



Aula 7

Herança e Polimorfismo

Iniciativa Conjunta:



Com o apoio de:



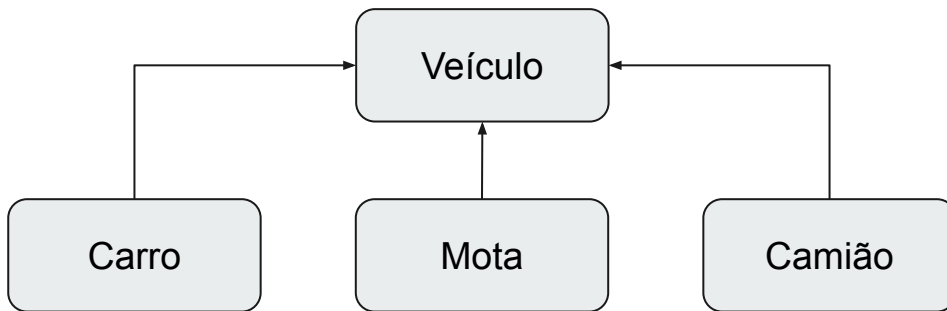


Herança

- Refere-se à capacidade no Java de uma classe herdar propriedades e funções de outra classe. Também conhecido como uma **extensão** à classe.
 - Uma classe pode **estender** outra e herdar as suas propriedades.
- Quando uma classe herda de outra classe no Java, as duas ficam categorizadas da seguinte forma:
 - A classe que **estende** (herda da outra classe) é a **subclasse**
 - A classe que **é estendida** (cuja propriedades são herdadas) é a **superclasse**

Herança

- A herança pode ser um método eficaz de partilhar código entre classes que têm características partilhadas, mas permitindo que algumas partes sejam diferentes.





Herança - Exemplo “employee”

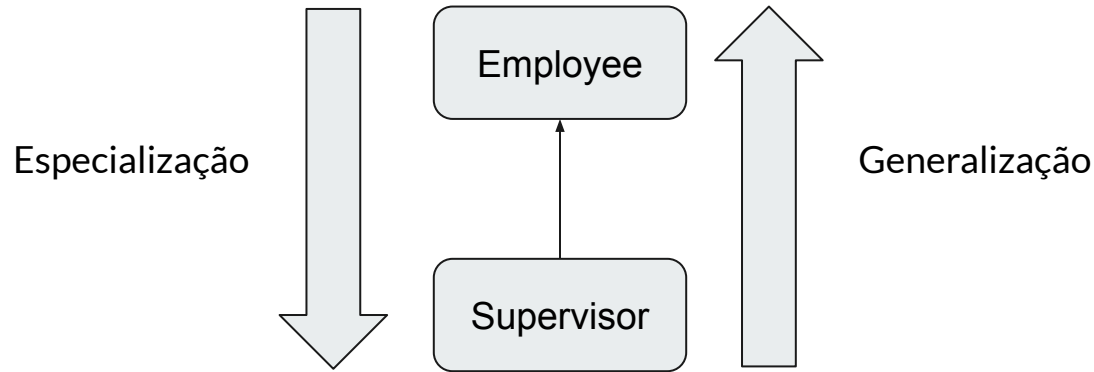
```
public class Employee {  
    private String name;  
    private String ssn;  
  
    public Employee(final String name, final String ssn) {  
        this.name = name;  
        this.ssn = ssn;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```



Herança - Exemplo “employee”

```
public String getSsn() {  
    return ssn;  
}  
  
@Override  
public String toString() {  
    return "(" + getName() + ", " + getSsn() + ")";  
}  
}
```

Relação de Generalização



- Um **Supervisor** é sempre um **Employee**
- Um **Employee** pode ser um **Supervisor**

Herança

```
public class Supervisor extends Employee {  
    private int level;
```

Um **Supervisor** é um **Employee**

```
    public Supervisor(final String name,  
        final String ssn, final int level) { ... }
```

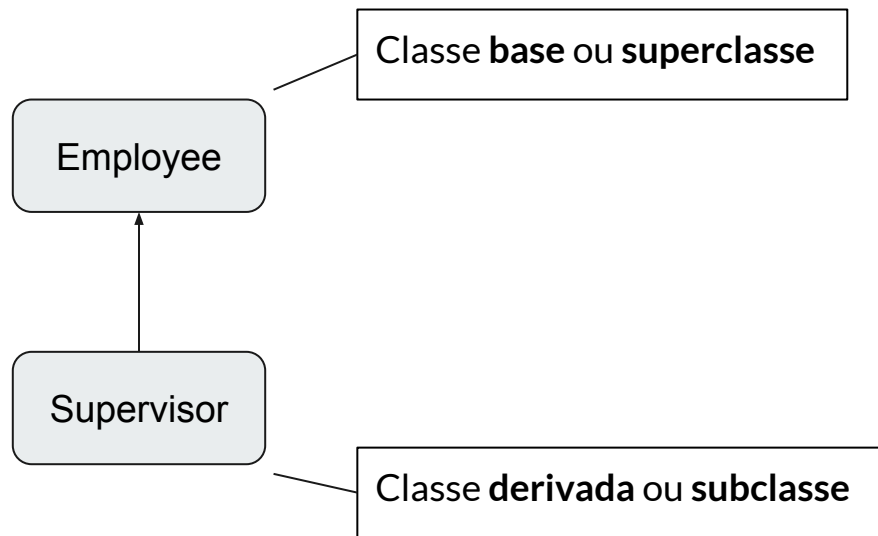
Novo método específico
ao **Supervisor**

```
    public int getLevel() {  
        return level;  
    }
```

Sobreposição do método com a mesma
assinatura da classe base

```
    @Override  
    public String toString() {  
        return "(" + getName() + ", " + getSsn() + ", " + getLevel() + ")";  
    }  
}
```

Relação





Herança

- Classe derivada **deriva** da classe base (subclasse deriva da superclasse)
- Membros são herdados e mantêm categoria de acesso
- Relação “**é um**” - Referências do tipo de classe base podem referir-se a objetos de classes derivadas
- Exemplo:

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
```

```
Employee employee = new Supervisor("Felisberto", "987654321", 5);
```



Herança

- Classe derivada **tem todas as propriedades da base**
- Exemplo:

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
```

```
Employee employee = new Supervisor("Felisberto", "987654321", 5);
```

```
String employee_ssn_id_1 = employee.getSsn();
```

```
String employee_ssn_id_2 = supervisor.getSsn();
```



Sobreposição

- Método de classe derivada pode **sobrepôr-se** a método da classe base
- Sobreposição é uma forma de **especialização**
- Regras
 - Mesma assinatura e tipo de devolução compatível
 - Método na classe base não privado e não final
 - Método na classe derivada com acessibilidade igual ou superior



Categorias de Acesso (Revisão)

- Características ou membros podem ser:
 - **private** - acesso apenas por outros membros da mesma **classe**
 - **package-private** - adicionalmente, acesso por membros do mesmo **pacote**
 - **protected** - adicionalmente, acesso por membros de **classes derivadas**
 - **public** - acesso universal



Interfaces de uma classe

- Dentro da própria classe temos acesso a:
 - Membros da classe
 - Membros não privados de classes base
- Nas classes do mesmo pacote tem-se acesso a:
 - Membros não privados da classe ou das suas bases
- Numa classe derivada:
 - Membros protegidos ou públicos da classe ou das suas bases
- Noutras classes:
 - Membros públicos da classe ou das suas bases

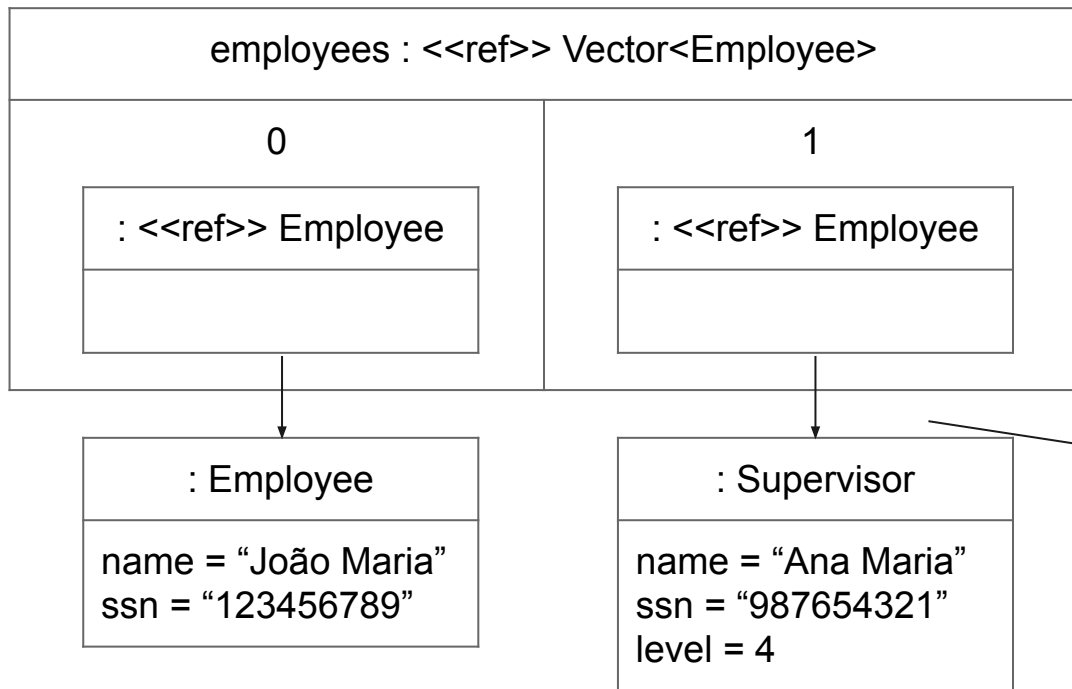


Exemplo

```
Vector<Employee> employees = new Vector<Employee>();  
  
employees.add(new Employee("João Maria", "123456789"));  
  
employees.add(new Supervisor("Ana Maria", "987654321", 4));  
  
...  
  
for (Employee employee : employees)  
    out.println(employee.toString());
```

Qual é o método
.toString() executado?

Organização



Possível porque a classe **Supervisor** deriva a classe **Employee**, ou seja, possível porque um Supervisor **é sempre também** um Employee



Resultado

- O resultado depende do tipo do objeto e não do tipo da referência!
- Isto acontece porque o método toString é **polimórfico** ou virtual

```
(João Maria, 123456789)  
(Ana Maria, 987654321, 4)
```




Polimorfismo

- Designa-se pela capacidade de um objeto tomar várias formas
 - A forma descrita pela classe a que pertence
 - As formas descritas pelas classes **acima** na hierarquia a que pertence
- Objeto pode ser referenciado por referências do tipo da classe a que pertence ou das classes acima na hierarquia (mais genéricas)



O que aparece na consola?

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
```

```
Employee anEmployee = new Supervisor("Felisberto", "987654321", 5);
```

```
Employee anotherEmployee = new Employee("Elvira", "111111111");
```

```
out.println(supervisor.toString());
```

```
out.println(anEmployee.toString());
```

```
out.println(anotherEmployee.toString());
```



O que aparece na consola?

```
Supervisor supervisor = new Supervisor("Guilhermina", "123456789", 3);
```

```
Employee anEmployee = new Supervisor("Felisberto", "987654321", 5);
```

```
Employee anotherEmployee = new Employee("Elvira", "111111111");
```

```
out.println(supervisor.toString());
```

```
out.println(anEmployee.toString());
```

```
out.println(anotherEmployee.toString());
```

(Guilhermina, 123456789, 3)

(Felisberto, 987654321, 5)

(Elvira, 111111111)



Polimorfismo - Operações e Métodos

- Uma **operação polimórfica** ou **virtual** pode ter várias implementações
- A uma implementação de uma operação chama-se **método**
- A uma operação polimórfica podem corresponder diferentes métodos, cada um em sua classe
- Todas as operações no Java são polimórficas, excepto as qualificadas com **private**
- Uma classe é polimórfica se tiver pelo menos uma operação polimórfica



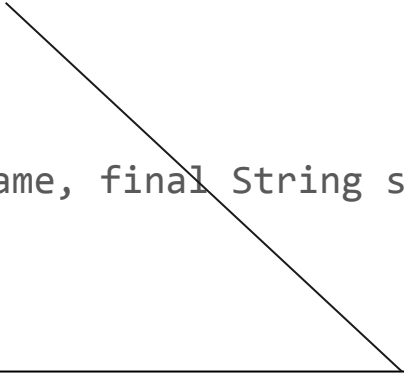
Polimorfismo - Operações e Métodos

- **Invoca-se** uma **operação** sobre um objeto de uma classe para atingir um objetivo
- Invocação de uma operação leva à execução do método apropriado, ou seja, leva à execução da implementação apropriada da operação
- Polimorfismo
 - Invocação de uma operação pode levar à execução de diferentes métodos
 - Método efectivamente executado depende da classe do objecto sobre a qual a operação é invocada
 - Método executado não depende do tipo da referência para o objeto utilizado



A classe “Object”

```
public class Employee extends Object {  
    private String name;  
    private String ssn;  
  
    public Employee(final String name, final String ssn) {  
        ...  
    }  
}
```



Se uma classe não derivar explicitamente de outra, derivará implicitamente da classe Object, que está no topo da hierarquia de classes do Java.



Métodos Finais

- Classe derivada não é obrigada a fornecer método para operação da classe base
- Classe base pode proibir às classes derivadas a sobreposição de um seu método, que se dirá ser um método **final**
- Razão para um método ser final:
 - Programador que forneceu o método na classe base entendeu que classes derivadas não deveriam poder especializar o modo de funcionamento desse método



Conceitos abstratos e concretos

- **Conceito abstrato** - Sem instâncias no domínio em causa
- **Conceito concreto** - Com instâncias no domínio em causa

Considera-se que uma classe abstrata não pode ser instanciada **logicamente**. Por exemplo no conceito de uma classe “Carro” e a classe que a estende “Peugeot 206”, não existe a possibilidade de ter um “Carro” sem que este seja especificamente de uma determinada marca/modelo.



Classes abstratas

- Uma operação com qualificador **abstract** é uma simples declaração da operação
- Uma operação sem esse qualificador inclui também a **definição** de um método correspondente
- Uma classe com uma operação abstrata tem de ser uma **classe abstrata**
- Uma classe é abstrata se tiver o qualificador **abstract**

Análise - Conceitos

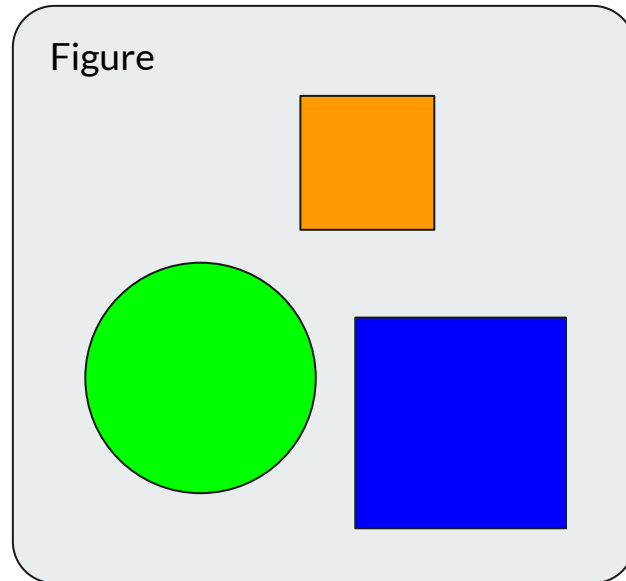
- Figura
- Forma (Abstrata)
- Círculo
- Quadrado

Figure

Shape

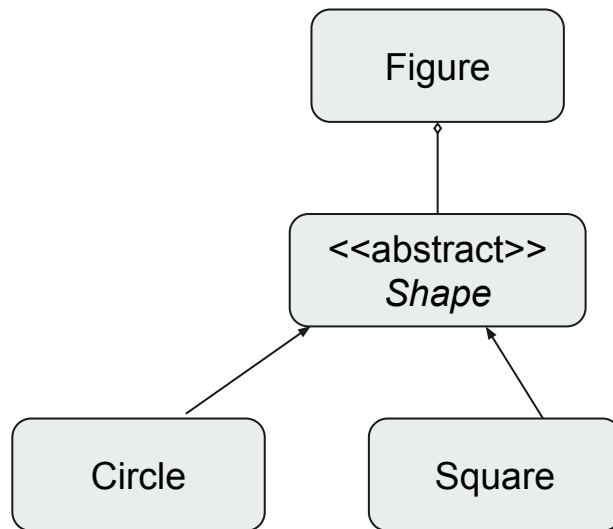
Circle

Square



Análise - Relações

- Uma Figura é composta de Formas
- Um Círculo é uma Forma
- Um Quadrado é uma Forma





Implementação

```
public class Figure {  
    private Vector<Shape> shapes;  
    ...  
}  
  
public abstract class Shape { ... }  
  
public class Circle extends Shape { ... }  
  
public class Square extends Shape { ... }
```



Implementação - *Shape*

```
public abstract class Shape {  
    private Position position;  
  
    public Shape(final Position position) {  
        this.position = position;  
    }  
  
    public final Position getPosition() {  
        return position;  
    }  
  
    public abstract double getArea();  
    public abstract double getPerimeter();  
    public abstract Box getBoundingBox();  
}
```

Operações abstractas, ou seja, operações **sem qualquer implementação** disponível até este nível da hierarquia.



Implementação - Circle

```
public class Circle extends Shape {  
    private double radius;
```

Um Circle é uma Shape e a classe Circle herda a implementação da classe Shape.

```
    public Circle(final Position position, final double radius) {  
        super(position);  
        this.radius = radius;  
    }
```

```
    public final double getRadius() {  
        return radius;  
    }  
    ...
```

É necessário apenas um atributo adicional, correspondente a uma das duas propriedades de um círculo (o raio), já que a posição do centro é herdada da classe *Shape*.

Implementação - Circle

...

```
@Override  
public double getArea() {  
    return Math.PI * getRadius() * getRadius();  
}
```

```
@Override  
public double getPerimeter() {  
    return 2.0 * Math.PI * getRadius();  
}
```

```
}
```

Qual é a área de um círculo?
Fácil: $\pi \times r^2$

Fornece-se implementações,
ou seja, métodos, para cada
uma das operações abstratas
da classe *Shape*



Exercício 1 - Empregados

Crie um programa que use uma lista de empregados para calcular o total dos salários a pagar de uma cadeia de lojas. A lista de empregados deve conter empregados (sem especialização), gerentes de loja e diretores regionais. Para cada classe de empregados, o salário é calculado da seguinte maneira:

- Empregados: valor fixo de 800€;
- Gerente de loja: valor fixo igual ao dos empregados sem especialização, acrescido de um prémio de 200€ que é atribuído sempre que a loja cumpre os objectivos das vendas.
- Diretor regional: valor fixo igual ao dobro do dos empregados sem especialização, acrescido de um prémio que corresponde a 1% do lucro mensal nas lojas da região.

Nota: para simplificar, considere que ter cumprido ou não os objectivos de vendas é um atributo dos gerentes e que o lucro mensal da região é um atributo dos diretores.



Exercício 2 - Jogadores de Futebol

Pretende-se escrever um programa para registar as estatísticas de jogadores de futebol.

- A classe (base) **Jogador** deve conter o nome e o número do jogador. Deve também ter um método para registar cada golo marcado e um inspetor para o número de golos marcados.
- A classe **GuardaRedes**, uma extensão de Jogador, que deve permitir registar e consultar o número de golos sofridos.
- A classe **JogadorDeCampo**, uma extensão de Jogador, que deve permitir registar e consultar o número de passes feitos e recebidos.

Teste criando um jogador de campo e um guarda-redes e use as funções próprias de cada classe para atribuir a cada jogador (desde que seja possível): 2 golos marcados, 3 golos sofridos, 4 passes feitos e 5 passes recebidos.

Sobreponha o método toString em ambas as classes para melhor visualizar os resultados do teste.