MAPA – Material de Avaliação Prática da Aprendizagem

Acadêmico: Douglas Marcelo Monquero	**R.A.:** 23343540-5
Curso: BACHARELADO EM ENGENHARIA DE SOFTWARE	
Disciplina: PROGRAMAÇÃO ORIENTADA A OBJETOS	
Valor da atividade: 3,50	**Prazo: 24/11/2024**

Instruções para Realização da Atividade

1. Todos os campos acima deverão ser devidamente preenchidos;
2. Utilize deste formulário para a realização do MAPA;
3. Esta é uma atividade INDIVIDUAL. Caso identificado cópia de colegas, o trabalho de ambos sofrerá decréscimo de nota;
4. Utilizando este formulário, realize sua atividade, salve em seu computador, renomeie e envie em forma de anexo;
5. Formatação exigida para esta atividade: documento Word, Fonte Arial ou Times New Roman tamanho 12, Espaçamento entre linhas 1,5, texto justificado;
6. Ao utilizar quaisquer materiais de pesquisa referencie conforme as normas da ABNT;
7. Critérios de avaliação: Utilização do Template; Atendimento ao Tema; Constituição dos argumentos e organização das Ideias; Correção Gramatical e atendimento às normas ABNT;
8. Procure argumentar de forma clara e objetiva, de acordo com o conteúdo da disciplina.

Em caso de dúvidas, entre em contato com seu Professor Mediador.

Bons estudos!

a) Explique o conceito de herança em Java e explique como ela pode ser utilizada para promover a reutilização de código. Dê um exemplo de como uma classe base e uma classe derivada podem ser implementadas em Java.
b) Defina polimorfismo e descreva como ele pode ser utilizado em Java para permitir a utilização de métodos de forma dinâmica. Forneça um exemplo de código em Java que demonstre o uso de polimorfismo.

Introdução

Para se enquadrar no paradigma de programação orientado a objetos, é necessário atender a alguns requisitos, esses são chamados de os quatro pilares da programação orientada a objetos, os quais são: abstração, encapsulamento, herança e polimorfismo.

Abstração seria transformar em código uma entidade do mundo real, definindo a essa classe atributos e métodos, os quais farão sentido para o nosso projeto. Ou seja, quando trazemos para o nosso código a definição de uma classe pessoa, definimos quais atributos, ou características, são importantes para a nossa aplicação, por exemplo, uma matricula, nome, endereço e telefone de contato. Também iremos abstrair os métodos importantes para definir o que a pessoa irá fazer em nosso código, como por exemplo, se essa pessoa irá recorrer um cadastro.

Desta forma, ao criarmos objetos desta classe pessoa, poderemos a cada objeto reutilizar esses atributos e métodos gerais e criarmos um ente único e determinado.

Encapsulamento, diz respeito a segurança, pelo fato de que essa prática esconde as propriedades da classe numa espécie de cápsula. Desta forma, somente terão acesso aos atributos da nossa classe os métodos pertencentes a ela. Este procedimento visa evitar o acesso direto as propriedades do objeto, aplicando uma camada de proteção a nossa aplicação. Júnior (2024) explica que “a ideia principal do encapsulamento envolve omitir os membros de uma classe, além de esconder como funcionam as rotinas ou regras de negócio” (pág. 103).

Herança em Java

Chegamos agora, ao contexto do nosso trabalho, o desenvolvimento de software orientado a objetos deve promover a modularidade, reutilização e extensibilidade do

código. As técnicas de herança e o polimorfismo, permitem a criação de sistemas mais flexíveis e adaptáveis às mudanças e demandas do cotidiano profissional.

A herança permite que uma classe (denominada classe derivada ou subclasse) herde as características de outra classe (denominada classe base ou superclasse). Essa abordagem promove a reutilização de código, uma vez que funcionalidades comuns podem ser definidas na superclasse e compartilhadas pelas subclasses, evitando a redundância.

Em Java, cabe ressaltar, só é permitido a herança simples, ou seja, uma classe não pode herdar métodos e atributos de mais de uma classe, ou seja, não é permitida a herança múltipla. Esse assunto é tratado com interfaces e não será tratado nesse contexto.

Superclasse	Subclasses
Aluno	AlunoDeGraduação, AlunoDePósGraduação
Forma	Círculo, Triângulo, Retângulo, Esfera, Cubo
Financiamento	FinanciamentoDeCarro, FinanciamentoDeCasa
Empregado	CorpoDocente, Funcionário
ContaBancária	ContaCorrente, ContaPoupança

Figura 01: Exemplos de Herança
Fonte: Deitel (2010, pág. 280)

Como exemplo prático de herança, imagine um sistema bancário com diferentes tipos correntistas. Podemos criar uma classe base Pessoa, que contém atributos e métodos comuns a todos as contas, e subclasses específicas como Pessoa Física e Pessoa Jurídica, que herdam esses atributos e métodos e, opcionalmente, adicionam suas particularidades.

Segue o exemplo de implementação em Java, adaptado do capítulo V – Herança e Polimorfismo em Java – Professores Edson A. de Oliveira Junior e ME. André A. Noel.

SuperClasse Pessoa:

```
package Dados;

import java.util.Scanner;
```

```
import Localizacao.Endereco;

public abstract class Pessoa {
    protected String nome;
    protected String telefone;
    protected Endereco e = new Endereco();

    public Pessoa() {
        // Construtor padrão explícito
    }

    public void cadastra(){
        Scanner scan = new Scanner(System.in);
        System.out.println("Bem vindo ao Cadastro de Conta");
        System.out.print("Digite seu nome: ");
        nome = scan.nextLine();
        System.out.print("Digite seu telefone: ");
        telefone = scan.nextLine();
        e.cadastraEndereco();

    }

    public void imprime(){
        System.out.println("Nome: " + getNome());
    }

    //getters and setters


    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getTelefone() {
        return telefone;
    }

    public void setTelefone(String telefone) {
```

```
        this.telefone = telefone;
    }

    public Endereco getE() {
        return e;
    }

    public void setE(Endereco e) {
        this.e = e;
    }

    @Override
    public String toString() {
        return "Dados.Pessoa{" +
                "nome='" + nome + '\'' +
                ", telefone='" + telefone + '\'' +
                ", e=" + e +
                '}';
    }
}
```

Sub classe Pessoa Física:

```
package Dados;

import java.util.Scanner;

public class Pfisica extends Pessoa {
    private String cpf;

    public Pfisica() {
        // Construtor padrão explícito
    }

    public void setCpf(String cpf){
        this.cpf = cpf;
    }

    public String getCpf(){
        return this.cpf;
    }
```

```
    @Override
    public void cadastra(){
        super.cadastra();
        System.out.println("---- Cadastro de Dados Pessoa Física ----");
        Scanner scan = new Scanner(System.in);
        System.out.print("Digite seu cpf: ");
        cpf = scan.nextLine();
        System.out.println("Cadastro PF realizado com sucesso");
        System.out.println("Pressione Enter para continuar...");
        scan.nextLine(); // Espera o usuário pressionar Enter
    }
    @Override
    public void imprime(){
        super.imprime();
        System.out.print("CPF: " + this.getCpf());
        System.out.println();
    }

    @Override
    public String toString() {
        return "Dados.Pfisica{" +
                "cpf='" + this.getCpf() + '\'' +
                ", nome='" + getNome() + '\'' +
                ", telefone='" + getTelefone() + '\'' +
                ", e=" + getE() +
                '}';
    }
}
```

Subclasse Pessoa Jurídica:

```
package Dados;

import java.util.Scanner;

public class Pjuridica extends Pessoa {
    private String cnpj;

    public Pjuridica() {
        // Construtor padrão explícito
    }
```

```
    public void setCnpj(String cnpj){
        this.cnpj = cnpj;
    }

    public String getCnpj(){
        return cnpj;
    }

    @Override
    public void cadastra(){
        super.cadastra();
        System.out.println("---- Cadastro de Dados Pessoa Jurídica ----");
        Scanner scan = new Scanner(System.in);
        System.out.print("Digite seu cnpj: ");
        cnpj = scan.nextLine();
        System.out.println("Cadastro PJ realizado com sucesso3");
        System.out.println("Pressione Enter para continuar...");
        scan.nextLine(); // Espera o usuário pressionar Enter
    }
    @Override
    public void imprime(){
        super.imprime();
        System.out.print("CNPJ: " + this.getCnpj());
        System.out.println();
    }

    @Override
    public String toString() {
        return "Dados.Pjuridica{" +
                "cnpj='" + this.getCnpj() + '\'' +
                ", nome='" + getNome() + '\'' +
                ", telefone='" + getTelefone() + '\'' +
                ", e=" + getE() +
                '}';
    }
}
```

Observe que as classes filhas possuem uma palavra reservada extends, essa indicação define as classes como classes filhas (Física e Jurídica) que herdam da classe pai (Pessoa), fazendo com que todas os atributos e métodos da classe pai

sejam herdados para a classe filha, não sendo necessário desta forma a repetição de códigos, pois utilizamos o principio da reutilização dos mesmos.

Veja que para a classe Pessoa Física, definimos somente o atributo cpf e para a classe Pessoa Jurídica definimos o atributo cnpj, sendo estes únicos atributos que diferenciam as classes filhas, os demais serão herdados da classe pai.

Preferimos nesse exemplo de superclasse, colocarmos o modificador abstract que, uma classe abstrata, como ensina Deitel, possuem o proposito de “fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim podem compartilhar um design comum.” (pag. 309).

Esse método de aproveitamento de código fornece benefícios notáveis no desenvolvimento de aplicações. Podemos citar a definição dos autores Edson A. de Oliveira Junior e André A. Noel (2024), “ao concentrarmos características comuns em uma classe e derivar as classes mais específicas a partir desta, nós estamos preparados para a adição de novas funcionalidades ao sistema. Se mais adiante uma nova propriedade comum tiver que ser adicionada, não precisaremos efetuar alterações em todas as classes. Basta alterar a superclasse e todas as outras classes derivadas serão automaticamente atualizadas.” (pág. 138).

Polimorfismo em Java

O polimorfismo, como nos ensina nossos professores no livro didático e em suas aulas, significa, “varias (poli) formas (morfo)” ou seja, refere-se à capacidade de um objeto de assumir diferentes formas (Júnior, 2024, pág. 138). Em Java, ele é amplamente utilizado por meio de sobrescrita de métodos, permitindo que o mesmo método seja chamado em objetos de diferentes classes derivadas de uma mesma superclasse, proporcionando flexibilidade e extensibilidade ao código, permitindo a construção de comportamentos distintos nas classes herdadas, dos comportamentos dos métodos da superclasse.

Vale ressaltar que o polimorfismo, como ressaltam os professores Noel e Junior, em Java se manifesta somente nos métodos e está intimamente ligado ao conceito de herança e é implementado utilizando-se o atributo @Override, pois para

que o método seja implementado de forma correta, deve-se utilizar exatamente a mesma identificação de assinatura (Júnior, 2024, pág. 138).

Um exemplo prático é quando diferentes tipos de objetos compartilham um método com comportamentos específicos para cada classe. Utilizando o mesmo sistema de gerenciamento bancário, podemos ilustrar o conceito:

Exemplo de implementação em Java:

Método cadastra em Pessoa:

```
public void cadastra(){
    Scanner scan = new Scanner(System.in);
    System.out.println("Bem-vindo ao Cadastro de Conta");
    System.out.print("Digite seu nome: ");
    nome = scan.nextLine();
    System.out.print("Digite seu telefone: ");
    telefone = scan.nextLine();
    e.cadastraEndereco();

}
```

Método cadastra em PFisica:

```
@Override
public void cadastra(){
    super.cadastra();
    System.out.println("---- Cadastro de Dados Pessoa Física ----");
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite seu cpf: ");
    cpf = scan.nextLine();
    System.out.println("Cadastro PF realizado com sucesso");
    System.out.println("Pressione Enter para continuar...");
    scan.nextLine(); // Espera o usuário pressionar Enter
}
```

Método cadastra em Pjuridica:

```
@Override
public void cadastra(){
    super.cadastra();
    System.out.println("---- Cadastro de Dados Pessoa Jurídica ----");
    Scanner scan = new Scanner(System.in);
    System.out.print("Digite seu cnpj: ");
    cnpj = scan.nextLine();
    System.out.println("Cadastro PJ realizado com sucesso3");
    System.out.println("Pressione Enter para continuar...");
    scan.nextLine(); // Espera o usuário pressionar Enter
}
```

Note-se que na classe Pessoa não utilizamos a anotação @Overrride, que identifica que este método está sendo reescrito, justamente por ela ser a superclasse, desta forma, utilizamos as notações nas classes que herdam da classe pai. Junior (2024) conclui que “o código, além de legível, obriga ao compilador a aplicar as regras de reescrita para essa marcação” (p. 153).

Em nosso projeto utilizamos o mesmo princípio e outros dois métodos, imprime() e toString(), vale ressaltar que esse último é herdado da classe Object que é o pai de todas as classes em Java. Sem a reescrita desse método nas classes filhas ao chamar o método, ele traria o endereço em memória que nosso objeto foi salvo no momento de sua criação. Essa reescrita nos faz capaz de imprimir como retorno os atributos de nossa classe.

Primeiramente o sistema busca o método cadastro generalizado na classe Pessoa e depois, dependendo da escolha, busca o método cadastro em cada classe estendida. Primeiro vemos o cadastro de pessoa física e depois vemos o cadastro de pessoa jurídica, ambos chamando o método polimorfo cadastra().

O mesmo método cadastro trazendo duas formas de cadastro:

```
Digite:
1: Cadastro de Dados Pessoa Física:
2: Cadastro de Dados Pessoa Juridica:
3: Imprime Clientes:
4: Sair
1
Bem vindo ao Cadastro de Conta
Digite seu nome: Douglas Marcelo Monquero
Digite seu telefone: 44999013434
Bem vindo ao Cadastro de Endereço
Digite sua rua: dos Ipes
Digite seu numero: 312
Digite sua Cidade: Maringa
Digite seu Estado: Pr
---- Cadastro de Dados Pessoa Física ----
Digite seu cpf: 021.123.123-11
Cadastro PF realizado com sucesso
Pressione Enter para continuar...
```

```
Digite:
1: Cadastro de Dados Pessoa Física:
2: Cadastro de Dados Pessoa Juridica:
3: Imprime Clientes:
4: Sair
2
Bem vindo ao Cadastro de Conta
Digite seu nome: Extreme Informática
Digite seu telefone: 4432568978
Bem vindo ao Cadastro de Endereço
Digite sua rua: do Silício
Digite seu numero: 821
Digite sua Cidade: Maringá
Digite seu Estado: Pr
---- Cadstro de Dados Pessoa Jurídica ----
Digite seu cnpj: 12.456.789/0001-23
Cadastro PJ realizado com sucesso3
Pressione Enter para continuar...
```

Figura 02: Cadastro Pessoa Física Figura 03: Cadastro Pessoa Jurídica

Ainda temos o método imprime(), o qual vemos claramente que é herdado da superclasse e depois complementado nas classes filhas.

Esse método busca o nome do cadastro na lista de pessoa e o cpf ou o cnpj nos métodos da classe filha, como ilustra a seguinte figura.

```
Clientes cadastrados:
Cliente(1)
Nome: Douglas Marcelo Monquero
CPF: 021.123.123-11

Cliente(2)
Nome: Lucas Evangelista
CPF: 123.456.789-99

Cliente(3)
Nome: Extreme Informática
CNPJ: 12.456.789/0001-23
```

Figura 03: resultado da impressão dos clientes cadastrados

Fonte: autor

Ao final do trabalho apresento o método principal e a classe endereço responsáveis por executar o nosso projeto e cadastrar o endereço respectivamente.

Conclusão

Os conceitos de herança e polimorfismo são pilares fundamentais da programação orientada a objetos, permitindo que desenvolvedores criem sistemas modulares, reutilizáveis e adaptáveis. A herança possibilita o reaproveitamento de código ao compartilhar atributos e métodos comuns entre classes, enquanto o polimorfismo adiciona flexibilidade, permitindo que métodos sejam utilizados de forma dinâmica em diferentes contextos.

O exemplo apresentado, baseado no gerenciamento de cadastros bancários, ilustra como essas técnicas podem ser aplicadas para resolver problemas reais, reduzindo a redundância e aumentando a eficiência no desenvolvimento de software. Além disso, o uso de práticas como classes abstratas e sobrescrita de métodos demonstra a capacidade de criar sistemas extensíveis e preparados para evoluir conforme as necessidades do projeto.

Por fim, o estudo desses conceitos contribui significativamente para a formação de profissionais capacitados, capazes de alinhar teoria e prática, e destaca a importância de boas práticas no desenvolvimento de software orientado a objetos.

Referência

DEITEL, Harvey M. Java - Como Programar. 8. ed. Prentice Hall Brasil, 2010.

Junior, Edson A. Oliveira, Noel, Andre Abdala. Programação Orientada a Objetos. Florianopolis, Sc: Arqué, 2023.

Hudson, 2012, Devmedia. https://www.devmedia.com.br/entendendo-e-aplicando-heranca-em-java/24544 acesso em 20/11/2024.

Igor, 2012, Devmedia. https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140 acesso em 20/12/2024

Anexo

Outras Classes do Projeto:

Classe Endereço:

```
package Localizacao;

import java.util.Scanner;

public class Endereco {
    private String rua;
    private int numero;
    private String cidade;
    private String uf;

    public Endereco() {
        // Construtor padrão explícito
    }

    public void cadastraEndereco(){
        Scanner scan = new Scanner(System.in);
        System.out.println("Bem vindo ao Cadastro de Endereço");
        System.out.print("Digite sua rua: ");
        rua = scan.nextLine();
        System.out.print("Digite seu numero: ");
        numero = scan.nextInt();
        scan.nextLine();
        System.out.print("Digite sua Cidade: ");
        cidade = scan.nextLine();
        System.out.print("Digite seu Estado: ");
        uf = scan.nextLine();



    }
```

```
    public void imprimeEndereco(){
        System.out.println("Seu endereço: " + this.toString());
    }

    @Override
    public String toString() {
        return "Localizacao.Endereco{" +
                "rua='" + rua + '\'' +
                ", numero=" + numero +
                ", cidade='" + cidade + '\'' +
                ", uf='" + uf + '\'' +
                '}';
    }
}
```

Classe Principal com método main:

```
import Dados.Pessoa;
import Dados.Pfisica;
import Dados.Pjuridica;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {

    static List<Pessoa> pessoa = new ArrayList<>();

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
```

```
int opt = 0;
while (opt!=4){
    System.out.println("Digite: ");
    System.out.println("1: Cadastro de Dados Pessoa Física: ");
    System.out.println("2: Cadastro de Dados Pessoa Juridica: ");
    System.out.println("3: Imprime Clientes: ");
    System.out.println("4: Sair");
    opt = scan.nextInt();
    scan.nextLine();
    switch (opt){
        case 1:
          Pfisica pf = new Pfisica();
          pf.cadastra();
          pessoa.add(pf);
          break;
        case 2:
          Pjuridica pj = new Pjuridica();
          pj.cadastra();
          pessoa.add(pj);
          break;
        case 3:
          System.out.println("Clientes cadastrados: ");
          int count = 0;
          for(Pessoa p : pessoa){
            count++;
            System.out.printf("Cliente(%d)%n", count);
            //System.out.println(p); //imprime o toString do objeto
            p.imprime();
            System.out.println();
          }
          break;
        case 4:
```

```
                System.out.println("----------------------");
                System.out.println("-- Saindo do Sistema --");
                System.out.println("-- !!Volte  Sempre!! --");
                System.out.println("----------------------");
                break;
              default:
                System.out.println("Opção inválida: ");
                break;
          }
          }
      scan.close();
   }
}
```