

Herança

Lilian Passos Scatalon
lpscatalon2@uem.br

Conceitos de POO

Abstração

Encapsulamento

Herança

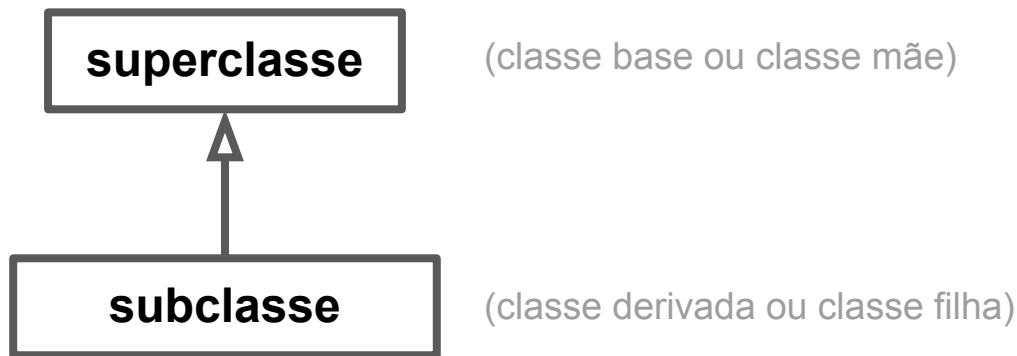
Polimorfismo

Herança

Mecanismo em que uma nova classe é criada a partir de uma classe existente

A **subclasse** (nova classe) herda os membros da **superclasse** (classe existente) e define novos membros

Promove reúso de código

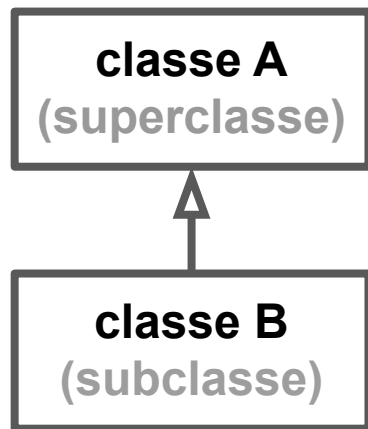


Herança - Exemplos

Classe básica	Classes derivadas
Aluno	AlunoDeGraduação, AlunoDePósGraduação
Forma	Círculo, Triângulo, Retângulo, Esfera, Cubo
Financiamento	FinanciamentoDeCarro, FinanciamentoDeReformaDeCasa, FinanciamentoDeCasaPrópria
Empregado	CorpoDocente, Funcionários
Conta	ContaCorrente, ContaPoupança

Herança

A palavra-chave **extends** estabelece a herança entre as classes



```
class B extends A {  
    .  
    . // membros herdados da classe A  
    . // novos membros da classe B  
}
```

Relacionamento é-um

A herança cria um **relacionamento é-um** (*is-a*)

A subclasse é mais específica que a superclasse

Exemplo:

Carro **é-um** veículo

Caminhão **é-um** veículo

Significa que a subclasse é um **tipo** da superclasse

```
Veiculo v;  
v = new Carro();
```

Hierarquia de classes

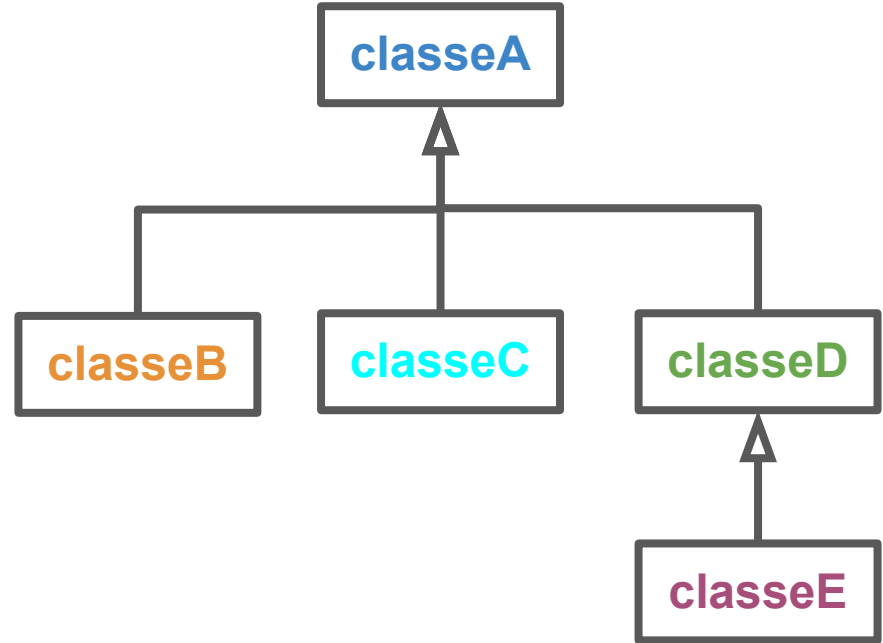
```
class classeA {  
    . . .  
}
```

```
class classeB extends classeA {  
    . . .  
}
```

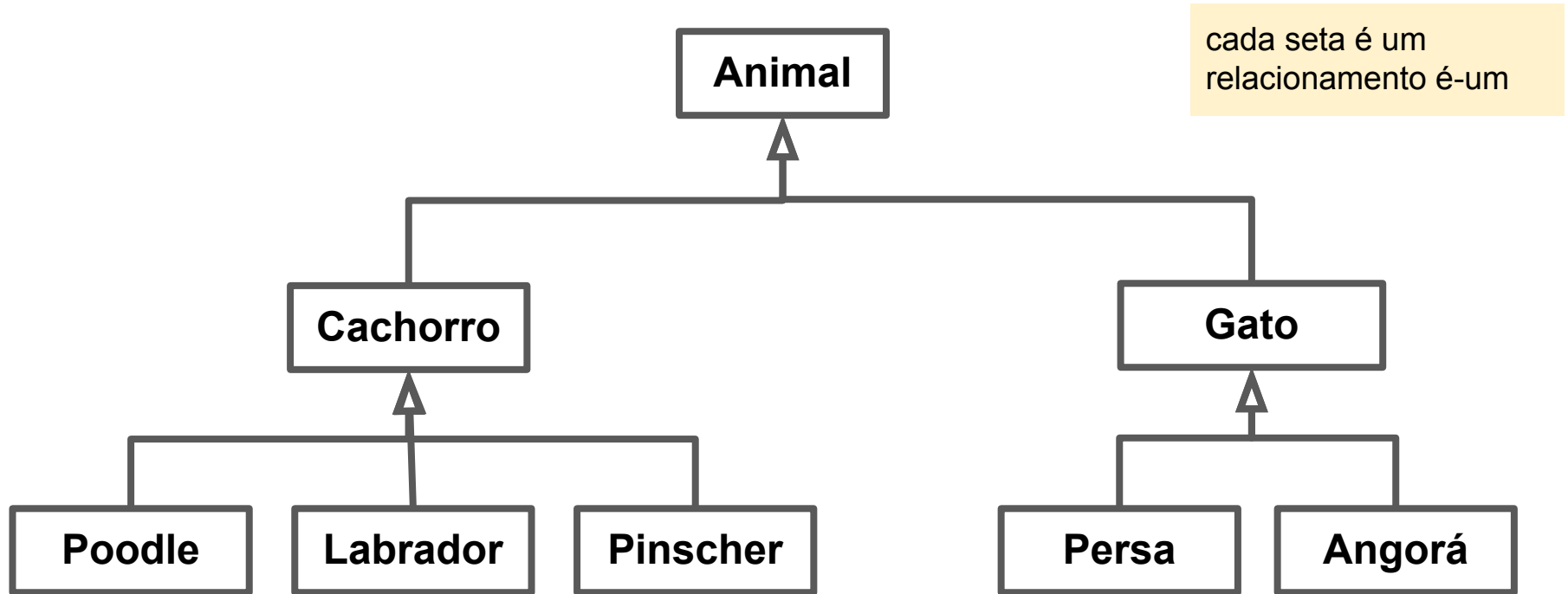
```
class classeC extends classeA {  
    . . .  
}
```

```
class classeD extends classeA {  
    . . .  
}
```

```
class classeE extends classeD {  
    . . .  
}
```



Hierarquia de classes - Exemplo



Hierarquia de classes

Relações dentro da hierarquia de classes

Classe **ancestral** (de uma determinada classe X)

Classe **descendente** (de uma determinada classe Y)

Exemplo

Animal é ancestral de Labrador

Poodle é descendente de Cachorro

Relacionamentos é-um e tem-um

Relacionamento é-um (is-a): uma classe herda de outra

Exemplo: Cachorro é-um Animal

```
class Cachorro extends Animal {  
    . . .  
}
```

Relacionamento tem-um (has-a): uma classe tem um atributo que é referência a um objeto de outra classe

Exemplo: Pessoa tem-um Animal

```
class Pessoa {  
    Animal pet;  
    . . .  
}
```

Herança

Uma subclasse herda os atributos e os métodos de sua superclasse, além de incluir os seus próprios

```
class Veiculo {  
    int numeroRegistro;  
    Pessoa proprietario;  
    void transferirPropriedade(Pessoa novoProprietario) {  
        . . .  
    }  
    . . .  
}
```

```
class Carro extends Veiculo {  
    int numeroPortas;  
    . . .  
}
```

```
class Caminhao extends Veiculo{  
    int numeroEixos;  
    . . .  
}
```

Membros protegidos

O modificador **protected** restringe o acesso às subclasses (e às classes do mesmo pacote)

```
class Veiculo {  
    protected int numeroRegistro;  
    protected Pessoa proprietario;  
  
    public void transferirPropriedade(Pessoa novoProprietario) {  
        . . .  
    }  
    . . .  
}
```

```
class Carro extends Veiculo {  
    private int numeroPortas;  
    . . .  
}
```

```
class Caminhao extends Veiculo{  
    private int numeroEixos;  
    . . .  
}
```

Herança - modificadores de acesso

```
class A{  
    public int x;  
    protected int y;  
    private int z;  
  
}
```

```
class B extends A{  
    // herda atributos x e y  
    // não herda atributo z  
  
}
```

Modificadores de acesso

	Dentro da mesma classe	Em uma classe derivada	Fora da classe
Private	Sim	Não	Não
Protected	Sim	Sim	Não
Public	Sim	Sim	Sim

Subclasse - Atributos

```
public class Funcionario {  
    protected String nome;  
    protected double salario;  
    ...  
}  
  
public class Gerente extends Funcionario {  
    // atributos herdados:  
    //     String nome;  
    //     double salario;  
  
    private double bonus;  
    ...  
}
```

A subclasse herda os atributos da superclasse e define novos atributos

Subclasse - Métodos

```
public class Funcionario {  
    ...  
  
    public void aumentaSalario(double porcentagem) {  
        double aumento = salario * porcentagem / 100;  
        salario += aumento;  
    }  
}  
  
public class Gerente extends Funcionario {  
    ...  
    // métodos herdados:  
    //     void aumentaSalario(double porcentagem)  
  
    public void setBonus(double bonus) {  
        this.bonus = bonus;  
    }  
}
```

A subclasse herda os métodos da superclasse e define novos métodos

Subclasse - Métodos

```
public class Funcionario {  
  
    ...  
  
    public double getSalario() {  
        return salario;  
    }  
  
}  
  
public class Gerente extends Funcionario {  
  
    ...  
  
    public double getSalario() {  
        return salario + bonus;  
    }  
  
}
```

A subclasse pode redefinir/sobrescrever métodos herdados

Subclasse - Construtor

```
public class Funcionario {  
  
    protected String nome;  
    protected double salario;  
  
    public Funcionario(String nome, double salario) {  
        this.nome = nome;  
        this.salario = salario;  
    }  
}
```

Os construtores da superclasse não são herdados...

```
public class Gerente extends Funcionario {  
  
    private double bonus;  
  
    public Gerente(String nome, double salario) {  
        super(nome, salario);  
        bonus = 0;  
    }  
}
```

... mas podem ser invocados com super

O construtor da subclasse inicializa os atributos definidos na subclasse

Referência super

super se refere à superclasse

Geralmente usada para invocar o construtor da superclasse

Deve ser o primeiro comando no construtor da subclasse

```
public Carro(int numeroPortas, int numeroRegistro){  
    super(numeroRegistro);  
    this.numeroPortas = numeroPortas;  
}
```

Métodos sobrescritos

Uma subclasse pode sobrescrever (*override*) um método herdado

O método sobrescrito deve ter a mesma assinatura, mas com o corpo redefinido

```
public class Forma {  
    double calculaArea(){  
        ...  
    }  
}
```

```
public class Triangulo extends Forma {  
    double calculaArea(){  
        return (base*altura)/2;  
    }  
}
```

```
public class Quadrado extends Forma {  
    double calculaArea(){  
        return lado*lado;  
    }  
}
```

Métodos sobrescritos


O método na superclasse pode ser invocado usando a referência `super`

Se um método for declarado com o modificador `final`, ele não pode ser sobrescrito

Métodos sobrescritos

Quando um método é sobrescrito, é preciso ter cuidado para manter os tipos dos parâmetros

```
public class Gerente extends Funcionario {  
    // ...  
    @Override  
    public boolean trabalhaPara(Funcionario supervisor){  
        //...  
    }  
  
    public boolean trabalhaPara(Gerente supervisor){  
        //...  
    }  
}
```



outro método distinto com mesmo nome,
mas assinatura diferente

Classe Object

Toda classe em Java herda direta ou indiretamente de Object

Quando uma classe não tem superclasse explícita, ela herda implicitamente de Object

```
public class Funcionario { ... }
```

é equivalente a

```
public class Funcionario extends Object { ... }
```

Classe Object

A classe Object define métodos que são aplicáveis a qualquer objeto Java

Esses métodos possuem uma implementação *default*, mas alguns deles podem ser redefinidos (sobrescritos)

Método	Descrição
<code>String toString()</code>	Fornece uma representação string o objeto, por default o nome da classe e o hash code
<code>boolean equals(Object other)</code>	Retorna true se o objeto deve ser considerado igual ao outro (parâmetro other), retorna false se other é null ou diferente de other
<code>int hashCode()</code>	Produz um hash code para o objeto. Objetos iguais devem ter o mesmo hash code. A menos que o método seja sobrescrito, o hash code é gerado de algum modo pela JVM.
<code>Class<?> getClass()</code>	Retorna um objeto Class, que descreve a classe à qual o objeto pertence
<code>protected Object clone()</code>	Faz uma cópia do objeto
<code>protected void finalize()</code>	Este método é chamado quando o objeto é reivindicado pelo coletor de lixo da JVM. Não se deve sobrescrevê-lo.
<code>wait, notify, notifyall</code>	Métodos relacionados a programação concorrente (threads)

Classe Object - método toString

Retorna uma descrição do objeto no formato de string

Muitos métodos toString seguem o formato: nome da classe, seguido dos atributos dentro de colchetes

```
public String toString() {  
    return getClass().getName() +  
        "[nome=" + nome + ",salário=" + salario + "];"  
}
```

Classe Object - método toString

Em uma subclasse, geralmente é feita a chamada ao método toString da superclasse

```
public class Gerente extends Funcionario {  
    public String toString() {  
        return super.toString() + "[bonus=" + bonus + "];"  
    }  
}
```

Classe Object - método toString

Sempre que um objeto é concatenado com uma string, o compilador Java automaticamente invoca o método toString do objeto

```
Ponto p = new Ponto(10, 20);  
String mensagem = "A posição atual é " + p; // p.toString()
```

Classe Object - método equals

Testa se um objeto é considerado igual ao outro

Do jeito que está implementado na classe Object, determina se as duas referências dos objetos são idênticas

É preciso sobrescrever o método para testar a igualdade com base no conteúdo dos objetos (valores dos atributos)

Por exemplo, a classe String sobrescreve equals para checar se duas strings são compostas pelos mesmos caracteres

Classe Object - método equals

```
public class Item {  
    private String descricao;  
    private double preco;  
    //...  
    public boolean equals(Object other) {  
  
        // Verifica se os objetos são idênticos  
        if (this == other) return true;  
  
        // Deve retornar false se o outro objeto é null  
        if (other == null) return false;  
        // Verifica se other é um Item  
        if (getClass() != other.getClass()) return false;  
  
        // Verifica se os atributos têm valores idênticos  
        Item outro = (Item) other;  
        if (descricao.equals(outro.getDescricao()) && preco == outro.getPreco())  
  
        // public int hashCode() { ... }  
    }  
}
```

Classe Object - método hashCode

Um hash code é um inteiro derivado de um objeto

Os métodos hashCode e equals devem ser compatíveis!

Se `x.equals(y)` for verdadeiro, então `x.hashCode() == y.hashCode()` também deve ser

O método `Object.hashCode` retorna um valor gerado pela JVM

Classe Object - método hashCode

Por exemplo, a classe String usa o seguinte algoritmo para computar o hash code:

```
int hash = 0;  
for (int i = 0; i < length(); i++)  
    hash = 31 * hash + charAt(i);
```

Ou seja, strings com os mesmos caracteres produzem o mesmo hash code

Classe Object - método hashCode

Se você redefinir o método equals, também precisa redefinir o método hashCode para que ambos os métodos sejam compatíveis

Uma possibilidade é combinar os hash codes dos atributos

```
class Item {  
    //...  
    public int hashCode() {  
        return descricao.hashCode() + preco;  
    }  
}
```

Classe Class

Em Java é possível descobrir em tempo de execução a classe à qual um determinado objeto pertence

```
Object obj = ...;  
Class<?> cl = obj.getClass();  
System.out.println("Este objeto é uma instância de " + cl.getName());
```

Bibliografia

H. Schildt. **Java: A Beginner's Guide**. McGraw-Hill Education, 8th edition, 2018.

C. S. Horstmann. **Core Java SE 9 for the Impatient**. Addison-Wesley, 2nd Edition, 2018.

Caelum. Apostila **Java e Orientação a Objetos**. Disponível em <https://www.alura.com.br/apostila-java-orientacao-objetos>