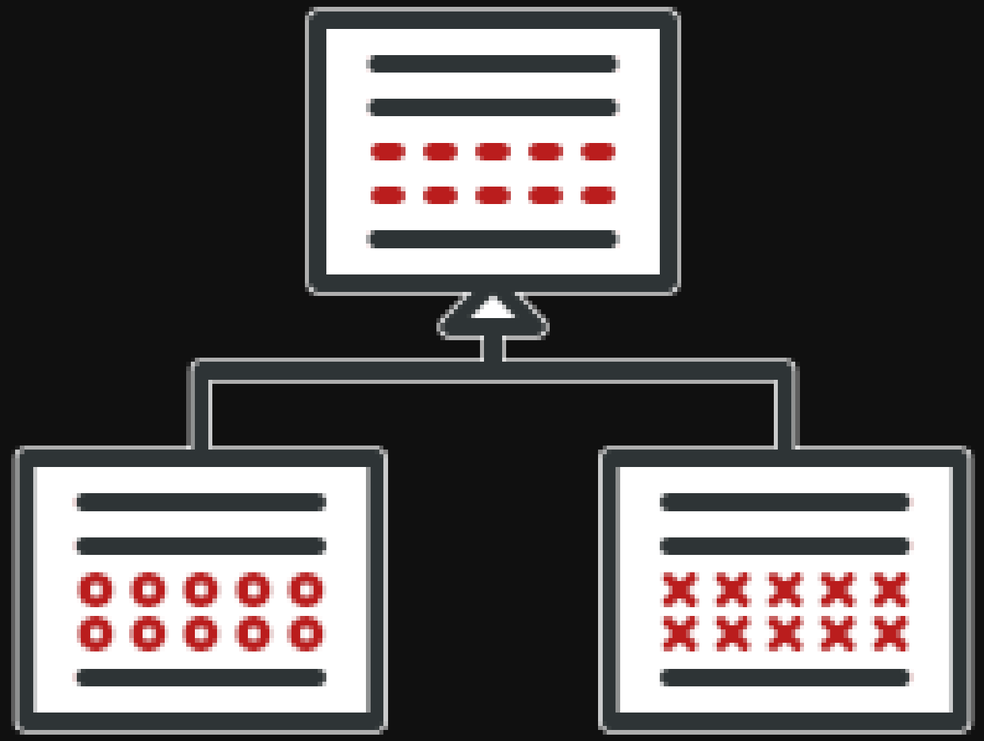
+DEVS2BLU

Curso de C#

TEMPLATE METHOD

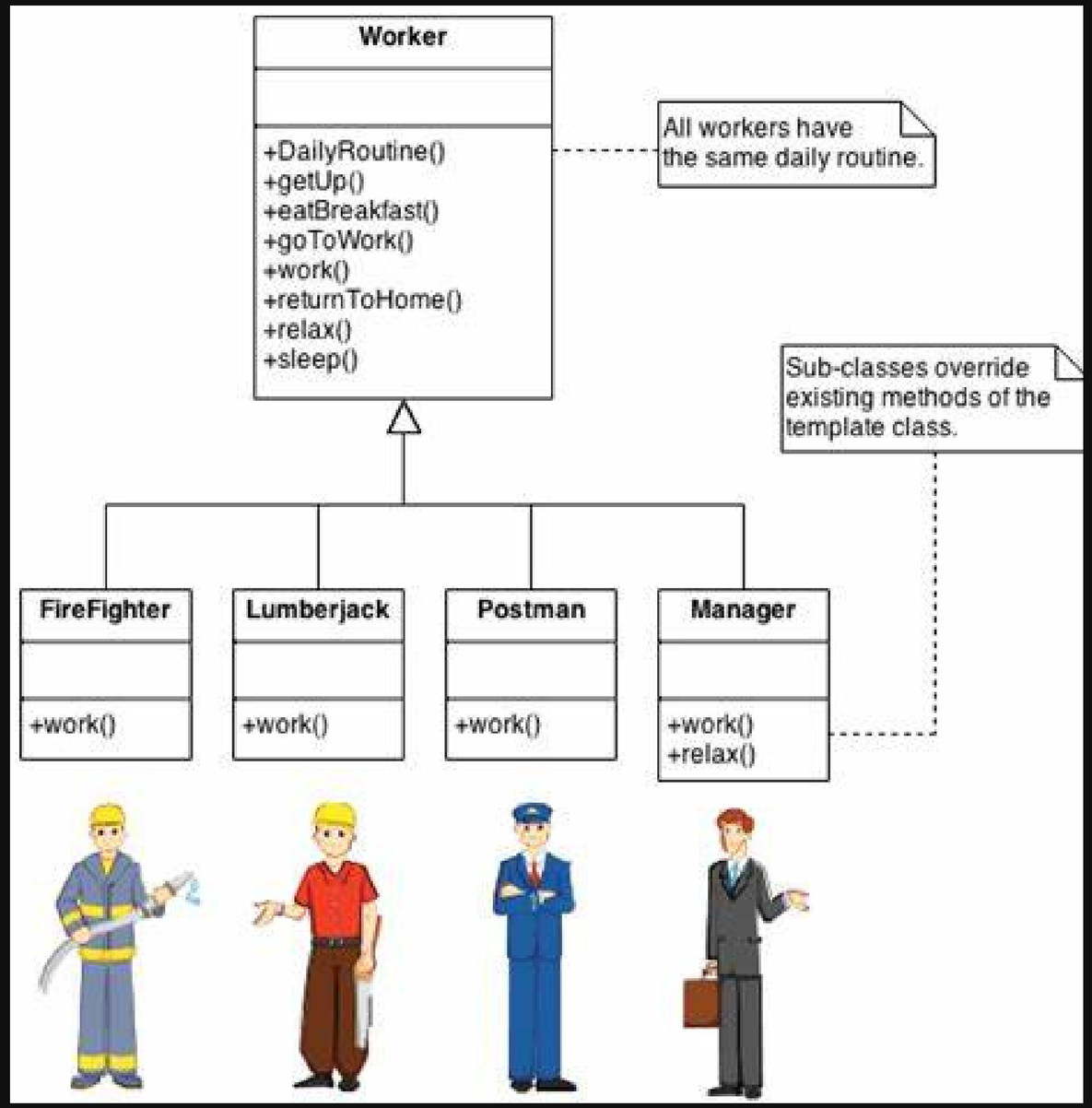


DESIGN PATTERN

O Livro

Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos" de Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides

Em 1994, eles publicaram este livro no qual eles aplicaram o conceito de padrões de projeto para programação.



Encapsulamento

Uma superclasse permite subclasses alterarem partes do algoritmo

Herança

A classe Worker define o esqueleto do algoritmo
'Classe Abstrata'

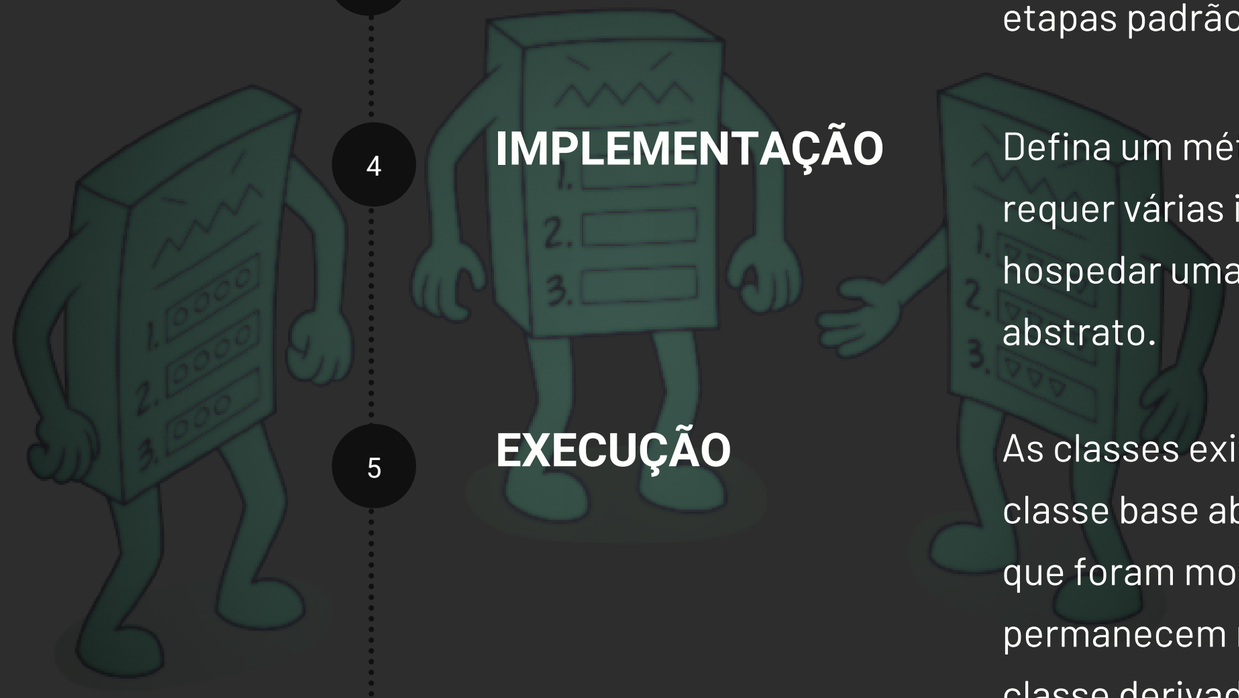
E as demais classes implementam as etapas individuais com seus comportamentos
'Classes Concretas'

COMO FUNCIONA

1. **ANÁLISE**
   Avalie o algoritmo e decida quais etapas são padrão e quais são específicas para cada classe atual.

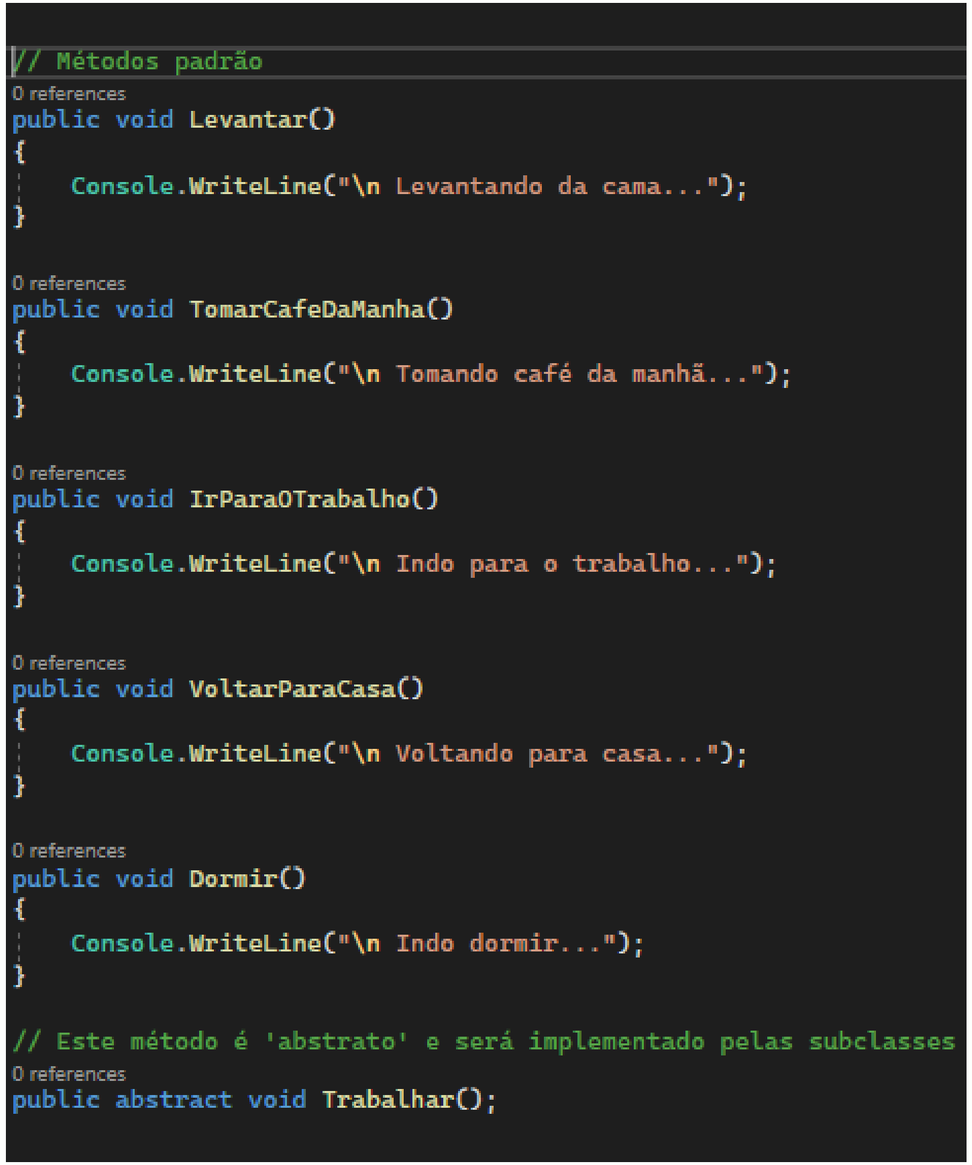
2. **CRIAÇÃO**
   Defina uma nova classe base abstrata para hospedar o esqueleto do algoritmo.

3. **DEFINIÇÃO**
   Mova o esqueleto do algoritmo e a definição de todas as etapas padrão para a nova classe base.



4. **IMPLEMENTAÇÃO**
   Defina um método "gancho" na classe base para cada etapa que requer várias implementações diferentes. Esse método pode hospedar uma implementação padrão ou pode ser definido como abstrato.

5. **EXECUÇÃO**
   As classes existentes declaram uma relação "é-um" com a nova classe base abstrata, e removem os detalhes de implementação que foram movidos para a classe base. Os únicos detalhes que permanecem nas classes existentes são os específicos para cada classe derivada.



```
// Métodos padrão
0 references
public void Levantar()
{
    Console.WriteLine("\n Levantando da cama...");
}

0 references
public void TomarCafeDaManha()
{
    Console.WriteLine("\n Tomando café da manhã...");
}

0 references
public void IrParaOTrabalho()
{
    Console.WriteLine("\n Indo para o trabalho...");
}

0 references
public void VoltarParaCasa()
{
    Console.WriteLine("\n Voltando para casa...");
}

0 references
public void Dormir()
{
    Console.WriteLine("\n Indo dormir...");
}

// Este método é 'abstrato' e será implementado pelas subclasses
0 references
public abstract void Trabalhar();
```

Análise e Criação

Métodos padrão

Métodos que podem variar

```
public abstract class Trabalhador
{
    //Este é o 'template method'
    1 reference
    public void RotinaDiaria()
    {
        Levantar();
        TomarCafeDaManha();
        IrParaOTrabalho();
        Trabalhar();
        VoltarParaCasa();
        Dormir();
    }
```

```
    // Este é um método 'HOOK'
    1 reference
    public abstract void Trabalhar();
}
```

Definição e Implementação

Método Template 'Modelo'

Método Hook 'Gancho'

```
namespace TemplateMethod
{
    0 references
    public class Carteiro : Trabalhador
    {
        2 references
        public override void Trabalhar()
        {
            Console.WriteLine("\n Entrega correspondências e encomendas...");
        }
    }
}
```

Execução

SubClasse01

```
namespace TemplateMethod
{
    1 reference
    public class Bombeiro : Trabalhador
    {
        2 references
        public override void Trabalhar()
        {
            Console.WriteLine("\n Combate a incêndios e salva vidas...");
        }
    }
}
```

SubClasse02

```
namespace TemplateMethod
{
    0 references
    public class Program
    {
        0 references
        static void Main(string[] args)
        {
            Trabalhador bombeiro = new Bombeiro();
            Console.WriteLine("\n Rotina de um bombeiro:");
            bombeiro.RotinaDiaria();

            Console.WriteLine();

            Trabalhador carteiro = new Carteiro();
            Console.WriteLine("\n Rotina de um carteiro:");
            carteiro.RotinaDiaria();

            Console.ReadKey();
        }
    }
}
```

Na prática

Partes de um algoritmo permanecem as mesmas, enquanto outras partes precisam variar

```
Rotina de um bombeiro:
Levantando da cama...
Tomando café da manhã...
Indo para o trabalho...
Combate a incêndios e salva vidas...
Voltando para casa...
Indo dormir...


Rotina de um carteiro:
Levantando da cama...
Tomando café da manhã...
Indo para o trabalho...
Entrega correspondências e encomendas...
Voltando para casa...
Indo dormir...
```

Benefícios para O SISTEMA

Torna-o mais compreensível e Fácil de manter

REUTILIZAÇÃO DO CÓDIGO

Reduz a quantidade de código necessário para implementar variações do algoritmo.

MINIMIZA DUPLICIDADE

Evita a duplicação de código ao isolar as partes comuns do algoritmo em uma superclasse.

LIMPEZA E MODULARIDADE

Reduz a redundância e aprimorando a leitura. Aumenta a modularidade, permitindo que partes específicas do algoritmo sejam alteradas ou estendidas de maneira independente, sem afetar o restante do sistema.

Referências

- https://learn.microsoft.com/en-us/shows/Visual-Studio-Toolbox/Design-Patterns-Template-Method
- https://sourcemaking.com/design_patterns/template_method
- https://refactoring.guru/design-patterns/template-method
- https://medium.com/xp-inc/design-patterns-parte-24-template-method-69e3a7927dcd
- https://www.devmedia.com.br/padrao-de-projeto-template-method-em-java/26656
- https://www.thiengo.com.br/padrao-de-projeto-template-method-metodo-template

ALUNO: PAULO ROBERTO SOUZA LISBOA

PROFESSOR: RALPH LIMA

DISCIPLINA: ARQUITETURA DE SOFTWARE

Agosto de 2023