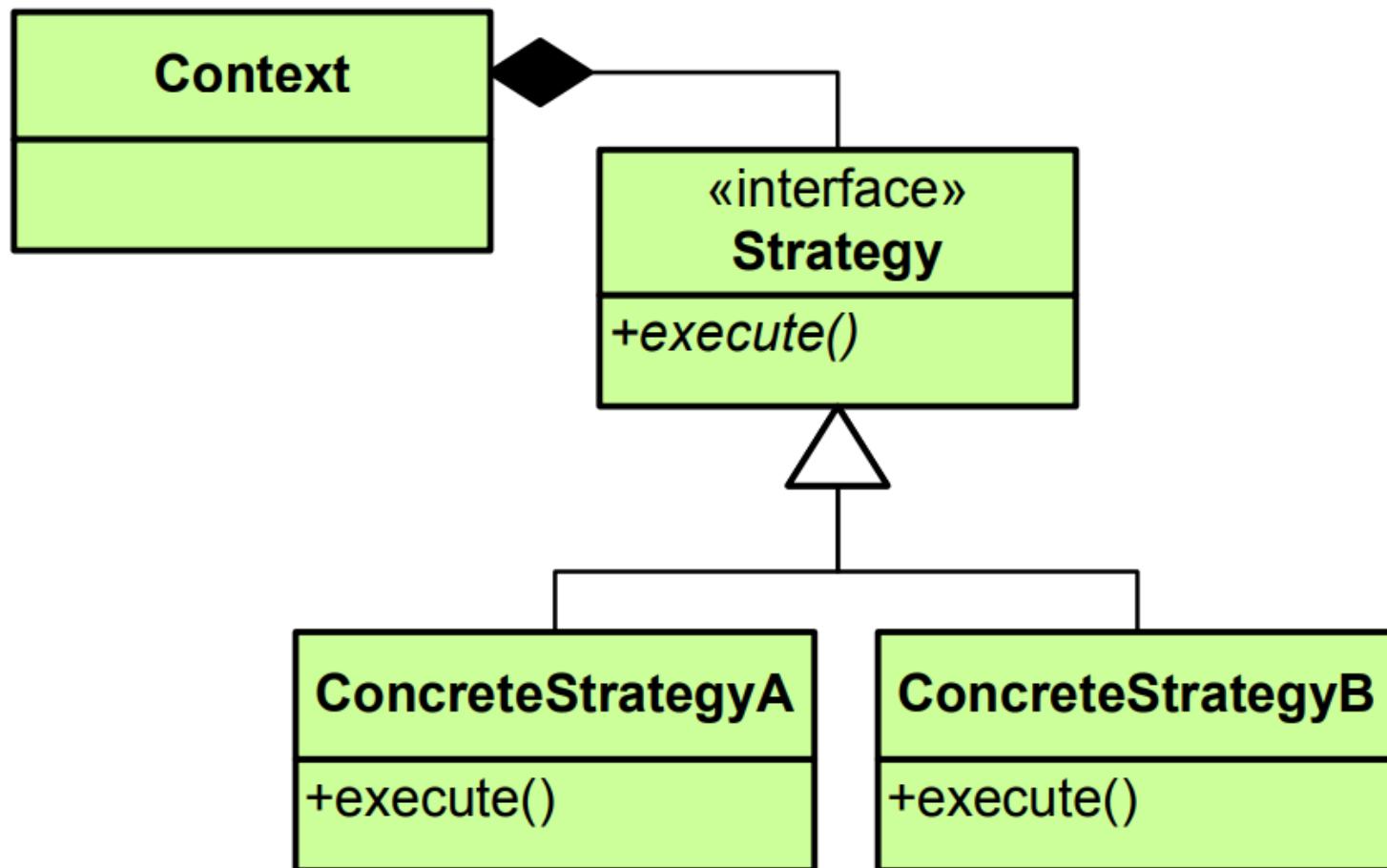
Stratégie

Strategy

Type: Behavioral

What it is:

Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



Stratégie – C# Version 1

```
public interface SaluerStrategieV1 {
    void Executer();
}

public class SaluerConsoleFrancaisStrategieV1 : SaluerStrategieV1 {
    public void Executer()
    {
        Console.Out.WriteLine("Bonjour !");
    }
}

public class SaluerConsoleAnglaisStrategieV1 : SaluerStrategieV1 {
    public void Executer()
    {
        Console.Out.WriteLine("Hello !");
    }
}
```

Stratégie – C# Version 1 – Application appelante

```
public class ApplicationXYZUIV1 {
    public SaluerStrategieV1 Saluer { get; set; }

    public ApplicationXYZUIV1(SaluerStrategieV1 p_saluer) {
        if (p_saluer is null) {
            throw new ArgumentNullException();
        }

        this.Saluer = p_saluer;
    }

    public void AccueillirUtilisateur() {
        if (this.Saluer is null) {
            throw new InvalidOperationException("La stratégie de salutation n'est pas définie");
        }

        this.Saluer.Executer();
    }
}
```

Stratégie – C# Version 1 – Utilisation

```
ApplicationXYZUI app = new ApplicationXYZUI(new SaluerConsoleFrancaisStrategieV1());
app.AccueillirUtilisateur();

// Ou

app.Saluer = new SaluerConsoleAnglaisStrategieV1();
app.AccueillirUtilisateur();
```

Stratégie – C# Version 2

```
public static class SaluerStrategieV2
{
    public static void SaluerConsoleAnglaisStrategieV2()
    {
        Console.Out.WriteLine("Hello !");
    }

    public static void SaluerConsoleFrancaisStrategieV2()
    {
        Console.Out.WriteLine("Bonjour !");
    }
}
```

Stratégie – C# Version 2 – Application appelante

```
public class ApplicationXYZUIV2 {
    public Action Saluer { get; set; }

    public ApplicationXYZUIV2(Action p_saluer) {
        if (p_saluer is null)
        {
            throw new ArgumentNullException();
        }

        this.Saluer = p_saluer;
    }

    public void AccueillirUtilisateur() {
        if (this.Saluer is null) {
            throw new InvalidOperationException("La stratégie de salutation n'est pas définie");
        }

        this.Saluer();
    }
}
```

Stratégie – C# Version 2 – Utilisation

```
ApplicationXYZUIV2 app2 = new ApplicationXYZUIV2(SaluerStrategieV2.SaluerConsoleFrancaisStrategieV2);
app2.AccueillirUtilisateur();

// Ou

app2.Saluer = SaluerStrategieV2.SaluerConsoleAnglaisStrategieV2;
app2.AccueillirUtilisateur();

// Ou

app2.Saluer = () => Console.Out.WriteLine("Kaixo !");
app2.AccueillirUtilisateur();
```



Références

- Livre : *Design Patterns: Elements of Reusable Software* » (Gamma, Helm, Johnson et Vlissides – Gang of Four GoF)
- Article cours en français :
<https://rpouiller.developpez.com/tutoriel/java/design-patterns-gang-of-four/>
- Cartes :
<http://www.mcdonaldland.info/files/designpatterns/designpatterns.pdf>
- Exemple de code : <https://github.com/iluwatar/java-design-patterns>
- Aller plus loin sur les accès concurrents et les différents modes de réservation : <http://www.albahari.com/threading/part2.aspx>