Programação Orientada a Objetos

Prof. Dr. Josenalde Barbosa de Oliveira

josenalde.oliveira@ufrn.br
https://github.com/josenalde/apds

Interfaces

- Relembrando:
  - Classes abstratas não possuem objetos, apenas concentram atributos e/ou métodos comuns às classes derivadas
  - Estas classes derivadas (subclasses) são classes CONCRETAS, ou seja, possuem objetos
  - As classes abstratas podem possuir métodos abstratos, que precisam ser sobrescritos (implementados) obrigatoriamente nas subclasses; mas também pode ter método concreto, como os setters e getters
  - Assim, a classe abstrata não impõe apenas obrigações (métodos abstratos) às subclasses, mas também oferece herança de métodos concretos que são herdados e são comuns às subclasses
  - **Mas se quisermos apenas definir assinaturas de métodos abstratos nesta classe, forçando a implementação nas subclasses, formando uma coleção de obrigações, compromissos ou operações necessárias (CONTRATO)?**
  - Este é o conceito de INTERFACE
  - Uma classe não estende a interface, ela IMPLEMENTA (*implements*) a interface
  - Uma INTERFACE pode ter atributos constantes (final) herdados pelas classes que a implementa

Interfaces

- Ideia:
  - Imagine uma empresa com vários processos a serem modelados em classes
  - Exemplo: contrato com fornecedores, pagamentos de impostos, contratação de funcionários, operações de vendas, etc.
  - Classes neste domínio teriam atributos e métodos bem distintos, provavelmente com hierarquias de classes diferentes
  - O que haveria de comum entre todas essas classes? (Ex: auditoria!)
  - Poderíamos ter uma interface com um método auditar que precisaria ser implementado em todas as classes que implementassem esta interface. Em Java é a única forma de atribuir obrigações comuns a hierarquias de classes distintas, pois não possui herança múltipla (uma subclasse não pode ter mais de uma superclasse)

Interfaces

- Ampliar o problema dos profissionais:
  - Vamos modelar um **balancete** no domínio do sistema

O balancete é um **demonstrativo financeiro** de caráter não obrigatório feito pelas empresas, sendo muito relevante para que se previnam possíveis erros tanto de crédito quanto de débito na contabilidade de um negócio. Entretanto, esse documento é de uso interno, sendo muitas vezes usado estrategicamente pela companhia em questão, e normalmente contém todas as contas e saldos em um determinado período.

  - O salário bruto pode ser um item declarado neste balancete
  - Embora seja um atributo da classe abstrata Profissional, não faz sentido que ela seja uma interface, pois descaracteriza-a como entidade do domínio
  - Já o custo associado aos salários dos profissionais são um dos itens do balancete
  - Agora imagine que há outro item, de natureza distinta, os imóveis da empresa com atributos (descricao:String, dataInicioContrato: Date, valorAluguel:float)

Diagrama de classes

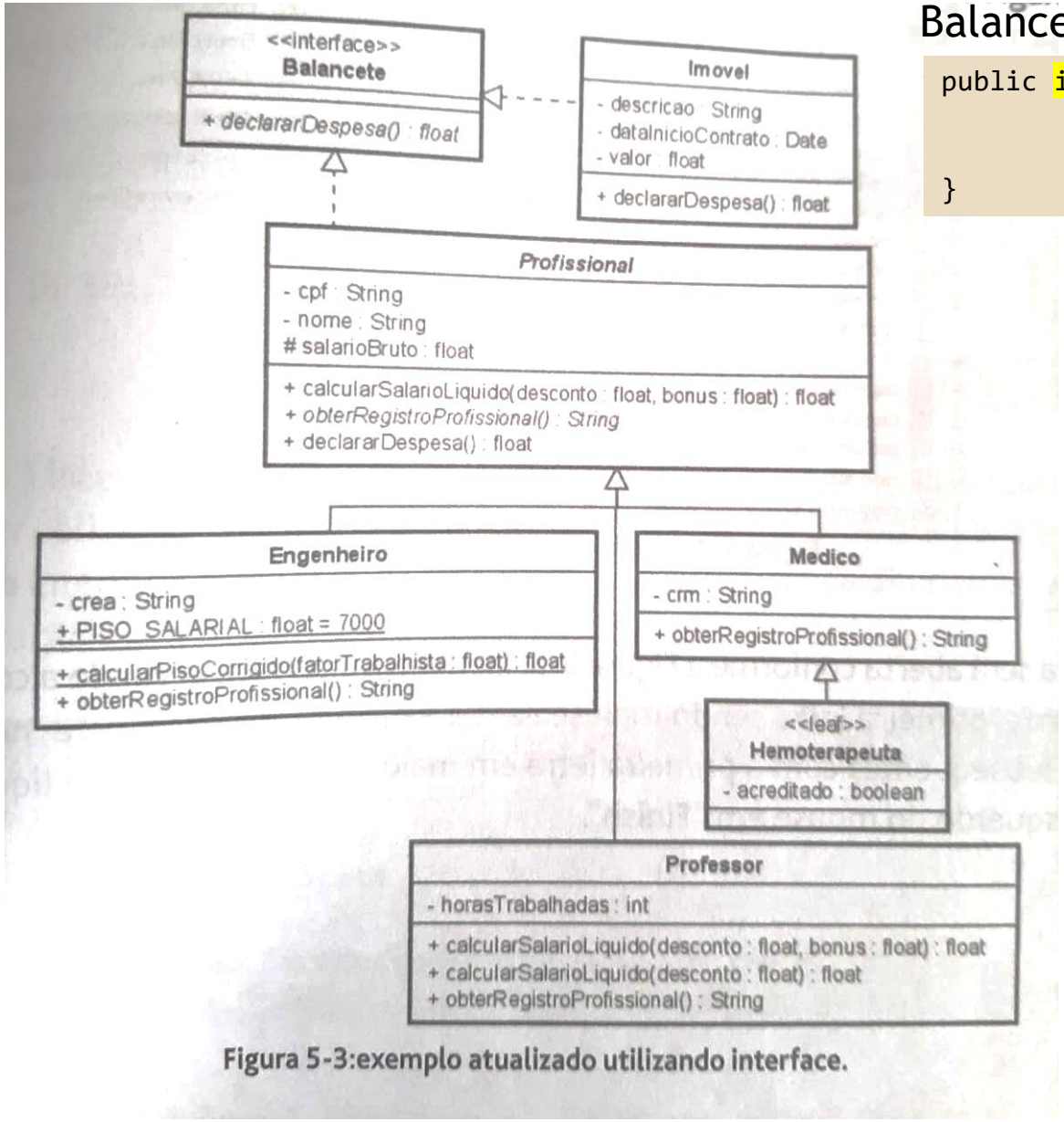


Figura 5-3:exemplo atualizado utilizando interface.

Balancete.java

```
public interface Balancete {

        public float declararDespesa();

}
```

Palavra abstract OPCIONAL

Imovel.java

```
import java.util.Date;

public class Imovel implements Balancete{

    private String descricao;

    private Date dataInicioContrato;

    private float valorAluguel;

    //setters and getters

    @Override

    public float declararDespesa() {

        return valorAluguel;

    }

}
```

Profissional.java

```
public abstract class Profissional implements Balancete{

    //atributos, construtores...;

    //setters and getters

    @Override

    public float declararDespesa() {

        return salarioBruto;

    }

}
```

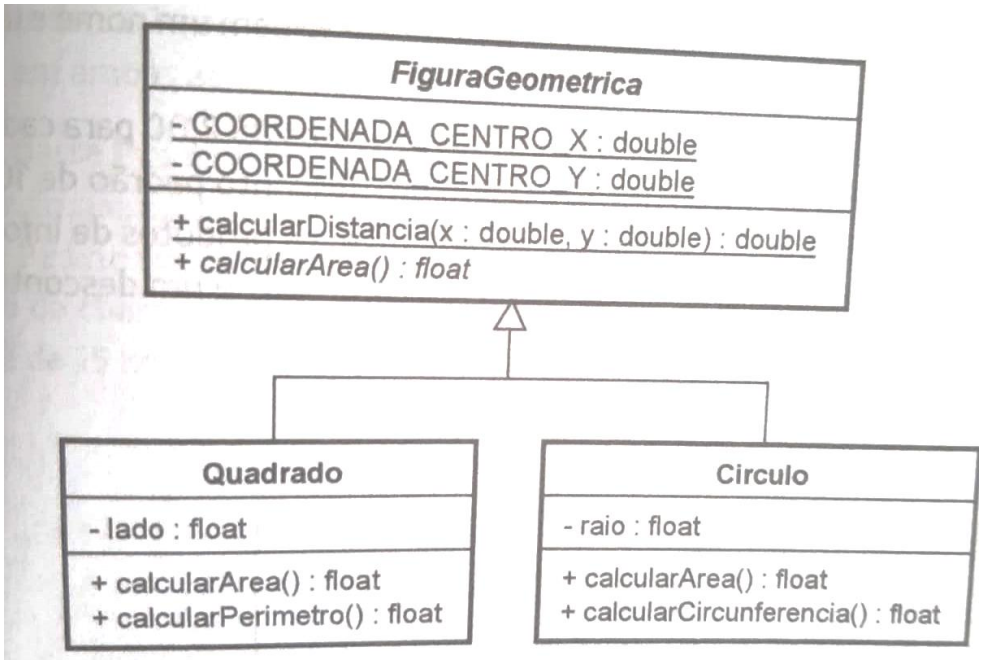
Principal.java

```
public class Principal() {

   public static void main(String[] args) {

      Profissional p3 = new Engenheiro("111.111.111-11", "José",100);
      ((Engenheiro) p3).setCrea("1234");
      Imovel i1 = new Imovel();
      LocalDate dataInicioContrato = new LocalDate(2024,07,09);
      i1.setDataInicioContrato(dataInicioContrato);
      i1.setDescricao("Sede da empresa");
      i1.setValorAluguel(1000.50f);
      imprimirDespesa(i1);
      imprimirDespesa(p3);

   }

   public static void imprimirDespesa(Balancete bal) {

      System.out.println("Despesa: " + bal.declararDespesa());

   }

}
```

- Qualquer objeto de qualquer classe que implemente a interface Balancete pode ser passado como parâmetro para imprimirDespesa

Exercício resolvido

1. Seja o diagrama de classes abaixo, criar um projeto que o implemente:



- Uma superclasse abstrata chamada FiguraGeometrica e as classes Quadro e Circulo
- Os atributos públicos, estáticos e finais COORDENADA_CENTRO_X e COORDENADA_CENTRO_Y dentro da classe FiguraGeometrica, atribuindo o valor 2 para ambos atributos
- Um método abstrato para cálculo de área, sobescrito nas subclasses.
- Um método estático calcularDistancia que recebe uma posição e retorna a dist. Euclidiana para o centro da figura
- Os atributos das subclasses devem ser iniciados com construtores com parâmetros
- Escrever classe Principal, contendo o método main que instancie um objeto Quadrado com lado 3cm e um objeto Circulo de raio 1cm. Exiba o perímetro e área de cada figura. Invoque o método calcularDistancia passando os valores 4 e 8, exibindo o resultado

Exercício resolvido

2. Construa o diagrama de classes UML que modele as seguintes ações:

- Uma interface Pontuacao que possua um método calcularPontos a ser implementado pelas subclasses Clube e CartaoProva
- Uma superclasse abstrata Clube que possua os atributos nome (que guarda o nome do clube). A classe deve possuir um construtor que receba o valor para inicialização do atributo, além dos getters e setters
- Uma classe Futebol, que possua os atributos numéricos vitorias, empates, derrotas. A classe deve possuir construtor parametrizado, getters e setters
- Um método calcularPontos, dentro da classe Futebol, que compute os pontos de um clube, que consiste no número de vitórias multiplicado por três mais os números de empates
- Uma classe Vantagem com o atributo milhas (guarda milhas voadas). A classe deve possuir um construtor para metrizado , getters e setters
- Um método calcularPontos dentro da classe Vantagem, que compute os pontos de um clube de vantagens, que consiste no número de milhas multiplicado por 1,5
- Uma classe CartaoProva que possua os atributos acertos (que guarda o número de acertos do candidato) e erros
- Um método calcularPontos, dentro da classe CartaoProva, que calcule os pontos de uma prova, que consiste no número de acertos multiplicados por 2 menos o número de erros
- Uma classe Principal, com método main, que cria uma lista de 3 objetos, tipificado pela interface Pontuacao. Instancie um objeto de cada classe, passando os parâmetros adequados ao construtor, com valores a sua escolha. A seguir, percorra a lista, enviando uma mensagem ao objeto, pedindo que calcule a qtd. de pontos, exibindo o valor recebido. Use ArrayList