

Creational Patterns :

Abstract Factory Design Pattern :

Les Classes et objets participants au patron :

Abstract Factory :

- A le rôle d'interface pour la création des produits abstraits.

ConcreteFactory :

- Concretise les méthodes CreateProduct de la classe AbstractFactory pour retourner un objet de type Product.

AbstractProduct :

- A le rôle d'une interface pour les objets de type Product.

Product :

- définit l'objet Product créer par la concreteFactory correspondante.
- Implémente l'interface AbstractProduct

Client:

- Utilise les interfaces déclarer par AbstractFactory et AbstractProduct.

Code Csharp généré automatiquement	Code PADL
<pre> //AbstractFactory public abstract class AbstractFactory { public abstract AbstractProductA CreateProductA(); public abstract AbstractProductB CreateProductB(); } //AbstractProductA public abstract class AbstractProductA { } //AbstractProductB public abstract class AbstractProductB { } public abstract void Interact(AbstractProductA a); } </pre>	<pre> //*****AbstractFactory Class***** IClass AbstractFa = Factory.getInstance().createClass("AbstractFactory"); AbstractFactory.addActor(AbstractFa); AbstractFa.setAbstract(true); //CreateProductA Method IMethod createPA = Factory.getInstance().createMethod("CreateProductA"); AbstractFa.addActor(createPA); createPA.setAbstract(true); createPA.setReturnType("AbstractProductA"); //CreateProductB Method IMethod createPB = Factory.getInstance().createMethod("CreateProductB"); AbstractFa.addActor(createPB); createPB.setAbstract(true); createPB.setReturnType("AbstractProductB"); //*****AbstractProductA Class***** IClass AbstractProductA = Factory.getInstance().createClass("AbstractProductA"); AbstractFactory.addActor(AbstractProductA); AbstractProductA.setAbstract(true); //*****AbstractProductB Class***** IClass AbstractProductB = Factory.getInstance().createClass("AbstractProductB"); AbstractFactory.addActor(AbstractProductB); AbstractProductB.setAbstract(true); //Interact Method IMethod interact = Factory.getInstance().createMethod("Interact"); interact.setAbstract(true); AbstractProductB.addActor(interact); IParameter AbstractProduct = Factory.getInstance().createParameter("a", "AbstractProductA"); interact.addActor(AbstractProduct); </pre>

```

//ConcreteFactory1
public class ConcreteFactory1 : AbstractFactory {
    public override AbstractProductA CreateProductA() {
        return new ProductA1();
    }
    public override AbstractProductB CreateProductB() {
        return new ProductB1();
    }
}
//ConcreteFactory2
public class ConcreteFactory2 : AbstractFactory {
    public override AbstractProductA CreateProductA() {
        return new ProductA2();
    }
    public override AbstractProductB CreateProductB() {
        return new ProductB1();
    }
}

//***** ConcreteFactory1 Class*****
IClass ConcreteFactory1 =
Factory.GetInstance().createClass("ConcreteFactory1");
AbstractFactory.AddActor(ConcreteFactory1);
ConcreteFactory1.AddInheritedActor(AbstractFa);
//CreateProductA Method
ICSharpMethod CreateFaA =
CSharpFactory.GetCSharpInstance().createNewMethod("CreateProductA");
ConcreteFactory1.AddActor(CreateFaA); CreateFaA.setOverride(true);
CreateFaA.setReturnType("AbstractProductA");
CreateFaA.SetReturnValue("new ProductA1()");
//CreateProductB Method
ICSharpMethod CreateFaB =
CSharpFactory.GetCSharpInstance().createNewMethod("CreateProductB");
ConcreteFactory1.AddActor(CreateFaB); CreateFaB.setOverride(true);
CreateFaB.setReturnType("AbstractProductB");
CreateFaB.SetReturnValue("new ProductB1()");

//***** ConcreteFactory2 Class*****
IClass ConcreteFactory2 =
Factory.GetInstance().createClass("ConcreteFactory2");
AbstractFactory.AddActor(ConcreteFactory2);
ConcreteFactory2.AddInheritedActor(AbstractFa);
//CreateProductA Method
ICSharpMethod CreateFaAA =
CSharpFactory.GetCSharpInstance().createNewMethod(
"CreateProductA"); ConcreteFactory2.AddActor(CreateFaAA);
CreateFaAA.setOverride(true);
CreateFaAA.setReturnType("AbstractProductA");
CreateFaAA.SetReturnValue("new ProductA2()");
//CreateProductB Method
ICSharpMethod CreateFaBB =
CSharpFactory.GetCSharpInstance().createNewMethod("CreateProductB");
ConcreteFactory2.AddActor(CreateFaB); CreateFaBB.setOverride(true);
CreateFaBB.setReturnType("AbstractProductB");
CreateFaBB.SetReturnValue("new ProductB2()");

```

```

//Client
public class Client {

    private AbstractProductA AbstractProductA;

    private AbstractProductB AbstractProductB;
    public Client(AbstractFactory factory) {

        this.AbstractProductA = factory.CreateProductA();
    }

    //ProductA1
    public class ProductA1 : AbstractProductA {

    }

    //ProductA2
    public class ProductA2 : AbstractProductB {

    }

    //ProductB1
    public class ProductB1 : AbstractProductB {

    }

    //ProductB2
    public class ProductB2 : AbstractProductB {

    }

} Le Code C# Correspondant Abstract Factory Design Motif

```

```

//*****La Classe Client*****
Iclass Client = Factory.GetInstance().createClass("Client");
AbstractFactory.addActor(Client); IField AbstractProductA =
Factory.GetInstance().createField("AbstractProductA", "AbstractProductA", 2);
Client.addActor(AbstractProductA); AbstractProductA.setPrivate(true);
IField AbstractProductB = Factory.GetInstance().createField(
"AbstractProductB", "AbstractProductB", 2);
Client.addActor(AbstractProductB); AbstractProductB.setPrivate(true);
//Client Constructor
IConstructor client =
Factory.GetInstance().createConstructor("Client");
Client.addActor(client); IParameter factory =
Factory.GetInstance().createParameter("factory", "AbstractFactory");
client.addActor(factory); IAffectation affecteA =
StatementFactory.getStateInstance().createAffectation(
AbstractProductA, "factory.CreateProductA()");
client.addActor(affecteA); IAffectation affecteB =
StatementFactory.getStateInstance().createAffectation(
AbstractProductB, "factory.CreateProductB()");
client.addActor(affecteB);
//*****ProductA1 Class*****
Iclass ProductA1 = Factory.GetInstance().createClass("ProductA1");
AbstractFactory.addActor(ProductA1);
ProductA1.addInheritedActor(AbstractProductA);
//*****ProductB1 Class*****
Iclass ProductB1 = Factory.GetInstance().createClass("ProductB1");
AbstractFactory.addActor(ProductB1);
ProductB1.addInheritedActor(AbstractProductB);
//*****ProductA2 Class*****
Iclass ProductA2 = Factory.GetInstance().createClass("ProductA2");
AbstractFactory.addActor(ProductA2);
ProductA2.addInheritedActor(AbstractProductB);
//*****ProductB2 Class*****
Iclass ProductB2 = Factory.GetInstance().createClass("ProductB2");
AbstractFactory.addActor(ProductB2);
ProductB2.addInheritedActor(AbstractProductB);

```

Factory Method Design Pattern :

Les Classes et objets participants au patron :

Product :

- A le rôle d'interface pour les objets que la méthode Factory crée.

ConcreteProduct :

- Implémente l'interface Product.

Creator :

- Déclare la méthode FactoryMethod qui retourne un objet de type Product
- Creator peut aussi définir une implémentation par défaut de la méthode FactoryMethod pour retourner un objet de type ConcreteProduct.
- Peut appeler la méthode Factory pour créer un objet de type Product.

ConcreteCreator :

- définit une implémentation concrète de la méthode Factory pour retourner une instance de ConcreteProduct.

Code Csharp généré automatiquement	Code PADL
<pre> //ConcreteCreatorA class ConcreteCreatorA : Creator { public override Product FactoryMethod() { return new ConcreteProductA(); } } //ConcreteCreatorB public class ConcreteCreatorB : Creator { public override Product FactoryMethod() { return new ConcreteProductB(); } } </pre>	<pre> //***** ConcreteCreatorA Class***** IClass concretecreatorA = Factory.GetInstance().createClass("ConcreteCreatorA"); factoryMethod.addAction(concretecreatorA); concretecreatorA.addInheritedActor(creator); concretecreatorA.setPublic(false); //FactoryMethod Concrete Method ICSharpMethod factoryM = CSharpFactory.getCSharpInstance().createNewMethod("FactoryMethod"); concretecreatorA.addAction(factoryM); factoryM.setOverride(true); factoryM.setReturnType("Product"); factoryM.SetValue("new ConcreteProductA()"); //***** ConcreteCreatorB Class***** IClass concretecreatorB = Factory.GetInstance().createClass("ConcreteCreatorB"); factoryMethod.addAction(concretecreatorB); concretecreatorB.addInheritedActor(creator); //FactoryMethod Concrete Method ICSharpMethod factoryMet = CSharpFactory.getCSharpInstance().createNewMethod("FactoryMethod"); concretecreatorB.addAction(factoryMet); factoryMet.setOverride(true); factoryMet.SetValue("new ConcreteProductB()"); </pre>

```

//ConcreteProductA
public class ConcreteProductA : Product {
//implmente l'interface product
}
//ConcreteProductB
public class ConcreteProductB : Product {
//implmente l'interface product
}
//Creator
public abstract class Creator {
    public abstract Product FactoryMethod();
}
//Product
public abstract class Product {
}
} Le Code C# Correspondant Factory Method Design Motif

```

```

//***** Product Class*****
IClass Product = Factory.GetInstance().createClass("Product");
factoryMethod.addActor(Aproduct); Aproduct.setAbstract(true);
//***** ConcreteProductA Class*****
IClass concreteproductA =
Factory.GetInstance().createClass("ConcreteProductA");
factoryMethod.addActor(concreteproductA);
concreteproductA.addInheritedActor(Aproduct);
concreteproductA.setComment("//implmente l'interface product");
//***** ConcreteProductB Class*****
IClass concreteproductB =
Factory.GetInstance().createClass("ConcreteProductB");
factoryMethod.addActor(concreteproductB);
concreteproductB.addInheritedActor(Aproduct);
concreteproductB.setComment("//implmente l'interface product");
//***** Creator Class*****
IClass creator = Factory.GetInstance().createClass("Creator");
factoryMethod.addActor(creator); creator.setAbstract(true);
//FactoryMethod Method
IMethod factory =
Factory.GetInstance().createMethod("FactoryMethod");
creator.addActor(factory); factory.setAbstract(true);
factory.setReturnType("Product");

```

Singleton Design Pattern:

Les Classes et objets participants au patron :

Singleton :

- Responsable de la création et du maintien d'une unique instance de la classe Singleton.

Code Csharp généré automatiquement	Code PADL
<pre> //Singleton public class Singleton { public static void GetInstance() { if (this.isInstanceNull) { instance = new Singleton(); } return this.instance; } private Singleton() { } } private static Singleton instance; public static boolean isInstanceNull() { } } Le Code C# Correspondant Singleton Design Motif </pre>	<pre> //*****Singleton Class***** IClass Singleton = Factory.GetInstance().createClass("Singleton"); SingletonDesign.addActor(Singleton); IField instance = Factory.GetInstance().createField("instance", "Singleton", 2); Singleton.addActor(instance); instance.setPrivate(true); instance.setStatic(true); //GetInstance Method ICSharpMethod getInstance = CSharpFactory.getCsharpInstance().createNewMethod("GetInstance"); Singleton.addActor(getInstance); getInstance.setStatic(true); getInstance.SetReturnValue("this.instance"); //isInstanceNull Method IMethod isnullInstance = Factory.GetInstance().createMethod("isInstanceNull"); Singleton.addActor(isnullInstance); isnullInstance.setReturnType(" boolean"); isnullInstance.setStatic(true); IBlock block = StatementFactory.getStateInstance().createBlock(); getInstance.addActor(block); IConditional structure = StatementFactory.getStateInstance().createConditionals(isnullInstance); block.addActor(structure); structure.SetIfBlock(block); IInstanciation inst = StatementFactory.getStateInstance().createInstanciation(instance,Singleton); structure.addActor(inst); //Singleton Constructor IConstructor SingletonC = Factory.GetInstance().createConstructor("Singleton"); Singleton.addActor(SingletonC); SingletonC.setPrivate(true); </pre>

Builder Design Pattern:

Les Classes et objets participants au patron :

Builder :

- A le rôle d'interface pour créer les différentes parties de l'objet de type Product.

ConcreteBuilder :

- Rassemble les différentes parties de l'objet Product en implémentant l'interface Builder.
- assure le suivi de la représentation qu'il créer
- Fournit une interface pour retrouver l'objet Product

Director :

- Construit un objet de type Product en utilisant l'interface Builder

Product :

- Représente l'objet complexe dans la construction

Code Csharp généré automatiquement	Code PADL
<pre> //Builder public abstract class Builder { public abstract void BuildPartA(); public abstract void BuildPartB(); public abstract Product GetResult(); } //Director public class Director { //Builder uses a complex series of steps public void Construct() { builder.BuildPartA(); builder.BuildPartB(); } private Builder builder; } </pre>	<pre> //***** Builder Class***** IClass builder=Factory.GetInstance().createClass("Builder"); Builder.addAction(builder); builder.setAbstract(true); //BuildPartA Method IMethod BuilderPartA=Factory.GetInstance().createMethod("BuildPartA"); builder.addAction(BuilderPartA); BuilderPartA.setAbstract(true); //BuildPartB Method IMethod BuilderPartB=Factory.GetInstance().createMethod("BuildPartB"); builder.addAction(BuilderPartB); BuilderPartB.setAbstract(true); //getResult Method IMethod GetResult=Factory.GetInstance().createMethod("getResult"); builder.addAction(GetResult); GetResult.setAbstract(true); GetResult.setReturnType("Product"); //*****Director Class***** IClass director=Factory.GetInstance().createClass("Director"); Builder addAction(director); director.setComment(" //Builder uses a complex series of steps"); IField builder1=Factory.GetInstance().createField("builder", "Builder", 2); director.addAction(builder1); builder1.setPrivate(true); //Construct Method IMethod Construct=Factory.GetInstance().createMethod("Construct"); director.addAction(Construct); IMethodInvocation invoc1=Factory.GetInstance().createMethodInvocation(2,1,1,director); invoc1.addCallingField(builder1); invoc1.setCalledMethod(BuilderPartA); Construct.addAction(invoc1); IMethodInvocation invoc2=Factory.GetInstance().createMethodInvocation(2,1,1,director); invoc2.addCallingField(builder1); invoc2.setCalledMethod(BuilderPartB); Construct.addAction(invoc2); </pre>

```

//ConcreteBuilder1
public class ConcreteBuilder1 : Builder {
    product = new Product();
    public override void BuilderPartA() {
        product.Add('PartA');
    }
    public override void BuilderPartB() {
        product.Add('PartB');
    }
    public override Product GetResult() {
        return product;
    }
    private Product product;
}

```

```

//*****ConcreteBuilder1 Class*****
IClass ConcreteBuilder1=Factory.GetInstance().createClass("ConcreteBuilder1");
Builder.addActor(ConcreteBuilder1);
ConcreteBuilder1.addInheritedActor(builder);

IField product=Factory.GetInstance().createField("product", "Product", 2);
ConcreteBuilder1.addActor(product);
IInstanciation inst=StatementFactory.getStateInstance().createInstanciation(product, Product);
ConcreteBuilder1.addActor(inst);
product.setPrivate(true);

//BuilderPartA Method
ICSharpMethod BuilderPartAA=CSharpFactory.getCSharpInstance().createNewMethod("BuilderPartA");
ConcreteBuilder1.addActor(BuilderPartAA);
BuilderPartAA.setOverride(true);
BuilderPartAA.setComment(" product.Add('PartA');");

//BuilderPartB Method
ICSharpMethod BuilderPartBB=CSharpFactory.getCSharpInstance().createNewMethod("BuilderPartB");
ConcreteBuilder1.addActor(BuilderPartBB);
BuilderPartBB.setOverride(true);
BuilderPartBB.setComment(" product.Add('PartB');");

//getResult Method
ICSharpMethod
getResultA=CSharpFactory.getCSharpInstance().createNewMethod("getResult");
ConcreteBuilder1.addActor(getResultA); getResultA.setOverride(true);
getResultA.setReturnType("Product");
getResultA.SetReturnValue("product");

```

```

//ConcreteBuilder2
public class ConcreteBuilder2 : Builder {

    product = new Product();
    public override void BuilderPartA() {
        product.Add('PartX');
    }
    public override void BuilderPartB() {
        product.Add('PartY');
    }
    public override Product GetResult() {
        return product;
    }
    private Product product;
}

```

```

//*****ConcreteBuilder2 Class*****
IClass ConcreteBuilder2Factory.GetInstance().createClass("ConcreteBuilder2");
Builder.addAction(ConcreteBuilder2);
ConcreteBuilder2.addInheritedActor(builder);

IField product1=Factory.GetInstance().createField("product", "Product", 2);
ConcreteBuilder2.addAction(product1);
IInstanciation inst1=StatementFactory.getStateInstance().createInstanciation(product1,Product);
ConcreteBuilder2.addAction(inst1);
product1.setPrivate(true);

//BuilderPartA Method
ICSharpMethod BuilderPartAA1=CSharpFactory.getCSharpInstance().createNewMethod("BuilderPartA");
ConcreteBuilder2.addAction(BuilderPartAA1);
BuilderPartAA1.setOverride(true);
BuilderPartAA1.setComment(" product.Add('PartX');");

//BuilderPartB Method
ICSharpMethod BuilderPartBB1=CSharpFactory.getCSharpInstance().createNewMethod("BuilderPartB");
ConcreteBuilder2.addAction(BuilderPartBB1);
BuilderPartBB1.setOverride(true);
BuilderPartBB1.setComment(" product.Add('PartY');");

//getResult Method
ICSharpMethod
getResultAA1=CSharpFactory.getCSharpInstance().createNewMethod("getResult");
ConcreteBuilder2.addAction(getResultAA1);
getResultAA1.setOverride(true);
getResultAA1.setReturnType("Product");
getResultAA1.SetReturnValue("product");

```

```

//Product
public class Product { ArrayList parts = new ArrayList();
public void Add(string part) {
    parts.Add(part);
}
public void Show() {
    Console.WriteLine('Product parts :');
    for(this.addedPart)
    {
        Console.WriteLine('part');
    }
}
public boolean addedPart() {
}
} Le Code C# Correpndant Builder Design Motif

```

```

//*****Product Class*****
IClass Product=Factory.getInstance().createClass("Product");
Builder.addAction(Product);
Product.setComment("ArrayList parts = new ArrayList();");
IParameter param=Factory.getInstance().createParameter("part", "string");
IMethod add=Factory.getInstance().createMethod("Add");
Product.addAction(add);
add.addAction(param);
add.setComment(" parts.Add(part);");
//Show Method
IMethod show=Factory.getInstance().createMethod("Show");
Product.addAction(show);
//addedPart Method
IMethod addedPart=Factory.getInstance().createMethod("addedPart");
Product.addAction(addedPart);
addedPart.setReturnType("boolean");
IOutPut out=StatementFactory.getInstance().createOutPut("Product parts :");
show.addAction(out);
IIterator iter=StatementFactory.getStateInstance().createIterators(addedPart);
show.addAction(iter);
IOutPut out1=StatementFactory.getStateInstance().createOutPut("part");
iter.addAction(out1);

```