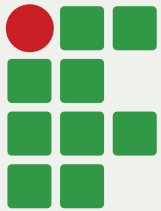
Linguagem de Programação 1

Herança

Prof. Fábio José Rodrigues Pinheiro

Dezembro de 2021



INSTITUTO FEDERAL
Catarinense
Campus Videira

Herança

Genética

Um organismo adquire características semelhantes à do organismo que o gerou

Herança

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

Herança

- 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 métodos de outra classe
- **classe pai**, **superclasse** ou **classe base**
  - A classe cujo membros são herdados por outras classes

Herança

- 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 métodos de outra classe
- **classe pai**, **superclasse** ou **classe base**
  - A classe cujo membros são herdados por outras classes

Quando usar herança?

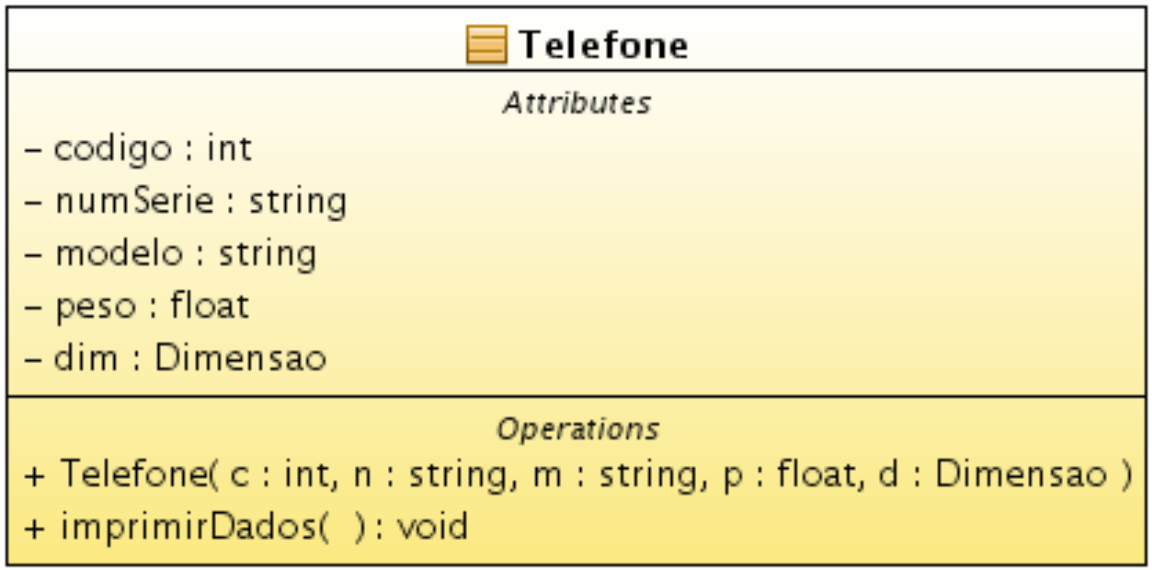
Ideal para casos onde são **necessárias classes distintas para atacar problemas específicos**. Porém, tais classes necessitam compartilhar um núcleo comum

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)



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?

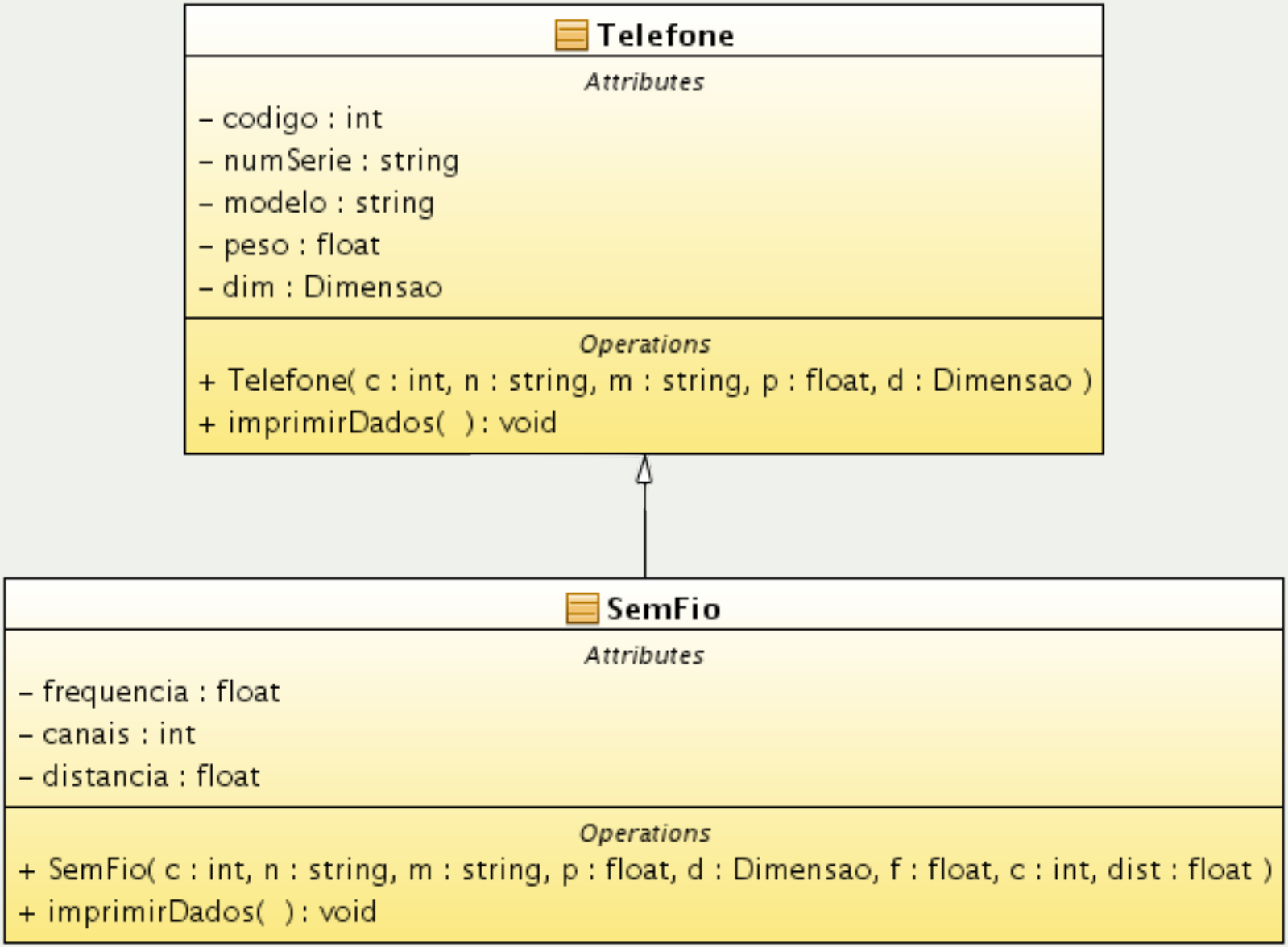
Exemplo: Sistema para cadastro de produtos

- 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

```
1  public class SemFio extends Telefone{
2    private float frequencia, distancia;
3    private int canais;
4
5    public SemFio(int c, String s, String m, float p, Dimensao d, int ca,
         float f, float dis){
6      super(c, s, m, p, d); // invocando o construtor da superclasse
7      this.frequencia = f;
8      this.distancia = dis;
9      this.canais = ca;
10   }
11
12   // sobrescrita do metodo da superclasse
13   public void imprimirDados(){
14     super.imprimirDados(); // invocando o metodo de mesmo nome da
       superclasse
15     System.out.println("Freq: " + this.frequencia);
16     ...
17   }
18 }
```


Herança: Criando instâncias do Telefone e SemFio

```
1  public class Principal{
2    public static void main(String[] args){
3      Telefone t = new Telefone(1,"ABC123","MesaTel",0.5, new Dimensao
       (10,10,5));
4
5      SemFio sf = new SemFio(2,"DEF456","LivreTel",0.7,new Dimensao
       (20,8,7), 11, 2400,100);
6
7      t.imprimirDados();
8      sf.imprimirDados();
9    }
10 }
```


Sobrescrita de método

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

```
1  public class Telefone{
2    public void ola(){
3      System.out.println("Ola, sou um telefone");
4    }
5  }
6  public class Semfio extends Telefone{
7    public void ola(){
8      System.out.println("Ola, sou um telefone sem fio");
9    }
10 }
11 public class Principal{
12   public static void main(String args[]){
13     Telefone t = new Telefone();
14     Semfio   s = new Semfio();
15     t.ola(); // Ola, sou um telefone
16     s.ola(); // Ola, sou um telefone sem fio
17   }
18 }
```


Membros públicos, privados e protegidos

- 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

```java
package produtos;

public class Telefone{
  private String marca;
  protected String modelo;
  public float peso;
}
```

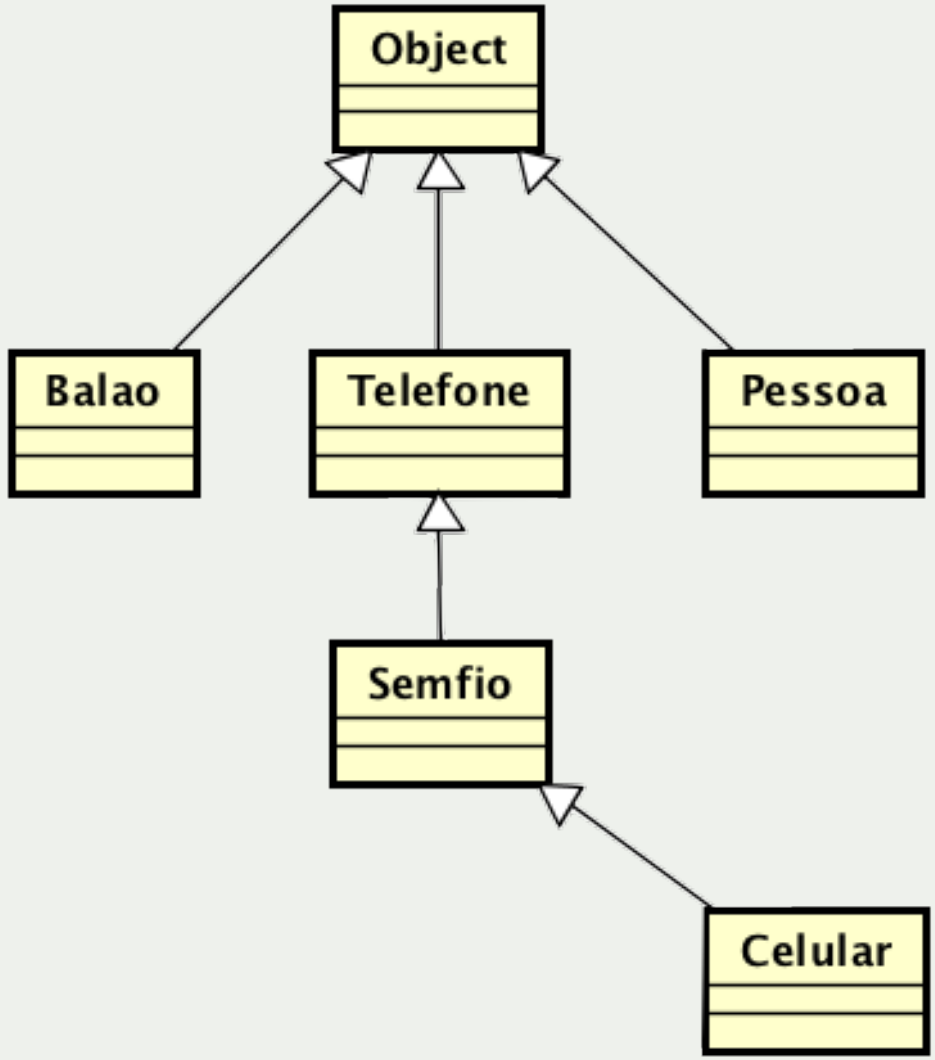
```java
package produtos;

public class SemFio extends Telefone{
  private float frequencia;
  public void modificador(){
    this.frequencia = 900; // acesso ok
    this.modelo = "ABC"; // acesso ok
    this.peso = 0.5; // acesso ok
    this.marca = "GrandTel"; // erro! Nao permitido
  }
}
```


Modificador de acesso protected: exemplo

```
1  package poo;
2  import produtos.Telefone;
3  import produtos.SemFio;
4
5  public class Principal{
6    public static void main(String[] args){
7      Telefone t = new Telefone();
8      SemFio sf = new SemFio();
9
10     // invocando um membro public
11     t.peso = 0.6; // acesso ok
12     // invocando um membro protected
13     t.modelo = "DEF"; // erro!
14     // invocando um membro private
15     t.marca = "GT"; // erro!
16   }
17 }
```


Associação do tipo Herança em Java

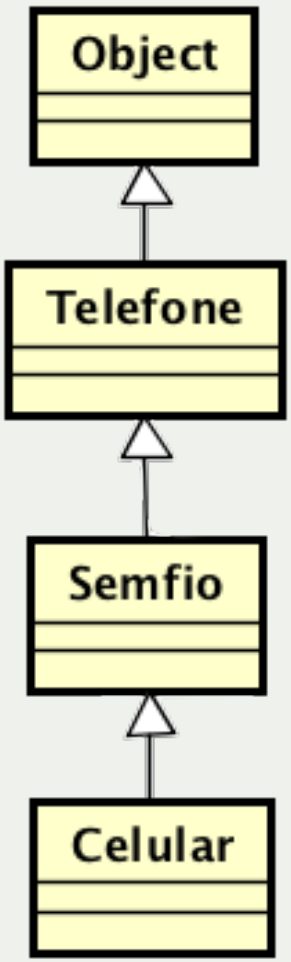
- Com exceção da classe `Object`, que não possui superclasse, toda classe Java tem uma e somente uma superclasse direta
  - Toda classe herda implicitamente da classe `Object`
- Uma classe pode ser derivada de uma outra classe e essa por sua vez pode ser derivada de outra classe, . . .



Herança pode ser lida como **é um**

- Celular **é um** Telefone

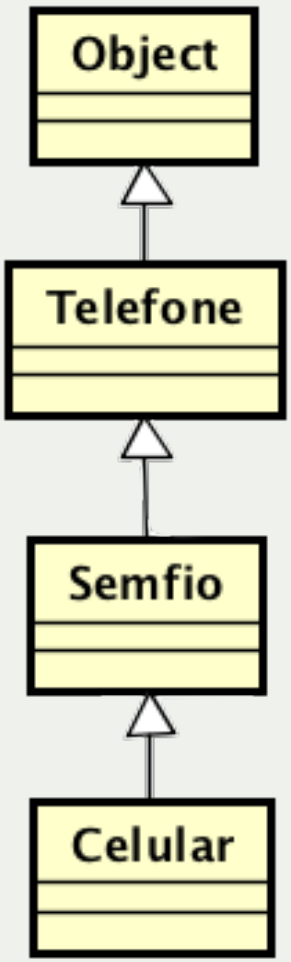
Coerção de tipos (*typecasting*) – ou conversão de tipos




```
1 Telefone a = new Telefone();
2 Semfio   b = new SemFio();
3 Celular  c = new Celular();
```

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

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



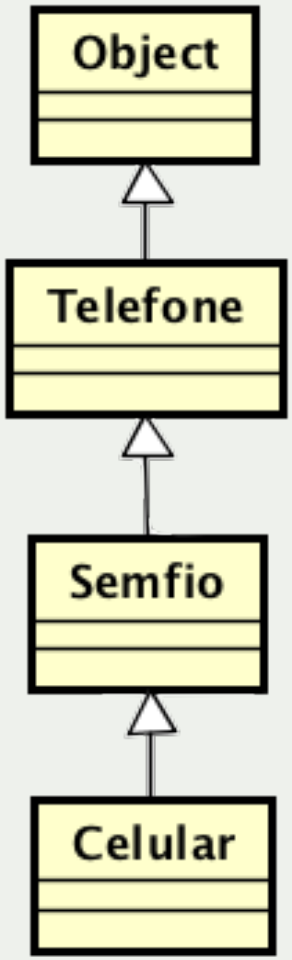

```
1 Telefone a = new Telefone();
2 Semfio   b = new SemFio();
3 Celular  c = new Celular();
```

- 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

```
1 Telefone d = new Celular(); // OK, coerção implítica
2 Object   e = new Semfio(); // OK, coerção implítica
3 Celular  f = (Celular) d; //OK, coerção explícita
4
5 Celular  g = a; // ERRO! Telefone não é Celular
6 Celular  h = (Celular) e; // ERRO! Semfio não é Celular
```

Operador `instanceof`

Teste lógico para verificar o tipo de um objeto

```
1  Telefone vetor[] = new Telefone[3];
2  vetor[0] = new Telefone();
3  vetor[1] = new SemFio();
4  vetor[2] = new Celular();
5
6  for(int i = 0; i < 3; i++ ){
7
8    if (vetor[i] instanceof Celular){
9
10     Celular c = (Celular) vetor[i];
11     ....
12   }
13 }
```

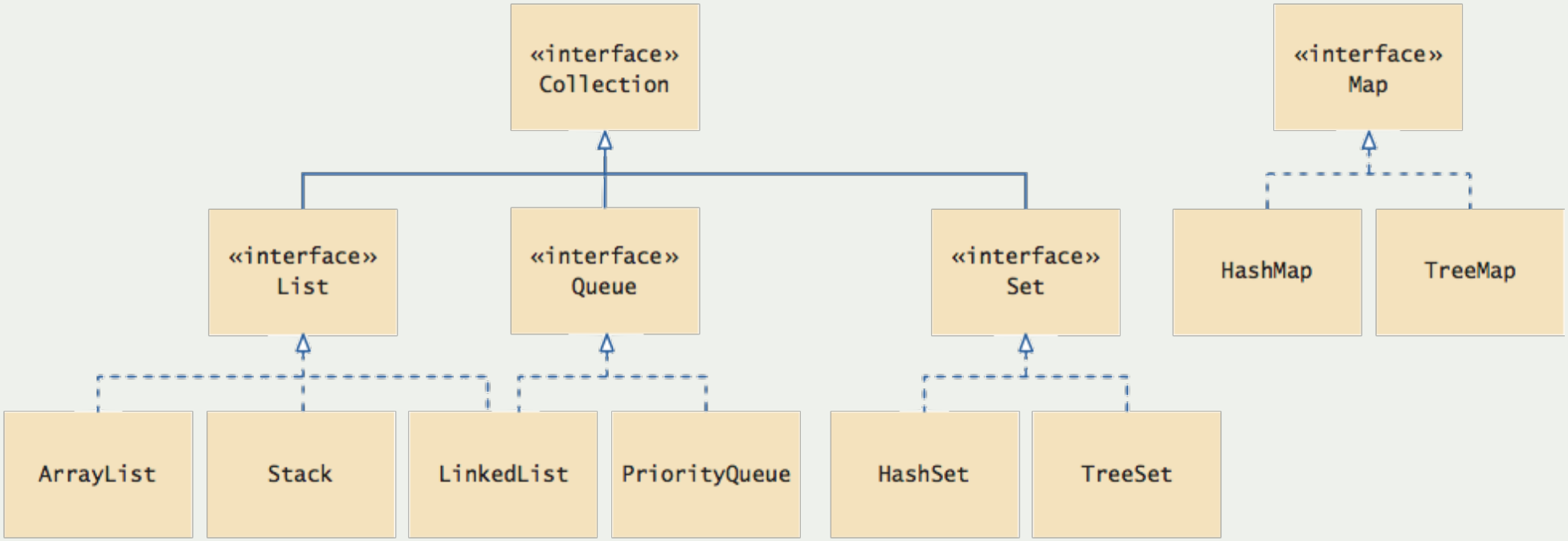
Sobrescrita dos método `equals`

- 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

```
1 String s = "dia"
2 String nova = "noite"
3
4 if (s.equals(nova)){
5   System.out.println("São iguais");
6 }else{
7   System.out.println("Não são iguais");
8 }
```

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

Hierarquia do Framework Collections




```
1 Collection<String> colecao = new ArrayList<>();
2 colecao.add("POO");
```


Exercícios

Diagrama de classes UML com associações de herança

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. Funcionário, vendedor, gerente, diretor

Exercício 02

- Desenvolva o diagrama do item 03 do slide anterior:
  - A classe **Funcionário** deve possuir dois atributos, nome e salário. Salário deve ser do tipo protected. Crie os métodos get e set, construtor com todos os atributos e o método toString.
  - Além dos atributos herdados a classe **Gerente** possui a informação sobre o departamento que trabalha. Crie os métodos get e set, e construtor necessários e sobrescreva o método toString.
  - A classe **Vendedor** possui um atributo que representa sua comissão (percentual), e um método denominado `calcularSalario` que calcula o valor do salário acrescido da comissão. O método toString deve apresentar as informações do empregado, salário sem comissão, salario com comissão e percentual de comissão.
  - A classe **Diretor** possui o atributo "participação nos lucros", que é uma gratificação adicionada ao salário duas vezes no ano: nos meses de junho e dezembro. Além dos métodos get, set e construtor, crie o método `calcularSalario`.

Leitura complementar

Caelum

Apostila Caelum FJ-11 Java e Orientação a Objetos

disponível na pasta compartilhada na página da disciplina

- Ler seções 9.1, 9.2 e 9.3 sobre Herança