Lição 11

Herança, polimorfismo e interfaces

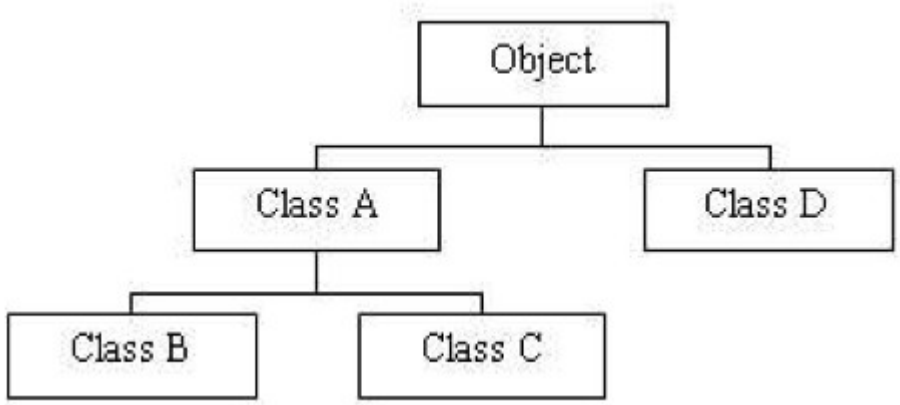
Objetivos

Ao final da lição, o estudante deverá estar apto a:

- Definir superclasses e subclasses
- Criar Override de métodos de superclasses
- Criar métodos e classes final

Herança

- Em Java, todas as classes, incluindo as que formam a API Java, são subclasses da classe **Object**



- Superclasse
  - Qualquer classe acima de uma classe específica na hierarquia de classes
- Subclasse
  - Qualquer classe abaixo de uma classe específica na hierarquia de classes

Herança

- Uma vez que um comportamento (método) é definido em uma superclasse, este comportamento é automaticamente herdado por todas as subclasses
- Permite codificar um método apenas uma única vez e este pode ser usado por todas as subclasses
- Uma subclasse necessita apenas implementar as diferenças entre ela própria e sua classe pai.

Herança

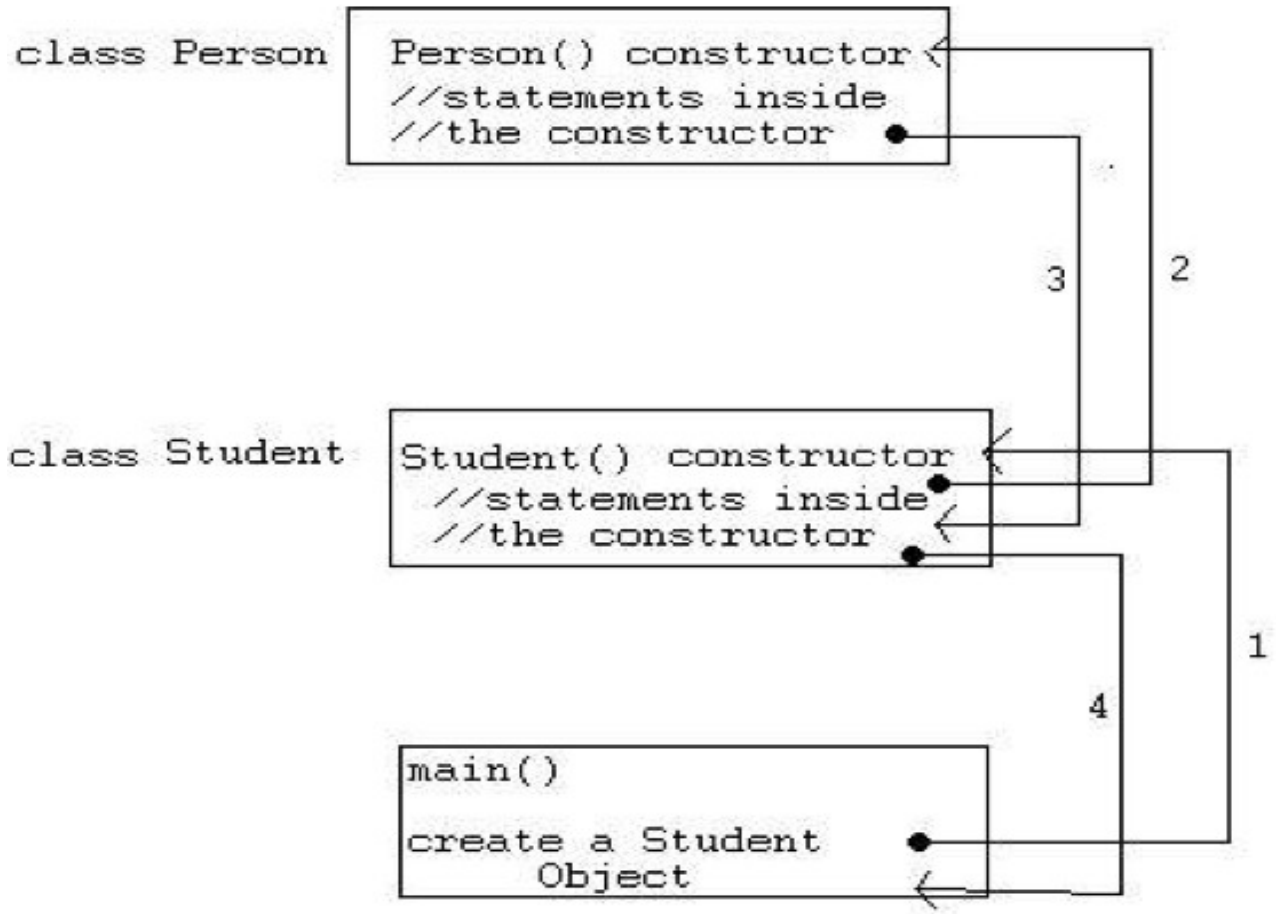
```
class Person {
   protected String name;
   protected String address;

   /**
    * Construtor padrão
    */
   public Person(){
      System.out.println(
         "Inside Person:Constructor");
      name = ""; address = "";
   }
}
```

Herança

```
class Student extends Person {
   public Student() {
      System.out.println(
         "Inside Student:Constructor");
   }
}
```

Herança

- O fluxo do programa é mostrado abaixo



super

- Uma subclasse pode também, **explicitamente,** chamar um construtor de sua superclasse imediata
- Feito através da chamada do construtor **super**
- Uma chamada a um construtor **super** no construtor de uma subclasse resultará na execução do construtor referente da superclasse, baseado nos argumentos passados

```
public Student() {
    super("SomeName", "SomeAddress");
    System.out.println("Inside Student:Constructor");
}
```

super

- Utilizado para se referir a membros da superclasse (assim como o objeto **this**)

```
public Student() {
    super.name = "Person name";
    this.name = "Student name";
}
```

Override de métodos

- Se, por alguma razão, uma classe derivada necessita ter uma implementação diferente de um certo método da superclasse, realizar **override de métodos** pode ser muito útil

- Uma subclasse pode **override de métodos** definido em sua superclasse provendo uma nova implementação para aquele método

Exemplo

```
class Person {
   public String getName() {
      System.out.println("Parent: getName");
      return name;
   }
}

class Student extends Person {
   public String getName() {
      System.out.println("Student: getName");
      return name;
   }
}
```

Classe Final

- Classes que não podem ter subclasses

```
<modificador>* final class <nomeClasse> {
    ...
}
```

Método Final

- Métodos que não podem ser modificados por polimorfismo de override

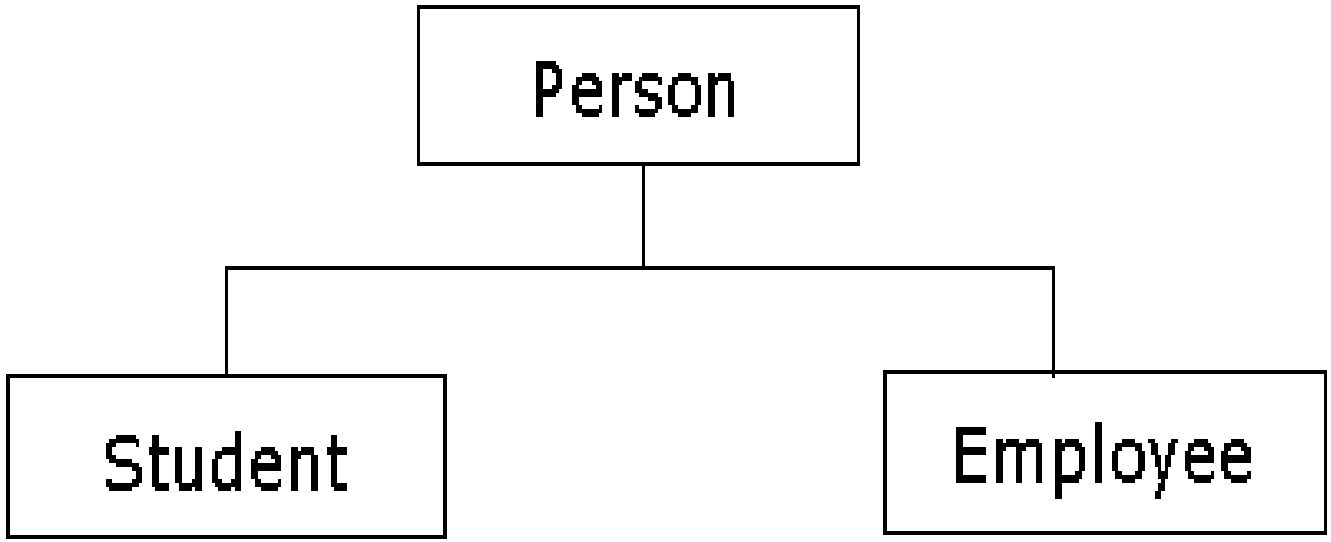
```
<modificador>* final <tipoRetorno> <nomeMétodo>(
      <argumentos>*) {
   ...
}
```

Polimorfismo

- Propriedade que permite a uma referência mudar o comportamento de acordo com o que o objeto está esperando

- Habilita múltiplos objetos de diferentes subclasses a serem tratados com objetos de uma única superclasse enquanto seleciona automaticamente os métodos adequados para aplicar em um objeto particular baseado em subclasses às quais pertence

Polimorfismo



Polimorfismo

```
public static main(String[] args) {

   Person  ref;
   Student studentObject = new Student();
   Employee employeeObject = new Employee();

   ref = studentObject;
}
```

Polimorfismo

```
class Student {
   public String getName(){
      System.out.println("Student Name:" + name);
      return name;
   }
}

class Employee {
   public String getName(){
      System.out.println("Employee Name:" + name);
      return name;
   }
}
```

Polimorfismo

```
public static main(String[] args) {
      Person ref;
      Student studentObject = new Student();
      Employee employeeObject = new Employee();
      ref = studentObject;
      String temp = ref.getName();
      System.out.println(temp);
      ref = employeeObject;
      String temp = ref.getName();
      System.out.println(temp);
}
```

Classes Abstratas

- É uma classe que não pode ser instanciada
- Freqüentemente aparece no topo de uma hierarquia de classes de programação orientada a objetos
- Define vários tipos de ações possíveis com os objetos de todas as subclasses desta classe

Métodos Abstratos

- Métodos criados classes abstratas sem implementação
- Para criar um método abstrato, escreva a declaração de método sem o corpo e use a palavra-chave **abstract**

```
<modificador>* abstract <tipoRetorno> <nomeMetodo>(
    <argumento>*);
```

Classe Abstrata de Exemplo

```
public abstract class LivingThing {
   public void breath(){
      System.out.println("Living Thing breathing...");
   }

   public void eat(){
      System.out.println("Living Thing eating...");
   }


   public abstract void walk();
}
```

Classes Abstratas

```
public class Human extends LivingThing {
    public void walk(){
        System.out.println("Human walks...");
    }
}
```

Interfaces

- É um tipo especial de classe contendo métodos abstratos e atributos finais
- Define um meio público e padrão de especificar o comportamento das classes
- Habilita classes, independentemente de sua posição na hierarquia de classes, para implementar comportamentos comuns

Porque utilizar Interfaces?

- Para ter métodos similares em classes não relacionadas
- Para revelar a interface de programação do objeto sem revelar sua classe
- Para modelar herança múltipla, que permite a uma classe a ter mais de uma superclasse

Criando Interfaces

```
[public] [abstract] interface <NomeDaInterface> {
    <[public] [final] <tipoAtributo> <atributo> = <valorInicial>;>*
    <[public] [abstract] <retorno> <nomeMetodo>(<argumento>*); >*
}
```

Criando Interfaces

```
public interface Relation {
   boolean isGreater(Object a, Object b);
   boolean isLess(Object a, Object b);
   boolean isEqual(Object a, Object b);
}
```

Criando Interfaces

```
public class Line implements Relation {
   public double getLength(){
      // instruções
   }
   public boolean isGreater( Object a, Object b) {
      // instruções
   }
   public boolean isLess( Object a, Object b) {
      // instruções
   }
   public boolean isEqual( Object a, Object b) {
      // instruções
   }
}
```

Criando Interfaces

- Erro da falta de implementação de um método:

```
Line.java:4: Line is not abstract and does not override abstract
  method isGreater(java.lang.Object,java.lang.Object) in Relation
public class Line implements Relation
       ^
1 error
```

Interface vs. Classe Abstrata

- TODO método de interface não tem implementação e é público
- A classe abstrata pode ter métodos implementados
- Uma interface pode definir apenas atributos finais
- Interfaces não têm relacionamento de herança direta com qualquer classe em particular, elas são definidas independentemente

Interface vs. Classe

- Interfaces e classes são tipos
- Uma interface pode ser usada em lugares onde pode se usar uma classe

```
PersonInterface pi = new Person();
Person pc = new Person();
```

- Não é permitido criar instância de uma interface

```
PersonInterface pi = new PersonInterface(); //ERRO!
```

Estendendo Classes vs. Implementação de Interfaces

- Uma classe pode ESTENDER UMA superclasse e IMPLEMENTAR VÁRIAS interfaces

```
public class Person implements PersonInterface,
             LivingThing,
             WhateverInterface {

   // instruções
}
```

Herança entre Interfaces

- Interfaces não são partes da hierarquia de classe. Entretanto, interfaces podem ter relacionamentos de herança entre elas próprias

```
public interface PersonInterface {
   . . .
}

public interface StudentInterface extends
    PersonInterface {
   . . .
}
```

Sumário

- Herança (superclasse, subclasse)
- Utilizando a palavra-chave **super** para acessar campos e construtores de superclasses
- Override de Métodos
- Métodos e Classes Final
- Classes Abstratas
- Interfaces

Parceiros

- **Os seguintes parceiros tornaram JEDI possível em Língua Portuguesa:**