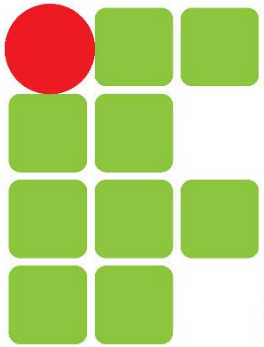
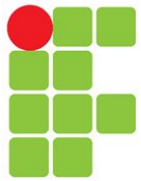


INSTITUTO FEDERAL
SÃO PAULO



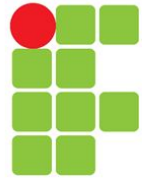
Implementando Componentes em Java

Desenvolvimento de Componentes (BRADECO)

Prof. Luiz Gustavo Diniz de Oliveira Véras

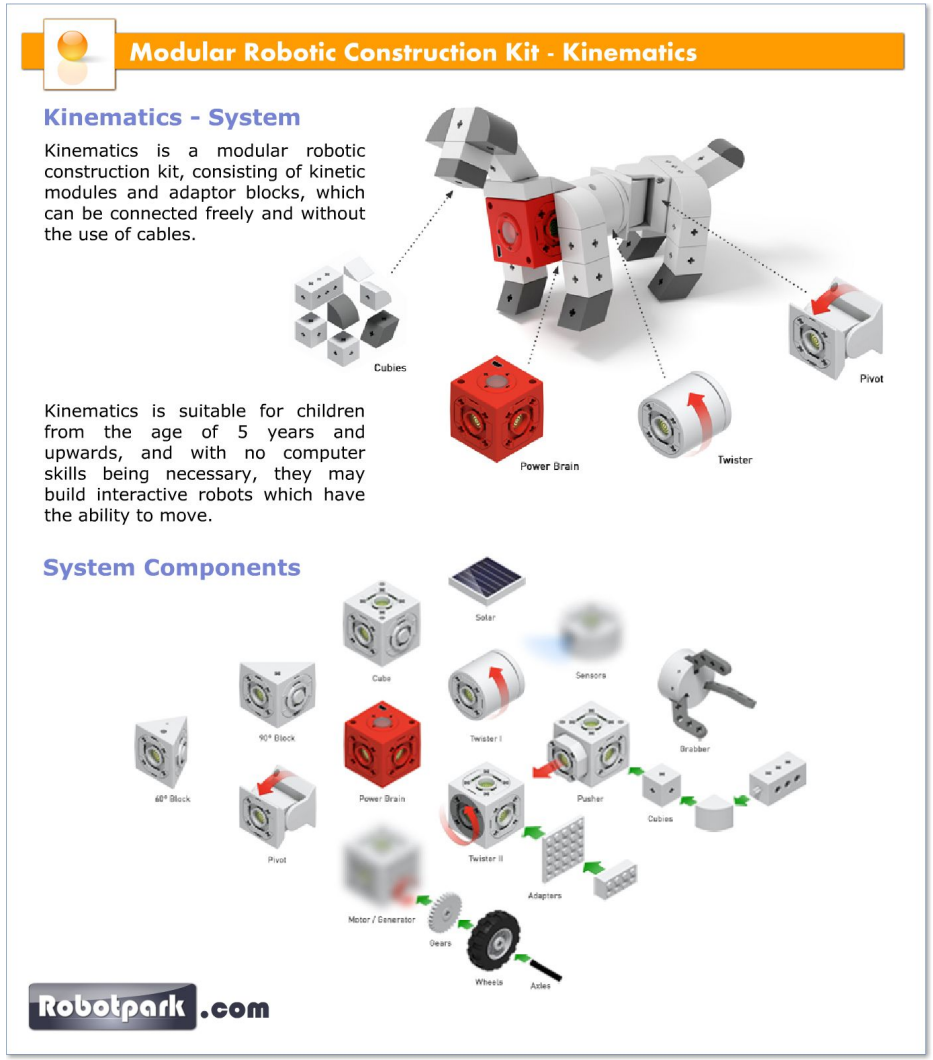
E-mail: gustavo_veras@ifsp.edu.br

Conteúdo



- ✔ O porquê de Componentes
- ✔ Componentes *WhiteBox* e *BlackBox*
- ✔ Etapas de implementação de componentes.
- ✔ Ilustrar as etapas com um exemplo.
- ✔ Exercícios

O porquê de componentes?



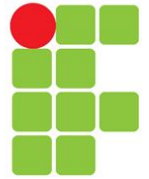
Reutilizável

O porquê de componentes?

Características desejáveis

Fonte: https://youtu.be/vyYN5Setfv0?si=wzGS9Nrxqi0sCicc

Componentes *WhiteBox* e *BlackBox*

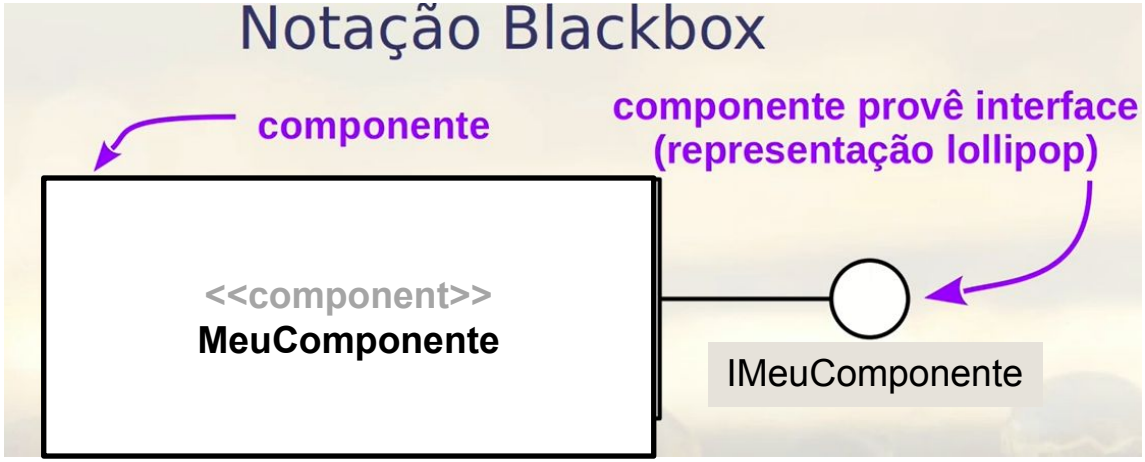


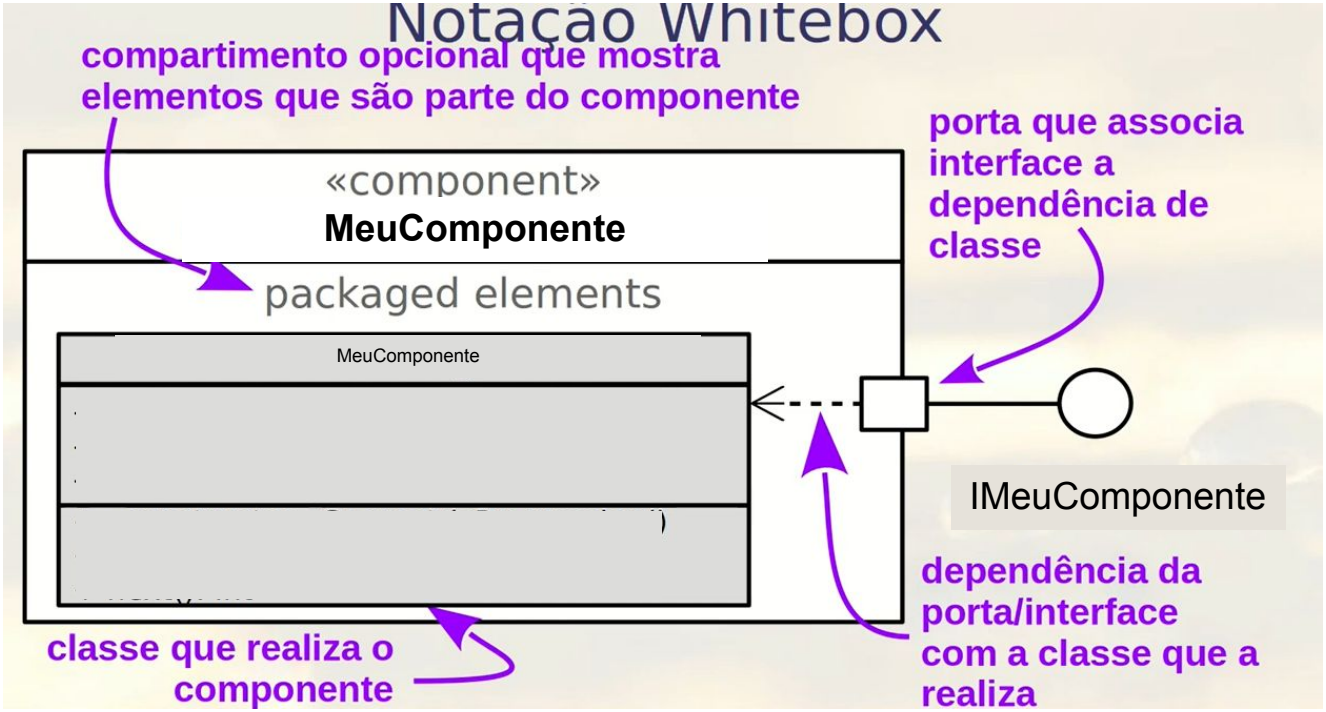
As abordagens de se trabalhar com componentes:

- **Encapsulamento caixa-branca(*White box wrapping*)** – aqui, a implementação do componente é diretamente modificada para resolver incompatibilidades. Isso é, obviamente, possível apenas se o código-fonte do componente estiver disponível, algo extremamente improvável em componentes proprietários.
- **Encapsulamento caixa-preta(*Black box wrapping*)** – Caso mais comum, onde não é possível o acesso ao código-fonte, e a única maneira de adaptar o componente é por pré/pós processamento a nível de interface.

Componentes *WhiteBox* e *BlackBox*

BlackBox e *WhiteBox* em diagramas





Fonte: https://www.youtube.com/watch?v=Y0S6RBIbXKk

Etapas de implementação de componentes.

ETAPAS

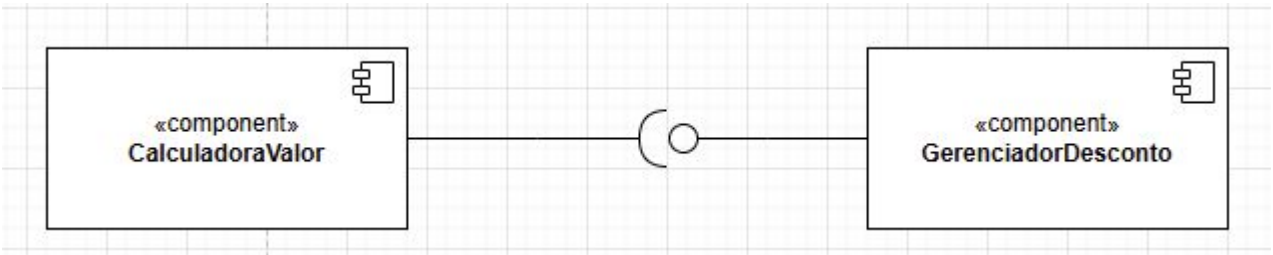
1. **Identificação de Funções com Dependências**
   - Identificar funções que possuem dependências entre si
   - Analisar o fluxo de dados e chamadas entre as funções
2. **Transformação em Classes com Dependência Direta**
   - Converter funções em métodos de classes . Seus parâmetros devem virar campo das classes.
   - Implementar dependência direta através da instanciação de objetos
   - Identificar problemas de acoplamento forte.
3. **Criação de Conectores entre Componentes**
   - Desenvolver métodos intermediárias (conectores), implementando métodos que encapsulam a comunicação entre componentes.
   - Reduzir o acoplamento direto entre as classes.
4. **Definição de Interface**
   - Criar interfaces Java para cada componente
   - Implementar as interfaces nas classes existentes
   - Estabelecer contratos claros entre os componentes

Exemplo

Descrição do Cenário

Um proprietário de uma pequena loja online precisa de um **sistema automatizado para gerenciar seus pedidos** em crescimento. Ele busca uma solução que **calcule** com precisão **o valor total dos pedidos** e **aplique descontos para clientes** frequentes, permitindo processar vendas rapidamente e recompensar a fidelidade dos compradores. A implementação baseada em componentes foi recomendada por uma consultoria, separando as responsabilidades entre um **componente de cálculo de valor**, **um gerenciador de descontos** e um **conector**, garantindo assim um **sistema flexível** que possa evoluir para acomodar novas regras de negócio e tipos de promoção conforme a loja cresce.

Exemplo



ETAPA 1. Identificação de Funções com Dependências

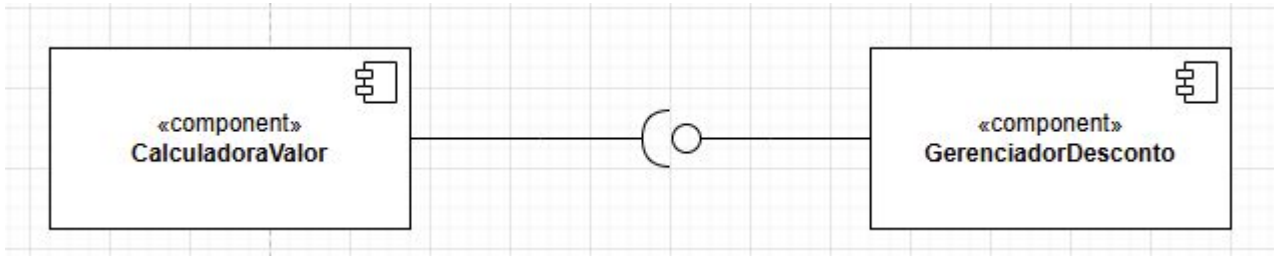
componente de cálculo de valor

```
    // Função para calcular o valor total do pedido
    public static double calcularValorTotal(String[] itens, double[] precos) {
        double total = 0;
        for (int i = 0; i < itens.length; i++) {
            total += precos[i];
        }
        return total;
    }
```

um gerenciador de descontos

```
// Função para aplicar desconto (depende da função de calcular valor total)
    public static double aplicarDesconto(String[] itens, double[] precos, double
percentualDesconto) {
        double valorTotal = calcularValorTotal(itens, precos); // aqui está a conexão
        double valorDesconto = valorTotal * (percentualDesconto / 100);
        return valorTotal - valorDesconto;
    }
```

Exemplo



ETAPA 2 - Transformação em Classes com Dependência Direta

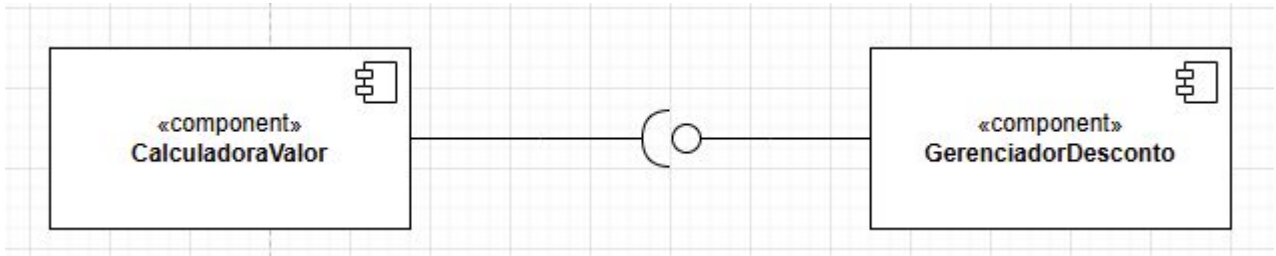
Calculadora Valor

```
// Classe para cálculo de valor
class CalculadoraValor {
    private String[] itens;
    private double[] precos;

    public CalculadoraValor(String[] itens, double[] precos) {
        this.itens = itens;
        this.precos = precos;
    }

    public double calcularValorTotal() {
        double total = 0;
        for (int i = 0; i < itens.length; i++) {
            total += precos[i];
        }
        return total;
    }
}
```

Exemplo



ETAPA 2 - Transformação em Classes com Dependência Direta

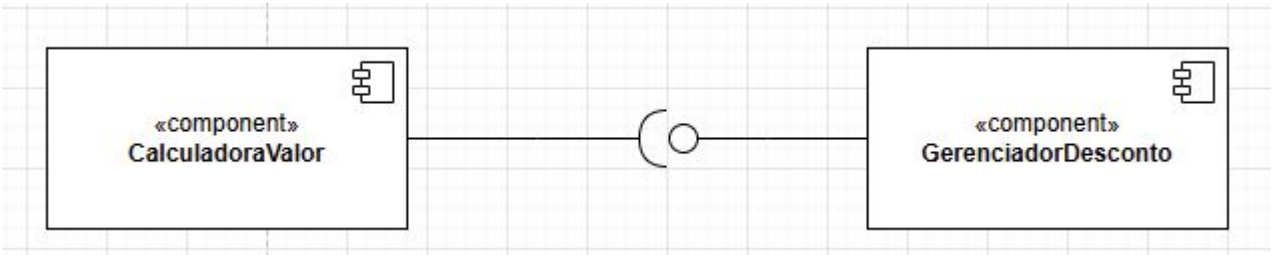
GerenciadorDesconto

```
// Classe para aplicação de desconto com dependência direta
class GerenciadorDesconto {
    private double  percentualDesconto;

    public GerenciadorDesconto(double percentualDesconto){
        this.percentualDesconto = percentualDesconto;
    }

    public double aplicarDesconto(CalculadoraValor calculadora) {
        // Dependência direta da classe CalculadoraValor
        double valorTotal = calculadora.calcularValorTotal(); //Dependência
        double valorDesconto = valorTotal * (this.percentualDesconto / 100);
        return valorTotal - valorDesconto;
    }
    // getters e setters omitidos
}
```

Exemplo



ETAPA 3 - Criação de Conectores entre Componentes

Calculadora Valor

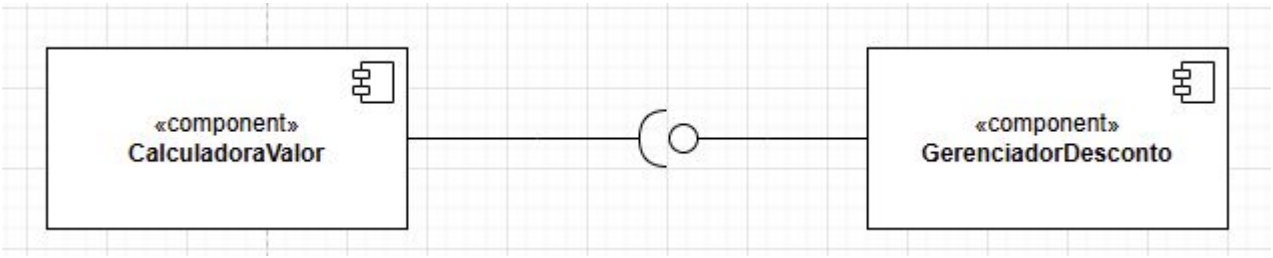
```
// Adicionar em CalculadoraValor
// Interface com GerenciadorDesconto
public void connect(GerenciadorDesconto componente){
    componente.update(this.calcularValorTotal());
}
```

GerenciadorDesconto

```
public double aplicarDesconto(double valorTotal) {
    // double valorTotal = calculadora.calcularValorTotal(); //Dependencia
    double valorDesconto = valorTotal * (this.percentualDesconto / 100);
    return valorTotal - valorDesconto;
}

// Interface com CalculadoraValor
public void update(double valorTotal){
    System.out.println("Desconto: " + aplicarDesconto(valorTotal));
}
```

Exemplo

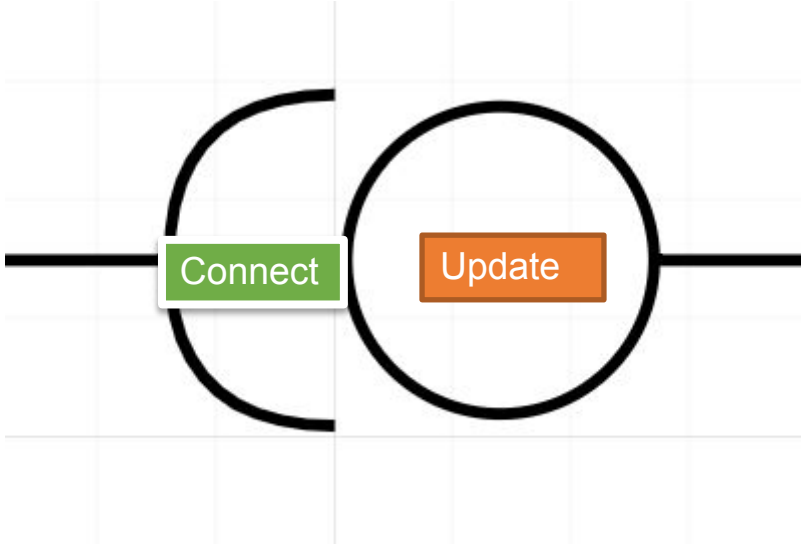


ETAPA 4 - Definição de Interface

Calculadora Valor

```
class CalculadoraValor implements ICalculadoraValor{
```

```
public interface IRCalculadoraValor {
    public void connect(IGerenciadorDesconto componente);
}
```



GerenciadorDesconto

```
class GerenciadorDesconto implements IGerenciadorDesconto{
}
```

```
public interface IGerenciadorDesconto {
    public void update(double valorTotal);
}
```

Exercícios

Nos exercícios que seguem, aplique o mesmo passo a passo demonstrado no exemplo, mas em diferentes contextos. Para cada exercício, você deve:

Criar duas funções com dependência entre si -> Transformar essas funções em classes, onde uma chama diretamente um objeto da outra -> Criar um conector entre a interface requerida e fornecida (usando métodos) -> Implementar interfaces Java para finalizar o modelo de componentes -> Realizar a conexão entre eles no *main* -> Desenhe o Diagrama de Componentes em UML.

Exercício 1: Sistema Financeiro

Implemente um sistema onde uma função calcula juros compostos e outra função gera um plano de amortização de empréstimo. A função de amortização depende do cálculo de juros.

Contexto: Sistema bancário para simulação de empréstimos.

Exercícios

Nos exercícios que seguem, aplique o mesmo passo a passo demonstrado no exemplo, mas em diferentes contextos. Para cada exercício, você deve:

Criar duas funções com dependência entre si -> Transformar essas funções em classes, onde uma chama diretamente um objeto da outra -> Criar um conector entre a interface requerida e fornecida (usando métodos) -> Implementar interfaces Java para finalizar o modelo de componentes -> Realizar a conexão entre eles no *main* -> Desenhe o Diagrama de Componentes em UML.

Exercício 2: E-commerce

Crie um sistema onde uma função valida o estoque disponível e outra processa um pedido. A função de processamento depende da validação do estoque.

Contexto: Plataforma de comércio eletrônico.

Exercícios

Nos exercícios que seguem, aplique o mesmo passo a passo demonstrado no exemplo, mas em diferentes contextos. Para cada exercício, você deve:

Criar duas funções com dependência entre si -> Transformar essas funções em classes, onde uma chama diretamente um objeto da outra -> Criar um conector entre a interface requerida e fornecida (usando métodos) -> Implementar interfaces Java para finalizar o modelo de componentes -> Realizar a conexão entre eles no *main* -> Desenhe o Diagrama de Componentes em UML.

Exercício 3: Gerenciamento de Recursos Humanos

Desenvolva um sistema onde uma função calcula horas trabalhadas e outra calcula a folha de pagamento. A função de cálculo de pagamento depende da contabilização das horas.

Contexto: Sistema de RH para empresas.

Fontes

Canal do Professor André Santachè - Unicamp

https://www.youtube.com/c/Andr%C3%A9Santanch%C3%A8

Vídeos

Componentização Passo a Passo em Python - Passo 3 / Relação - Componentes de Software e Reúso 2024

https://youtu.be/m3l6Cm-nte4?si=2pfcjWyRg1o58nHH

Aplicação baseada em Componentes em Java - Aula 17/05 – Programação Orientada a Objetos 2022

https://www.youtube.com/watch?v=ORFV0CWtqkE