



Interface Classes Abstratas

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br



Aula 9: Interface

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br

Interface

- Fronteira de comunicação entre dois componentes de software.
- Na prática
 - Uma interface é vista como uma classe abstrata
 - Suas instâncias não podem ser diretamente criadas delas.
 - Essas classes (interfaces) podem conter operações mas não podem conter atributos.
 - Ou seja: possuem métodos (modificador + assinatura)
 - Uma classe pode derivar de uma interface (através da realização de uma relação de associação) e instâncias podem então ser feitas destas relações.

Interface



- Cuidado
 - Não confundir o conceito de interface apresentado com um outro, nesse caso aplicado a design,
 - Em design
 - é relacionado a uma tela (com janelas, botões e outros componente visuais)
 - Interfaces de Programa Aplicativos, do inglês APIs (*Application Program Interfaces*), ou GUIs (Graphical User Interfaces)

Interface

- Pode servir de base para uma classe
- Mas como assim: “servirá de base”?
 - A questão é:
 - “*e se desejássemos limitar o acesso a determinados métodos?*”
 - Por isso a interface funcionará como uma espécie de validador, que permitirá o acesso ou não a determinados métodos que estejam definidos na interface da classe.

Interface



- Exemplo
 - A Aplicação **Funcionario.java** envolve parte de um sistema lojista.
 - Serão dois os atores: Caixa e Gerente.
 - Inicialmente vejamos uma possível solução SEM FAZER o uso de interfaces

Exemplo – Sem Interface



```
public class Funcionario {

    //declaração de constates
    public static final int FUNCIONARIO_CAIXA = 0;
    public static final int FUNCIONARIO_GERENTE = 1;

    //declaração de atributos
    private String nome;
    private String matricula;
    private int rg;
    private double salario;
    private int cargo;

    //método construtor
    public Funcionario (String n, int rg, int cg,
double sal){
        this.nome = n;
        this.rg = rg;
        this.cargo = cg;
        this.salario = sal;
    }

    //método construtor
    public Funcionario (String n, String mat, int cg,
double sal){
        this.nome = n;
        this.matricula = mat;
        this.cargo = cg;
        this.salario = sal;
    }

    //retorna o nome do funcionário
    public String getNome(){return this.nome;}

    //retorna a matrícula do funcionário
    public String getMatricula(){return
this.matricula;}

    //retorna o RG do funcionário
    public int getRg(){return this.rg;}

    //retorna o salário do funcionário
    public double getSalario(){return this.salario;}

    //exibe informações do Funcionario
    public void getInformacaoFuncao(){
        if (this.cargo == FUNCIONARIO_CAIXA)
            System.out.println("O caixa " +
this.getNome() + ", RG " + this.getRg() + ", recebe
R$" + this.getSalario());

        if (this.cargo == FUNCIONARIO_GERENTE)
            System.out.println("O gerente " +
this.getNome()+ ", matricula " + this.getMatricula() +
", recebe R$" + this.getSalario());
    }
}
```

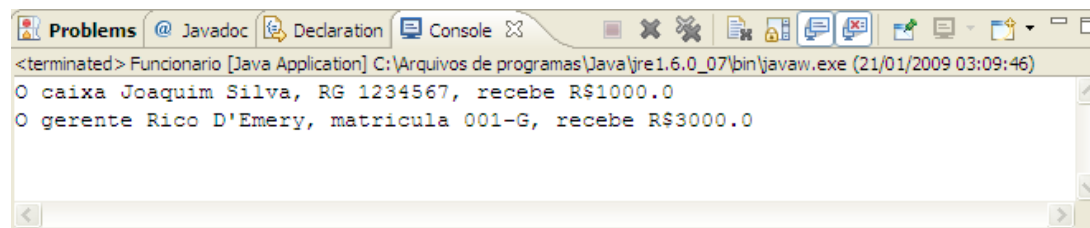
Exemplo – Sem Interface



```
//método principal (aplicação)
public static void main (String [] args){
    //cria os objetos
    Funcionario a = new Funcionario ("Joaquim Silva", 1234567, 0, 1000.00);
    Funcionario b = new Funcionario ("Rico D'Emery", "001-G", 1, 3000.00);

    //chamada de método para exibir as informações dos funcionários
    a.getInformacaoFuncao();
    b.getInformacaoFuncao();

    //exemplo de instrução que utiliza um método que não deveria ser permitido
    //System.out.println("RG do gerente: " + b.getRg());
}
}
```



Exemplo – Sem Interface

- Analisando:
 - Analisando o código acima podemos notar um objeto `Funcionario` pode se relacionar com outras classes de outras maneiras:
 - “*retornando seu nome*”, “sua matrícula”, “seu RG”, “o valor do seu salário”, e “os dados de sua função” (a partir dos métodos `getNome()`, `getMatricula()`, `getRg()` e `getSalario()` e `getInformacaoFuncao()`).
 - Aos outros objetos, pouco importa como o objeto calcula estes valores. Através do modificador de acesso `public` é informado o tipo de acesso a informação (interface do objeto).

Exemplo – Sem Interface

- Analisando:
 - Os objetos `a` e `b` podem acessar qualquer método da classe `Funcionário`,
 - e assim o objeto `b`, por exemplo, não deveria acessar o método `getRg()`,
 - assumindo o fato que utiliza o método construtor `Funcionario (String n, String mat, int cg, double sal)` que não faz uso do atributo `rg`, e se o objeto `b` chamar `getRg()` exibirá o valor do atributo, porém com o valor default (zero) por não ter sido inicializado.

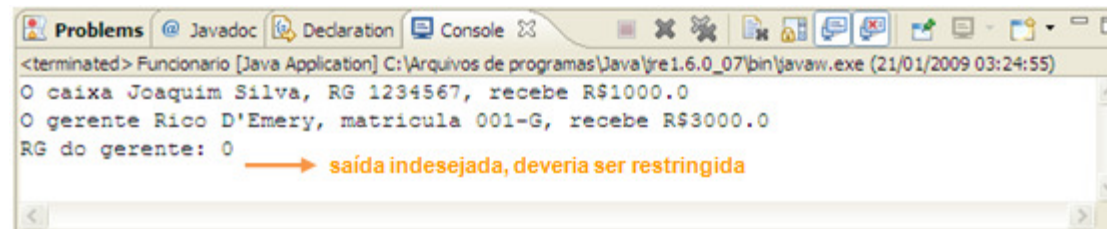
Exemplo – Sem Interface

- Analisando:
 - Logo se liberarmos a instrução que está comentada na aplicação (método `main()`) o programa executará a instrução sem nenhum problema, pois não existe qualquer restrição para a sua utilização

```
System.out.println("RG do gerente: " + b.getRg());
```

Método que deveria ser restrinido

- Vejamos o resultado no console:



```
<terminated> Funcionario [Java Application] C:\Arquivos de programas\Java\jre1.6.0_07\bin\javaw.exe (21/01/2009 03:24:55)
O caixa Joaquim Silva, RG 1234567, recebe R$1000.0
O gerente Rico D'Emery, matricula 001-G, recebe R$3000.0
RG do gerente: 0
```

saída indesejada, deveria ser restringida

Interface



- E como resolver a questão do acesso?
 - A utilização de **interface** nesse tipo de acesso não seria permitido, pois quando um objeto se relacionar com os métodos, ele não acessará seus recursos diretamente, dependerá do que ele poderá reconhecer através das permissões de sua **interface**.
 - Java estendeu o conceito de interfaces à um nível ainda mais flexível e ao modelar um sistema
 - o desenvolvedor poderá pensar apenas nas interfaces de seus objetos (ou seja, nos métodos de cada um deles, e nos seus relacionamentos), criando assim uma camada extra de abstração.

Exemplo – Com Interface

//Início do arquivo Funcionario.java

Modificador que implementa uma interface (Caixa) para a classe Funcionario.
Definirá tudo que será acessível a objetos do tipo Caixa na classe Funcionário.

```
public interface Caixa {  
    public static final int FUNCIONARIO_CAIXA = 0;  
  
    public String getNome();  
    public int getRg();  
    public double getSalario();  
    public void getInformacaoFuncao();  
}
```

Constante declarada para objetos Caixa

Métodos que poderão ser utilizados.
Observe que o método getMatricula() não foi definido, logo um objeto Caixa não poderá utilizá-los.

Modificador que implementa uma interface (Gerente) para a classe Funcionario.
Definirá tudo que será acessível a objetos do tipo Gerente na classe Funcionário.

```
public interface Gerente {  
    public static final int FUNCIONARIO_GERENTE = 1;  
  
    public String getNome();  
    public String getMatricula();  
    public double getSalario();  
    public void getInformacaoFuncao();  
}
```

Constante declarada para objetos Gerente

Métodos que poderão ser utilizados.
Observe que o método getRg() não foi definido, logo um objeto Gerente não poderá utilizá-los.

Exemplo – Com Interface

Uma vez definida, a interface é compilada normalmente (é gerado um arquivo `.class` para cada interface, como se fosse uma classe qualquer). A sintaxe utiliza a palavra-chave `implements` para dizer que uma classe implementa uma ou mais interfaces informadas.

```
public class Funcionario implements Caixa, Gerente{

    //Declarando os atributos
    private String nome;
    private String matricula;
    private int rg;
    private double salario;
    private int cargo;

    //Método construtor
    public Funcionario (String n, int rg, int cg, double sal){
        this.nome = n;
        this.rg = rg;
        this.cargo = cg;
        this.salario = sal;
    }

    //Método construtor (difere do primeiro, polimorfismo)
    public Funcionario (String n, String mat, int cg, double
sal){
        this.nome = n;
        this.matricula = mat;
        this.cargo = cg;
        this.salario = sal;
    }
}
```

A classe Funcionario poderá fazer uso de duas interfaces (Caixa e Gerente)

Exemplo – Com Interface



Implementação de método. Observe que a interface apenas contém a assinatura do método, pois é dessa forma que define quais métodos poderão ser utilizados pelos objetos, e é necessário implementar o método na classe, dizendo o que ele faz.

```
public String getNome() {  
    return nome;  
}  
  
public String getMatricula() {  
    return matricula;  
}  
  
public int getRg() {  
    return rg;  
}  
  
public double getSalario() {  
    return salario;  
}  
  
public void getInformacaoFuncao() {  
    if (this.cargo == FUNCIONARIO_CAIXA)  
        System.out.println("O caixa " + this.getNome() +  
            ", RG " + this.getRg() + ", recebe R$" + this.getSalario());  
  
    if (this.cargo == FUNCIONARIO_GERENTE)  
        System.out.println("O gerente " + this.getNome()  
            + ", matricula " + this.getMatricula() + ", recebe R$" +  
            this.getSalario());  
}
```

Exemplo – Com Interface



```
//método principal (aplicação)
public static void main (String [] args){

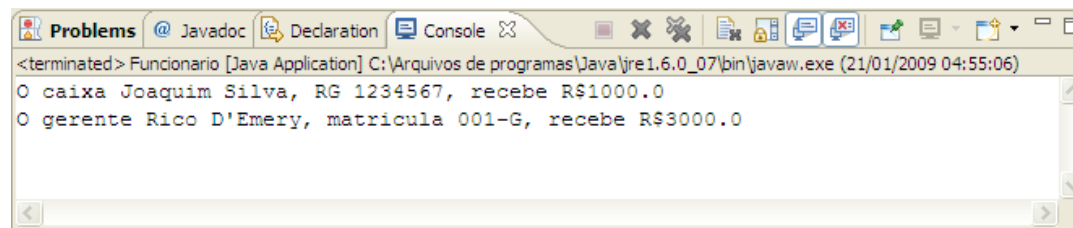
    Observe que o objeto a pertence a classe Caixa, logo só poderá
    fazer uso dos métodos e constantes declarados na interface Caixa.
    Da mesma forma que o objeto b só poderá fazer uso do que foi
    liberado pela interface Gerente

    Caixa a = new Funcionario ("Joaquim Silva", 1234567, 0,
1000.00);
    Gerente b = new Funcionario ("Rico D'Emery", "001-G",
1, 3000.00);

    //chamada de método para exibir as informações dos
funcionários
    a.getInformacaoFuncao();
    b.getInformacaoFuncao();
}
}
//Fim do arquivo Funcionario.java
```


Exemplo – Com Interface

- Vejamos o resultado:



```
<terminated> Funcionario [Java Application] C:\Arquivos de programas\Java\jre1.6.0_07\bin\javaw.exe (21/01/2009 04:55:06)
O caixa Joaquim Silva, RG 1234567, recebe R$1000.0
O gerente Rico D'Emery, matricula 001-G, recebe R$3000.0
```

– Analisando:

- uma classe pode implementar mais de uma interface,
 - são colocadas lado a lado, separadas por vírgula juntamente com a cláusula **implements**.
- Quando uma classe implementa uma interface, ela deve obrigatoriamente implementar os métodos definidos na interface.
 - E é dessa forma que é possível tratar objetos do tipo `Funcionario` como se fossem do tipo `Caixa` ou `Gerente`.

Exemplo – Com Interface

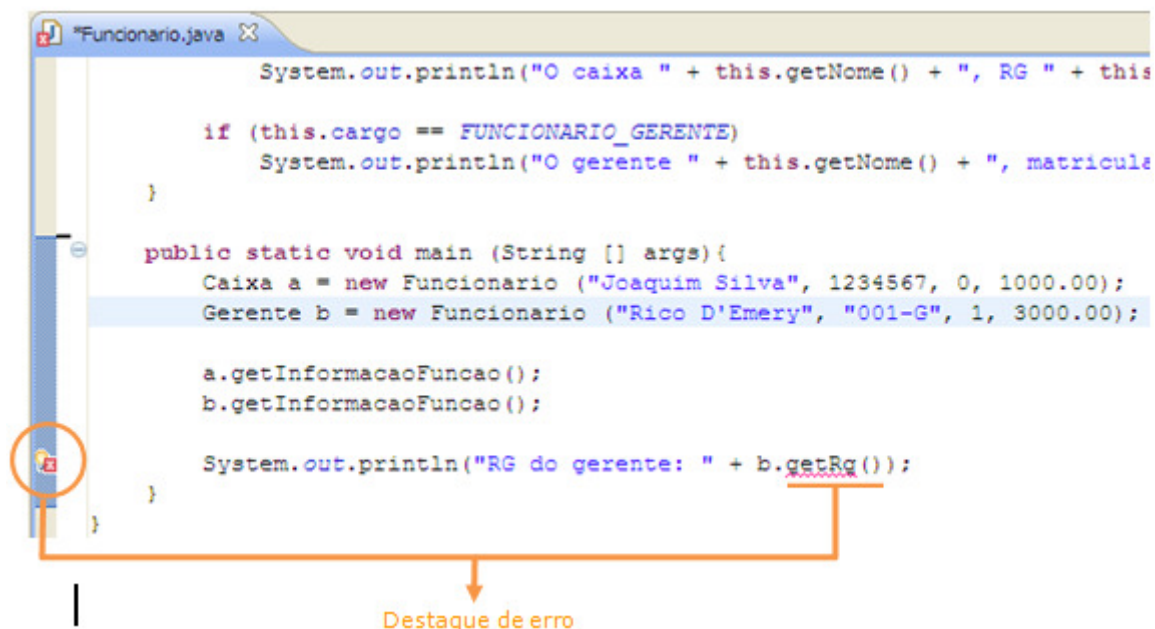
– Analisando

- Finalizando: o que aconteceria se um objeto regido por uma interface tentasse fazer uso de um método não definido por ela.
 - No primeiro exemplo, onde não fazíamos uso de interface, o objeto **b** por ser da classe **Funcionário** conseguia fazer uso do método `getRg()`, por não existir nenhuma restrição, então vamos ver o que aconteceria se o objeto **b** (agora regido pela interface **Gerente**) tentasse fazer uso do método `getRg()` (que não foi definido na interface `Gerente`). Para isso utilizaremos a mesma instrução do programa anterior:

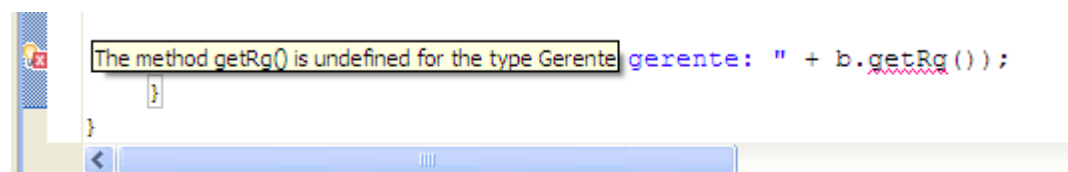
```
System.out.println("RG do gerente: " + b.getRg());
```

Exemplo – Com Interface

– Analisando

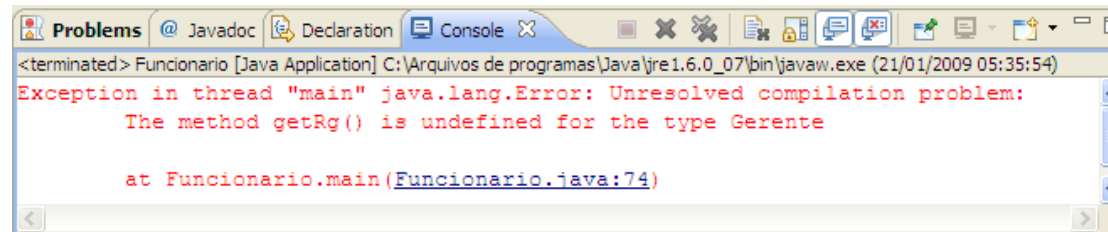


- Se passarmos o mouse sobre o erro apresentado (🔍), o IDE Eclipse exibirá o erro:



Exemplo – Com Interface

- Ou seja, é informado que o método `getRg()` não foi declarado na interface `Gerente`, e não poderá fazer uso do método. Mesmo com o erro em vista se desejar compilar, poderá observar que o problema será apresentado no console:



```
<terminated> Funcionario [Java Application] C:\Arquivos de programas\Java\jre1.6.0_07\bin\javaw.exe (21/01/2009 05:35:54)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    The method getRg() is undefined for the type Gerente

    at Funcionario.main(Funcionario.java:74)
```

- E da mesma forma aconteceria se o objeto `a` (regido pela interface `Caixa`) tentasse chamar o método `getMatrícula()` (que não está definido na interface `Caixa`)

Exemplo – Com Interface

- Artífício para Herança Múltipla
 - Java não permite a implementação de **herança múltipla**, mas com o auxílio do conceito de **interface**, podemos obter um resultado semelhante, como mostrado no exemplo, onde pode-se dizer que "um funcionário pode ser um caixa".

```
Caixa a = new Funcionario ();
```

- e "um outro funcionário é um gerente":

```
Gerente b = new Funcionario();
```

Exercício Exemplo



- Implemente em Java a seguinte situação:
 - Uma animal poder ser racional ou irracional
 - Se irracional possui o comportamento de latir, se racional de falar
 - Escreva uma aplicação que possui um homem (racional) e um cachorro (irracional), que acessam seus comportamentos.
- Qual a mensagem de erro emitida pelo eclipse quando se tenta criar um homem irracional?



Aula 10: Classe Abstrata

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

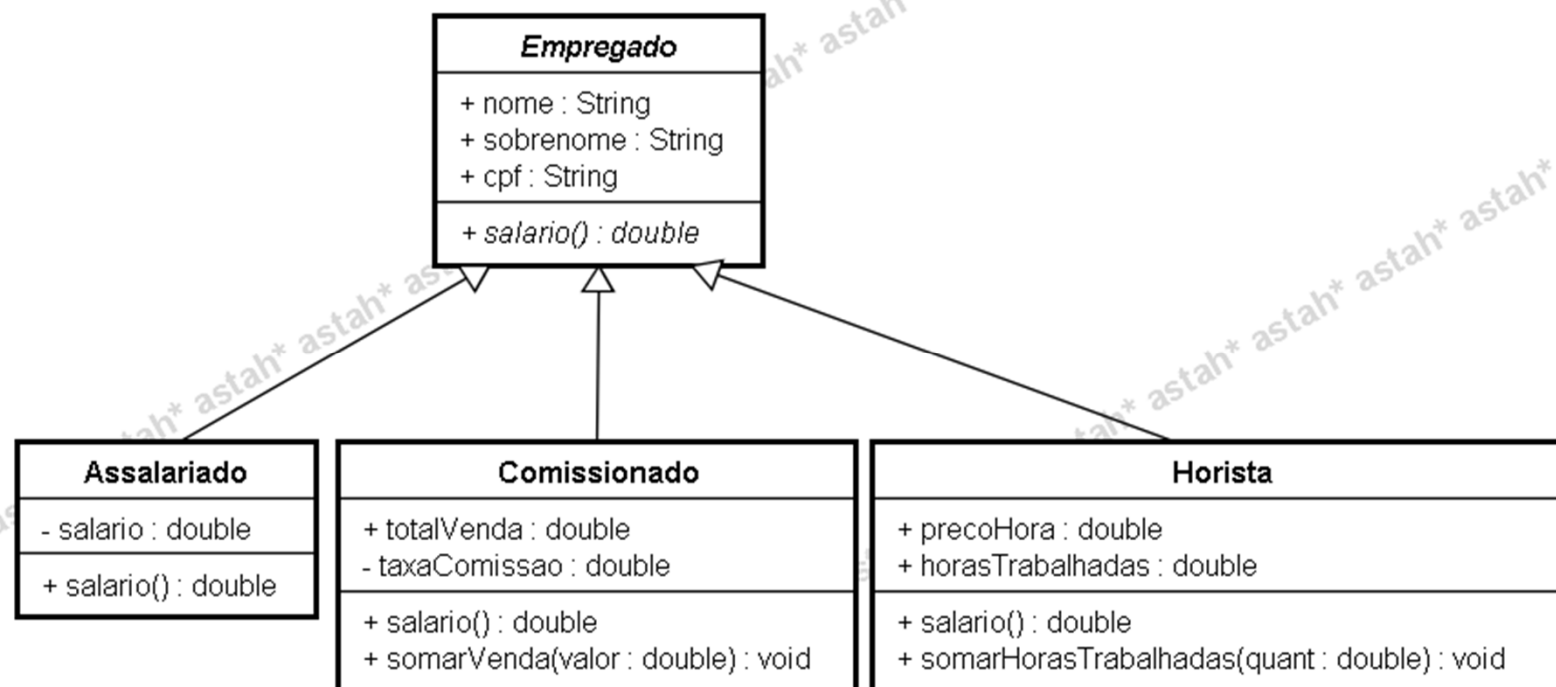
contato: rico_demery@yahoo.com.br

Classe Abstratas

- Serve apenas como modelo
 - Para uma classe concreta (classe que comumente usamos).
 - Então: não podem ser instanciadas diretamente com o **new**,
 - Devem ser herdadas por classes concretas.
 - Podem conter ou não métodos abstratos
 - Possui a mesma definição da assinatura de método encontrada em interfaces. Ou seja, uma classe abstrata pode implementar ou não um método.
 - Devem obrigatoriamente ser implementados em uma classe concreta. Mas se uma classe abstrata herdar outra classe abstrata, a classe que herda não precisa implementar os métodos abstratos.

Classe Abstratas

- Implementação
 - Para uma classe ou método abstrato usa-se a palavra-chave **abstract**.
 - Exemplo:



Exemplo



```
1 package exemplo1Abstracao;
2
3 public abstract class Empregado {
4     String nome, sobrenome, cpf;
5
6     public Empregado(String nome, String sobrenome, String cpf){
7         this.nome=nome;
8         this.sobrenome=sobrenome;
9         this.cpf=cpf;
10    }
11
12    public abstract double salario();//método abstrato (funciona como interface)
13 }
```

Exemplo



```
1 package exemplo1Abstracao;
2
3 public class Assalariado extends Empregado {
4
5     private double salario;
6
7     public Assalariado(String nome, String sobrenome, String cpf, double salario) {
8         super(nome, sobrenome, cpf);
9         this.salario=salario;
10    }
11
12    //deve implementar o método definido como abstract na superclasse
13    //exemplo de sobreposição
14    public double salario() {//poderia ser outra coisa, não necessariamente a mesma funcionalidade do get
15        return this.salario;
16    }
17
18    public double getSalario() {return this.salario;}
19    public void setSalario(double salario) {this.salario=salario;}
20 }
```

Exemplo



```
1 package exemplo1Abstracao;
2
3 public class Comissionado extends Empregado {
4
5     double totalVenda;
6     private double taxaComissao;
7
8     public Comissionado(String nome, String sobrenome, String cpf) {
9         super(nome, sobrenome, cpf);
10    }
11
12    public double salario() {//redefinição de método atendendo as particularidades da classe
13        return (this.totalVenda * this.taxaComissao);
14    }
15
16    public void somarVenda(double valor){
17        this.totalVenda+=valor;
18    }
19
20    public double getTaxaComissao() {return this.taxaComissao;}
21    public void setTaxaComissao(double taxaComissao) {this.taxaComissao = taxaComissao;}
22 }
```

Exemplo



```
1 package exemplo1Abstracao;
2
3 public class Horista extends Empregado {
4
5     double precoHora;
6     double horasTrabalhadas;
7
8     public Horista(String nome, String sobrenome, String cpf, double precoHora) {
9         super(nome, sobrenome, cpf);
10        this.precoHora=precoHora;
11    }
12
13    public double salario() {
14        return this.precoHora*this.horasTrabalhadas;
15    }
16
17    public void somarHorasTrabalhadas(double quant){
18        this.horasTrabalhadas+=quant;
19    }
20 }
```

Exercício 1

- Eletronico
 - Implemente em Java a seguinte que representa uma classe abastrata de Eletrônicos aos quais possuem as características consumo, voltagem e status; assim como os comportamentos ligar, desligar e isLigado(que informa se o eletrônico está ligado)
 - O status será ligado ou desligado dependendo da utilização do comportamento.
 - Os comportamentos ligar e deligar são abstratos.
- Uma TV é um eletrônico que possui polegada e tipo. Ao utilizar o comportamento ligar seu status será ligado e o seu canal será modificado para 12.

Exercício 1 (cont.)

- Um Radio também é um eletrônico que possui:
 - Características
 - AM / FM – ambos static final com valor 1 e 2, respectivamente
 - banda, sintonia e volume
 - Comportamentos
 - desligar
 - Torna seu status desligado e volume zero
 - ligar
 - Torna seu status ligado , sintonia 88.1 e volume 10.



Exercícios

Interface versus Classe Abstrata

Richarlyson A. D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br

Exercício Exemplo

- No Exercício Exemplo do slide 22 vimos a utilização de interface, agora uniremos interface e classe abstrata. Implemente em Java a situação abaixo:
 - Racional é uma interface que permite falar e andar
 - Irracional é uma interface que permite latir
 - Um Animal é uma class abstrata que possui os comportamentos de latir e falar
 - Um SerHumano é um Animal porém Racional
 - Ilustre um homem falando.

Exercício 2



- Fazendo uso dos conceitos de Interface e Classe Abstrata, implemente em Java uma solução que representa a seguinte situação:

Um homem é um animal que ao andar fala "Eu amo"

Exercício 3



- Fazendo uso dos conceitos de Interface, Classe Abstrata e método Abstrato, implemente em Java uma solução para o exemplo ilustrado na Aula 9 (Funcionario / Caixa / Gerente).



FIM

Prof. Richarlyson D'Emery

site: <https://sites.google.com/site/profricodemery/mpoo>

grupo: http://groups.google.com/group/mpoo_uast

email grupo: mpoo_uast@googlegroups.com

contato: rico_demery@yahoo.com.br