POO

Classes Abstratas

Profa.: Márcia Sampaio Lima

EST - UEA

Classes Abstratas

- Conceito:
  - Define um modelo (*template*) para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade) que é compartilhada por um grupo de classes derivadas.
  - Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

Classes Abstratas

- Conceito:
  - Classes abstratas são definidas para representar entidades e conceitos abstratos.
  - São sempre superclasses que não possuem instâncias → não é possível instanciar objetos.

Classes Abstratas

- Conceito:
  - Uma classe abstrata possui métodos abstratos.
  - **Método abstrato** é o método que não possui implementação.
  - São implementados nas suas classes derivadas concretas com o objetivo de definir o comportamento específico.
  - O método abstrato define apenas a assinatura do método e, portanto, não contém código.

Classes Abstratas

- Na classe abstrata, o método abstrato é definido com palavra reservada **abstract** em sua assinatura.
- A implementação do mesmo é feita na classe filha, através de **sobrescrita de método**.

```
abstract public String som();
```

Classes Abstratas

- As classes concretas implementam todos os seus métodos e permitem a criação direta de instâncias (objetos).
- Estão “prontas” para serem usadas para gerar seus objetos.
- Uma classe concreta não possui métodos abstratos e quando utilizadas neste contexto, são classes derivadas de uma classe abstrata.

Classes Abstratas

- Importância das classes abstratas:
  - Principalmente em tempo de projeto do sistema, podemos criar um nível a mais de abstração e tornar o projeto mais reutilizável.

- Classe abstrata é declarada como tal, através da palavra reservada **abstract e contém** pelo menos um método abstrato.

```
package animais;

public abstract class Bicho {
   String nome;

   public Bicho(){
       this.nome = "";
   }

   public Bicho(String pNome){
       this.nome = pNome;
   }
   abstract public String som();
}
```

Classes Abstratas

- Se eu quiser instanciar bicho?

```
Bicho b = new Bicho();
```

- Compilador gera um erro:

  Bicho.java: Bicho is abstract; cannot be instantiated

Cachorro.java

```
package animais;

public class Cachorro extends Bicho {

  public Cachorro (String pNome){
     super(pNome);
  }

  public String som(){
     return "Au au!!";
  }
}
```

Gato.java

```
package animais;

public class Gato extends Bicho{
  public Gato (String pNome){
      super(pNome);
  }

  public String som(){
      return "Miau miau!!";
  }
}
```

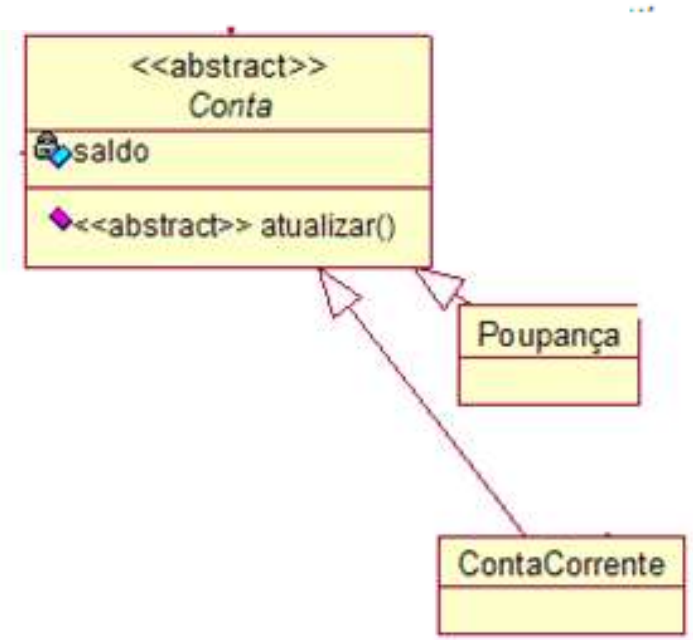
```
package animais;

public class Principal {

  public static void main(String[] args) {
      bicho[] bs = new Bicho[2];
      bs[0] = new Cachorro("Fred");
      bs[1]= new Gato("Shalon");

      for (int i=0; i < bs.length;i++){
            System.out.println(bs[i].som());
      }

  }
}
```

Classes Abstratas

- **Observações:**
  - Métodos abstratos só podem ser declarados em classes abstratas.
  - Em geral, classes abstratas também possuem métodos concretos.
  - Se uma classe só tem métodos abstratos, é melhor declará-la como interface.

Classes Abstratas

- Representação gráfica UML
  - A representação de uma classe abstrata em UML é quase igual à representação de uma classe concreta, a única diferença é o estilo da fonte do nome da classe, que, neste caso, está em itálico.



Classes Abstratas

- Classes abstratas podem possuir atributos e/ou métodos que serão herdados e que ela por si só não faz sentido sozinha.

- Por exemplo, em uma aplicação acadêmica podemos ter classes como Aluno, Professor e Coordenador e todas estas herdam da classe abstrata Pessoa.

- Todos professores, alunos e coordenadores possuem todas as características de uma pessoa como nome e data de nascimento, todavia não faz sentido termos uma instância da classe pessoa na aplicação, mas sim de uma de suas classes especialistas.

Interfaces

- Conceito:
  - As interfaces são padrões definidos através de contratos ou especificações.
  - Um contrato define um determinado conjunto de métodos que serão implementados nas classes que assinarem esse contrato.
  - Uma interface é 100% abstrata:
    - Os seus métodos são definidos como **abstract.**
    - E as variáveis por padrão são sempre constantes (static final).

Interfaces

- Uma interface é definida através da palavra reservada "**interface**".
- Para uma classe implementar uma interface é usada a palavra "**implements**".

- Java não permite herança múltipla:
  - Interfaces ajudam nessa questão;
  - Uma classe pode herdar apenas uma superclasse.
  - Porém, pode implementar inúmeras interfaces.
  - As classes que forem implementar uma interface terão de adicionar todos os métodos da interface ou se transformar em uma classe abstrata.

Interfaces

- Não podem ser instanciadas.

“Interface é um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.”

Interfaces

Exemplo:

```
public interface FiguraGeometrica {
    public String getNomeFigura();
    public int getArea();
    public int getPerimetro();
}
```

- A seguir é possível ver duas classes que implementam a interface FiguraGeometrica, uma chamada Quadrado e outra Triangulo.

```
public class Quadrado implements FiguraGeometrica {
    private int lado;
    public int getLado() {
        return lado;
    }
    public void setLado(int lado) {
        this.lado = lado;
    }
    @Override
    public int getArea() {
        int area = 0;
        area = lado * lado;
        return area;
    }
```

```
    @Override
    public int getPerimetro() {
        int perimetro = 0;

        perimetro = lado * 4;
        return perimetro;
    }
    @Override
    public String getNomeFigura() {
        return "quadrado";
    }
}
```

Triangulo.java

```
public class Triangulo implements FiguraGeometrica {
    private int base;
    private int altura;
    private int ladoA;
    private int ladoB;
    private int ladoC;

  //gera os getters e setters

    @Override
    public String getNomeFigura() {
        return "Triangulo";    }
```

```
    @Override
    public int getArea() {
        int area = 0;
        area = (base * altura) / 2;
        return area;    }

    @Override
    public int getPerimetro() {
        int perimetro = 0;
        perimetro = ladoA + ladoB + ladoC;

        return perimetro;    }
}
```

Interfaces

- Como é possível ver:
  - Ambas as classes seguiram o contrato da interface `FiguraGeometrica`, porém cada uma delas a implementou de maneira diferente.

- Ao contrário da herança que limita uma classe a herdar somente uma classe pai por vez, é possível que uma classe implemente varias interfaces ao mesmo tempo.

Interfaces

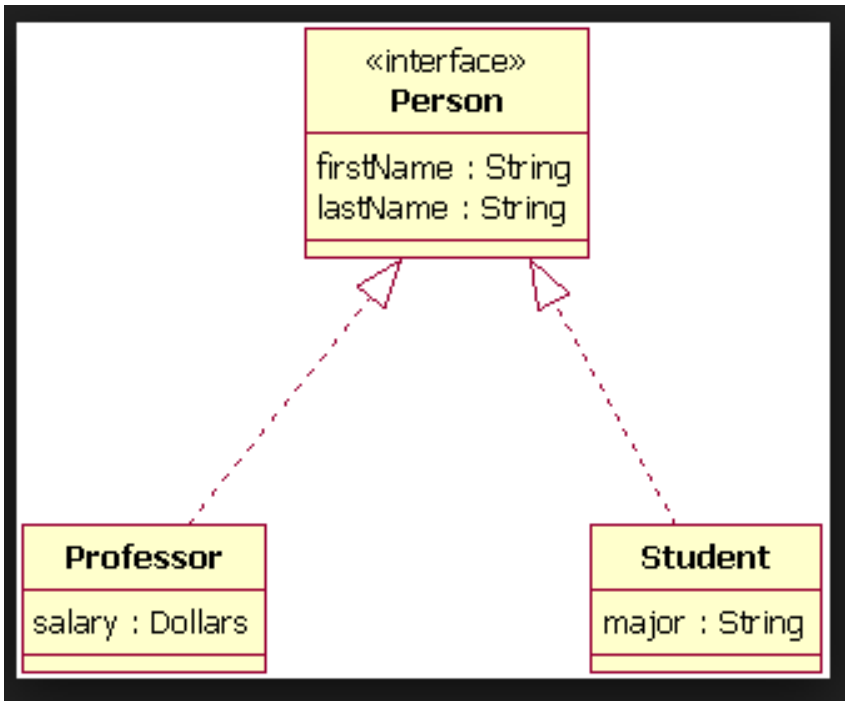
- **Resumindo ....**
  - **Interface é um contrato de regras que uma classes deve seguir em um determinado contexto.**
  - **Como em Java não existe herança múltipla, a interface passa a ser uma alternativa.**
  - **Interfaces contém apenas assinaturas dos métodos sem sua respectiva implementação.**

Interfaces

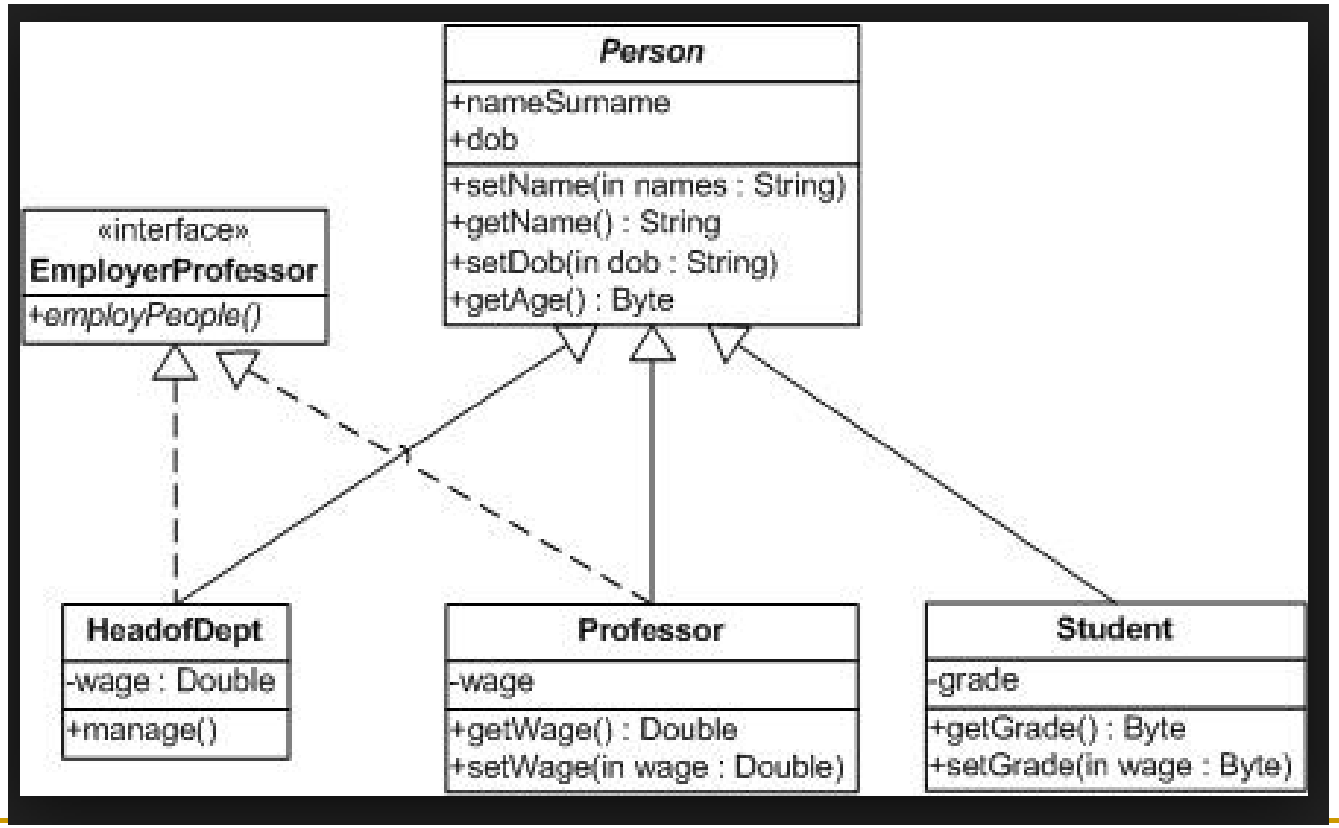
- **Resumindo ....**
  - **Classes Abstratas podem conter a implementação de seus métodos, neste caso, a classe especialista decide se sobrescreve ou não um destes métodos.**
  - **Interfaces não podem definir construtores, Classes Abstratas podem!**

Interfaces

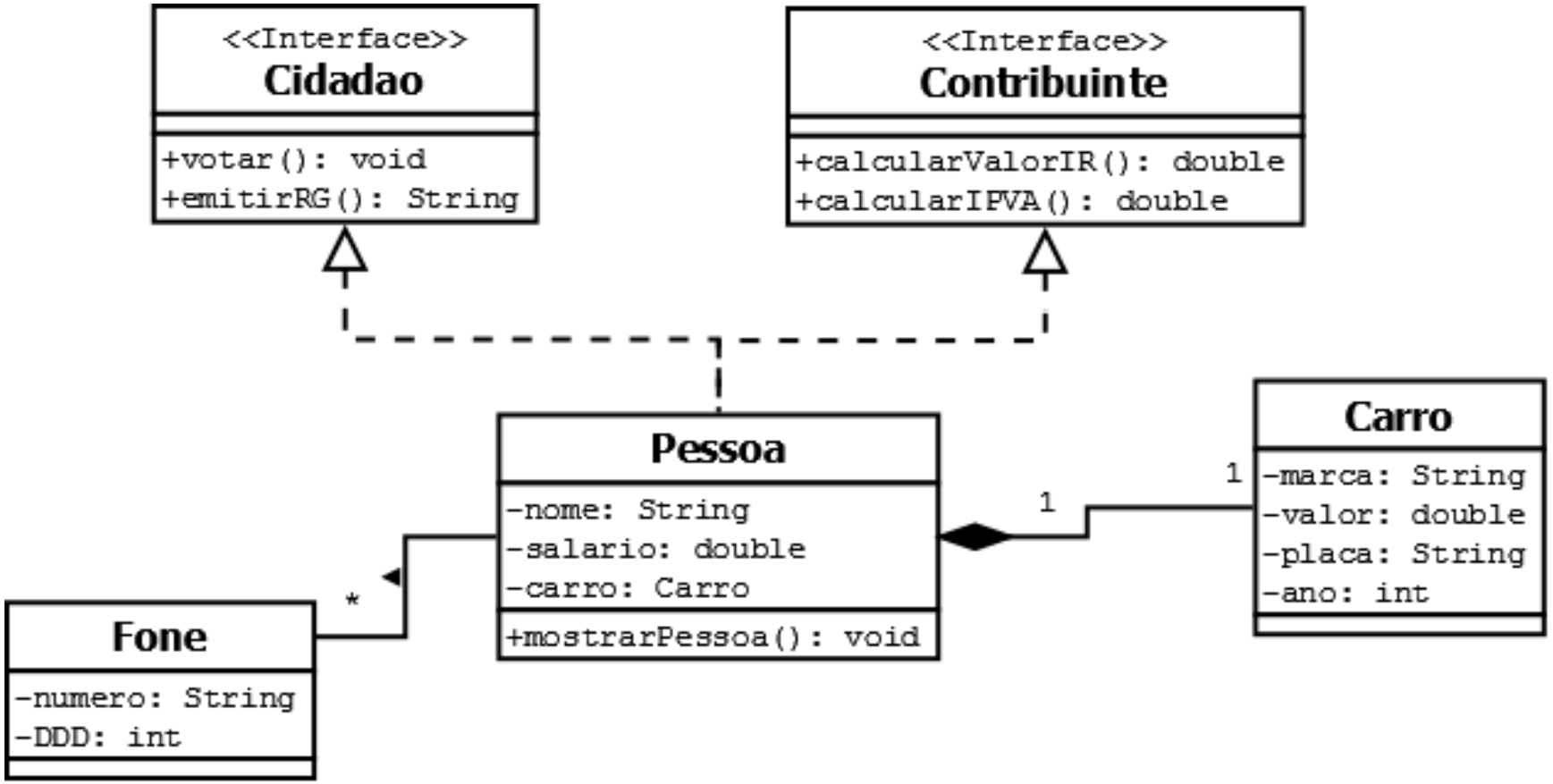
- Representação UML



Exemplo


Person
+nameSurname
+dob
+setName(in names : String)
+getName() : String
+setDob(in dob : String)
+getAge() : Byte
«interface»
EmployerProfessor
+employPeople()
HeadofDept
-wage : Double
+manage()
Professor
-wage
+getWage() : Double
+setWage(in wage : Double)
Student
-grade
+getGrade() : Byte
+setGrade(in wage : Byte)

Implemente


<<Interface>>
Cidadao
+votar(): void
+emitirRG(): String
<<Interface>>
Contribuinte
+calcularValorIR(): double
+calcularIPVA(): double
Pessoa
-nome: String
-salario: double
-carro: Carro
+mostrarPessoa(): void
Carro
-marca: String
-valor: double
-placa: String
-ano: int
Fone
-numero: String
-DDD: int
1
1
*