Padrões de projeto criacionais

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

Padrões de projeto criacionais

Padrões Criacionais

Lidam com a criação dos objetos. Ao invés de se criar um objeto diretamente, esses padrões oferecem flexibilidade na decisão de que objeto criar em cada caso.

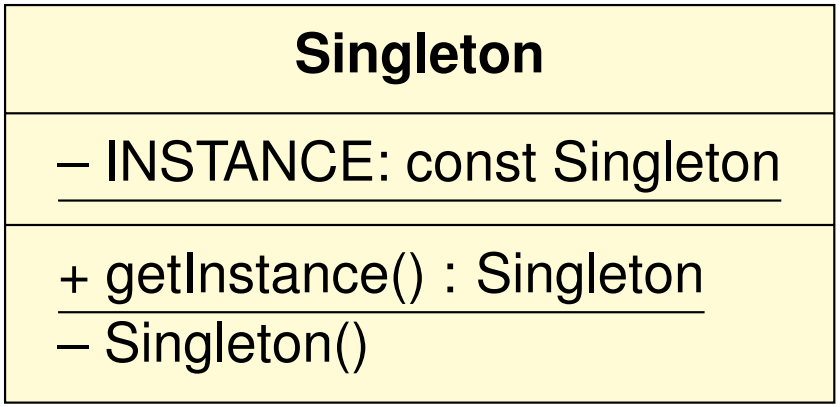
- Exemplos mais comuns: Factory Method, Abstract Factory, Singleton.
- Exemplos menos comuns: Builder, Prototype.

Singleton

Singleton

Garantir que uma classe possui apenas uma instância e provê acesso global a esta instância.

- Útil quando exatamente um único objeto é necessário para coordenar ações.



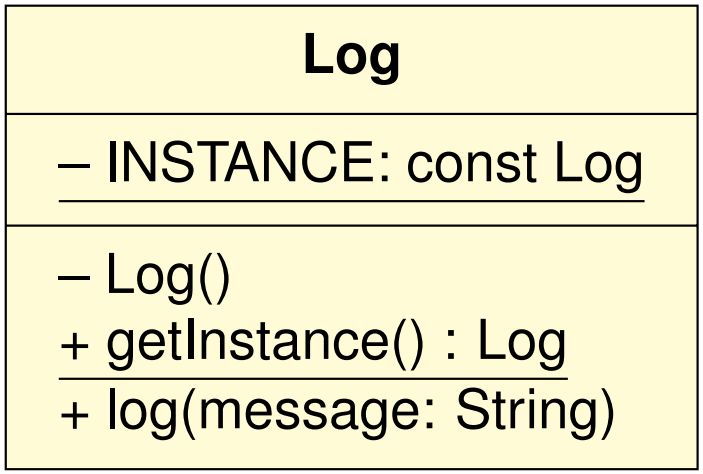
Singleton em Java – com acesso global

```
public class Singleton {
  private static final Singleton INSTANCE = new Singleton();

  private Singleton() {}

  public static Singleton getInstance() {
     return INSTANCE;
  }

  // Outros métodos públicos
}

// Uso

Singleton.getInstance().metodoDoSingleton();
```

Singleton em Java – sem acesso global

```
public class Singleton {
  private static boolean CREATED = false;
  public Singleton() {
     if (CREATED)
        throw new RuntimeException(
           "Classe " + this.getClass() +
           " só pode ser instanciada uma vez.");
     CREATED = true;
  }

  // Outros métodos aqui.

  public static void main(String args[]) {
    Singleton primeiro = new Singleton();
    Singleton segundo = new Singleton();
  }
}
```

Exemplo: Logging




```
public class Log {
 private static final Log INSTANCE
                              = new Log();

 private Log() {}

 public static Log getInstance() {
    return INSTANCE;
 }

 public void logging(String message)
   LocalD
   System.out.println();
 }
}
```

Classe utilitária estática

- Alternativa ao padrão Singleton.
- Útil quando o comportamento geral não necessita de um estado e não será estendido.
- Não é adequada para armazenar estado (atributos).
- Por não possuírem estado, muitos consideram uma violação dos princípios da POO.
- Exemplo: Classe `System` e classe `Math`.

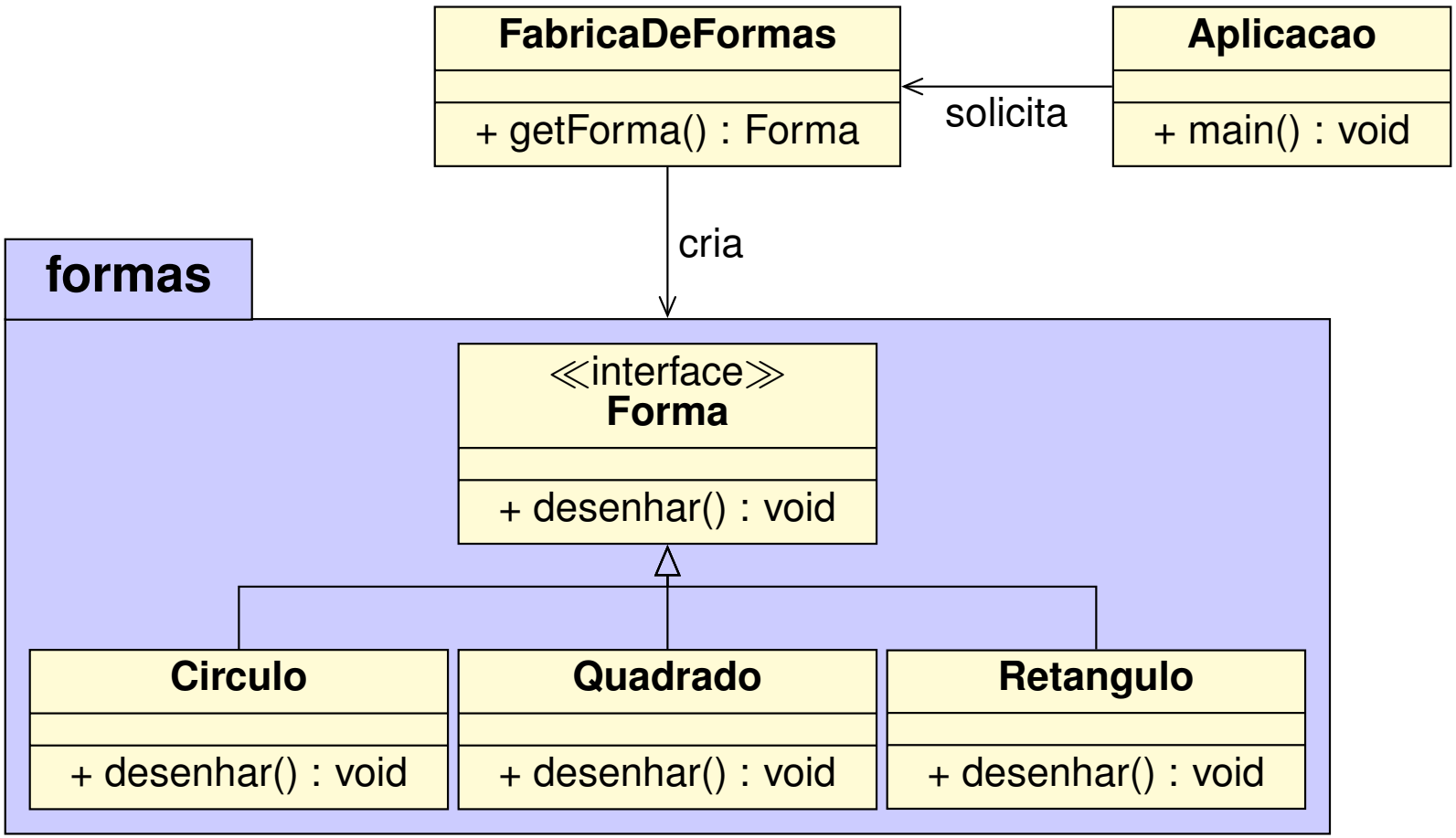
Factory Method

Factory Method

Objetos são criados sem expor a lógica de criação dos mesmos ao cliente, e referenciam o novo objeto a partir de uma interface comum.

- Útil quando se deseja encapsular a instanciação de tipos concretos, que serão escolhidos posteriormente por subclasses.

Factory Method: exemplo Fábrica de formas



Factory Method: exemplo Fábrica de formas

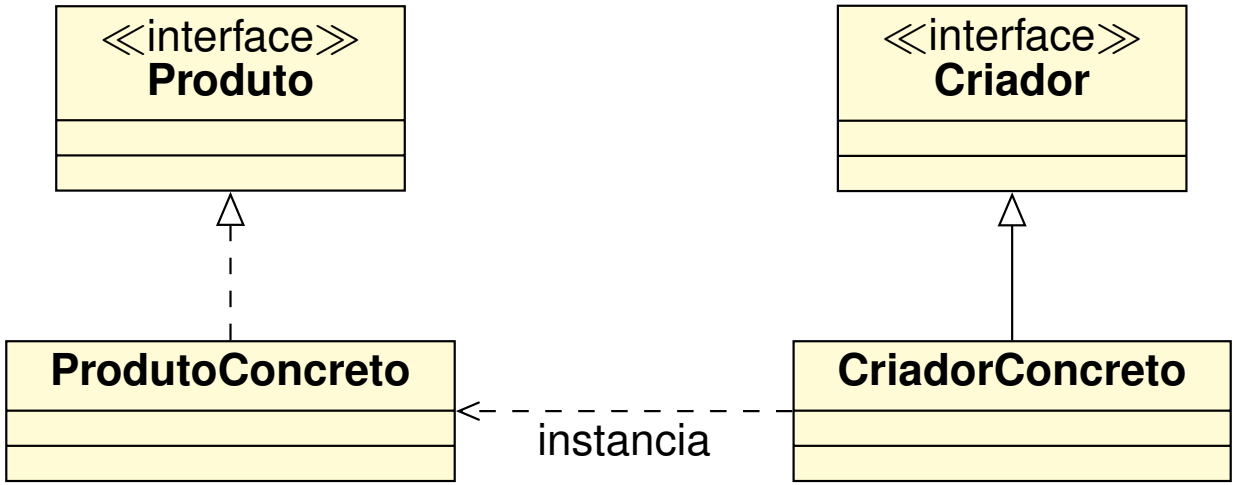
```
public enum TipoForma {
 CIRCULO, QUADRADO, RETANGULO;
}

public interface Forma {
 void desenhar();
}

public class Circulo implements Forma {

 @Override
 public void desenhar() {
  System.out.println("Circulo::desenhar()");
 }
}
```

```
public class Quadrado implements Forma {

  @Override
  public void desenhar() {
    System.out.println("Quadrado::desenhar()");
  }
}

public class Retangulo implements Forma {

  @Override
  public void desenhar() {
    System.out.println("Retangulo::desenhar()");
  }
}
```

Factory Method: exemplo Fábrica de formas

```
public class FabricaDeFormas {
    public Forma getForma(TipoForma tipoForma) {
        if (tipoForma == null) {
            return null;
        } else if (tipoForma.equals(TipoForma.CIRCULO)) {
            return new Circulo();
        } else if (tipoForma.equals(TipoForma.QUADRADO)) {
            return new Retangulo();
        } else if (tipoForma.equals(TipoForma.RETANGULO)) {
            return new Quadrado();
        }
        return null;
    }
}

public class Aplicacao {
    public static void main(String[] args) throws Exception {
        FabricaDeFormas fabricaDeFormas = new FabricaDeFormas();

        Forma forma1 = fabricaDeFormas.getForma(TipoForma.CIRCULO);
        forma1.desenhar();

        Forma forma2 = fabricaDeFormas.getForma(TipoForma.QUADRADO);
        forma2.desenhar();

        Forma forma3 = fabricaDeFormas.getForma(TipoForma.RETANGULO);
        forma3.desenhar();
    }
}
```

Factory Method: solução geral



- **Produto**: interface dos objetos que serão criados.
- **ProdutoConcreto**: implementa a interface.
- **Criador**: declara o *factory method* que irá retornar um objeto do tipo Produto. Pode-se usar método *default*
- **CriadorConcreto**: implementa o método que retorna um ProdutoConcreto.