Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 4 Padrões de Criação - Prototype

Intenção

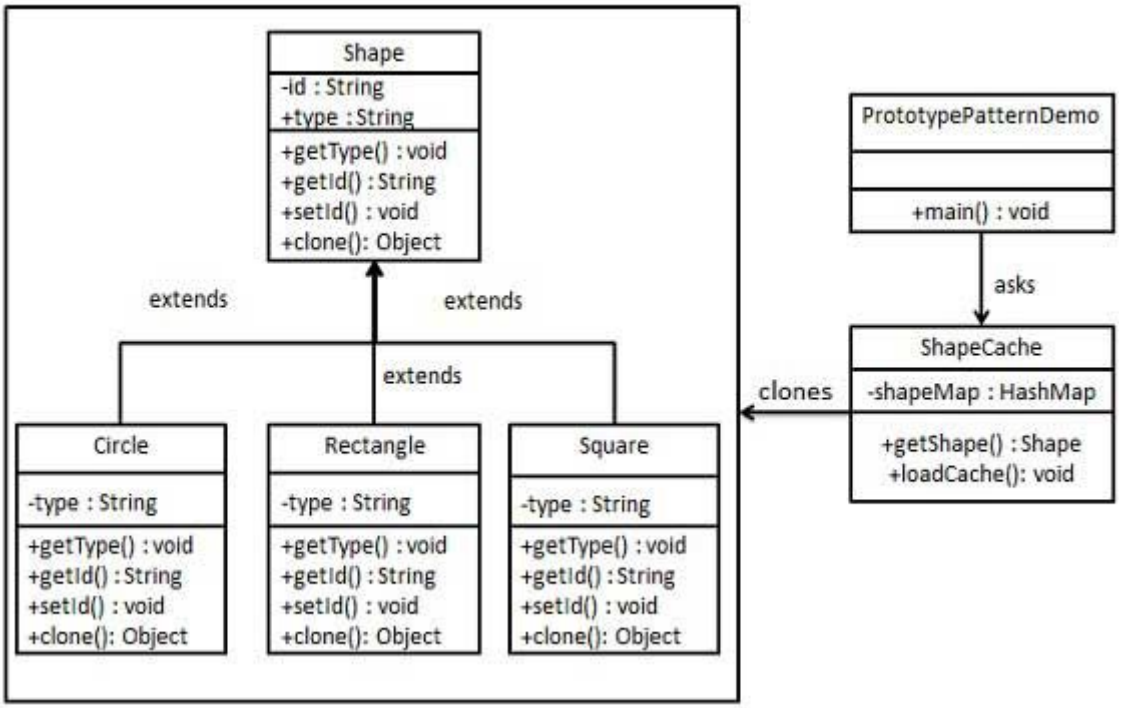
Especificar o tipo de objeto a ser criado utilizando uma instância como protótipo e criar novos objetos copiando este protótipo.

Usar este padrão quando…

- O sistema deve ser independente de como seus produtos são criados, compostos e representados e…
  - as classes que devem ser criadas são especificadas em tempo de execução;
  - ou você não quer construir uma fábrica para cada hierarquia de produtos;
  - ou as instâncias da classe clonável só tem alguns poucos estados possíveis, e é melhor clonar do que criar objetos.

Vantagens e desvantagens

- Esconde a implementação do produto;
- Permite adicionar e remover produtos em tempo de execução (configuração dinâmica da aplicação);
- Não necessita de uma fábrica para cada hierarquia de objetos;
- Implementar clone() pode ser complicado.



Passo 1
Crie uma classe abstrata implementando a interface Cloneable.

Shape.java

```
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;
   }
}
```

Passo 2
Crie classes concretas estendendo a classe acima.

Rectangle.java

```
public class Rectangle extends Shape {

   public Rectangle(){
     type = "Rectangle";
   }

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

Square.java

```
public class Square extends Shape {

   public Square(){
     type = "Square";
   }

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

Circle.java

```
public class Circle extends Shape {

   public Circle(){
     type = "Circle";
   }

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

Passo 3
Crie uma classe para obter classes concretas do banco de dados e armazená-las em uma Hashtable.

ShapeCache.java

```
import java.util.Hashtable;

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();
   }

   // for each shape run database query and create shape
   // shapeMap.put(shapeKey, shape);
   // for example, we are adding three shapes

   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);
   }
}
```

Passo 4
PrototypePatternDemo usa a classe ShapeCache para obter clones de formas armazenadas em uma Hashtable.

PrototypePatternDemo.java

```
public class PrototypePatternDemo {
   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());
   }
}
```

Passo 5

Teste sua implementação!