Análise e Projeto Orientados por Objetos

Aula 10 – Padrões GoF (Protoype e Façade)

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

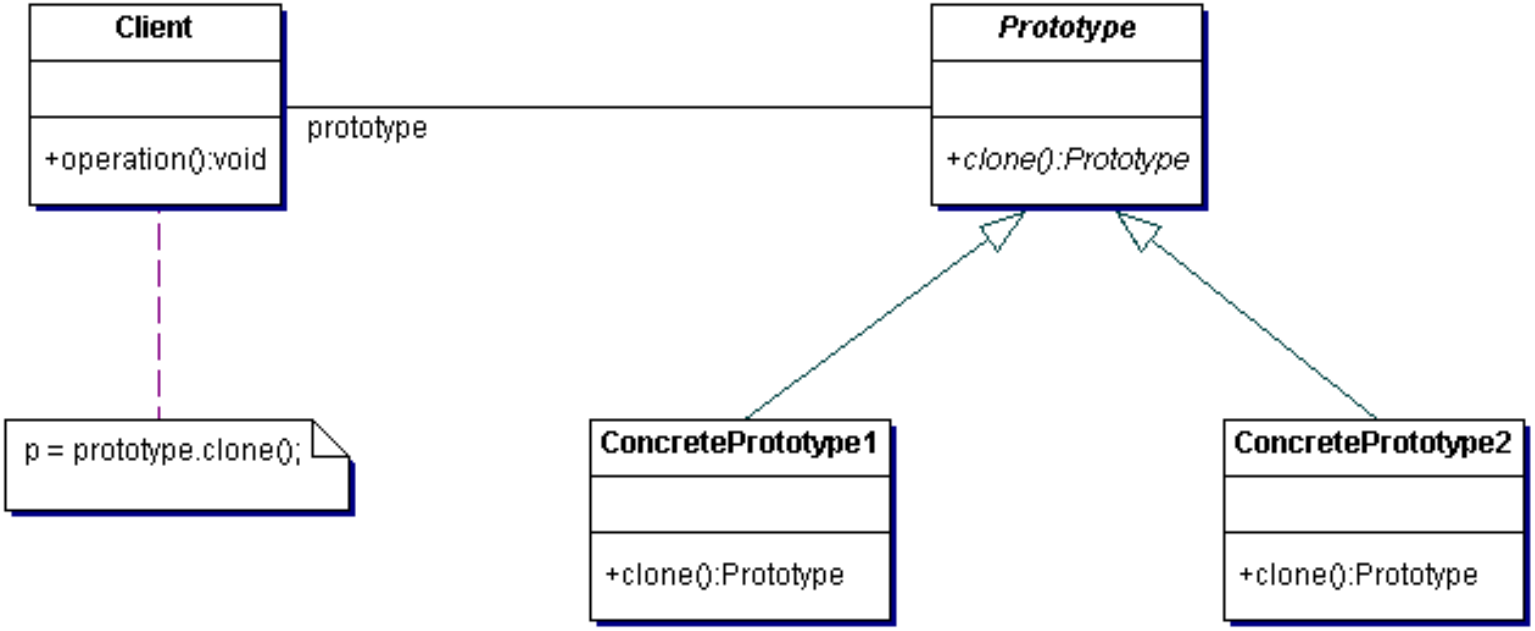
Padrões GoF

- 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

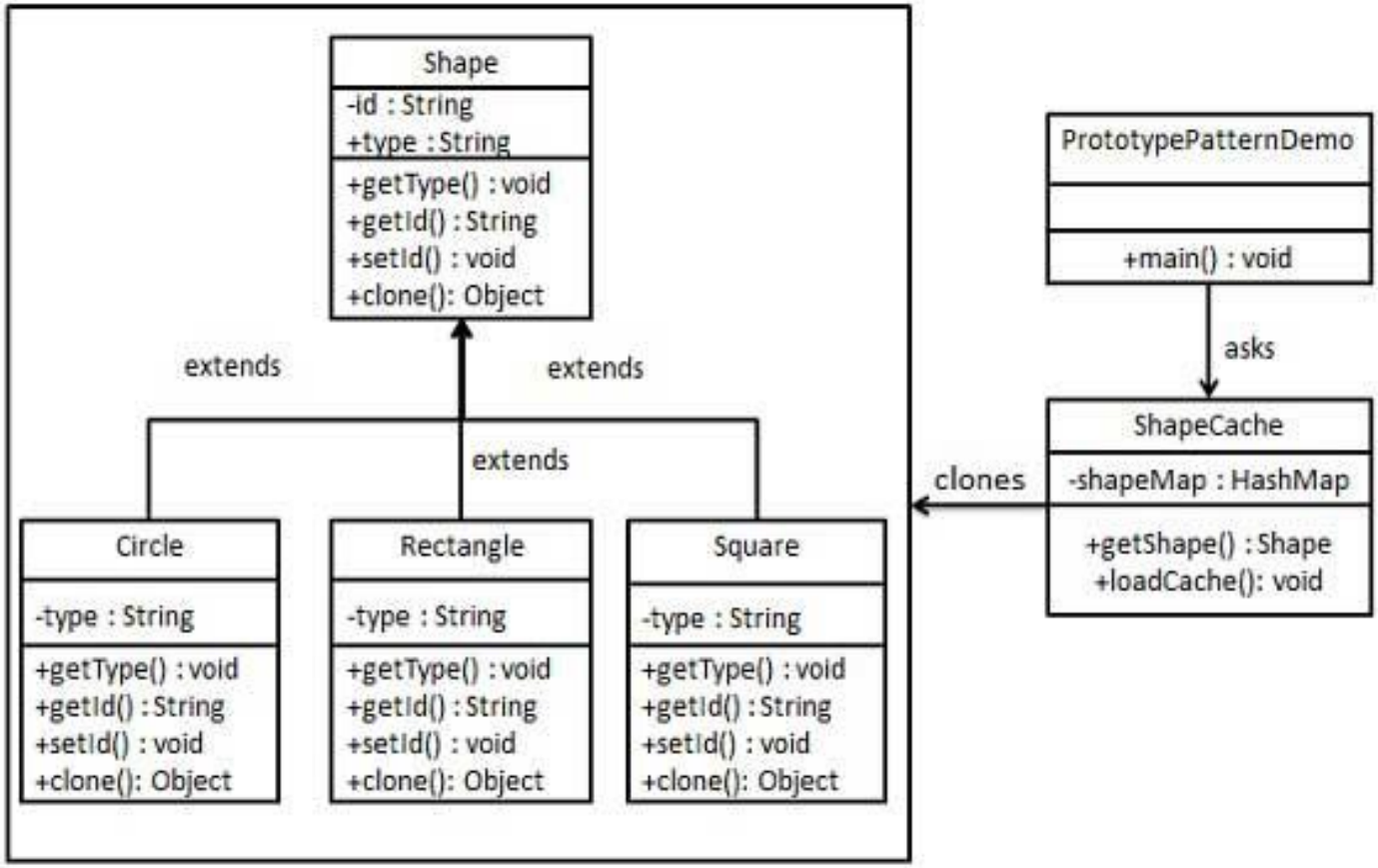
Prototype

- **Intenção:** permitir a criação de objetos a partir de um modelo, ou seja, permitir a especificação de tipos de objetos a serem criados usando uma **instância-protótipo** e criar novos objetos pela cópia desse protótipo.

Prototype


Client
+operation():void
prototype
Prototype
+clone():Prototype
p = prototype.clone();
ConcretePrototype1
+clone():Prototype
ConcretePrototype2
+clone():Prototype

Prototype – Exemplo



Prototype – Implementação

```
public abstract class Shape implements Cloneable {
   private String id;
   protected String type;
   abstract void draw();
   public String getType(){
      return type;
   }
   public String getId() {
      return id;
   }
   public void setId(String id) {
      this.id = id;
   }
   public Object clone() {
      Object clone = null;
      try {
         clone = super.clone();
      } catch (CloneNotSupportedException e){e.printStackTrace();}
      return clone;
   }
}
```

Prototype – Implementação

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

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

Prototype – Implementação

```
public class Circle extends Shape
{
   public Circle(){
     type = "Circle";
   }

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

Prototype – Implementação

```
public class ShapeCache {
   private static Hashtable<String, Shape> shapeMap  = new
                                        Hashtable<String, Shape>();
   public static Shape getShape(String shapeId){
      Shape cachedShape = shapeMap.get(shapeId);
      return (Shape) cachedShape.clone();
   }
   public static void loadCache() {
      Circle circle = new Circle();
      circle.setId("1");
      shapeMap.put(circle.getId(),circle);

      Square square = new Square();
      square.setId("2");
      shapeMap.put(square.getId(),square);

      Rectangle rectangle = new Rectangle();
      rectangle.setId("3");
      shapeMap.put(rectangle.getId(), rectangle);
   }
}
```

Prototype – Implementação

```
public static void main(String[] args)
{
   ShapeCache.loadCache();

   Shape clonedShape = (Shape) ShapeCache.getShape("1");
   System.out.println("Shape : " + clonedShape.getType());

   Shape clonedShape2 = (Shape) ShapeCache.getShape("2");
   System.out.println("Shape : " + clonedShape2.getType());

   Shape clonedShape3 = (Shape) ShapeCache.getShape("3");
   System.out.println("Shape : " + clonedShape3.getType());
}
```

Prototype – Aplicabilidade

- Quando a instânciação de um novo objeto demanda a execução de **operações complexas** que requerem tempo.

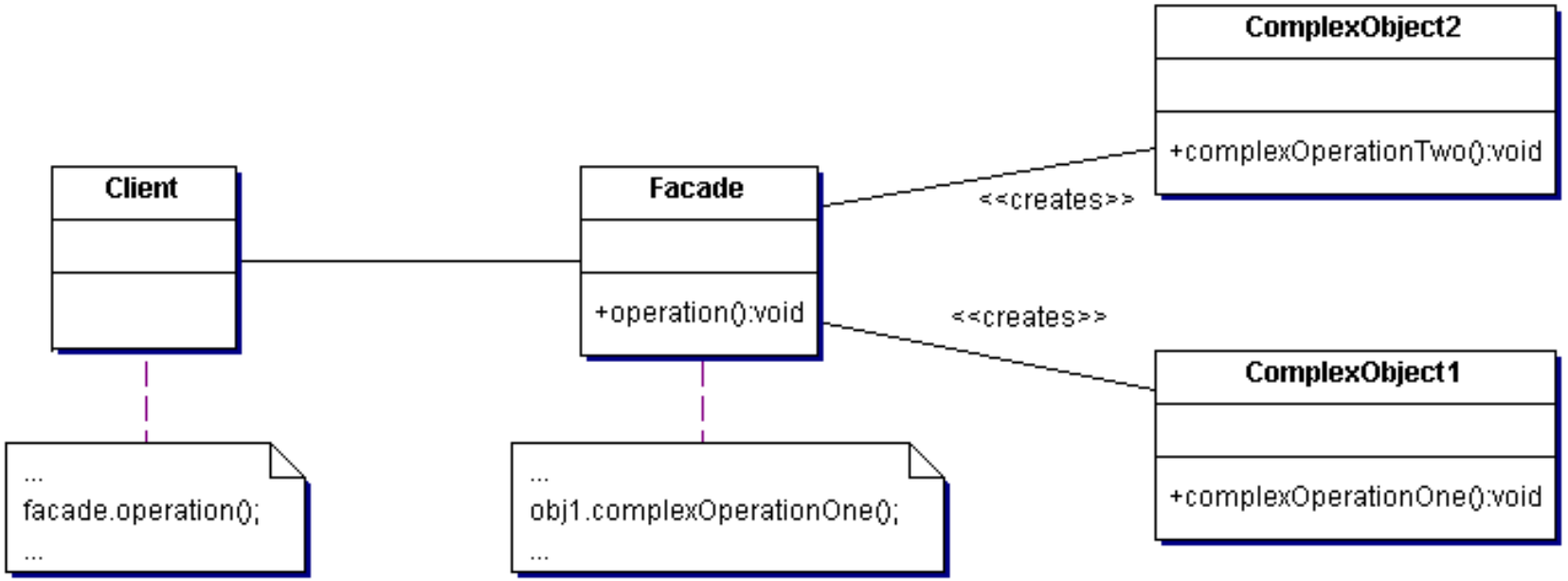
- Quando as instâncias de uma classe puderem ter uma dentre **poucas combinações** diferentes de estados.
  - Pode ser mais conveniente definir um número correspondente de protótipos e cloná-los, ao invês de instanciar a classe manualmente, cada vez com um estado apropriado.

Façade

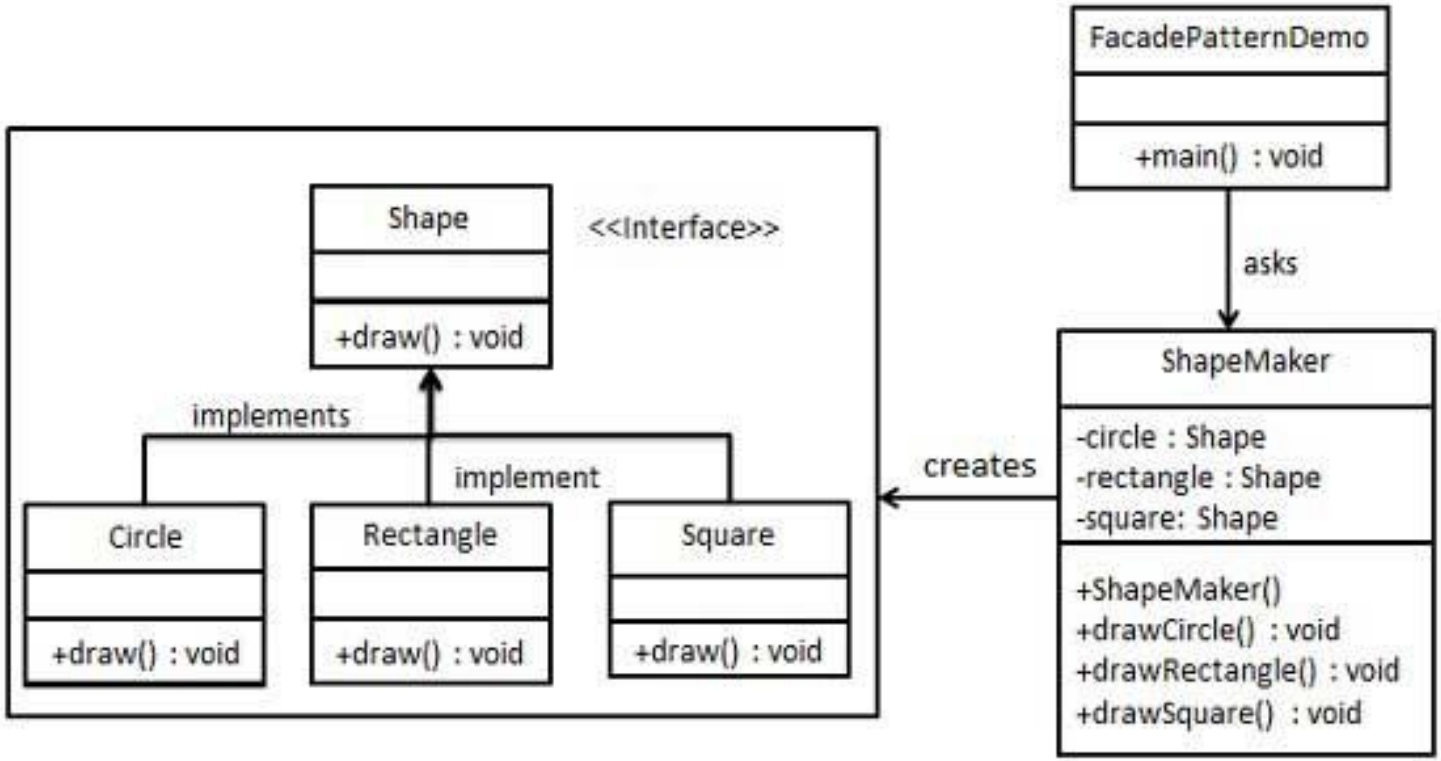
- **Intenção:** disponibilizar uma interface simplificada para as funcionalidades de partes do sistema ou de uma biblioteca.

- O Façade pode:
  - Tornar uma biblioteca de software mais fácil de entender e usar;
  - Tornar o código que utiliza esta biblioteca mais fácil de entender;
  - Reduzir as dependências em relação às características internas de uma biblioteca, trazendo flexibilidade no desenvolvimento do sistema;
  - Envolver uma interface mal desenhada, com uma interface melhor definida.

Façade


ComplexObject2
+complexOperationTwo():void
<<creates>>
Client
Facade
+operation():void
<<creates>>
ComplexObject1
+complexOperationOne():void
...
facade.operation();
...
...
obj1.complexOperationOne();
...

Façade – Exemplo



Façade – Implementação

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

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

Façade – Implementação

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

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

Façade – Implementação

```
public class ShapeMaker {
   private Shape circle;
   private Shape rectangle;
   private Shape square;

   public ShapeMaker() {
      circle = new Circle();
      rectangle = new Rectangle();
      square = new Square();
   }

   public void drawCircle(){
      circle.draw();
   }
   public void drawRectangle(){
      rectangle.draw();
   }
   public void drawSquare(){
      square.draw();
   }
}
```

Façade – Implementação

```
public static void main(String[] args)
{

   ShapeMaker shapeMaker = new ShapeMaker();

   shapeMaker.drawCircle();
   shapeMaker.drawRectangle();
   shapeMaker.drawSquare();
}
```