Análise e Projeto Orientados por Objetos

Aula 07 – Padrões GoF (Command e Template Method)

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

Padrões GoF

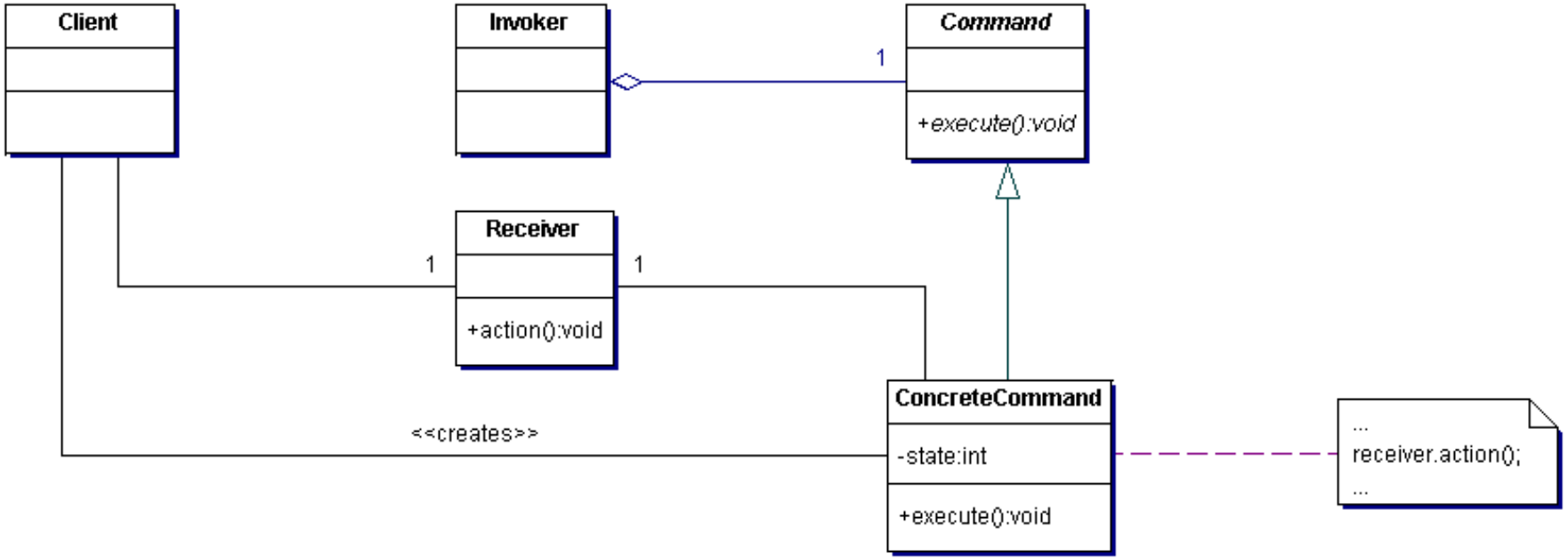
- Criação:
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- Estruturais:
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- Comportamentais:
  - Chain of Responsibility
  - **Command**
  - Interpreter
  - Iterator
  - Mediator
  -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

Command

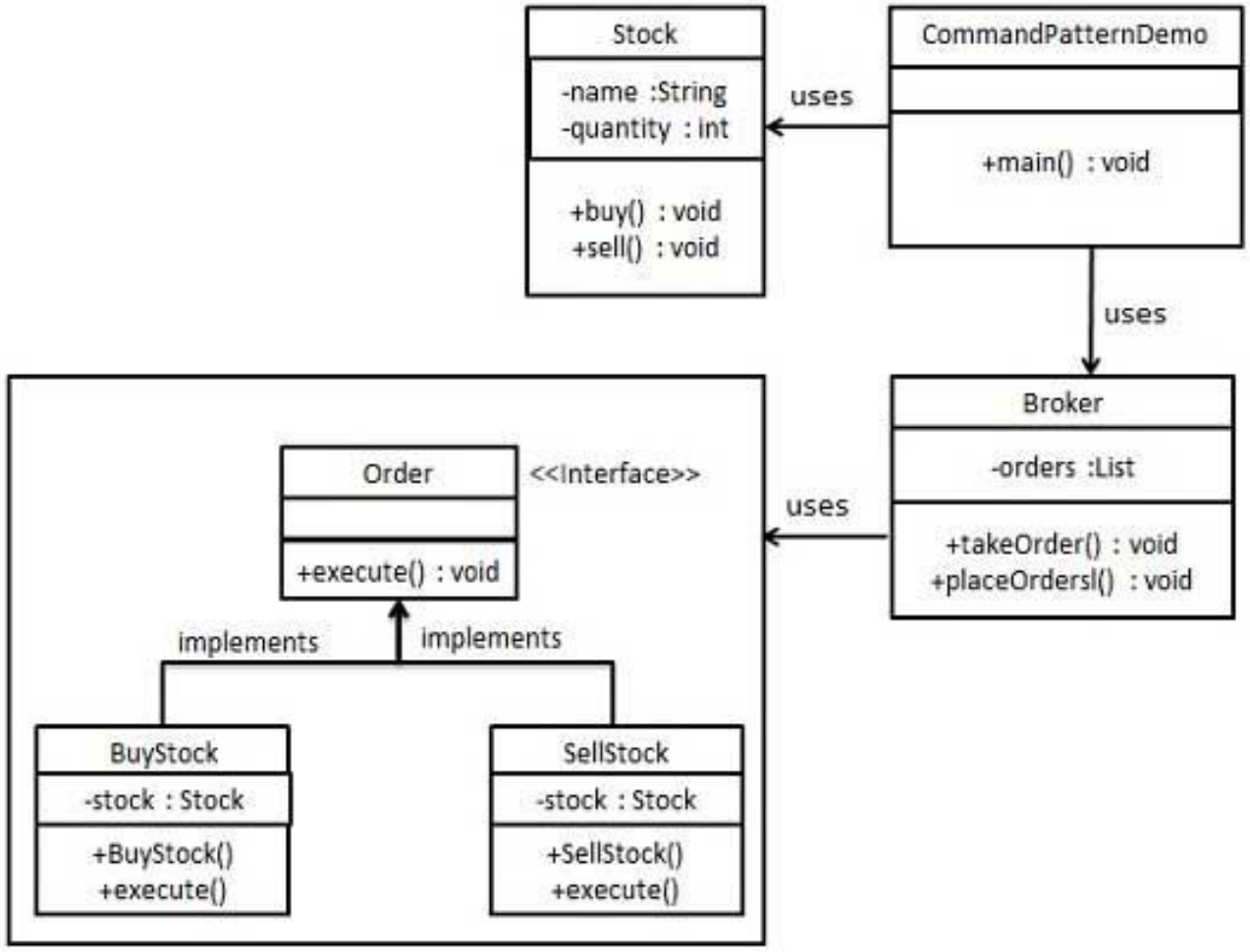
- **Intenção:** associar uma ação a diferentes objetos através de uma interface conhecida. Permitir que objetos deste tipos tenham tais ações executadas sem que conheçamos o tipo de tais objetos ou a natureza das ações.

- **Solução:** encapsular uma requisição como um objeto, permitindo a parametrização de clientes com diferentes requisições.

Command


Client
Invoker
Command
1
+execute():void
Receiver
1
1
+action():void
ConcreteCommand
-state:int
+execute():void
<<creates>>
...
receiver.action();
...

Command – Exemplo



Command – Implementação

```
public interface Order
{
   void execute();
}
```

```
public class Stock {
   private String name = "ABC";
   private int quantity = 10;

   public void buy(){
      System.out.println("Stock [ Name: "+name+",
                          Quantity: " + quantity +" ] bought");
   }
   public void sell(){
      System.out.println("Stock [ Name: "+name+",
                          Quantity: " + quantity +" ] sold");
   }
}
```

Command – Implementação

```
public class BuyStock implements Order {
   private Stock abcStock;

   public BuyStock(Stock abcStock){
      this.abcStock = abcStock;
   }
   public void execute(){
      abcStock.buy();
   }
}
```

```
public class SellStock implements Order {
   private Stock abcStock;

   public SellStock(Stock abcStock){
      this.abcStock = abcStock;
   }
   public void execute() {
      abcStock.sell();
   }
}
```

Command – Implementação

```
public class Broker
{
   private List<Order> orderList = new ArrayList<Order>();

   public void takeOrder(Order order)
   {
      orderList.add(order);
   }

   public void placeOrders()
   {
      for (Order order : orderList)
      {
         order.execute();
      }
      orderList.clear();
   }
}
```

Command – Implementação

```
public static void main(String[] args)
{
   Stock abcStock = new Stock();

   BuyStock buyStockOrder = new BuyStock(abcStock);
   SellStock sellStockOrder = new SellStock(abcStock);

   Broker broker = new Broker();
   broker.takeOrder(buyStockOrder);
   broker.takeOrder(sellStockOrder);

   broker.placeOrders();
}
```

Command – Consequências

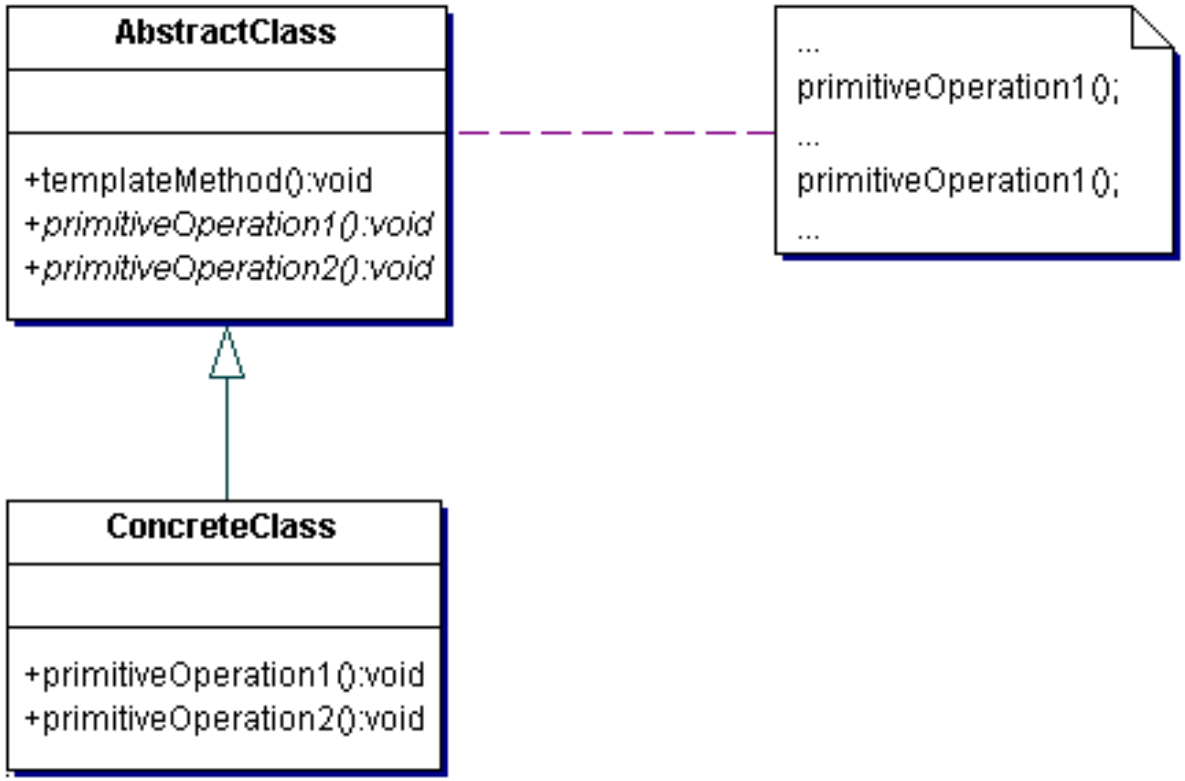
- Isola requisitante do executor;
- Permite registro (log) de ações;
- Permite execução em instante posterior à requisição;
  - Enfileirar ações para processamento em outro momento.

Template Method

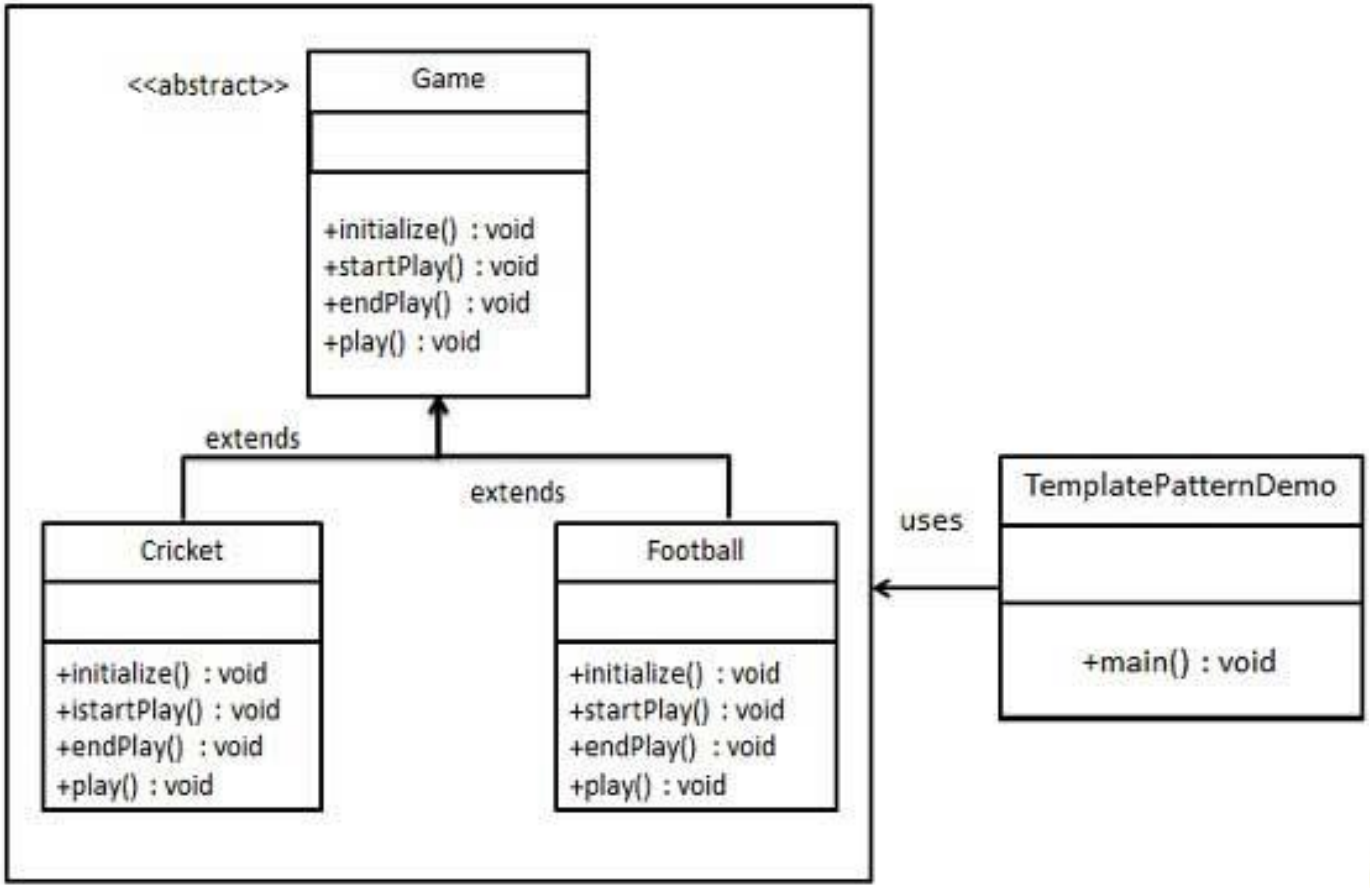
- **Intenção:** definir o esqueleto de um algoritmo em uma operação, postergando (delegando) a definição de alguns passos desse algoritmo para subclasses.

- **Solução:**
  - Em uma classe X, definir a parte invariável do algoritmo em uma operação. Essa operação é denominada método template (template method).
  - Nesta mesma operação, fazer chamadas a operações que representam a parte variável do algoritmo.
  - Essas operações devem então ser implementadas pelas subclasses de X.

Template Method



Template Method – Exemplo



Template Method – Implementação

```
public abstract class Game
{
   abstract void initialize();
   abstract void startPlay();
   abstract void endPlay();

   public final void play()
   {
      initialize();

      startPlay();

      endPlay();
   }
}
```

Template Method – Implementação

```
public class Cricket extends Game
{
   @Override
   void endPlay()
   {
      System.out.println("Cricket Game Finished!");
   }

   @Override
   void initialize()
   {
      System.out.println("Cricket Game Initialized! Start playing.");
   }

   @Override
   void startPlay()
   {
      System.out.println("Cricket Game Started. Enjoy the game!");
   }
}
```

Template Method – Implementação

```
public class Football extends Game
{
    @Override
    void endPlay()
    {
        System.out.println("Football Game Finished!");
    }

    @Override
    void initialize()
    {
        System.out.println("Football Game Initialized! Start playing.");
    }

    @Override
    void startPlay()
    {
        System.out.println("Football Game Started. Enjoy the game!");
    }
}
```

Template Method – Implementação

```
public static void main(String[] args)
{
    Game game = new Cricket();
    game.play();

    game = new Football();
    game.play();
}
```

Template Method – Aplicabilidade

- Quando queremos implementar **partes invariáveis** de um algoritmo e deixar que as subclasses implementem os comportamentos variáveis;
- Quando **comportamentos comuns** entre subclasses devem ser fatorados e localizados em uma superclasse comum;
  - Evitando assim duplicação de código;
- Quando queremos **controlar a extensão** das subclasses;
- Permite que as subclasses **redefinam certos passos** de um algoritmo sem mudar a estrutura desse algoritmo.