Etec
Escola Técnica Estadual

Técnicas de Orientação a Objeto

A07 - Classes e métodos abstratos e Polimorfismo

Classes Abstratas

- **Classe Abstrata** é aquela que não pode ter objetos instanciados. Ela é apenas um conceito.
- Uma classe é declarada abstrata utilizando a palavra-chave `abstract` na sua declaração, da forma:

```
public abstract class Classe{}
```

- Uma classe abstrata está sempre ligada ao conceito de **herança**, mas nem toda herança implica que a **superclasse** deve ser **abstrata**.

Classes Abstratas

- No nosso exemplo `Pessoa`/`PessoaJuridica`/ `PessoaFisica`, vimos que não existe pessoa que não seja jurídica ou física. Não devemos criar nenhum objeto da classe `Pessoa` nesse caso, ou seja, ela **é uma classe abstrata**.
- Ainda assim a classe `Pessoa` é um conceito importante para o nosso exemplo, e pode ser utilizada como referência (upcasting) de objetos das classes `PessoaJuridica` e `PessoaFisica`.

Classes Abstratas

- Tentar criar objetos de uma classe **abstrata** gera **erro de compilação**.
- Uma classe abstrata tem, **necessariamente** que ser uma **superclasse**.
- Uma classe que estende uma classe normal, **também pode ser abstrata**. Ela não poderá ser instanciada, mas sua **superclasse sim**.
- Classes que podem efetivamente ter objetos instanciados são chamadas de **classes concretas**.

Métodos Abstratos

- **Métodos abstratos** são métodos de classes abstratas que devem, **obrigatoriamente ser implementado**s nas subclasses concretas.
- A definição de um método abstrato se faz com a palavra chave `abstract` antes do tipo de retorno dele.
- Um método abstrato **não contém código**, apenas define uma assinatura do método que todas as subclasses devem seguir para implementá-lo, por isso não tem **chaves {}** mas um **ponto e virgula** no final da sua declaração.

Métodos Abstratos

- Todo **método abstrato** deve ser de uma **classe abstrata**, porém uma classe abstrata **não precisa obrigatoriamente** ter um método abstrato.
- **Construtores** não podem ser abstratos, por que?

Classes e Métodos Abstratos

- No nosso exemplo, `Pessoa`/`PessoaFisica`/ `PessoaJuridica`, o método `getNome()` e `setNome()` pode ser o mesmo para pessoas jurídicas e físicas, então não precisam ser implementados, pois isso implicaria em repetição de código.
- `PessoaFisica` e `PessoaJuridica` ambas têm um identificador, a primeira o `cpf` e a segunda o `cnpj`;
- Se queremos **obrigar** que ambas tenham **identificadores**, podemos criar **métodos abstratos** que definam esse comportamento e implementá-los nas classes filhas.

Classes e Métodos Abstratos

```
1  public abstract class Pessoa{
2    private String nome;
3    public Pessoa() {}
4    public Pessoa(String $nome) {
5      this.nome=$nome;
6    }
7    public String getNome() {
8      return this.nome;
9    }
10   public void setNome(String $nome) {
11     this.nome = $nome;
12   }
13   public abstract String getIdentificador();
14   public abstract void setIdentificador(String $identificador);
15 }
```

Classes e Métodos Abstratos

```
1  public class PessoaJuridica extends Pessoa{
2    private String cnpj;
3    public PessoaJuridica() {}
4    public PessoaJuridica(String $nome, String $cnpj) {
5      this.nome = $nome;
6      this.cnpj = $cnpj;
7    }
8    public String getCnpj() {
9      return this.cnpj;
10   }
11   public void setCnpj(String $cnpj) {
12     this.cnpj = $cnpj;
13   }
14   public String getIdentificador(){
15     return(this.getCnpj());
16   }
17   public void setIdentificador(String $identificador){
18     this.setCnpj($identificador);
19   }
20 }
```

Classes e Métodos Abstratos

```php
public class PessoaFisica extends Pessoa{
  private String cpf;
  public PessoaFisica() {}
  public PessoaFisica(String $nome, String $cpf) {
    this.nome = $nome;
    this.cpf = $cpf;
  }
  public String getIdentificador(){
    return(this.getCpf());
  }
  public void setIdentificador(String $identificador){
    this.setCpf($identificador);
  }
  public String getCpf() {
    return this.cpf;
  }
  public void setCpf(String $cpf) {
    this.cpf = $cpf;
  }
}
```

Classes e Métodos Abstratos

```
1  public class TestaAbstratos{
2    public static void main (String[] args) {
3      PessoaFisica p1 = new PessoaFisica();
4      PessoaJuridica p2 = new PessoaJuridica();
5      p1.setNome("Reinaldo");
6      p1.setIdentificador("123.456.789-00");
7      p2.setNome("39Dev");
8      p2.setIdentificador("123.456.789/0001-00");
9      System.out.println("Nome : " + p1.getNome());
10     System.out.println("CPF  : " + p1.getIdentificador());
11     System.out.println("Nome : " + p2.getNome());
12     System.out.println("CNPJ : " + p2.getIdentificador());
13   }
14 }
```

Exercícios - Classes e Métodos Abstratos

Atenção: Não copie ou cole nenhum exercício. A repetição é intencional para criar fluência na linguagem.

1. Na classe `Cliente` faça os métodos `setIdentificador()` e `getIdentificador()` abstratos.
2. Faça a classe `Cliente` ser abstrata;
3. Na classe `Conta` faça os métodos `setTaxa()` e `getTaxa()` serem abstratos;
4. Faça a classe `Conta` ser abstrata;
5. Compile a classe `AplicacaoFinanceira` e corrija os possíveis erros.

Polimorfismo

- **Polimorfismo** é a capacidade de um objeto poder ser **referenciado e manipulado** como se fosse da **superclasse** que estende ou **interfaces** que implementa (*);
- Essa definição porém isso não quer dizer que o objeto poderá se modificar ao longo do tempo.
- Para existir, o polimorfismo depende do conceito de **herança**; sem esta não há como se utilizar do polimorfismo na programação em Java (exceto por Interface*).
- Permite que se programe de maneira mais **genérica** e menos **específica**.

* veremos a frente

Polimorfismo

- Podemos trabalhar a parte em comum de várias classes que **compartilham a mesma superclasse** da mesma maneira.
- Uma das grandes vantagens do polimorfismo é eliminar conjuntos de if's e else's encadeados que checam o tipo de um objeto antes de executar um método.
- É um recurso muito útil dentro da programação orientada a objetos, do qual muitos **padrões de projetos** (*design patterns*) dependem.
- Polimorfismo depende de vários conceitos:
  - herança/interface;
  - sobrescrita;
  - casting de referências.

Polimorfismo

- As **classes e métodos abstratos** são uma ferramenta para **assegurar o polimorfismo** (embora não sejam uma obrigatoriedade) , pois garantem que todas as classes que compartilham uma superclasse abstrata possam ser tratadas genericamente, utilizando os **métodos abstratos** definidos na superclasse.
- O casting de referência permite que um objeto tenha como referência um superclasse e ainda assim possa ter seus atributos e métodos específicos acessados.
- É possível criar vetores de uma superclasse e em cada posição do vetor armazenar uma instância de uma subclasse.

Polimorfismo

- Podemos observar o polimorfismo implementado na classe `Sistema`

```
1  public class TestaPolimorfismo{
2    public static void main (String[] args) {
3      Pessoa p1 = new PessoaFisica("Marcia","123.456.678-00");
4      Pessoa p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
5
6      System.out.println(p1.getNome() + " - " + p1.getIdentificador());
7      System.out.println(p2.getNome() + " - " + p2.getIdentificador());
8
9      }
10 }
```

Polimorfismo

- Polimorfismo com vetores:

```
1  public class TestaVetorPolimorfismo
2  {
3    public static void main (String[] args) {
4      Pessoa p[] = new Pessoa[4];
5      PessoaFisica p1 = new PessoaFisica("Marcia","123.456.678-00");
6      PessoaJuridica p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
7      Pessoa p3 = new PessoaFisica("Reinaldo","987.654.321-00");
8      Pessoa p4 = new PessoaJuridica("Cast","111.111.111/0001-00");
9
10     p[0]=p1;
11     p[1]=p2;
12     p[2]=p3;
13     p[3]=p4;
14
15     for(int i=0;i<4;i++){
16       System.out.println(p[i].getNome() + " - " + p[i].getIdentificador());
17     }
18   }
19 }
```

Exercícios - Polimorfismo

Exercício 1: A empresa VED93 precisa de um controle de pagamento de funcionários.

- Dos funcionários a empresa precisa dos dados:
  - nome
  - cpf
  - salário base
  - senha
- Há 4 tipos de funcionários: Diretor, gerentes, programadores e secretárias.

Exercícios - Polimorfismo

- As bonificações são pagas à parte e são calculadas da forma:
  - **gerentes** - tem uma bonificação que depende do número de funcionários que gerenciam. A cada 5 funcionários que gerenciam, aumenta 500 reais.;
  - **programadores** - a bonificação depende dos anos de experiência. A cada 2 anos de experiência o salário aumenta 10%.
  - **diretor** tem uma bonificação fixa;
  - **secretária** não tem bonificação :(

Exercícios - Polimorfismo

- Faça uma aplicação ControlePagamento que:
  - Entre com os dados cada tipo de funcionário;
  - Mostre depois do cadastro o nome, salário e a bonificação de cada funcionário;
  - Mostre a soma do montante das bonificações.

Exercícios - Polimorfismo

Exercício 2: Observe esta classe para leitura de teclado:

```
1  import java.io.Console;
2  public class KeyboardReader {
3    public static void main(String[] args) {
4      Console c = null;
5      String nome = null;
6      try{
7        c = System.console();
8        if (c!=null){
9          nome = c.readLine("Digite seu nome: ");
10         System.out.println("O nome digitado foi: " + nome);
11       }
12     }catch (Exception e) {
13         e.printStackTrace();
14     }
15   }
16 }
```

Observação: Esta classe só funciona no terminal

Exercícios - Polimorfismo

Exercício 2: Observe esta classe para leitura de teclado:

```
1  import java.io.Console;
2  public class KeyboardReader {
3    public static void main(String[] args) {
4      Console c = null;
5      String nome = null;
6      try{
7        c = System.console();
8        if (c!=null){
9          nome = c.readLine("Digite seu nome: ");
10         System.out.println("O nome digitado foi: " + nome);
11       }
12     }catch (Exception e) {
13         e.printStackTrace();
14     }
15   }
16 }
```

Importação da classe Console

Lançamento de Exceção

Lê o que foi digitado no teclado e retorna uma string

Exercícios - Polimorfismo

- Adapte o código do leitor de teclado para uma classe que lê do teclado e cria leitores para `int`, `String` e `double`.
- Crie um objeto dessa classe no exercício de polimorfismo (`Pessoa`/`PessoaFisica`/`PessoaJuridica`)
- Utilize para converter Strings respectivamente para `int` e `double`:
    - `Integer.parseInt("string")`
    - `Double.parseDouble("string")`

Datas

- Existem algumas bibliotecas em java para manipulação e armazenamento de datas:
  - java.util.Date - é a mais tradicional. Embora seja bastante utilizada tem muitos comandos marcados como deprecated. Ela representa um momento específico no tempo, com precisão de mili-segundos.
  - java.time - Representa datas, horas, instantes e durações;
  - java.sql.Date - Representação de tempo que permite ao JDBC manipular uma data em SQL.

Datas

```
1  import java.util.Date;
2  import java.text.*;
3
4  public class Data {
5    public static void main(String[] args) throws ParseException{
6      //cria x com a data atual
7      Date x = new Date();
8      System.out.println(x);
9
10     SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
11
12     Date data1 = sdf.parse("25/10/2015");
13     Date data2 = sdf.parse("25/08/2015");
14     //getTime() transforma a data em milisegundos
15     long diferenca = data1.getTime() - data2.getTime();
16     //Diferença em dias:
17     System.out.println(diferenca/(24*60*60*1000));
18   }
19 }
```

Exercícios : Polimorfismo

Atenção: Não copie ou cole nenhum exercício. A repetição é intencional para criar fluência na linguagem.

1. Crie a classe ContaInvestimento que estende Conta com os seguintes atributos:
   - dataInicio do tipo Date;
   - taxaRendimento do tipo double;
2. Crie a classe Produto com os seguintes atributos e métodos:
   - id do tipo long;
   - totalDeMeses do tipo short;
   - mesesDecorridos do tipo short;
   - montante do tipo double;

Exercícios : Polimorfismo

3. Na classe Produto crie o método pagaParcela(short numeroDeMeses) - diminui o valor de mesesDecorridos em numeroDeMeses;
4. Ainda na classe Produto crie o método calculaParcela, que retorna o valor do montante dividido pelo totalDeMeses.
5. Crie a classe Consorcio que estende Produto com os seguintes atributos e métodos:
   - “sorteado" do tipo boolean, inicializado como false;
   - isSorteado() - retorna o valor do atributo sorteado (é como um get)
   - foiSorteado() -altera o valor do atributo “sorteado" para true;
6. Crie a classe Financiamento que estende Produto com os seguintes atributos e métodos:
   - “taxaDeJuro" do tipo double;

Exercícios : Polimorfismo

7. Crie construtores para Produto, Consorcio e Financiamento.
8. Na classe `AplicacaoFinanceira`:
    1. Crie um vetor da classe `Produto` com 4 posições;
    2. Atribua a ele 2 objetos da classe `Consorcio` e 2 objetos da classe `Finaciamento`;
    3. Em um laço mostre o montante e o total de meses dos produtos;

Exercícios : Polimorfismo

8. Na classe `AplicacaoFinanceira` (continuação):
    4. Crie um vetor da classe `Conta` com 5 posições;
    5. Atribua a ele 2 objetos da classe `ContaCorrente`, 2 objetos da classe `ContaPoupanca` e 1 objeto `ContaInvestimento (dataInicial` = hoje`)`;
    6. Em um laço mostre o número e saldo de todas as contas;
    7. Crie um vetor da classe `Cliente` com 4 posições;
    8. Atribua a ele 2 objetos da classe `ClientePessoaFisica` e 2 objetos da classe `ClientePessoaJuridica`;
    9. Em um laço mostre o nome e identificador de todas os clientes;