Análise e Projeto Orientados por Objetos

Aula 06 – Padrões GoF (Factory Method e Abstract Factory)

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

Padrões GoF

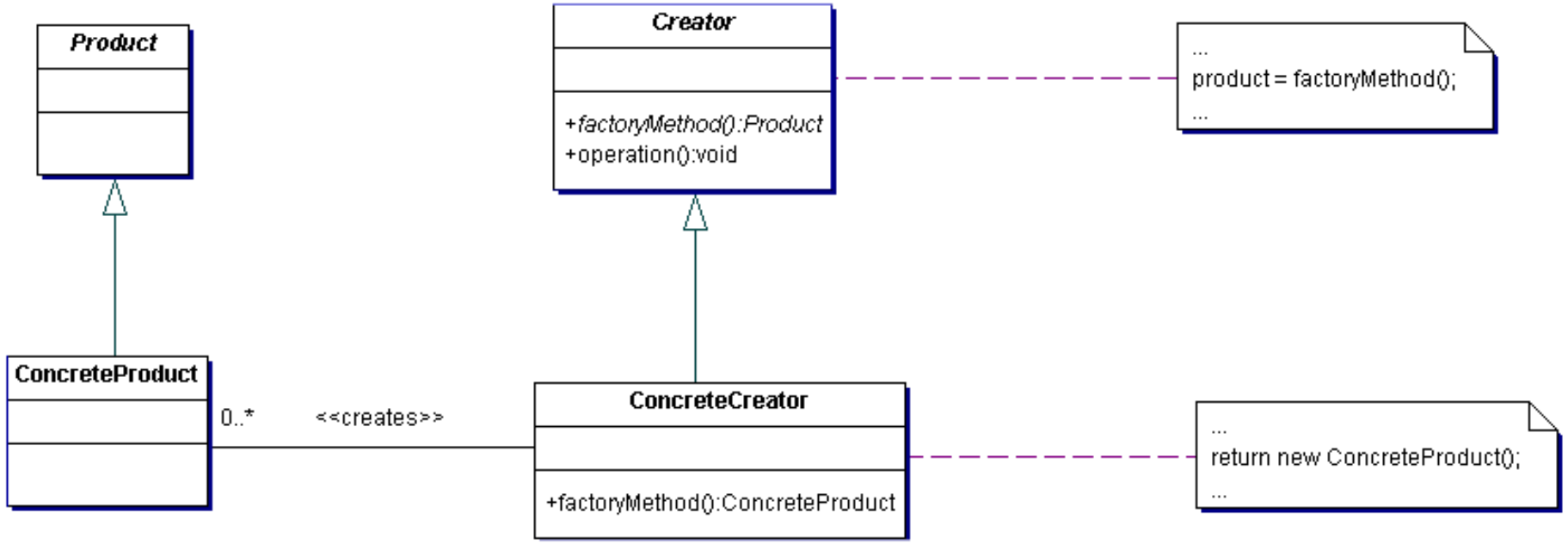
- Criação:
  - **Abstract Factory**
  - Builder
  - **Factory Method**
  - Prototype
  - Singleton
- Estruturais:
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Comportamentais:
  - Chain of Responsibility
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Template Method
  - Visitor

Factory Method

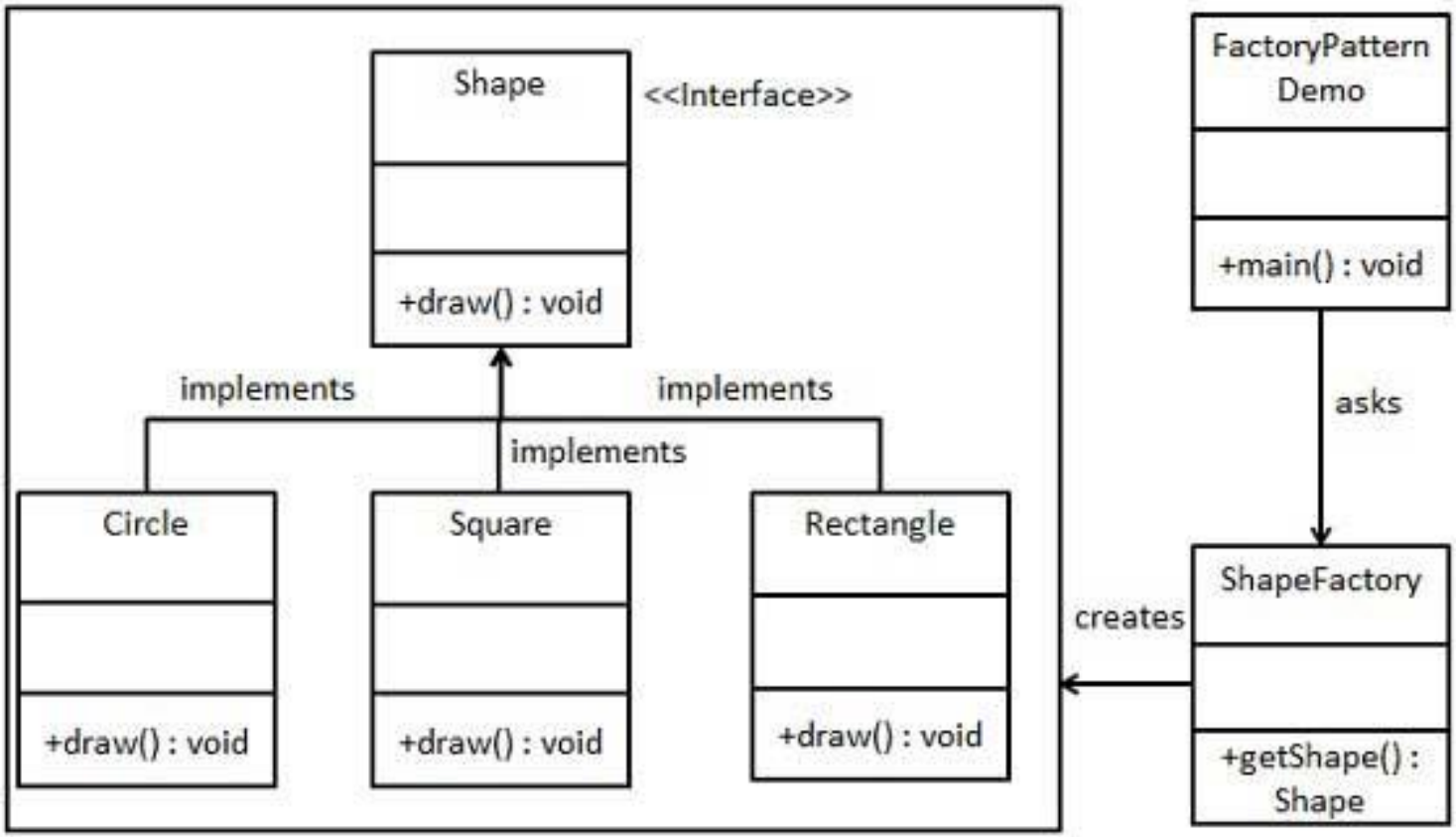
- **Intenção:** definir uma interface para criação de um objeto, permitindo que as suas subclasses decidam qual classe instanciar. O Factory Method deixa a responsabilidade de instanciação para as subclasses.

- **Aplicabilidade:**
  - Classe não conhece antecipadamente a classe dos objetos que deve criar;
  - Classe quer que suas subclasses especifiquem os objetos que criam;

Factory Method



Factory Method – Exemplo



Factory Method – Implementação

```
public interface Shape
{
   void draw();
}
```

```
public class Rectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Factory Method – Implementação

```
public class Square implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Square::draw() method.");
   }
}
```

```
public class Circle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

Factory Method – Implementação

```
public class ShapeFactory
{
   public Shape getShape(String shapeType) //factory method
   {
      if(shapeType == null)
         return null;

      if(shapeType.equalsIgnoreCase("CIRCLE"))
         return new Circle();
      else if(shapeType.equalsIgnoreCase("RECTANGLE"))
         return new Rectangle();
      else if(shapeType.equalsIgnoreCase("SQUARE"))
         return new Square();

      return null;
   }
}
```

Factory Method – Implementação

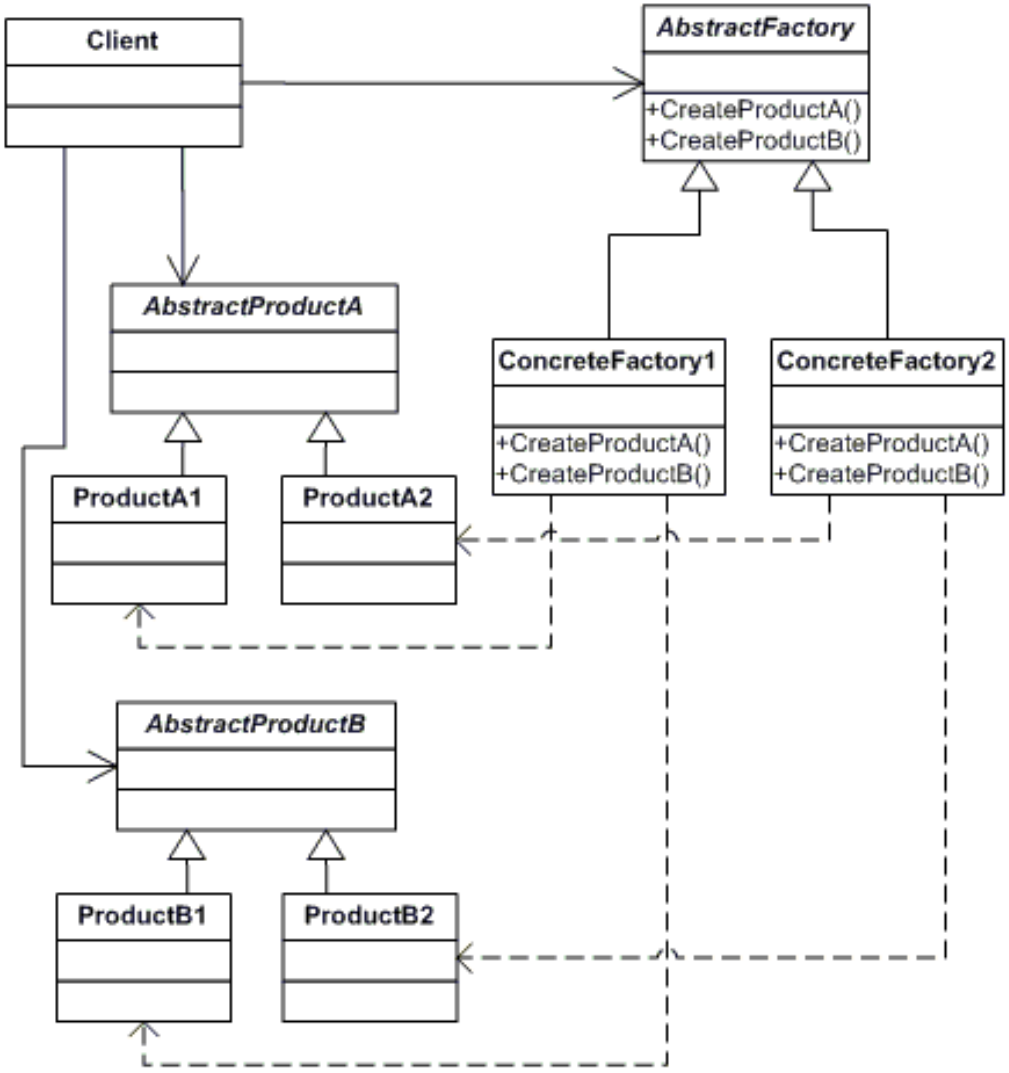
```
public static void main(String[] args)
{
    ShapeFactory shapeFactory = new ShapeFactory();

    Shape shape1 = shapeFactory.getShape("CIRCLE");
    shape1.draw();

    Shape shape2 = shapeFactory.getShape("RECTANGLE");
    shape2.draw();

    Shape shape3 = shapeFactory.getShape("SQUARE");
    shape3.draw();
}
```

Abstract Factory

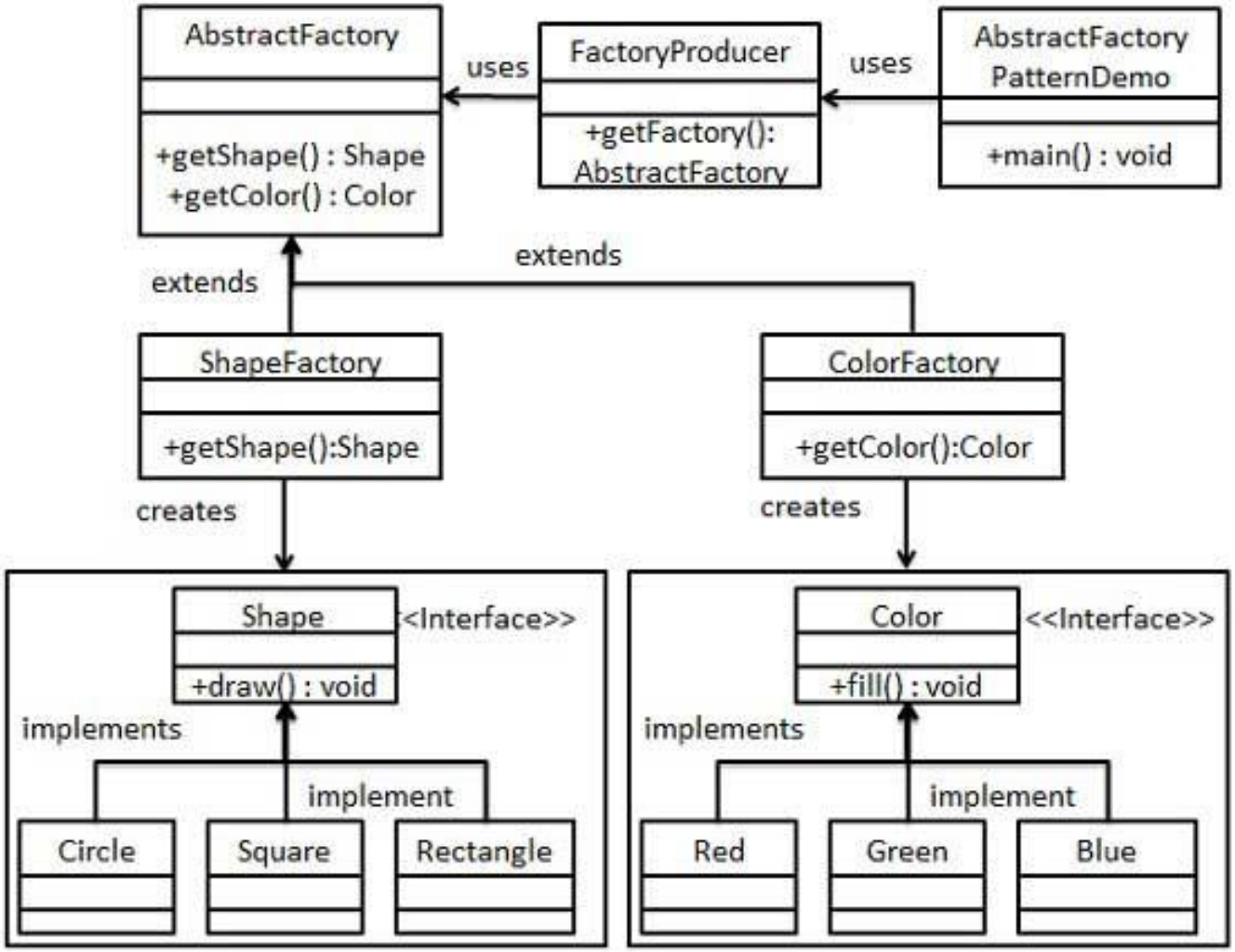
- **Intenção:** fornecer uma interface comum para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

- **Solução:** crie uma interface para representar uma fábrica para cada família de objetos. As subclasses concretas instanciam cada família específica.
  - Baseia-se no uso de uma super-fábrica, responsavel por criar outras fábricas.

Abstract Factory



Abstract Factory – Exemplo



Abstract Factory – Implementação

```
public interface Shape
{
   void draw();
}
```

```
public class Rectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Abstract Factory – Implementação

```
public class Square implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Square::draw() method.");
   }
}
```

```
public class Circle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

Abstract Factory – Implementação

```
public interface Color
{
   void fill();
}
```

```
public class Red implements Color
{
   @Override
   public void fill()
   {
      System.out.println("Inside Red::fill() method.");
   }
}
```

Abstract Factory – Implementação

```
public class Green implements Color
{
   @Override
   public void fill()
   {
      System.out.println("Inside Green::fill() method.");
   }
}
```

```
public class Blue implements Color
{
   @Override
   public void fill()
   {
      System.out.println("Inside Blue::fill() method.");
   }
}
```

Abstract Factory – Implementação

```
public abstract class AbstractFactory
{
   abstract Color getColor(String color);
   abstract Shape getShape(String shape);
}
```

Abstract Factory – Implementação

```
public class ShapeFactory extends AbstractFactory
{
   @Override
   public Shape getShape(String shapeType)
   {
      if(shapeType == null)
         return null;
      if(shapeType.equalsIgnoreCase("CIRCLE"))
         return new Circle();
      else if(shapeType.equalsIgnoreCase("RECTANGLE"))
         return new Rectangle();
      else if(shapeType.equalsIgnoreCase("SQUARE"))
         return new Square();
      return null;
   }

   @Override
   Color getColor(String color){
      return null;
   }
}
```

Abstract Factory – Implementação

```
public class ColorFactory extends AbstractFactory
{
   @Override
   public Shape getShape(String shapeType){
      return null;
   }

   @Override
   Color getColor(String color)
   {
      if(color == null)
         return null;
      if(color.equalsIgnoreCase("RED"))
         return new Red();
      else if(color.equalsIgnoreCase("GREEN"))
         return new Green();
      else if(color.equalsIgnoreCase("BLUE"))
         return new Blue();
      return null;
   }
}
```

Abstract Factory – Implementação

```
public class FactoryProducer
{
   public static AbstractFactory getFactory(String choice)
   {
      if(choice.equalsIgnoreCase("SHAPE"))
         return new ShapeFactory();
      else if(choice.equalsIgnoreCase("COLOR"))
         return new ColorFactory();
      return null;
   }
}
```

Abstract Factory – Implementação

```
public static void main(String[] args)
{
  AbstractFactory shapeFactory = FactoryProducer.getFactory("SHAPE");
  Shape shape1 = shapeFactory.getShape("CIRCLE");
  shape1.draw();
  Shape shape2 = shapeFactory.getShape("RECTANGLE");
  shape2.draw();
  Shape shape3 = shapeFactory.getShape("SQUARE");
  shape3.draw();

  AbstractFactory colorFactory = FactoryProducer.getFactory("COLOR");
  Color color1 = colorFactory.getColor("RED");
  color1.fill();
  Color color2 = colorFactory.getColor("Green");
  color2.fill();
  Color color3 = colorFactory.getColor("BLUE");
  color3.fill();
}
```

Abstract Factory – Aplicabilidade

- Quando o sistema deve ser independente de como seus produtos são **criados**, **compostos** e **representados**.

- Quando o sistema deve ser configurado com uma dentre várias famílias de produtos.
  - É necessário fornecer uma biblioteca de classes, mas não é desejável revelar que produto particular está sendo usado.

- Quando uma família de produtos relacionados foi projetada para ser **usada em conjunto**, e esta restrição tem de ser garantida.

Abstract Factory – Consequências

- **Isola classes concretas:** uma vez que uma fábrica encapsula a responsabilidade e o processo de criação de objetos-produto, ela isola clientes das classes de implementação.

- Fica mais fácil a troca de uma família de produtos, bastando trocar a fábrica concreta usada pela aplicação.

- Promove consistência entre produtos.