# A4 – APLICANDO CONHECIMENTO

| | |
|---|---|
| **COMPONENTE CURRICULAR:** | DESENVOLVIMENTO DE SISTEMAS II |
| **NOME DO ALUNO:** | JOÃO PEDRO LIMA LUSTOSA AMORIM |
| **RA:** | 10289920 |

1. **Definição de Polimorfismo:**

   Polimorfismo é um dos pilares da Programação Orientada a Objetos (POO) que permite o tratamento de uniformidade entre diferentes objetos por meio de uma interface ou classe. Existem vários tipos de polimorfismo, por exemplo: Subtyping, Overloading, Coercion Polymorphism, Parametric Polymorphism, Structural Polymorphism e Row Polymorphism. Podemos destacar os dois mais conhecidos que são Overloading (também conhecido como sobre carga) que ocorre quando várias funções ou métodos são definidos com parâmetros diferentes, e o Subtyping (ou Polimorfismo de Inclusão) ocorre quando subclasses fornecem diferentes implementações de algum método da superclasse, ou seja, a classe filha pode atuar como base através da herança sem alterar o comportamento do programa.

2. **Exemplo prático:**

```
public class Supermercado {
    public static void processarPagamento(FormaPagamento metodo, double valor)
{
        metodo.realizarPagamento(valor);
    }

    public static void main(String[] args) {
        FormaPagamento pagamento1 = new Dinheiro();
        FormaPagamento pagamento2 = new CartaoCredito();
        FormaPagamento pagamento3 = new ValeRefeicao();
        FormaPagamento pagamento4 = new Cheque();

        processarPagamento(pagamento1, 150.00);
        processarPagamento(pagamento2, 300.00);
```

```
        processarPagamento(pagamento3, 200.00);
        processarPagamento(pagamento4, 400.00);
    }

    @Override
    public String toString() {
        return "Supermercado []";
    }
}

interface FormaPagamento {
    void realizarPagamento(double valor);
}

class Dinheiro implements FormaPagamento {
    @Override
    public void realizarPagamento(double valor) {
        System.out.println("O pagamento foi realizado em dinheiro: R$" +
valor);
    }
}

class CartaoCredito implements FormaPagamento {
    @Override
    public void realizarPagamento(double valor) {
        if (valor > 150) {
            System.out.println("O pagamento foi realizado com cartão de
crédito e pode ser parcelado em até 2x.");
            double parcela = valor / 2;
            System.out.println("Valor total: R$" + valor + ". Parcelas de R$"
+ parcela + " em 2 vezes.");
        } else {
            System.out.println("O pagamento foi realizado com cartão de
crédito: R$" + valor);
        }
    }
}

class ValeRefeicao implements FormaPagamento {
    @Override
    public void realizarPagamento(double valor) {
        System.out.println("O pagamento foi realizado com vale-refeição no
valor de: R$" + valor);
    }
}

class Cheque implements FormaPagamento {
```

```
    @Override
    public void realizarPagamento(double valor) {
        System.out.println("O pagamento foi realizado com cheque no valor de:
R$" + valor);
    }
}
```

Para a construção deste código foi utilizado a interface FormaPagamento para definir o método realizarPagamento que é implementado pelas formas de pagamento Dinheiro, CartaoCredito, ValeRefeicao e Cheque que tem comportamento próprio. Na classe CartaoCredito foi adicionado uma lógica para parcelamento do pagamento em até 2 vezes para compras acima de 150 reais. Para testar a aplicabilidade do polimorfismo, no método main são testadas as formas de pagamento com o intuito de provar a flexibilidade e a extensibilidade do sistema para caso seja necessário adicionar novas formas de pagamento sem precisar modificar a lógica principal do sistema Supermercado.

3. **Pure Fabrication:**

O Padrão GRASP (General Respojnsibility Assignment Software Patterns) foi criado com o intuito de tornar o código mais flexível, facilitando sua manutenção e a extensibilidade. O Padrão GRASP Pure Fabrication (ou Padrão de Pura Fabricação) é uma classe que não pertence diretamente ao domínio do programa. Esta classe tem a função de prestar suporte às outras classes, contribuindo assim com o alívio de acoplamentos nas outras classes e aumentando a coesão do sistema. Como benefício de aplicabilidade deste padrão temos a reutilização do código com mais facilidade.

Como um exemplo prático deste padrão, podemos citar um sistema de controle de estoque no qual é possível criar uma classe lógica ControladorEstoque que se comunicaria diretamente com o banco de dados para armazenar e recuperar produtos, não necessariamente tendo que fazer parte do programa, mas caso definida, ajuda na organização do mesmo.

4. **Indirection:**

O Padrão GRASP de indireção ou *Indirection* ajuda a manter o baixo acoplamento designando reponsabilidades através de uma classe mediadora.

Um exemplo de Padrão GRASP aplicado é um sistema de mensagens no qual uma classe mediadora GerenciadorMensagens gerencia o envio e recebimento de mensagens, seja por sms, email ou aplicativos, sem ter que alterar o objeto Usuário.

5. **Protected Variations:**

O padrão GRASP Proteção Contra Variações (ou *Protected Variations*) protege o sistema de mudanças através de pontos de variação, ou seja encapsulamentos de comportamentos, que permitem alterações sem impactar o restante do sistema.

Um exemplo de Protected Variations aplicado é criar uma interface GatewayPagamento para prevenir um sistema de pagamento das mudanças de um gateway específico, preservando assim a lógica principal do sistema.