AULA TEÓRICA 10

Tema 9. Interface

Interface

Uma **interface** é como uma classe que não pode ser instanciada. Seu objectivo é espicificar um conjunto de métodos que uma classe deverá implementar.

Uma classe pode implementar várias interfaces e uma interface pode ser implementada por várias classes (Polimorfismo).

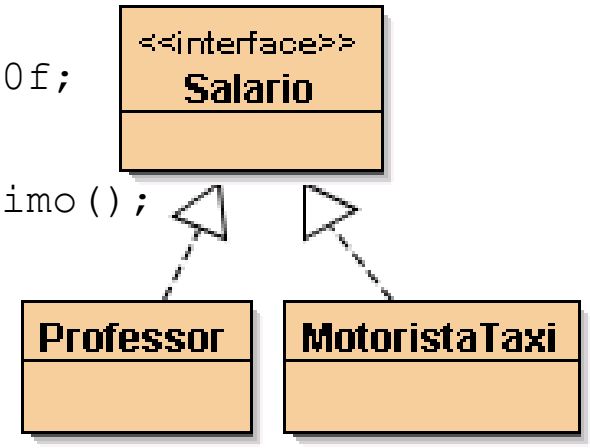
- Métodos em uma interface são sempre `public` e `abstract`
- Dados em uma interface são sempre constantes públicas
  declarados como `static` e `final` (devem ser inicializados na declaração)
- Portanto,
  uma **interface** é, essencialmente, uma **colecção de constantes e métodos abstratos**, não fornecendo nenhuma implementação.
- Interfaces são bastante úteis para especificar o que uma classe deve estar apta a fazer, além disso podem ser úteis para implementar bibliotecas de constantes.
- Mas, em alguns casos, interfaces são demasiado restrictivas. Elas não permitem a definição de variáveis (excepto constantes) ou comandos executáveis. Nestes casos classes abstractas devem ser usadas.

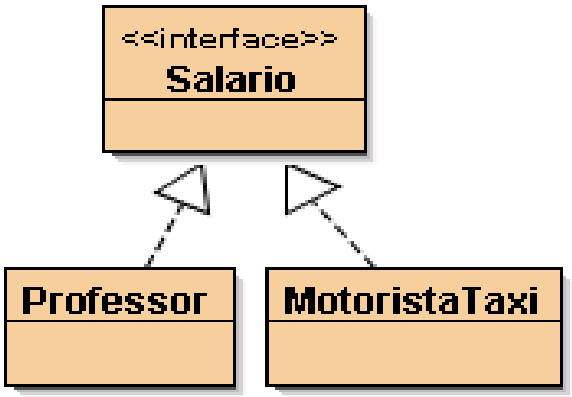
```
public interface Salario {
  public static final float SALARIO_MINIMO = 1412.0f;
    public abstract float getSalarioLiquido();
    public abstract float getQuantidadeSalariosMinimo();
 }
public class Professor implements Salario {
  private String nome;
  private int cargaHoraria;
  private float valorHora;

  public Professor(String nome,int cargaHoraria,float valorHora){
    this.nome = nome;
    this.cargaHoraria = cargaHoraria;
    this.valorHora = valorHora;
  }

  public String getNome()    { return(nome); }
  public int getChsemanal() { return cargaHoraria; }
  public float getValorHora() { return valorHora; }

  public float getSalarioLiquido(){
    float salBruto = valorHora*cargaHoraria*(float)4.5;
    return (salBruto-(salBruto*(float)0.08));
  }
  public float getQuantidadeSalariosMinimo(){
    return ( getSalarioLiquido()/SALARIO_MINIMO);}
}
```






```
public class MotoristaTaxi implements Salario{
  private String nome;
  private int numCarteira;
  private int numeroTaxi;
  private float totKmRodados,valorKm;

  public MotoristaTaxi(String n,int nc,int num,float t,float v){
    nome = n; numCarteira = nc;
    numeroTaxi = num;
    totKmRodados = t;
    valorKm = v;
  }
  public String getNome(){ return nome; }
  public int getNumCarteira() { return numCarteira; }
  public float getTotKmRodados(){ return totKmRodados;}
  public float getValorKm() { return valorKm; }

  public float getSalarioLiquido(){
    float salBruto = totKmRodados * valorKm;
    return (salBruto - (salBruto*(float)0.05)); }

  public float getQuantidadeSalariosMinimo(){
    return (2 * getSalarioLiquido() / SALARIO_MINIMO); }
}
```

```
public class TesteSalario {
  public static void main(String[] args){
    Salario a = new Professor("Joao Paulo",12,500);
    System.out.println("PROFESSOR:\nSalario Liquido: "+
                       a.getSalarioLiquido()+"\nQde de Sal.minimo:"+
                       a.getQuantidadeSalariosMinimo());

    Salario b = new MotoristaTaxi("Jose Bata",112233,32,300,50);
    System.out.println("MOTORISTA\nSalario Liquido: "+
                       b.getSalarioLiquido()+"\nQde de Sal.minimo:"+
                       b.getQuantidadeSalariosMinimo());
  }
}
```

Usando Interfaces se pode trabalhar com polimorfismo

Uma referência do tipo da *Interface* pode apontar para qualquer objecto que implementa aquela *Interface*. Entretanto, **nunca** se pode criar um objetco da *interface*!

Criando uma referência da interface, é possível invocar os métodos definidos na *interface*.

Considerando o exemplo anterior

```
Salario s;
```

O objecto ao qual **`s`** se refere não tem um tipo **`Salario`** (nenhum objecto tem um tipo **`Salario`**). O tipo do objecto é uma classe que implementa a *interface* **`Salario`**, como **`Professor`** ou **`MotoristaTaxi`.**

```
public class TesteInterfacePolimorfismo
{ public static void main(String args[]) {
   Salario ref1, ref2;
   ref1 = new Professor("Jose",20,(float)20.5);
   ref2 = new MotoristaTaxi("Luis",123456,8745,25f,0.5f);
  System.out.println("Motorista "+((MotoristaTaxi)ref2).getNome()+
                  " andou "+((MotoristaTaxi)ref2).getTotKmRodados()+
                  " km no mês de julho.");
  System.out.println("Salario líquido do motorista = "+
                       ref2.getSalarioLiquido());
  System.out.println("O professor "+((Professor)ref1).getNome()+
     " tem "+((Professor)ref1).getChsemanal()+" h/a por semana.");
  System.out.println("Salario líquido do professor = " +
                       ref1.getSalarioLiquido());
   }
  }
```

A referência é **polimórfica**. A linha de código pode executar diferentes métodos em diferentes momentos se o objecto para o qual a referência aponta for diferente. Note que referências polimórficas podem ser resolvidas em tempo de execução. Isto é chamado **ligação dinâmica** (*dynamic binding).* Como saber se **`ref1`** é um objecto **`Professor`** ou **`MotoristaTaxi`**?

Resumindo

- Uma interface define um conjunto de métodos que outras classes devem implementar, mas não define como esses métodos devem ser implementados
- Uma classe que implementa uma interface deve implementar TODOS os métodos listados
- Em Java, classes só podem ter um ancestral direto, ou seja, só podem ser derivadas de uma única classe (embora esta possa ser derivada de outra e assim por diante)
- Existem situações, porém, onde pode ser necessário que uma classe herde **características** de mais de uma “superclasse” simultaneamente
- Como a herança múltipla não é permitida em Java, a linguagem oferece o conceito de interface como opção
- **Uma classe só poder herdar de uma única superclasse, mas pode implementar diversas interfaces**

```
public class Exemplo implements Interf1, Interf2,
  Interf3
```

Referência bibliográfica:

António José Mendes; Maria José Marcelino.
"Fundamentos de programação em Java 2". FCA. 2002.

Elliot Koffman; Ursula Wolz.
"Problem Solving with Java". 1999.

F. Mário Martins;
"Programação Orientada aos objectos em Java 2", FCA, 2000,

John Lewis, William Loftus;
"Java Software Solutions: foundation of program design", **2nd edition, Addision-Wesley**

John R. Hubbard.
"Theory and problems of programming with Java". Schaum's Outline series. McGraw-Hill.

H. Deitel; P. Deitel.
"Java, como programar". 4 edição. 2003. Bookman.

Rui Rossi dos Santos.
"*Programando em Java 2– Teoria e aplicações"*. Axcel Books. 2004