Programação Orientada a Objetos Java

Alexandre Mello

Fatec Campinas

2024

Roteiro

Resumo

	Objetos	Herança	Métodos	Atributos
Interface	Não pode ter instâncias	Pode ser implementada (**implements**)	Somente assinatura dos métodos	Somente constantes
Classe Abstrata	Não pode ter instâncias	Pode ser estendida (**extends**)	Métodos concretos e abstratos	Constantes e atributos
Classe Final	Pode ter instâncias	Não pode ser estendida	Somente métodos concretos	Constantes e atributos

Superclasse Object

Ancestral mais básico em Java e toda classe estende a classe Object
Principais métodos:

- `Class getClass()` retorna um objeto Class que contém informações sobre o objeto instânciado (ex.: `getName()`)
- `boolean equals(Object obj)` compara se 2 objetos apontam para a mesma região de memória
- `Object clone()` copia um objeto na memória (clone)
- `String toString()` retorna uma String que representa o valor do objeto

Genéricos

- Permite a definição de tipo quando da declaração da classe
- Especificado pelos sinas <>
- Utilizada em classes que representam coleções
- Exemplo de um vetor genérico de Strings:

```
ArrayList<String> l = new ArrayList<String>();
l.add("teste");
String item = l.get(0);
```

Ao invés de (com *casting*):

```
ArrayList l = new Arraylist();
l.add("teste");
String item = (String)l.get(0);
```

Classes com genéricos

Definindo um tipo genérico para a classe:

```
public class classExemplo<A> {
   private A var1;
   ...
   public A getVar1() {
      return var1;
   }
}
```

O tipo de A deve ser definido na instanciação:

```
classExemplo<String> obj1 = new classExemplo<String>();
classExemplo<Integer> obj2 = new classExemplo<Integer>();

String item = obj1.getVar1(); // não precisa de casting
```

Exemplo1: sem genérico

```
...
Pessoa p1 = new Pessoa();
p1.setNome("Jonas");
Pessoa p2 = new Pessoa();
p2.setNome("Maria");

List lista = new ArrayList();
lista.add(p1);
lista.add(p2);

Iterator iterator = lista.iterator();
while (iterator.hasNext()) {
   Pessoa pessoa = (Pessoa) iterator.next();
   System.out.println("Nome: " + pessoa.getNome());
}
```

Exemplo2: com genérico

```
...
Pessoa p1 = new Pessoa();
p1.setNome("Jonas");
Pessoa p2 = new Pessoa();
p2.setNome("Maria");

List<Pessoa> lista = new ArrayList<>();
lista.add(p1);
lista.add(p2);

Iterator<Pessoa> iterator = lista.iterator();
while (iterator.hasNext()) {
   Pessoa pessoa = iterator.next();
   System.out.println("Nome: " + pessoa.getNome());
}
```

Exemplo3: com foreach

```
...
Pessoa p1 = new Pessoa();
p1.setNome("Jonas");
Pessoa p2 = new Pessoa();
p2.setNome("Maria");

List<Pessoa> lista = new ArrayList<>();
lista.add(p1);
lista.add(p2);

Iterator<Pessoa> iterator = lista.iterator();
for (Pessoa p : lista) {
   System.out.println(p.getNome());
}
```

Exercício

Crie uma classe `Produto` com os atributos `codigo`, `descricao` e `preco`, sendo `codigo` um tipo genérico e os outros dois `String` e `double`, respectivamente.

Adicione um construtor, os métodos get e sobreescreva o método `toString()`

Exercício - solução

```
public class Produto<T> {
    private T codigo;
    private String descricao;
    private double preco;

    public Produto(T cod,String descr, double pr) {
        codigo = cod;
        descricao = descr;
        preco = pr;
    }

    public T getCodigo() { return codigo; }
    public String getDescricao() { return descricao; }
    public double getPreco() { return preco; }

    @Override
    public String toString() {
        return "Produto{" + "codigo=" + codigo + ", descricao=" +
        descricao + ", preco=" + preco + "}";
     }
}
```