
Padrões Comportamentais

— Padrões de Projeto —
Dra. Alana Moraes

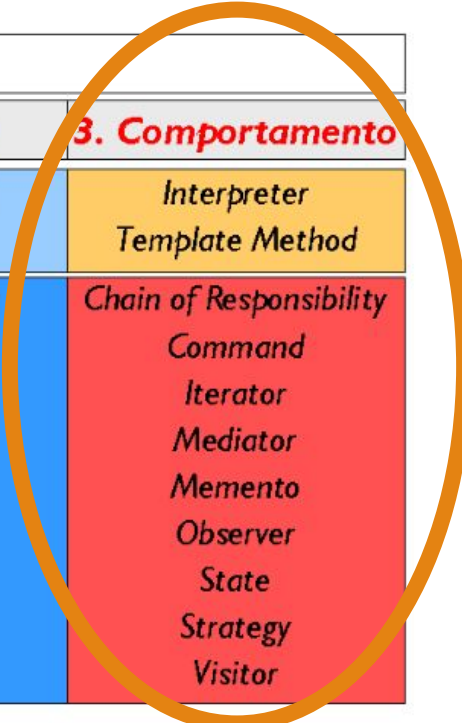
Aula Passada

- Padrões Estruturais Finalizados
- Comportamentais:
 - State e Observer



Padrões GoF – Padrões de Comportamento

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor



Padrões GoF – Padrões de Comportamento

Descrevem padrões de comunicação entre objetos;

- Fluxos de comunicação complexos;
- Foco na interconexão entre objetos.

“Preocupam-se com algoritmos e a delegação de responsabilidades entre objetos.”

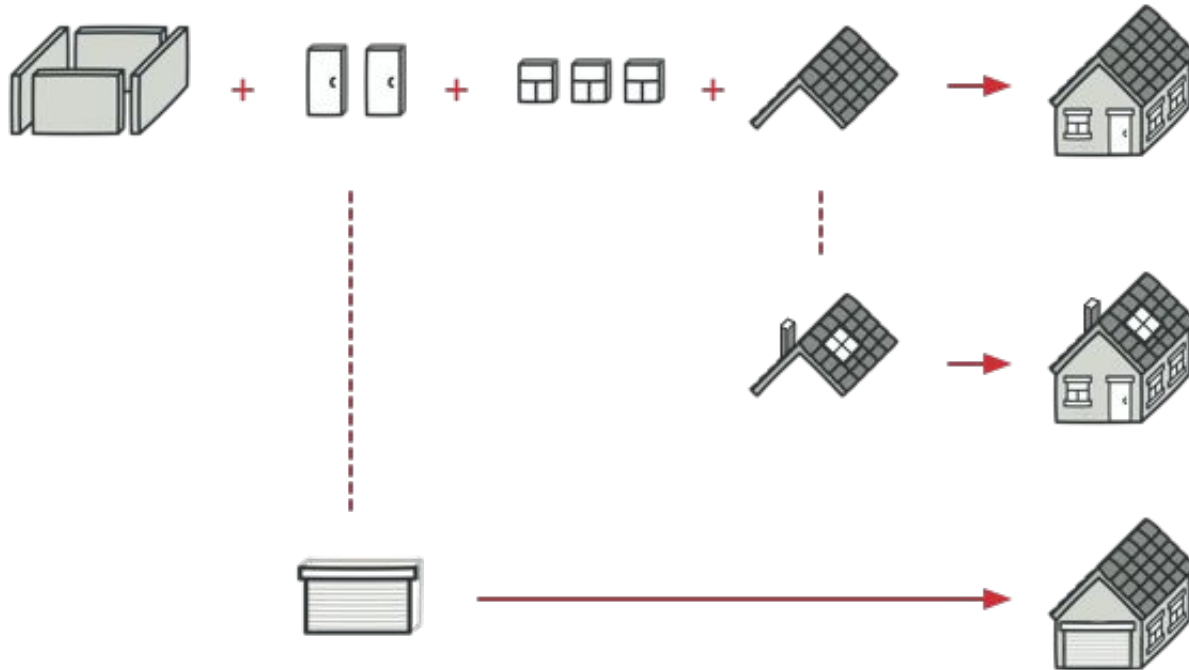
Template Method

“DEFINIR O ESQUELETO DE UM ALGORITMO DENTRO DE UMA OPERAÇÃO, DEIXANDO ALGUNS PASSOS A SEREM PREENCHIDOS PELAS SUBCLASSES. TEMPLATE METHOD PERMITE QUE SUAS SUBCLASSES REDEFINAM CERTOS PASSOS DE UM ALGORITMO SEM MUDAR SUA ESTRUTURA”.

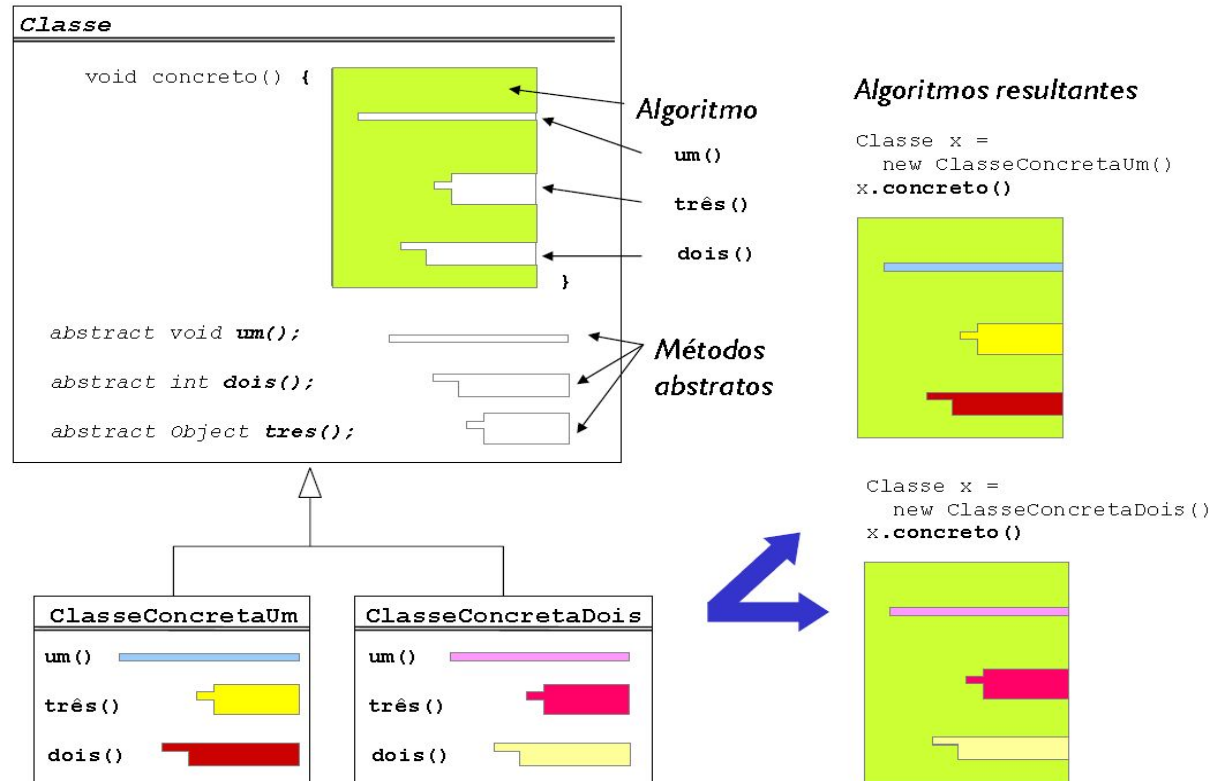
Template Method

- Implementa um algoritmo em um método adiando a definição de alguns passos do algoritmo para que subclasses possam defini-los
- Define o esqueleto de um algoritmo em uma operação, deferindo alguns passos para as subclasses
- Use Template Method para:
 - Implementar a parte invariante de um algoritmo uma vez e deixar para as subclasses a implementação do comportamento que pode variar.

Template Method - Problema

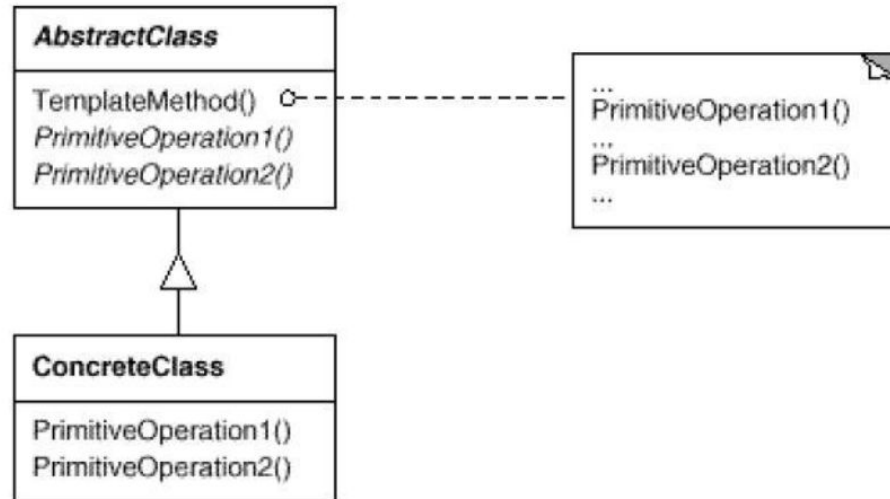


Template Method - Problema



Template Method - Solução

- Template Method define um algoritmo em termos de operações abstratas que subclasses sobrepõem para oferecer comportamento concreto



Template Method - Solução

```
public abstract class AbstractClass {  
  
    public final void templateMethod() {  
        //aqui vai alguma implementação FIXA  
        primitiveOperation1();  
        primitiveOperation2();  
    }  
  
    public abstract void primitiveOperation1();  
    public abstract void primitiveOperation2();  
}
```

Classe abstrata, com o método template definido

```
public class Concrete1 extends AbstractClass {  
  
    public void primitiveOperation1() {  
        //implementação específica  
    }  
  
    public void primitiveOperation2() {  
        //implementação específica  
    }  
}
```

```
public class Concrete2 extends AbstractClass {  
  
    public void primitiveOperation1() {  
        //implementação específica  
    }  
  
    public void primitiveOperation2() {  
        //implementação específica  
    }  
}
```

Classes concretas, com implementações específicas

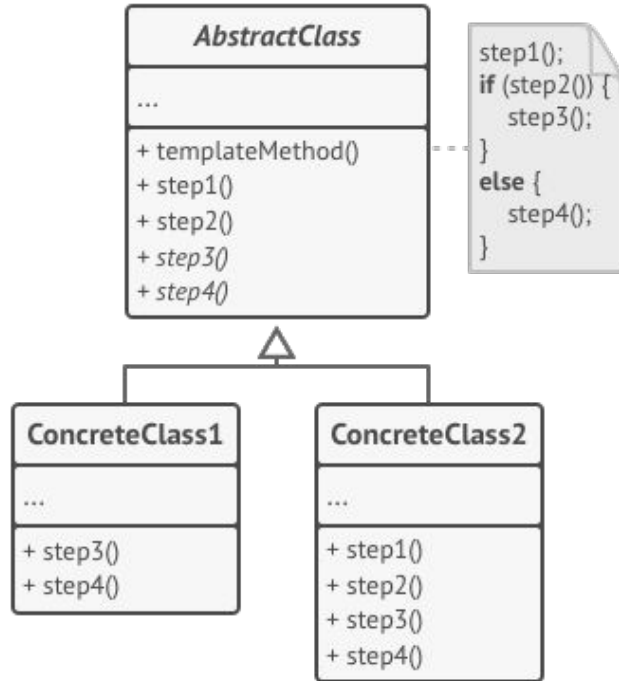
Template Method

Cuidado!

Podemos encontrar este tipo de implementação:

```
1. public abstract class AbstractClass {  
2.  
3.     public final void templateMethod() {  
4.         if (condicao()) {  
5.             fazAlgo();  
6.         }else{  
7.             fazOutraCoisa();  
8.         }  
9.     }  
10.  
11.     protected abstract boolean condicao();  
12.     protected abstract void fazAlgo();  
13.     protected abstract void fazOutraCoisa();  
14.  
15. }
```

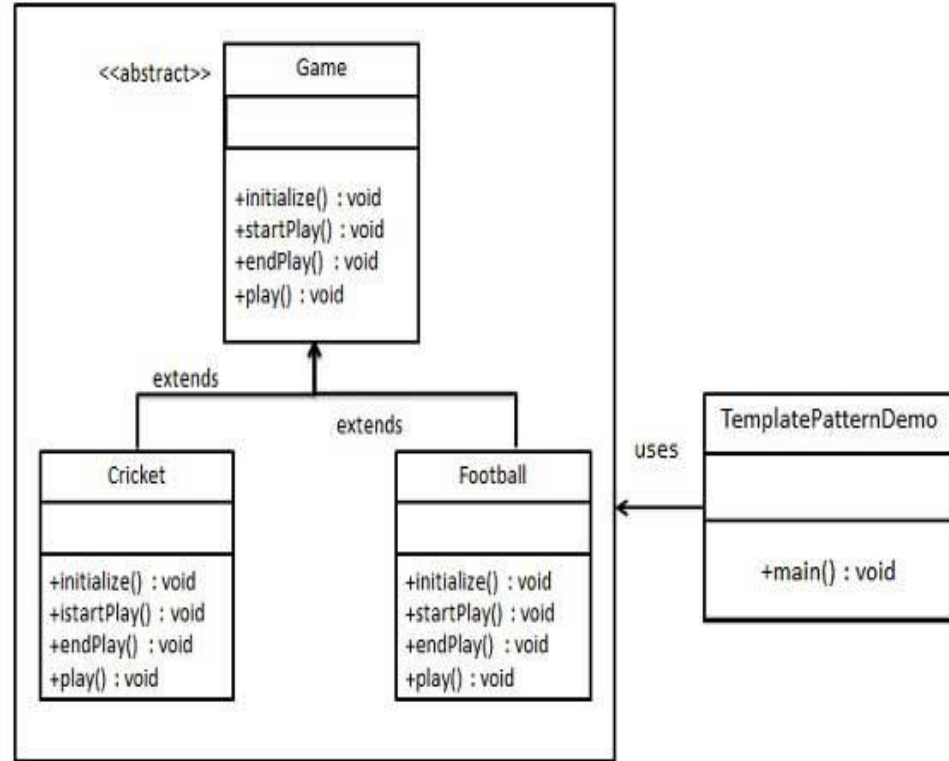
Template Method - Exemplo



- Imagine que você está escrevendo um aplicativo para documentos de mineração de dados. Os usuários alimentam documentos em vários formatos (PDF, DOC, CSV) e recebem dados uniformes na saída.
- Em algum momento, você notou que todos os três algoritmos de análise possuem códigos semelhantes. Por isso, seria bom livrar-se da duplicação de código, deixando a estrutura do algoritmo intacta.
- Houve outro problema relacionado ao código do cliente que usou esses algoritmos. Ele tinha muitas condicionais que estavam escolhendo um curso de ação adequado, dependendo dos algoritmos selecionados. Se todas as três classes analisadas compartilhassem uma interface comum ou uma classe base, essas condicionais poderiam ser eliminadas em favor do uso do polimorfismo.

Template Method Exercício

- Crie uma classe abstrata *Game* definindo operações com um template method definido como final para que não possa ser substituído.
- *Cricket* e *Football* devem ser classes concretas que estendam a *Game* e sobrescrevam seus métodos.
- *TemplatePatternDemo*, deve ser a classe de demonstração e deve usar o *Game* para demonstrar o uso do padrão Template Method.



Consequências

- Vantagens

- Pode deixar que os clientes substituam apenas algumas partes de um algoritmo grande, tornando-os menos afetados por alterações que acontecem em outras partes do algoritmo.
- Pode puxar o código duplicado para uma superclasse.

- Desvantagens

- Alguns clientes podem ser limitados pelo esqueleto fornecido de um algoritmo.
- Pode violar o Princípio de Substituição de Liskov suprimindo uma implementação de etapa padrão por meio de uma subclasse.
- Os métodos de modelo tendem a ser mais difíceis de manter quanto mais etapas tiverem.

Dúvidas?

=D