

Herança

POO29004 – Engenharia de Telecomunicações

Prof. Emerson Ribeiro de Mello

<http://docente.ifsc.edu.br/mello/poo>

22 DE ABRIL DE 2020



**INSTITUTO
FEDERAL**
Santa Catarina

Câmpus
São José

Genética

Um organismo adquire características semelhantes à do organismo que o gerou



Genética

Um organismo adquire características semelhantes à do organismo que o gerou

Programação Orientada a Objetos

Uma classe herda atributos e métodos de uma outra classe



- O conceito de **herança** pode tornar mais rápido o desenvolvimento de softwares complexos
 - Novas classes são criadas baseadas em classes existentes
- **classe filha, subclasse ou classe derivada**
 - A classe que herda os atributos e funções de outra classe
- **classe pai, superclasse ou classe base**
 - A classe cujo membros são herdados por outras classes



- O conceito de **herança** pode tornar mais rápido o desenvolvimento de softwares complexos
 - Novas classes são criadas baseadas em classes existentes
- **classe filha, subclasse** ou **classe derivada**
 - A classe que herda os atributos e funções de outra classe
- **classe pai, superclasse** ou **classe base**
 - A classe cujo membros são herdados por outras classes

Quando usar herança?

Ideal para casos onde são **necessárias classes distintas para atacar problemas específicos**. Porém, tais classes necessitam compartilhar um núcleo comum




Exemplo: Sistema para cadastro de produtos

- Uma indústria da área de telecomunicações necessita de um sistema para cadastrar os produtos que fabrica
 - **Aparelho telefônico**
- As informações necessárias para o cadastro são:
 - código, número de série, modelo, cor, peso, dimensões (AxLxP)



Exemplo: Sistema para cadastro de produtos

- Uma indústria da área de telecomunicações necessita de um sistema para cadastrar os produtos que fabrica
 - **Aparelho telefônico**
- As informações necessárias para o cadastro são:
 - código, número de série, modelo, cor, peso, dimensões (AxLxP)

 Telefone
<i>Attributes</i> <ul style="list-style-type: none">- codigo : int- numSerie : string- modelo : string- peso : float- dim : Dimensao
<i>Operations</i> <ul style="list-style-type: none">+ Telefone(c : int, n : string, m : string, p : float, d : Dimensao)+ imprimirDados() : void



Exemplo: Sistema para cadastro de produtos

- A empresa começou a fabricar também **telefones sem fio**
- Os **telefones sem fio** compartilham todas as características de um **telefone**, porém possuem novas características
 - frequência, quantidade de canais, distância de operação
- O atual sistema não permite cadastrar essas novas informações



Exemplo: Sistema para cadastro de produtos

- A empresa começou a fabricar também **telefones sem fio**
- Os **telefones sem fio** compartilham todas as características de um **telefone**, porém possuem novas características
 - frequência, quantidade de canais, distância de operação
- O atual sistema não permite cadastrar essas novas informações

O que fazer?

- 1 Criar uma nova classe telefone sem fio e colocar nela tudo o que tem na classe telefone mais as características do telefone sem fio?
- 2 Herdar as características comuns da classe telefone e adicionar as particulares do telefone sem fio?



Exemplo: Sistema para cadastro de produtos

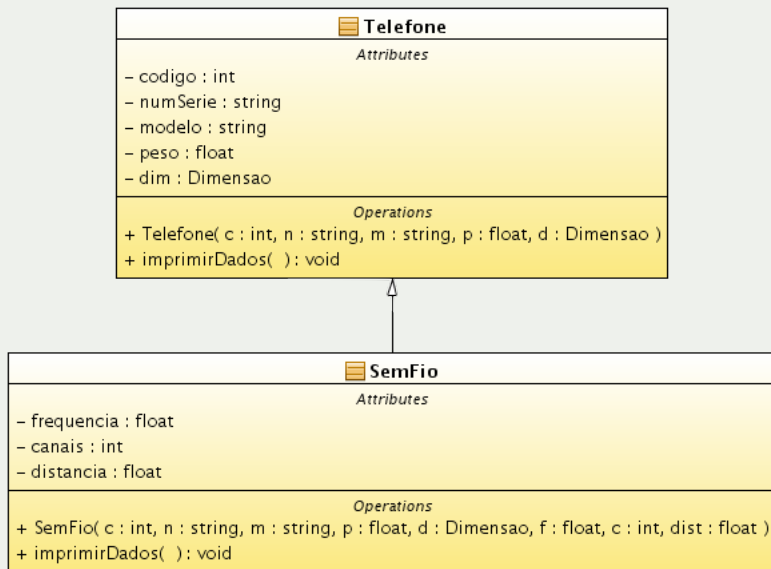
- A empresa começou a fabricar também **telefones sem fio**
- Os **telefones sem fio** compartilham todas as características de um **telefone**, porém possuem novas características
 - frequência, quantidade de canais, distância de operação
- O atual sistema não permite cadastrar essas novas informações

O que fazer?

- 1 Criar uma nova classe telefone sem fio e colocar nela tudo o que tem na classe telefone mais as características do telefone sem fio?
- 2 Herdar as características comuns da classe telefone e adicionar as particulares do telefone sem fio?



Herança: exemplo



Herança: Superclasse Telefone

```
1 public class Telefone{
2     private int codigo;
3     private String numSerie, modelo;
4     private float peso;
5     private Dimensao dim;
6
7     public Telefone(int c, String s, String m, float p, Dimensao d)
8     {
9         this.codigo = c; this.peso = p; this.dim = d;
10        this.numSerie = s;this.modelo = m;
11    }
12
13    public void imprimirDados(){
14        System.out.println("Codigo: " + this.codigo);
15        ...
16        this.dim.imprimirDados();
17    }
18 }
```



Herança: Subclasse SemFio

```
19 public class Semfio extends Telefone{
20     private float frequencia, distancia;
21     private int canais;
22
23     public SemFio(int c, String s, String m, float p, Dimensao d, int ca,
24         float f, float dis){
25         super(c, s, m, p, d); // invocando o construtor da superclasse
26         this.frequencia = f;
27         this.distancia = dis;
28         this.canais = ca;
29     }
30
31     // sobrescrita do metodo da superclasse
32     public void imprimirDados(){
33         super.imprimirDados(); // invocando o metodo de mesmo nome da
34         // superclasse
35         System.out.println("Freq: " + this.frequencia);
36         ...
37     }
38 }
```



Herança: Criando instâncias do Telefone e SemFio

```
37 public class Principal{
38     public static void main(String[] args){
39         Telefone t = new Telefone(1,"ABC123","MesaTel",0.5, new Dimensao
40             (10,10,5));
41
42         SemFio sf = new SemFio(2,"DEF456","LivreTel",0.7,new Dimensao
43             (20,8,7), 11, 2400,100);
44
45         t.imprimirDados();
46         sf.imprimirDados();
47     }
48 }
```



- Um subclasse pode sobrescrever um método da superclasse que tenha a mesma assinatura

```
47 public class Telefone{
48     public void ola(){
49         System.out.println("Ola, sou um telefone");
50     }
51 }
52 public class Semfio extends Telefone{
53     public void ola(){
54         System.out.println("Ola, sou um telefone sem fio");
55     }
56 }
57 public class Principal{
58     public static void main(String args[]){
59         Telefone t = new Telefone();
60         Semfio s = new Semfio();
61         t.ola(); // Ola, sou um telefone
62         s.ola(); // Ola, sou um telefone sem fio
63     }
64 }
```



- Os membros **privados** de uma classe só podem ser acessados pelos demais membros desta mesma classe
- Os membros **públicos** de uma classe podem ser acessados por qualquer outra classe
- O modificador de acesso **protected** apresenta uma restrição intermediária entre o **private** e o **public**
- Membros **protegidos** podem ser acessados pelos demais membros da classe, pelas demais classes do pacote e pelas **classes derivadas**



Modificador de acesso protected: exemplo

```
66 package produtos;
67
68 public class Telefone{
69     private String marca;
70     protected String modelo;
71     public float peso;
72 }
```

```
73 package produtos;
74
75 public class SemFio extends Telefone{
76     private float frequencia;
77     public void modificador(){
78         this.frequencia = 900; // acesso ok
79         this.modelo = "ABC"; // acesso ok
80         this.peso = 0.5; // acesso ok
81         this.marca = "GrandTel"; // erro! Nao permitido
82     }
83 }
```



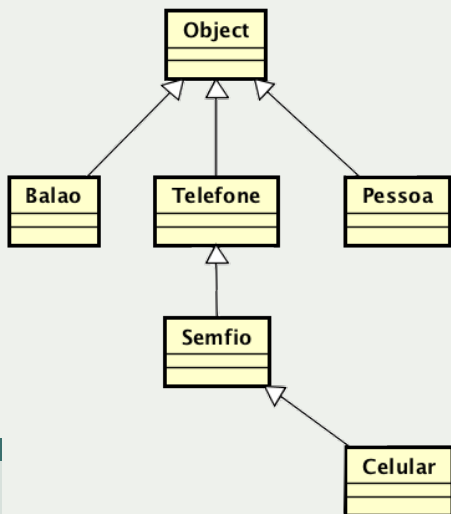
Modificador de acesso protected: exemplo

```
84 package poo;
85 import produtos.Telefone;
86 import produtos.SemFio;
87
88 public class Principal{
89     public static void main(String[] args){
90         Telefone t = new Telefone();
91         SemFio sf = new SemFio();
92
93         // invocando um membro public
94         t.peso = 0.6; // acesso ok
95         // invocando um membro protected
96         t.modelo = "DEF"; // erro!
97         // invocando um membro private
98         t.marca = "GT"; // erro!
99     }
100 }
```



Associação do tipo Herança em Java

- Com exceção da classe `Object`, que não possui superclasse, toda classe Java tem uma e somente uma superclasse direta
 - Toda classe herda implicitamente da classe `Object`
- Uma classe pode ser derivada de uma outra classe e essa por sua vez pode ser derivada de outra classe, ...



Herança pode ser lida como é um

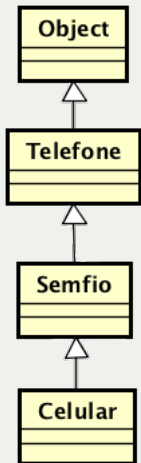
- **Celular é um Telefone**



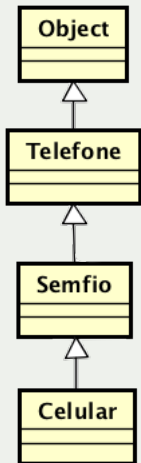
Coerção de tipos (*typecasting*) – ou conversão de tipos

```
101 Telefone a = new Telefone();  
102 Semfio   b = new SemFio();  
103 Celular  c = new Celular();
```

- Celular **é** um Telefone?
- Um Telefone pode ser um Celular?



Coerção de tipos (*typecasting*) – ou conversão de tipos



```
110 Telefone a = new Telefone();
111 Semfio b = new SemFio();
112 Celular c = new Celular();
```

- Celular **é** um Telefone? **SIM!**
- Um Telefone pode ser um Celular? **Não**

typecasting

O uso do objeto de um tipo na referência de um outro tipo

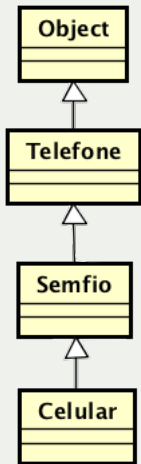
```
113 Telefone d = new Celular(); // OK, coerção implícita
114 Object e = new Semfio(); // OK, coerção implícita
115 Celular f = (Celular) d; //OK, coerção explícita
116
117 Celular g = a; // ERRO! Telefone não é Celular
118 Celular h = (Celular) e; // ERRO! Semfio não é Celular
```



Coerção de tipos (*typecasting*) – ou conversão de tipos

Operador instanceof

Teste lógico para verificar o tipo de um objeto



```
119 Telefone vetor[] = new Telefone[3];
120 vetor[0] = new Telefone();
121 vetor[1] = new Semfio();
122 vetor[2] = new Celular();
123
124 for(int i = 0; i < 3; i++ ){
125
126     if (vetor[i] instanceof Celular){
127
128         Celular c = (Celular) vetor[i];
129         ....
130     }
131 }
```



Sobrescrita dos método equals

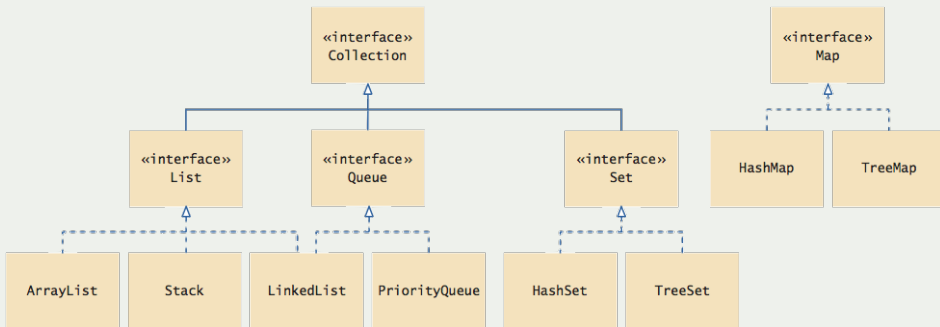
- Por padrão toda classe Java herda da classe `Object` e essa define o método `public boolean equals(Object o)`
- A implementação do método `equals` pela classe `String` pode ser usada para verificar se duas `Strings` são iguais

```
132 String s = "dia"
133 String nova = "noite"
134
135 if (s.equals(nova)){
136     System.out.println("São iguais");
137 }else{
138     System.out.println("Não são iguais");
139 }
```

- Você poderá sobrescrever o método `equals` em suas classes, caso deseje comparar atributos dos objetos dessas classes



Hierarquia do Framework Collections



```
140 Collection<String> colecao = new ArrayList<>();  
141 colecao.add("P00");
```



Exercícios

Você pode criar as classes que julgar necessário, contudo desde que não inclua herança múltipla

- 1 Pessoa, estudante, professor
- 2 Bicicleta, carro, moto, navio, avião, helicóptero
- 3 Conta corrente, poupança





CAELUM

APOSTILA CAELUM FJ-11 JAVA E ORIENTAÇÃO A OBJETOS

<http://docente.ifsc.edu.br/mello/livros/java/apostila-caelum-java-orientacao-objetos-FJ11.pdf>

- Ler seções 9.1, 9.2 e 9.3 sobre Herança

