

Palavras-chave

## Objetivos

## Estrutura de Conteúdo

- [illegible]

São usadas como moldes para a criação de outras classes e podem encapsular atributos e comportamentos comuns.

No exemplo da concessionária temos:

Carro é um tipo de *Veículo*

Caminhão é um tipo de *Veículo*

*Veículo* possui modelo e ano de fabricação

**Perguntas:**

Se algum cliente entrar na concessionária e pedir para comprar um *Veículo*, o vendedor saberá o que deve vender?

Se a concessionária anunciar que está vendendo um *Veículo*, o cliente tem condições de saber exatamente o que está comprando ?

- Isso ocorre porque *Veículo* é uma abstração para Carro e Caminhão nessa concessionária.
- Classes abstratas são definidas em Java com o uso da palavra `abstract` na sua definição.
- Por não representarem objetos concretos, classes abstratas não podem ser criadas com o operador `new`.

Exemplo :

```
public abstract class Veiculo {  
    protected String modelo;  
    protected int anoFabricacao;  
    public Veiculo(String modelo, int anoFabricacao)  
    {  
        this.modelo = modelo;  
        this.anoFabricacao = anoFabricacao;  
    }  
}
```

}

Veiculo v = new Veiculo("gol", 2011); //ERRO de compilação

- Métodos definidos nas classes abstratas, mas não implementados são chamados métodos abstratos.
- As classes concretas derivadas da classe abstrata devem, obrigatoriamente, definir esses métodos e de preferência implementá-los, caso contrário, nada ocorrerá ao chamá-los.
- Com o mecanismo de herança, nem sempre a instância referenciada por uma variável é do tipo dessa variável.

Poligono p = new Retangulo(0, 0, 10, 20);

- Para verificarmos a instância referenciada por uma variável, em tempo de execução, usamos o operador **instanceof**.
- Para evitar que sejam criadas sub-classes a partir de uma classe podemos defini-la com o modificador final.
- Podemos usar o modificador final para evitar que um método seja sobrescrito pelas sub-classes.

## Interface

Interface é um recurso muito utilizado em Java. Assim, boa parte da API do Java é formada por Interfaces.

Uma definição de interface inicia com a palavra interface e contém um conjunto de métodos públicos abstratos e/ou constantes públicas estáticas.

Uma classe indica que vai implementar uma interface através da palavra implements. Dessa forma, a classe tem que definir todos os métodos da interface. Caso a classe não defina os métodos da interface, ocorrerá erro de compilação, indicando que a classe deve ser abstract.

Exemplo :

```
public interface Autenticavel {  
  
    public boolean autenticar(String login, String senha); //Note : Sem corpo.
```

```
}
```

```
public class Cliente implements Autenticavel {  
  
    public boolean autenticar(String login, String senha) {  
  
        // aqui escreve o código que implementa o metodo  
  
        // para o cliente  
  
    }  
}
```

Implementar uma interface é como assinar um contrato com o compilador que declara :  
" *Definirei todos os métodos especificados pela interface.*" Fonte : Deitel & Deitel

- Uma classe pode implementar várias interfaces. É só separar o nomes das interfaces por vírgula.
- É permitido que uma mesma classe use o `extends` e o `implements` ao mesmo tempo. Ou seja, a classe herda de uma classe mãe e implementa uma ou mais interfaces.
- Não se pode criar objetos usando uma interface. No entanto, é possível ter variáveis do tipo da interface referenciando objetos.

Exemplo :

Autenticavel a;

a = new Cliente();

a.autenticar("joao?", "123?"); //Chama o método **autenticar()** do Cliente

A definição de Interfaces deve seguir algumas regras:

1. Interfaces não podem ter construtores;
2. Interfaces só podem ter membros públicos (*private* e *protected* não são permitidos);

3. Interfaces não podem implementar métodos, somente defini-los;
4. Interfaces não podem ter atributos, somente constantes;
5. Uma Interface pode estender outra interface (com `extends`) para criar uma hierarquia de interfaces.
6. Importante: as constantes definidas em uma interface são herdadas pelas classes que implementam essa interface. Entretanto, como elas são públicas, elas podem ser usadas em qualquer lugar.

## Estratégias de Aprendizagem

Para que o aprendizado seja proveitoso, o aluno deve se envolver ativamente na participação da aula, deve realizar as tarefas propostas, realizar testes por conta própria nos programas desenvolvidos e compartilhar sua experiência/conclusão com todos.

Toda tarefa realizada pode ser conferida com o professor, para que haja certeza se está ou não correta.

## Indicação de Leitura Específica

### Aplicação: articulação teoria e prática

1) Considere, conforme especificado abaixo, a [superclasse abstrata Empregado](#) e suas [sub-classes concretas Chefe, Comissionado e Horista](#), além da [main](#) fornecida. Depois, faça o que se pede, sabendo ainda que qualquer empregado recebe um salário semanal acrescido ou não de um extra, dependendo se é chefe ou comissionado. O salário do horista, no entanto, está de acordo com o número de horas trabalhadas e o valor da hora.

#### Classe Abstrata : Empregado

Atributos privados : `primeiroNome` e `ultimoNome` (ambos `String`)

Construtor : Deverá receber o primeiro e último nomes

Métodos concretos:

- 1) getPrimeiroNome()
- 2) getUltimoNome()
- 3) toString() é retorna o primeiro nome seguido do último nome

Método abstrato : getSalario()

Objetivo : retornar o salário semanal (double)

#### Classe concreta : Chefe

Atributo privado : salarioSemanal (double)

Construtor : Deverá receber todos os dados necessários p/a construção de um chefe.

Métodos:

- 1) setSalarioSemanal
- 2) toString() é retorna ?Chefe : ? seguido do primeiro nome, que por sua vez, deverá ser seguido do último nome
- 3) para retornar o valor do salário semanal do chefe.

#### Classe concreta : Comissionado

Atributos privados : salarioSemanal (double), comissao (double) e quantidade (int)

Construtor : Deverá receber todos os dados necessários à construção de um comissionado.

Métodos :

- 1) setSalarioSemanal
- 2) setComissao
- 3) setQuantidade
- 4) toString() é retorna ?Comissionado : ? seguido do primeiro nome, que deverá ser seguido do último nome

5) calcular o salário do comissionado e retorná-lo. Para este cálculo, usa-se o salário semanal, acrescido do resultado de comissão x quantidade.

### Classe concreta : Horista

Atributos privados :

- peso (double) è remuneração paga por hora
- horas (double) è número de horas trabalhadas por semana

Construtor : Deverá receber todos os dados necessários à construção de um horista.

Métodos :

- 1) setPeso
- 2) setHoras
- 3) toString() è retorna ?Horista: ? seguido do primeiro nome, que deverá ser seguido do último nome
- 4) calcular o salário do horista, conforme já especificado.

Monte uma classe teste conforme abaixo. Observe o uso do método printf que permite formatar o número de casas decimais.

```
public class ExemploClasseAbstrataEmpregadoTeste {  
    public static void main(String[] args) {
```

```
        Empregado e;
```

```
        Chefe      cf = new Chefe("João", "Silva",1000.0);
```

```
        Comissionado c = new Comissionado ("Maria", "Silva", 400.0, 3.0, 150);
```

```
        Horista     h = new Horista("Pedro","Silva",15.00, 40);
```

```
e = cf ; // empregado recebe chefe - referência Empregado para um Chefe
```

```
System.out.printf("\n%s recebeu R$%.2f\n", e.toString(), e.getSalario());
```

```
System.out.printf("\n%s recebeu R$%.2f\n", cf.toString(), cf.getSalario());
```

```
e = c;
```

```
System.out.printf("\n%s recebeu R$%.2f\n", e.toString(), e.getSalario());
```

```
System.out.printf("\n%s recebeu R$%.2f\n", c.toString(), c.getSalario());
```

```
e = h;
```

```
System.out.printf("\n%s recebeu R$%.2f\n", e.toString(), e.getSalario());
```

```
System.out.printf("\n%s recebeu R$%.2f\n", h.toString(), h.getSalario());
```

```
}
```

```
}
```

2) Considere o projeto TestaInterface formado pelas classes Quadrado, ConjuntoQuadrados e TestaInterface, além da **interface Ordenacao**. Depois, faça o que se pede nos itens a, b, c, ao longo do código :

```
//Classe Quadrado
```

```
public class Quadrado
```

```
{
```

```
    private double lado;
```

```
    public Quadrado ()          {
```



```
        this.lado = 0;
    }

    public Quadrado (double l)    {
        this.lado = l;
    }

    public double getLado()        {
        return lado;
    }


    public void setLado(double lado)    {
        this.lado = lado;
    }


    public double calcularArea()  {
        return (lado * lado);
    }


    public double calcularPerimetro()    {
        return (4 * lado);
    }


    public String getNome()    {
        return "quadrado";
    }
}
```

```
} // fim classe Quadrado
```

### **//Classe ConjuntoQuadrados**

```
import java.util.Scanner;
```

```
public class ConjuntoQuadrados implements Ordenacao
```

```
{
```

```
private int n;           // No. de quadrados
```

```
private Quadrado v[] ;   // declara vetor de quadrados
```

```
public ConjuntoQuadrados(int quant)
```

```
{
```

```
    n = quant;
```

```
    v = new Quadrado[n];
```

```
}
```

```
public void inserir()
```

```
{
```

```
    Scanner teclado = new Scanner(System.in);
```

```
    double lado;
```

```
    System.out.print("Entrada dos lados dos quadrados : " );
```

```
    //Item a)
```

```
    // Monte o vetor de quadrados com dados obtidos via console.
```

```
    // Capture exceção se necessário.
```

```

    }

    public void listar()
    {
        System.out.println("\nLados dos quadrados : ");
        for (int i = 0; i < v.length; i++)
            System.out.println(v[i].getLado());

    }

    public void ordenar()
    {
        // Item b) Implemente o método ordenar, usando algum
        // método de ordenação já estudado.

    } // fim ordenar

} // fim classe ConjuntoQuadrados


// Classe TestaInterface
public class TestaInterface
{
    public static void main(String[] args)
    {
        /***** Item c) TAREFAS :

```

1. Crie uma instância de ConjuntoQuadrados com 4 quadrados
2. Invoque o método inserir
3. Liste a coleção de quadrados
4. Ordene a coleção de quadrados
5. Liste a coleção novamente

```
*****  
*****/  
  
    }  
  
}
```

#### // Interface

```
public interface Ordenacao {  
  
    public abstract void ordenar();  
  
}
```

#### GABARITO :

1) Veja com seu professor, por favor.

2)

```
import java.util.Scanner;  
  
public class ConjuntoQuadrados implements Ordenacao  
{  
    private int n;           // No. de quadrados  
    private Quadrado v[] ;   // declara vetor de quadrados  
  
    public ConjuntoQuadrados(int quant)  
    {  
        n = quant;  
        v = new Quadrado[n];  
    }  
}
```

```
}
```

```
public void inserir()
```

```
{
```

```
Scanner teclado = new Scanner(System.in);
```

```
double lado;
```

```
System.out.print("Entrada dos lados dos quadrados : " );
```

```
for (int i = 0; i < v.length; i++)
```

```
{
```

```
System.out.print("\nLado ? ");
```

```
lado = teclado.nextDouble();
```

```
v[i] = new Quadrado(lado);
```

```
}
```

```
}
```

```
public void listar()
```

```
{
```

```
System.out.println("\nLados dos quadrados : ");
```

```
for (int i = 0; i < v.length; i++)
```

```
    System.out.println(v[i].getLado());
```

```
}
```

```
public void ordenar()
```

```
{
```

```
    int    menor, i, j ;    // índices
```

```
    Quadrado aux;          // armazena o dado do vetor
```

```
for ( j= 0; j < v.length - 1; j++)
```

```
{
```

```
    menor = j;
```

```
    for (i = j+1; i < v.length ; i++)
```

```
if (v[i].getLado() < v[menor].getLado())
```

```
    menor = i;
```

```
    // realizando a troca entre v[j] e v[menor], usando aux (fora do 2º for )
```

```
    aux    = v[j];
```

```
    v[j]    = v[menor];
```

```
    v[menor] = aux;
```

```
}// fim 1º for
```

```
    } // fim ordenar
```

```
} // fim classe ConjuntoQuadrados
```

```
////////////////////////////////
```

```
public class Quadrado
```

```
{  
    private double lado;
```

```
  
    public Quadrado ()        {  
        this.lado = 0;  
    }
```

```
  
    public Quadrado (double l) {  
        this.lado = l;  
    }
```

```
  
    public double getLado() {  
        return lado;  
    }
```

```
  
    public void setLado(double lado) {  
        this.lado = lado;  
    }
```

```
  
    public double calcularArea() {  
        return (lado * lado);  
    }
```

```
  
    public double calcularPerimetro() {  
        return (4 * lado);  
    }
```

```
  
        public String getNome()    {  
            return "quadrado";  
        }
```

```
} // fim classe Quadrado
```

```
////////////////////////////////
```

```
public interface Ordenacao {
```

```

public abstract void ordenar();
}

////////////////////////////////////

public class TestaInterface {

    public static void main(String[] args) {

        /***** TAREFAS :
        1. Crie uma instância de ConjuntoQuadrados com 4 quadrados
        2. Invoque o método inserir
        3. Liste a coleção de quadrados
        4. Ordene a coleção de quadrados
        5. Liste a coleção novamente
        *****/

        ConjuntoQuadrados c = new ConjuntoQuadrados(4);

        System.out.println("\nInserindo ... ");
        c.inserir();

        System.out.println("\nListando após inserção ... ");
        c.listar();

        System.out.println("\nOrdenando...");
        c.ordenar();

        System.out.println("\nListando após ordenação...");
        c.listar();

    }
}

```