

Linguagem de Programação 2 – IMD0040

Aula 15 – Classes Abstratas e Interfaces

João Carlos Xavier Júnior

jcxavier@imd.ufrn.br

Classes Abstratas

- ❑ As classes abstratas devem conter pelo **menos um método abstrato**, que não tem corpo.
- ❑ É um tipo **especial de classe** que não há como criar **instâncias dela**. É usada apenas para ser **herdada**, funciona como uma super classe.
- ❑ Uma **grande vantagem** é que força a hierarquia para todas as sub-classes.
- ❑ É um **tipo de contrato** que faz com que as sub-classes contemplem as mesmas hierarquias e/ou padrões.

Classes Abstratas

- ❑ Pode-se dizer que as **classes abstratas** servem como “**modelo**” para outras classes que dela herdem.

```
3 abstract class Conta {  
4  
5     private double saldo;  
6  
7     public void setSaldo(double saldo) {  
8         this.saldo = saldo;  
9     }  
10  
11     public double getSaldo() {  
12         return saldo;  
13     }  
14  
15     public abstract void imprimeExtrato();  
16 }
```

Classes Abstratas

❑ Herança em classes abstratas:

```
3 import java.text.SimpleDateFormat;
4 import java.util.Date;
5
6 public class ContaPoupanca extends Conta {
7
8     @Override
9     public void imprimeExtrato() {
10         System.out.println("### Extrato da Conta ###");
11
12         SimpleDateFormat sdf =
13             new SimpleDateFormat("dd/MM/aaaa HH:mm:ss");
14         Date date = new Date();
15
16         System.out.println("Saldo: "+this.getSaldo());
17         System.out.println("Data: "+sdf.format(date));
18     }
19 }
```

Classes Abstratas

❑ Testando classe abstrata:

```
3 public class TestaConta {  
4  
5     public static void main(String[] args) {  
6         // criando objeto cp  
7         Conta cp = new ContaPoupanca();  
8         cp.setSaldo(212.15);  
9         cp.imprimeExtrato();  
10    }  
11 }
```



The screenshot shows an IDE console window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the Java application. The output consists of three lines: a separator line of three hash symbols, the text 'Extrato da Conta', another separator line of three hash symbols, the balance 'Saldo: 212.15', and the date and time 'Data: 26/09/AM 09:59:55'.

```
<terminated> TestaConta [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (26/09/2016 09:59:55)  
### Extrato da Conta ###  
Saldo: 212.15  
Data: 26/09/AM 09:59:55
```

Classes Abstratas

❑ Métodos Abstratos:

- ❖ São obrigatoriamente implementados pelas subclasses;
- ❖ Utilizado como método genérico, que obrigatoriamente será especificado dentro de cada subclasse.

```
public abstract class Animal {  
    public abstract void falar();  
}
```

Métodos abstratos não
são implementados

Classes Abstratas

- ❑ Qual a diferença entre os métodos abaixo?

```
3 public class Animal02 {  
4  
5     public void falar() {  
6         // sem implementação  
7     }  
8 }
```

```
3 public abstract class Animal {  
4  
5     public abstract void falar();  
6  
7 }
```

Classes Abstratas

❏ Observação:

- ❖ Classes abstratas não precisam obrigatoriamente ter métodos abstratos.
- ❖ Métodos abstratos só podem existir em classes abstratas.

Interfaces

- ❑ 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**, ou seja, os seus métodos são **implicitamente abstratos**, e as **variáveis** são **implicitamente** constantes (**static final**).

Interfaces

- ❑ Uma interface é definida através da palavra reservada “**interface**”. Para uma classe implementar uma interface é usada a palavra “**implements**”.
- ❑ Como a linguagem Java **não tem herança múltipla**, as interfaces ajudam nessa questão.
- ❑ Uma classe pode ser herdada apenas uma vez, mas **pode implementar inúmeras interfaces**.
- ❑ As classes que forem implementar uma interface terão de adicionar todos os métodos da interface.

Interfaces

❑ Definindo uma **Interface**:

```
3 public interface Conta {  
4  
5     // métodos do contrato  
6     void depositar(double valor);  
7     void sacar(double valor);  
8     double getSaldo();  
9 }
```

Interfaces

❑ Implementando uma Interface 01:

```
3 public class ContaCorrente implements Conta {
4
5     private double saldo;
6     private double taxaOperacao = 0.45;
7
8     @Override
9     public void depositar(double valor) {
10         this.saldo += valor - taxaOperacao;
11     }
12
13     @Override
14     public void sacar(double valor) {
15         this.saldo -= valor + taxaOperacao;
16     }
17
18     @Override
19     public double getSaldo() {
20         return this.saldo;
21     }
22 }
```

Interfaces

❑ Implementando uma Interface 02:

```
3 public class ContaPoupanca implements Conta {
4
5     private double saldo;
6
7     @Override
8     public void depositar(double valor) {
9         this.saldo += valor;
10    }
11
12    @Override
13    public void sacar(double valor) {
14        this.saldo -= valor;
15    }
16
17    @Override
18    public double getSaldo() {
19        return this.saldo;
20    }
21 }
```

Interfaces

❑ Implementando uma Classe genérica:

```
3 public class GeradorExtratos {
4
5     public void geradorConta(Conta conta) {
6         if (conta instanceof ContaCorrente) {
7             System.out.println("#####");
8             System.out.println("Saldo da Conta Corrente");
9             System.out.println("Saldo: " + conta.getSaldo());
10        }
11        else{
12            System.out.println("#####");
13            System.out.println("Saldo da Conta Poupança");
14            System.out.println("Saldo: " + conta.getSaldo());
15        }
16    }
17 }
```

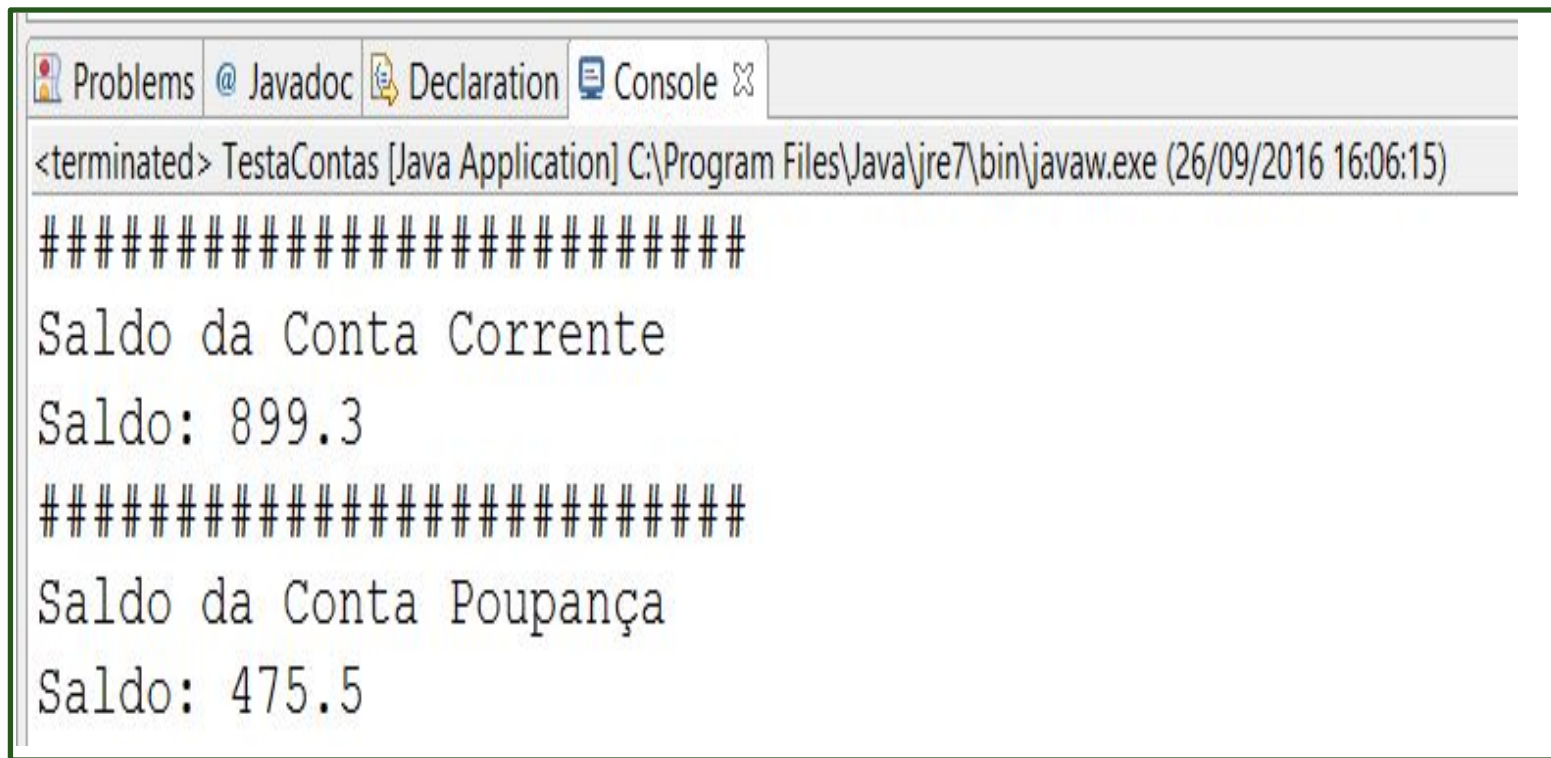
Interfaces

❑ Testando a Interface e suas implementações:

```
3 public class TestaContas {
4
5     public static void main(String[] args) {
6         ContaCorrente cc = new ContaCorrente();
7         cc.depositar(1200.20);
8         cc.sacar(300);
9
10        ContaPoupanca cp = new ContaPoupanca();
11        cp.depositar(500.50);
12        cp.sacar(25);
13
14        GeradorExtratos gerador = new GeradorExtratos();
15        gerador.geradorConta(cc);
16        gerador.geradorConta(cp);
17    }
18 }
```

Interfaces

- ❑ Testando a Interface e suas implementações:



```
Problems @ Javadoc Declaration Console X
<terminated> TestaContas [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (26/09/2016 16:06:15)
#####
Saldo da Conta Corrente
Saldo: 899.3
#####
Saldo da Conta Poupança
Saldo: 475.5
```


Interfaces

❏ Observação:

- ❖ Interfaces podem estender (**extends**) outras interfaces;
- ❖ Classes podem estender outra classe, mas apenas podem implementar interfaces;
- ❖ Uma classe pode implementar uma ou mais interfaces.

Perguntas ...



Métodos Default

- ❑ A partir do Java 8, é possível definir métodos dentro de interfaces, fornecendo uma implementação **default**.
- ❑ Nas versões anteriores, as interfaces se limitavam apenas a **especificar contratos**, agora elas podem também **fornecer comportamento**.
- ❑ A maior motivação para a criação de **métodos default** foi a necessidade de adicionar **novas funcionalidades** às interfaces existentes, **sem quebrar (bugar)** o código que faz uso delas.

Métodos Default

- ❑ Nas versões anteriores do Java, um **novo método** numa interface, obrigava a **alterar todas as classes** que herdavam dessa interface.

```
3 public interface TV {  
4     void ligar();  
5 }  
6  
7 class LG implements TV {  
8     @Override  
9     public void ligar() {  
10         System.out.println("ligou LG");  
11     }  
12 }  
13  
14 class Sony implements TV {  
15     @Override  
16     public void ligar() {  
17         System.out.println("ligou Sony");  
18     }  
19 }
```

Métodos Default

❑ Testando

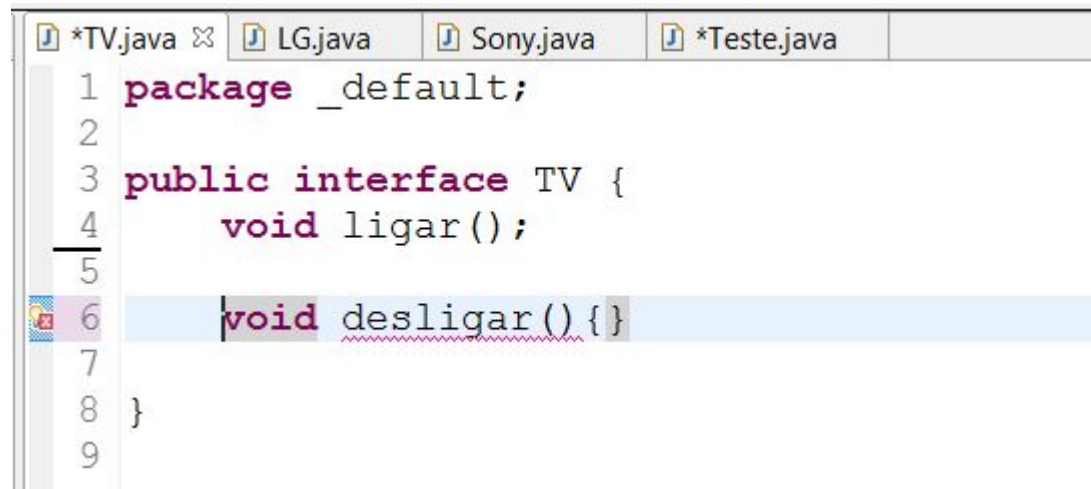
```
3 public class Teste {  
4  
5     public static void main(String[] args) {  
6         LG lg    = new LG();  
7         Sony sn  = new Sony();  
8  
9         lg.ligar();  
10        sn.ligar();  
11    }  
12 }
```

Métodos Default

- ❑ Adicione um método `void desligar() {} :`
- ❑ O que aconteceu?

Métodos Default

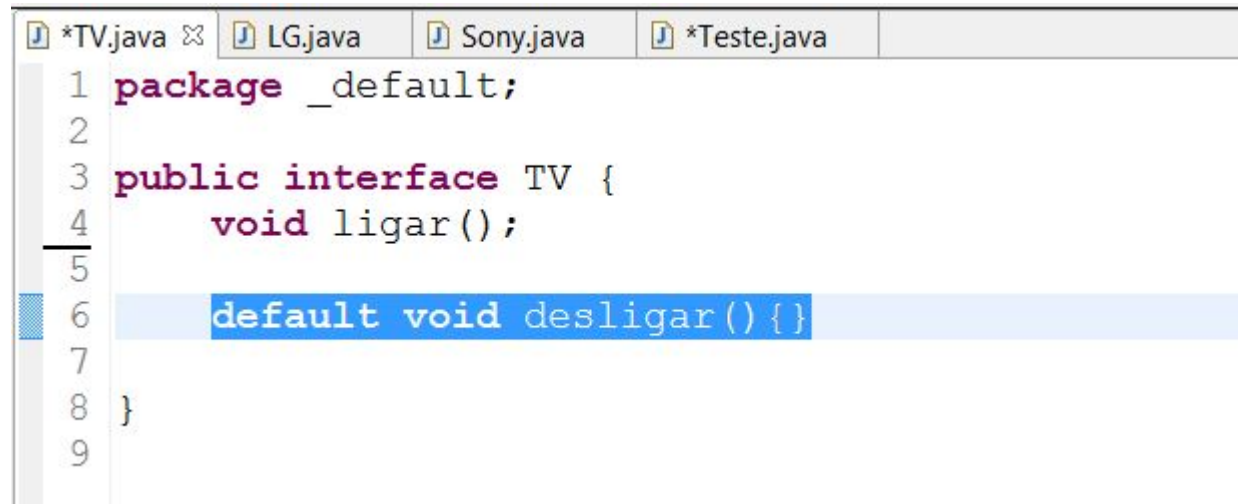
- ❑ Adicione um método `void desligar() {}` :
- ❑ O que aconteceu?



```
*TV.java LG.java Sony.java *Teste.java
1 package _default;
2
3 public interface TV {
4     void ligar();
5
6     void desligar() {}
7
8 }
9
```

Métodos Default

- ❑ Agora mude o modificador de acesso para **default**.



The screenshot shows an IDE with four tabs: *TV.java, LG.java, Sony.java, and *Teste.java. The *TV.java tab is active, displaying the following code:

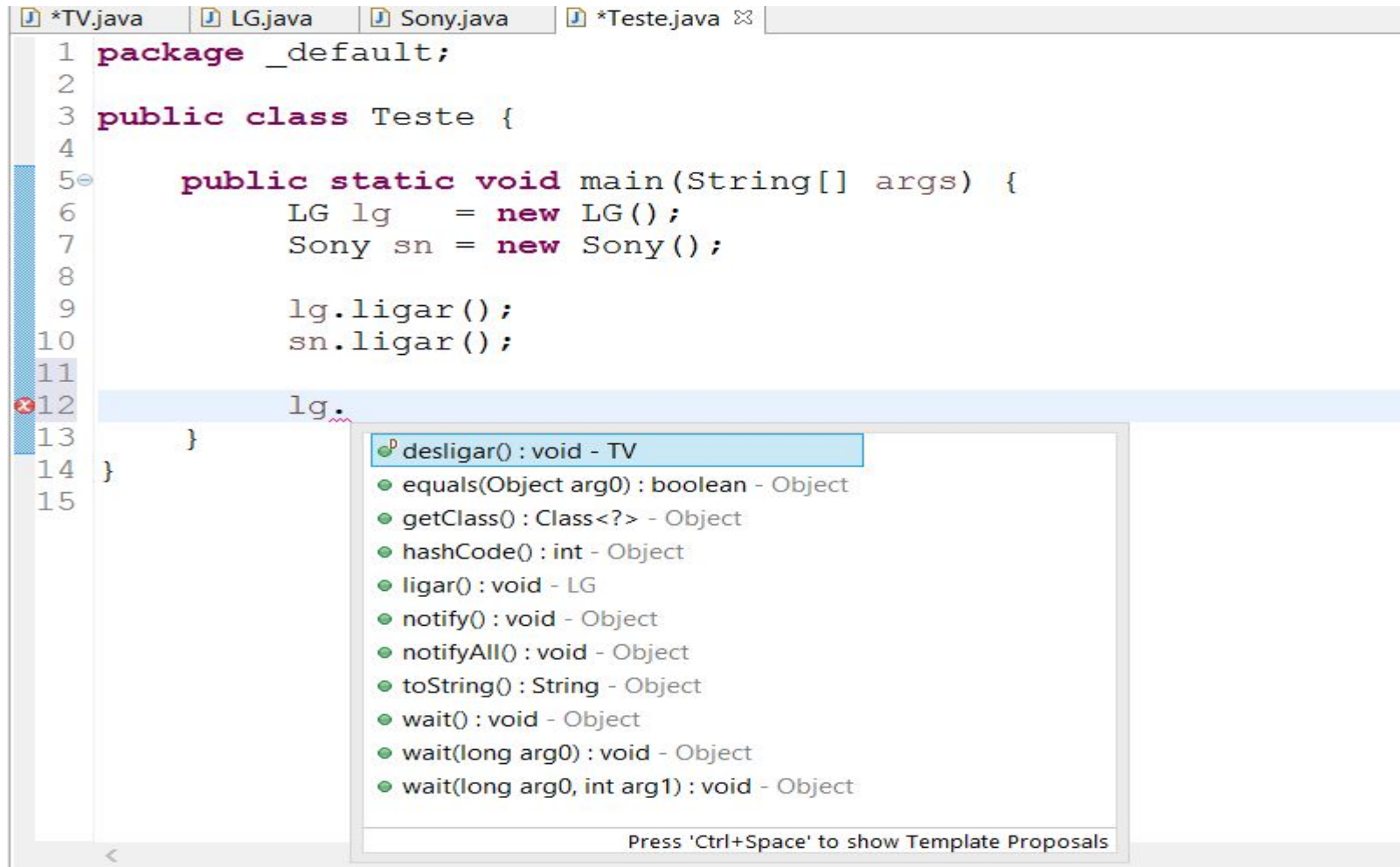
```
1 package _default;
2
3 public interface TV {
4     void ligar();
5
6     default void desligar() {}
7
8 }
9
```

Line 6, containing the default method `default void desligar() {}`, is highlighted in blue.

Métodos Default

- Agora as classes LG e Sony herdam automaticamente o comportamento default de desligar da interface TV e podem ou não sobrescrever esse comportamento.

Métodos Default



```
1 package _default;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         LG lg = new LG();
7         Sony sn = new Sony();
8
9         lg.ligar();
10        sn.ligar();
11
12        lg.
13    }
14 }
15
```

Code completion popup for `lg.`:

- `desligar() : void - TV`
- `equals(Object arg0) : boolean - Object`
- `getClass() : Class<?> - Object`
- `hashCode() : int - Object`
- `ligar() : void - LG`
- `notify() : void - Object`
- `notifyAll() : void - Object`
- `toString() : String - Object`
- `wait() : void - Object`
- `wait(long arg0) : void - Object`
- `wait(long arg0, int arg1) : void - Object`

Press 'Ctrl+Space' to show Template Proposals

Métodos Estáticos

- ❑ Os métodos **static** não dependem de nenhuma variável de instância.
- ❑ Quando invocados executam uma função sem a dependência de um objeto.
- ❑ Podem manipular as variáveis de instância do objeto.

Métodos Estáticos

❑ Exemplo:

```
3 public class Soma {  
4  
5     public static int resultado(int num1, int num2) {  
6         return (num1 + num2);  
7     }  
8 }
```

```
3 public class TestaSoma {  
4  
5     public static void main(String[] args) {  
6  
7         System.out.println(Soma.resultado(10,50));  
8     }  
9 }
```

Classes Abstratas ou Interfaces?

- ❑ A escolha tem dois aspectos que precisam ser considerados:
 - ❖ Conceitual:
 - Classes abstratas não podem ter instâncias;
 - Interfaces determinam contratos.
 - ❖ Prático:
 - Uma classe pode implementar mais de uma interface;
 - Uma classe pode ser abstrata e pode ter atributos.
- ❑ Importante: classes abstratas e interfaces têm como objetivo comum favorecer o uso de polimorfismo.

Perguntas ...

