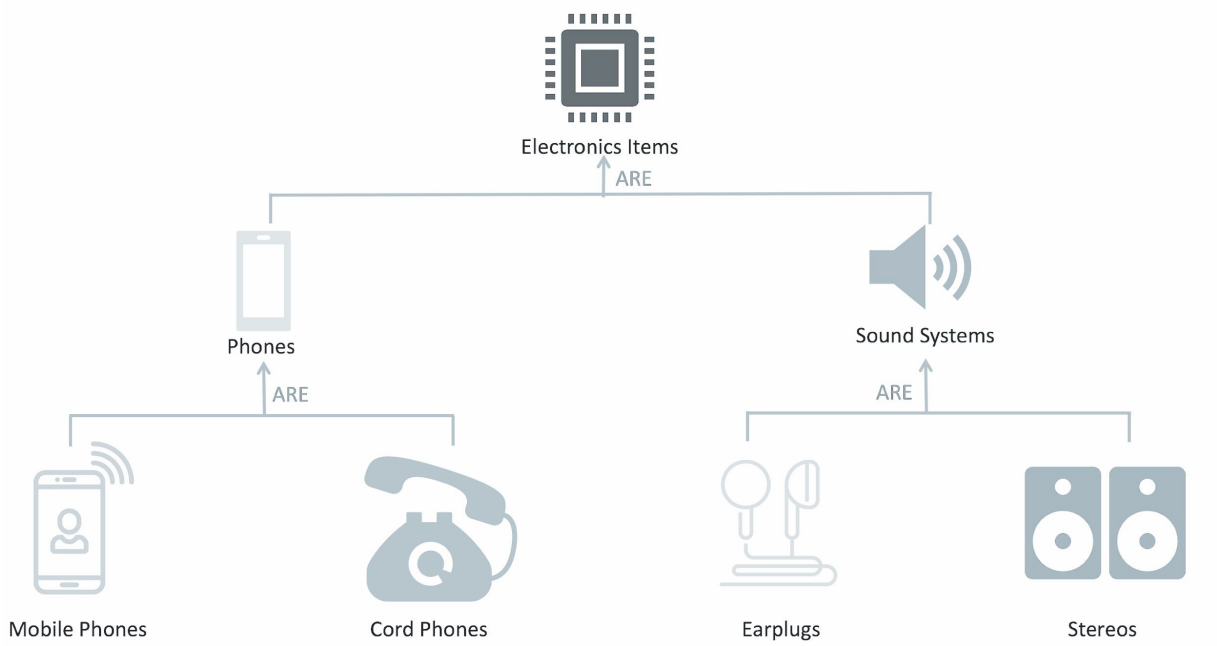


Aula 9
Herança

Rafael Geraldeli Rossi

Introdução

- Definição Jurídica [Wikipedia, 2015]: (do latim *haerentia*) é o conjunto de princípios jurídicos que disciplinam a transmissão do patrimônio (bens, direitos e obrigações), de uma pessoa que morreu, a seus sucessores legais.

Introdução

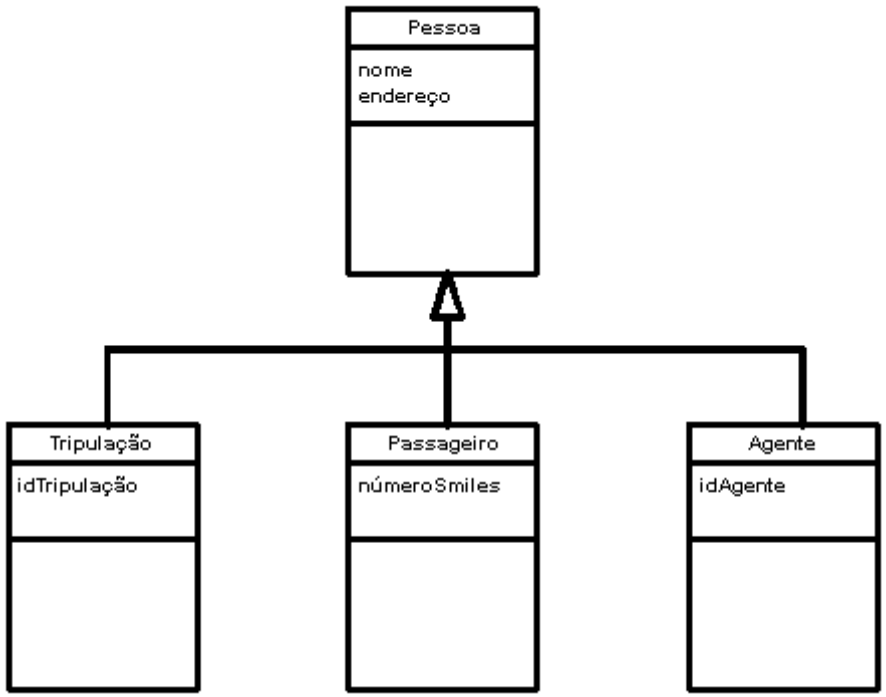
- Definição genética [Wikipedia, 2017a]: é processo pelo qual um organismo ou célula adquire ou torna-se predisposto a adquirir características semelhantes à do organismo ou célula que o gerou

Introdução

- Definição genética [Wikipedia, 2017a]: é processo pelo qual um organismo ou célula adquire ou torna-se predisposto a adquirir características semelhantes à do organismo ou célula que o gerou

Introdução

- Definição POO [Wikipedia, 2017b]: Herança é um princípio de orientação a objetos, que permite que classes **compartilhem atributos e métodos** através de "heranças".



Introdução

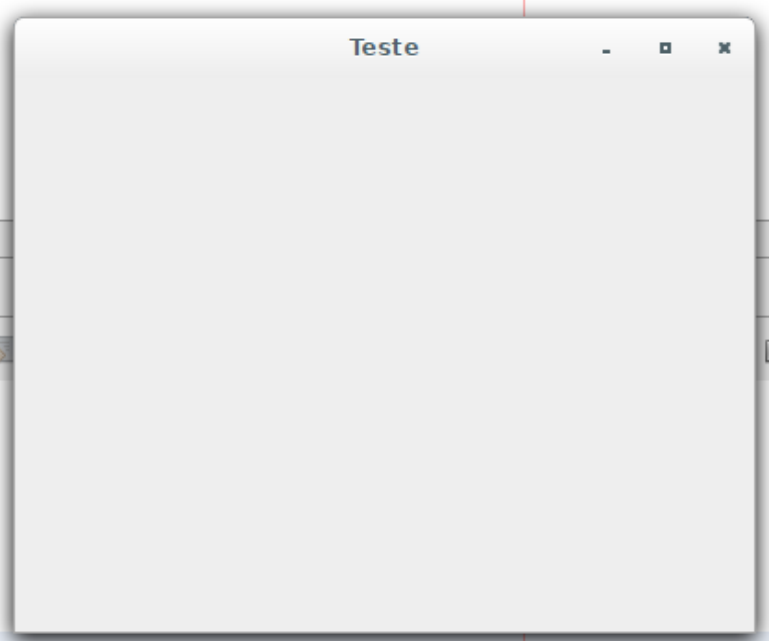
- Em uma definição padrão e básica, **HERANÇA** é uma forma de reutilização de software em que
  - **Uma nova classe é criada absorvendo membros de uma classe existente**
  - **A nova classe é aprimorada com capacidades novas ou modificadas de acordo com as características da nova classe**
- **Aumenta a velocidade de desenvolvimento**
- **Garante a qualidade das novas classes com base na qualidade das classes já existentes**

Introdução

- A ideia de herança é **SIMPLES e PODEROSA**:
  - Quando você quer criar uma nova classe e já existe uma classe que inclui algum código que você quer, você pode derivar sua nova classe a partir da classe existente
  - Ao fazer isso, você pode reusar os campos e métodos existentes da uma classe existente sem ter que escrevê-la

Introdução

```
6   package teste;
7
9   import java.util.Scanner;
10
11  public class Teste {
12
13      public static void main(String[] args) {
14
15          TesteFrame frame = new TesteFrame();
16          frame.setTitle("Teste");
17
18
19      }
20
21  }
```



teste.Teste

TesteFrame.java

Código-Fonte | Projeto | Histórico

```
1   package teste;
2
3   import javax.swing.JFrame;
4
5   public class TesteFrame extends JFrame {
6
7       public TesteFrame() {
8           initComponents();
9           this.setVisible(true);
10      }
```

Conceitos

- Ao criar uma classe, em vez de declarar membros completamente novos, pode-se designar que a nova classe herde membros de uma classe existente
- A classe existente (que passa os membros para a nova classe) é chamada de **SUPERCLASSE**, **classe básica** ou **classe pai**
- A nova classe (que recebe ou herda os membros de uma classe existente) é chamada de **SUBCLASSE**, **classe derivada** ou **classe filha**

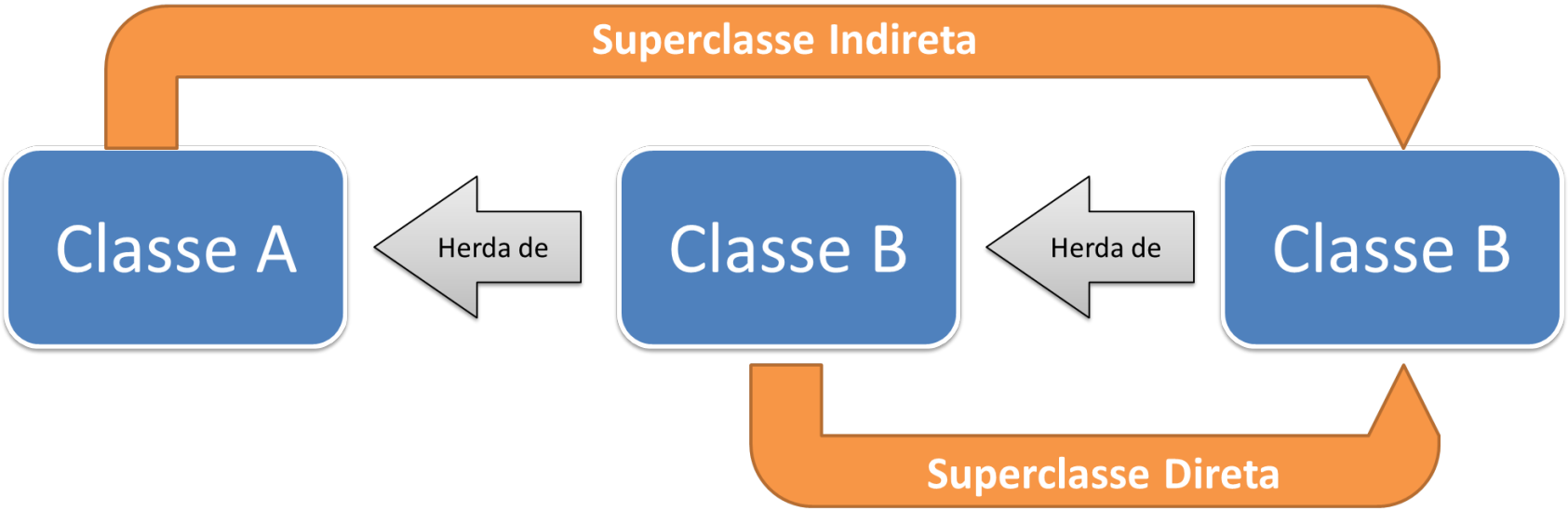
Conceitos

- Uma **subclasse pode adicionar seus próprios campos e métodos**
- **Portanto, uma subclasse é mais específica que sua superclasse e presenta um grupo mais especializado de objetos**
- A subclasse expõe comportamentos da sua superclasse e pode adicionar comportamentos que são mais específicos à subclasse

Conceitos

- É por isso que a herança é conhecida também como **especialização**
- Declarar uma subclasse não afeta o código-fonte da sua superclasse $\rightarrow$ a herança preserva a integridade da superclasse
- **OBSERVAÇÃO:** Uma subclasse, pode, sem problema, se tornar uma superclasse para outras subclasses

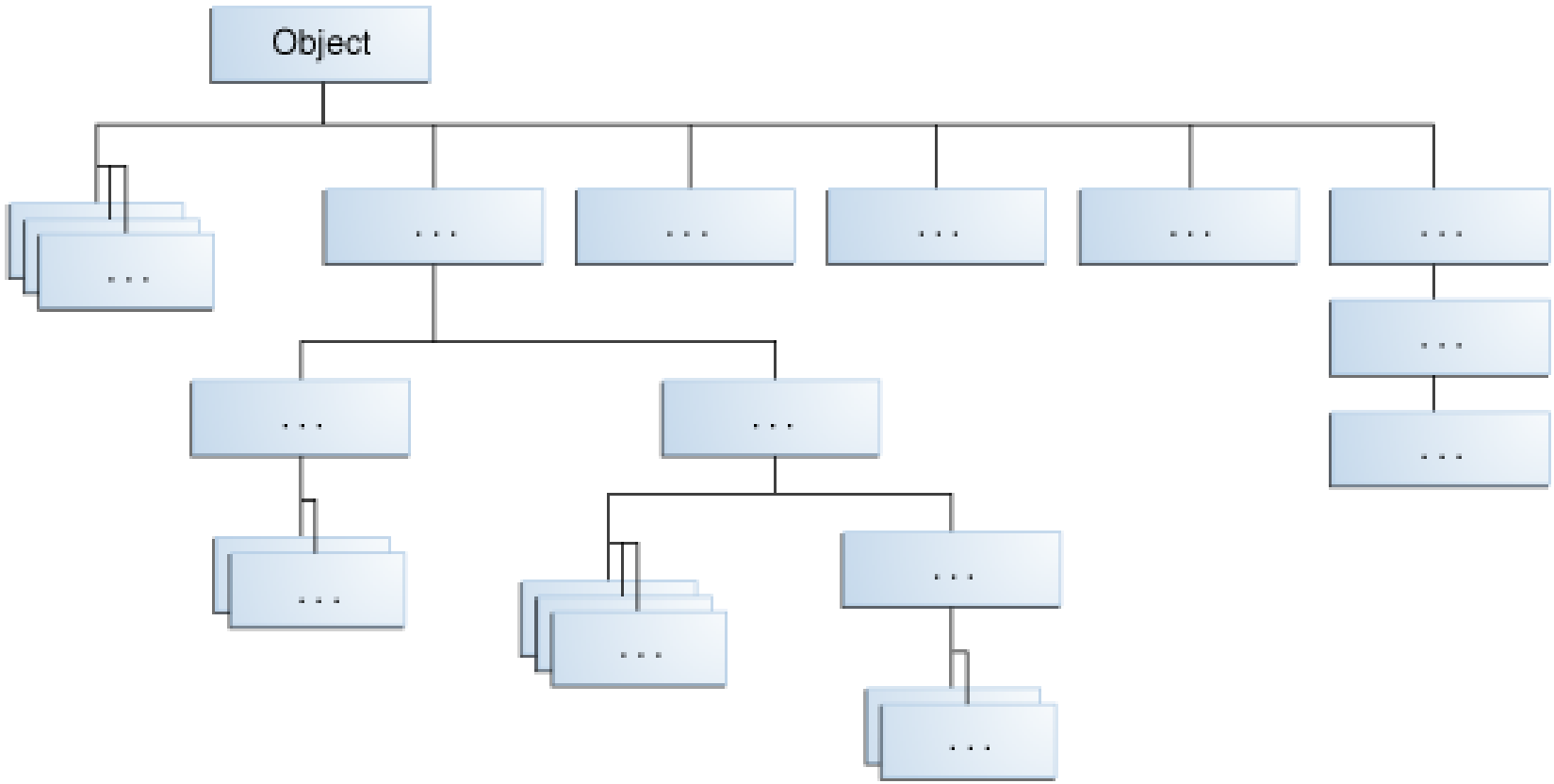
Conceitos



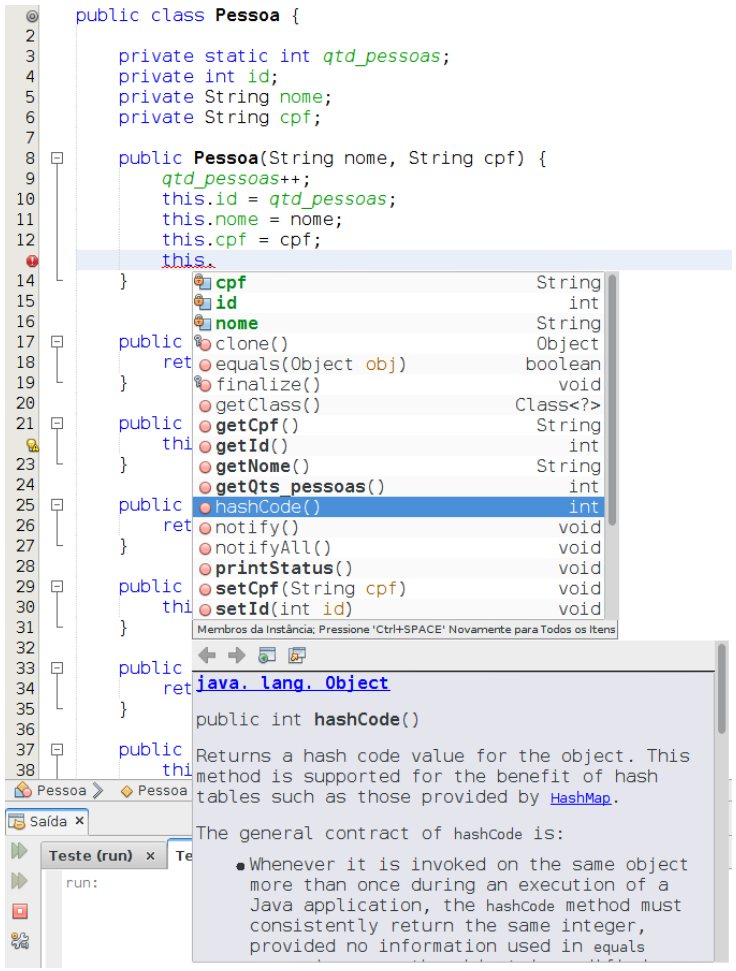
- Uma **superclasse direta** é a superclasse a partir da qual a subclasse herda explicitamente
- Uma **superclasse indireta** é qualquer superclasse acima da classe direta na hierarquia de classes

Conceitos

OSERVAÇÃO: no java todos herdam da classe **Objetc**



Conceitos


public class Pessoa {
private static int qtd_pessoas;
private int id;
private String nome;
private String cpf;
public Pessoa(String nome, String cpf) {
qtd_pessoas++;
this.id = qtd_pessoas;
this.nome = nome;
this.cpf = cpf;
this.
cpf String
id int
nome String
clone() Object
equals(Object obj) boolean
finalize() void
getClass() Class<?>
getCpf() String
getId() int
getNome() String
getQts_pessoas() int
hashCode() int
notify() void
notifyAll() void
printStatus() void
setCpf(String cpf) void
setId(int id) void
Membros da Instância; Pressione 'Ctrl+SPACE' Novamente para Todos os Itens
java.lang.Object
public int hashCode()
Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by HashMap.
The general contract of hashCode is:
Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals
Pessoa
Pessoa
Saída
Teste (run)
run:

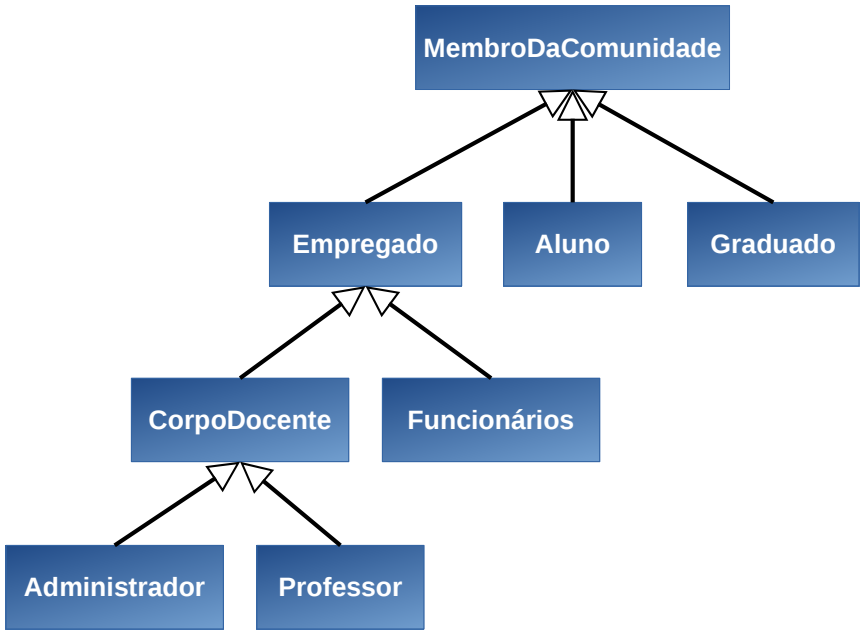
Conceitos

- Uma herança representa um relacionamento do tipo **É UM**
- Em um relacionamento É UM, **um objeto de uma subclasse também pode ser tratado como um objeto de sua superclasse**
- **Exs:**
  - Se a classe `Aluno` herda da classe `Pessoa`, um `Aluno` é uma `Pessoa`
  - Se a classe `Carro` herda da classe `Veículo`, um `Carro` é um `Veículo`

Conceitos

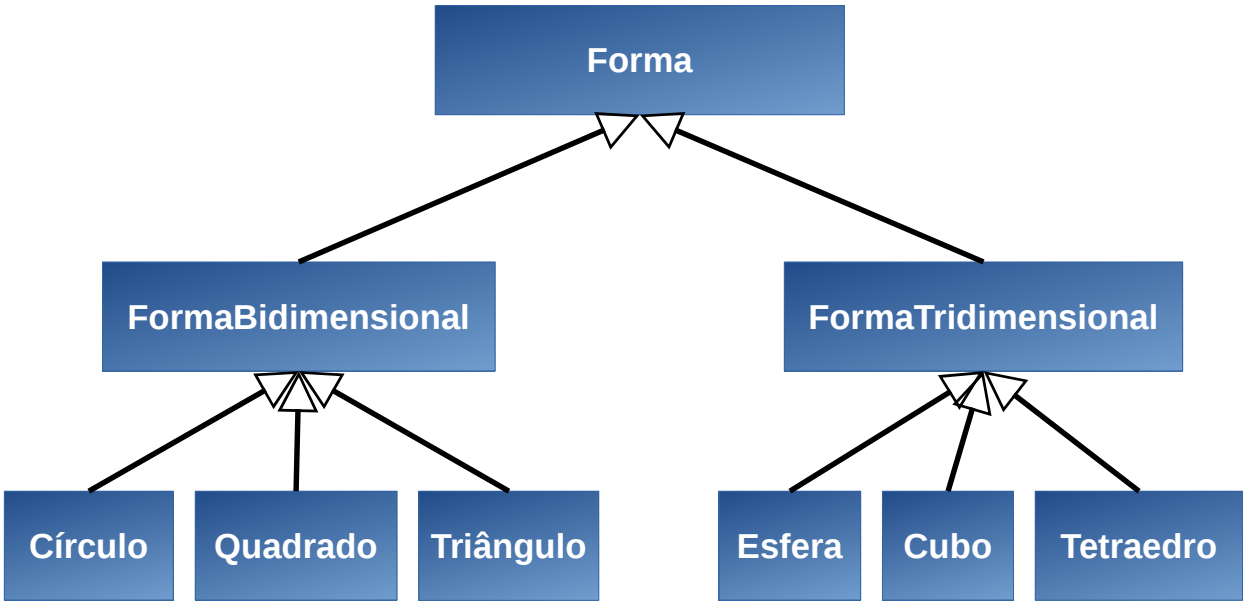
- As relações de herança formam estruturas hierárquicas parecidas com uma árvore (**hierarquia de herança**)

Exemplo de hierarquia de classes de membros de uma comunidade científica



Conceitos

Exemplo de hierarquia para os conceitos de formas



Trabalhando com Heranças

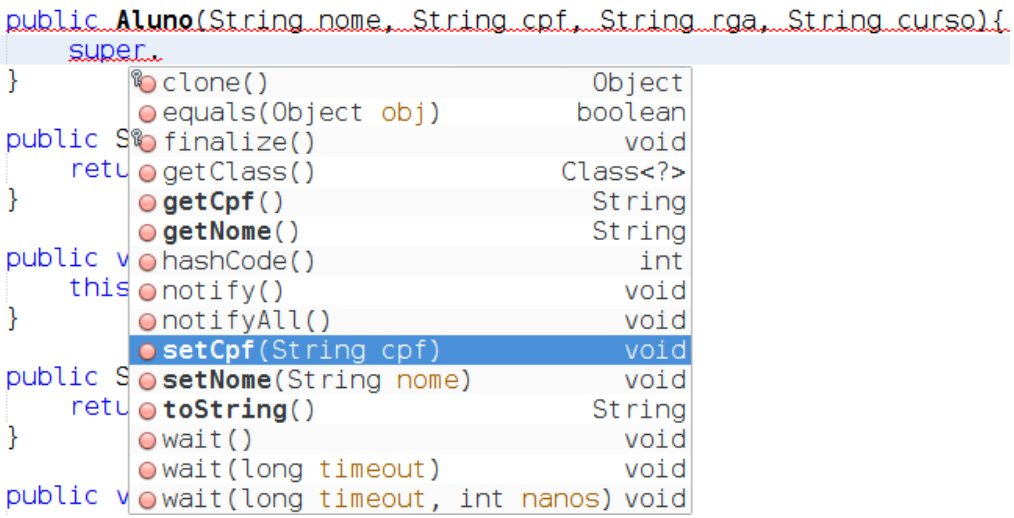
- Para fazer com que uma classe herde de outra classe (ou ainda estenda outra classe), devemos utilizar o comando `extends` na frente no nome da classe (na estrutura de declaração de classe) e em seguida o nome da classe que queremos herdar

```
public class Aluno extends Pessoa{
```

Trabalhando com Heranças

- A superclasse é **identificada** pela palavra-chave **super**
- Portando, podemos acessar membros e chamar métodos por meio de **super.[campo ou método]**
- **OBSERVAÇÃO:** pode-se acessar membros da superclasse desde estes sejam `public` ou `protected`

Trabalhando com Heranças



Trabalhando com Heranças

- **OBSERVAÇÃO 1:** todos os membros de superclasse `public` e `protected` retêm seu modificador de acesso original quando se tornam membros da subclasse

- **OBSERVAÇÃO 2:** os membros da superclasse `private` não são acessíveis fora da `própria` classe $\rightarrow$ eles permanecem ocultos nas suas subclasses e só podem ser acessados pelos métodos `public` ou `protected` herdados da superclasse

Métodos Construtores

- **OBSERVAÇÃO IMPORTANTE:**
  - Uma subclasse herda todos os membros (campos e métodos) da sua superclasse
  - **CONSTRUTORES NÃO SÃO MEMBROS**
  - Portanto, construtores não são herdados pelas subclasses
  - Porém, o construtor da superclasse pode ser invocado pela subclasse

Métodos Construtores

- **A primeira tarefa de qualquer construtor de subclasse é chamar o construtor de sua superclasse direta**, de maneira **explícita** ou **implícita** (se nenhuma chamada de construtor for especificada)
- Isso é feito de forma a assegurar que as **variáveis de instância herdadas da superclasse são inicializadas adequadamente**

  Pode-se chamar o construtor de uma superclasse da seguinte forma: `super(`*Argumentos*`)`
- **OBSERVAÇÃO:** classes chamam os construtores da classe `Object` implicitamente

Métodos Construtores

```
public Aluno(String nome, String cpf, String rga, String curso){
    super(nome,cpf);
    this.rga = rga;
    this.curso = curso;
}
```

Sobrescrita e a Anotação `@Override`

- Um problema com herança é que uma subclasse pode herdar métodos de que não necessita
- Mesmo quando um método de uma superclasse é adequado a uma subclasse, **essa subclasse precisa frequentemente de uma versão personalizada do método**
- Nesses casos, a subclasse pode **sobrescrever** (redefinir) o método de superclasse com uma implementação adequada
- A sobrescrita de um método é utilizada para "adaptar" ou para atender melhor as necessidade de uma subclasse

Sobrescrita e a Anotação `@Override`

- Por exemplo, a classe `Object` contém a classe toString() → portanto, sempre que quiser imprimir o conteúdo dos campos de um objeto que você criou e você implementa o método `toString()`, você está sobrescrevendo o método `toString()` da classe `Object`

- Para sobrescrever um método de superclasse, uma subclasse deve declarar um método com a mesma assinatura (nome do método, número de parâmetros, tipos de parâmetro e ordem dos tipos de parâmetro)

Sobrescrita e a Anotação `@Override`

- A notação `@Override` é utilizada para explicitar que uma subclasse está sobrescrevendo um método de uma superclasse
- **OBSERVAÇÃO:** a notação `@Override` **não é obrigatória**
- Quando o compilador encontra um método declarado com @Override, ele compara a assinatura do método com as assinaturas de método da superclasse
- **Se não houver uma correspondência exata, o compilador emite uma mensagem de erro** $\rightarrow$ *"method does not override or implement a method from a supertype"*

Sobrescrita e a Anotação @Override

```
1   public class Pessoa {
2
3       private static int qtd_pessoas;
4       private int id;
5       private String nome;
6       private String cpf;
7
8       public Pessoa(String nome, String cpf) {
9           qtd_pessoas++;
10          this.id = qtd_pessoas;
11          this.nome = nome;
12          thi
13      }
14
15      @Override
16      public String toString(String a){
17
18      }
```

method does not override or implement a method from a supertype
---
(Alt-Enter mostra dicas)

Conceitos

- **CURIOSIDADES/OBSERVAÇÕES:**
  - Pode-se declarar um campo em uma subclasse com o mesmo nome de um campo da superclasse → oculta o campo da superclasse (não recomendado)
  - Pode-se escrever um novo método estático na subclasse que tem a mesma assinatura da superclasse → oculta o método da superclasse

Trabalhando com Heranças

- **Os métodos de subclasse podem referir-se a membros `public` e `protected` herdados da superclasse simplesmente utilizando os nomes de membro**

- Quando um método de subclasse sobrescrever um método de superclasse herdado, **o método da superclasse pode ser acessado a partir da subclasse precedendo o nome do método de superclasse com a palavra-chave `super` e um separador de ponto (.)**

Trabalhando com Heranças

- **OBSERVAÇÃO:** Se um método realizar todas ou algumas das ações necessárias por outro método, chame esse método em vez de duplicar seu código e adicione as novas possibilidades, se for o caso e se for possível

Criando Heranças

```java
public class Pessoa {

    private String nome;
    private String cpf;

    public Pessoa(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    @Override
    public String toString(){
        return "Nome: " + this.nome + "\t CPF: " + this.cpf;
    }

}
```

```java
public class Aluno extends Pessoa{

    private String rga;
    private String curso;

    public Aluno(String nome, String cpf, String rga, String curso){
        super(nome,cpf);
        this.rga = rga;
        this.curso = curso;
    }

    public String getRga() {
        return rga;
    }

    public void setRga(String rga) {
        this.rga = rga;
    }

    public String getCurso() {
        return curso;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    @Override
    public String toString(){
        String retorno = super.toString();
        retorno += "\tRGA: " + this.rga + "\tCurso: " + this.curso;
        return retorno;
    }

}
```

```java
public class Teste {

    public int teste;

    public static void main(String args[]){

        Aluno aluno = new Aluno("Rafael", "346.XXX.XXX-XX", "46547456", "Sistemas de Informação");
        aluno.
    }
}
```

```
equals(Object obj)              boolean
getClass()                      Class<?>
getCpf()                        String
getCurso()                      String
getNome()                       String
getRga()                        String
hashCode()                      int
notify()                        void
notifyAll()                     void
setCpf(String cpf)              void
setCurso(String curso)          void
setNome(String nome)            void
setRga(String rga)              void
toString()                      String
wait()                          void
wait(long timeout)              void
wait(long timeout, int nanos)   void
```

Conceitos

- **OBSERVAÇÃO IMPORTANTE:**
  - Uma subclasse herda todos os membros (campos e métodos) da sua superclasse
  - **CONSTRUTORES NÃO SÃO MEMBROS**
  - Portanto, construtores não são herdados pelas subclasses
  - Porém, o construtor da superclasse pode ser invocado pela subclasse

Construtores

- **A primeira tarefa de qualquer construtor de subclasse é chamar o construtor de sua superclasse direta**, de maneira **explícita** ou **implícita** (se nenhuma chamada de construtor for especificada)

- Isso é feito de forma a assegurar que as **variáveis de instância herdadas da superclasse são inicializadas adequadamente**

- **OBSERVAÇÃO:** classes chamam os construtores da classe `Object` implicitamente

Construtores

- **OBSERVAÇÃO 1:** classes chamam os construtores da classe `Object` implicitamente

- **OBSERVAÇÃO 2:** sempre que uma superclasse direta tiver um construtor vazio, i.e., construtor sem argumentos, é feita a chamada desse construtor implicitamente (`super()`)

Construtores

- **Se o código não incluir uma chamada explícita para o construtor de superclasse, o Java chama implicitamente o construtor padrão ou sem argumentos da superclasse**

- De modo semelhante, se a superclasse é derivada de outra classe, como é naturalmente toda e qualquer classe, exceto a classe `Object`, o construtor de superclasse invoca o construtor da próxima classe no topo da hierarquia e assim por diante

Exemplo: Empregados Comissionados

- Vamos considerar agora um exemplo em que temos dois tipos de empregados
  - **Empregado comissionado**: recebe uma porcentagem das suas vendas
  - **Empregado comissionados com salário-base**: recebem um salário-base mais uma porcentagem das suas vendas
- Um empregado comissionado tem: **nome, sobrenome, CPF, taxa de comissão e valor total das vendas**
- Um empregado comissionado com salário-base tem: **nome, sobrenome, CPF, taxa de comissão, valor total das vendas das vendas e salário-base**

Exemplo: Empregados Comissionados

Notaram que os campos do empregado comissionado e os campos do empregado comissionado com salário base são bastante semelhantes??

- Um `empregado comissionado com salário base` nada mais é que um `empregado comissionado` + um salário base
- Portanto criar um `empregado comissionado com salario base` utilizando todo o código gerado para o `empregado comissionado` agilizaria a definição dos campos da classe uma vez que só seria preciso adicionar o campo salário base

Exemplo: Empregados Comissionados

- Todos os métodos `gets` e `sets`, e outros métodos pertinentes desenvolvidos para a classe empregado comissionado podem ser aproveitados pela classe empregado comissionado com salário base sem precisar reimplementá-los

- **CONSEQUÊNCIA DISSO:** maior agilidade no desenvolvimento

Exemplo: classe EmpregadoComissionado

```
12  public class EmpregadoComissionado {
13
14      private String primeiroNome;
15      private String segundoNome;
16      private String CPF;
17      private double totalVendas;
18      private double taxaComissao;
19
20      public EmpregadoComissionado(){
21          this.primeiroNome = "SEM_NOME";
22          this.segundoNome = "SEM_SEGUNDO_NOME";
23          this.CPF = "CPF";
24          this.totalVendas = 0.0;
25          this.taxaComissao = 0.0;
26      }
27
28      public EmpregadoComissionado(String primeiroNome, String segundoNome, String CPF, double totalVendas, double taxaComissao) {
29          this.primeiroNome = primeiroNome;
30          this.segundoNome = segundoNome;
31          this.CPF = CPF;
32          this.totalVendas = totalVendas;
33          this.taxaComissao = taxaComissao;
34      }
35
36      public double ganho(){
37          return totalVendas * taxaComissao;
38      }
39
40      @Override
41      public String toString(){
42          String saida = "Nome: " + this.primeiroNome + "\nSobrenome: " + this.segundoNome + "\nCPF: " + this.CPF + "\nTotal de Ve
43          return saida;
44      }
```

Exemplo: classe EmpregadoComissionado

```
12  public class EmpregadoComissionadoSalarioBase extends EmpregadoComissionado{
13
14      private double salarioBase;
15
16      public EmpregadoComissionadoSalarioBase(double salarioBase, String primeiroNome, String segundoNome, String CPF, double total
17          super(primeiroNome, segundoNome, CPF, totalVendas, taxaComissao);
18          this.salarioBase = salarioBase;
19      }
20
21      @Override
22      public double ganho(){
23          return salarioBase + (this.getTaxaComissao() * this.getTotalVendas());
24      }
25
26      public double getSalarioBase() {
27          return salarioBase;
28      }
29
30      public void setSalarioBase(double salarioBase) {
31          this.salarioBase = salarioBase;
32      }
33
34  }
```

Exemplo: classe EmpregadoComissionado

```
10  public class Teste {
11
12      public static void main(String[] args) {
13
14          EmpregadoComissionado empregado1 = new EmpregadoComissionado("Ricardo", "M", "6917124", 500, 0.1);
15
16          EmpregadoComissionadoSalarioBase empregado2 =
17                  new EmpregadoComissionadoSalarioBase("Rafael", "R", "007770003331", 25000, 0.2, 2000);
18
19          System.out.println("Informações empregado 1: " + empregado1.toString());
20          System.out.println("Ganho do empregado 1: " + empregado1.ganho());
21          System.out.println("================================");
22          System.out.println("Informações empregado 2: " + empregado2.toString());
23          System.out.println("Ganho do empregado 2: " + empregado2.ganho());
24
25      }
26
27  }
28
```

Localizar: Anterior Próximo

teste.Teste

Saída - Teste (run)

```
run:
Informações empregado 1: Nome: Ricardo
Sobrenome: M
CPF: 6917124
Total de Vendas: 500.0
Taxa de Comissão:0.1
Ganho do empregado 1: 50.0
================================
Informações empregado 2: Nome: Rafael
Sobrenome: R
CPF: 007770003331
Total de Vendas: 25000.0
Taxa de Comissão:0.2
SalárioBase: 2000.0
Ganho do empregado 2: 7000.0
```

Engenharia de Software com Herança

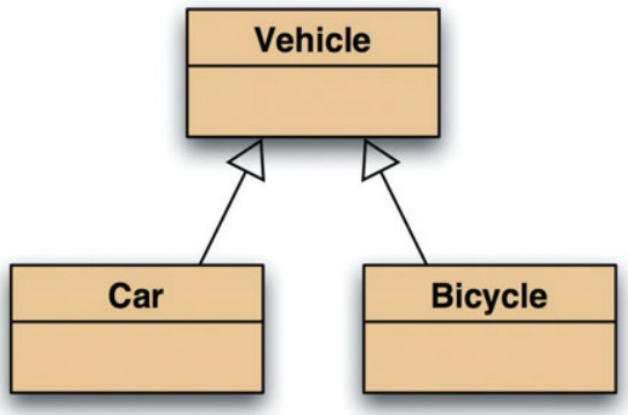
- O Java simplesmente requer acesso ao arquivo `.class` da superclasse para que possa compilar e executar qualquer programa que utiliza ou estenda a superclasse
- Essa capacidade é atraente para fornecedores de software independentes que podem desenvolver classes e disponibilizá-las para o usuário em formato de `bytecode`
- Os usuários podem então derivar novas classes dessas classe de biblioteca rapidamente e sem acessar o código-fonte proprietário

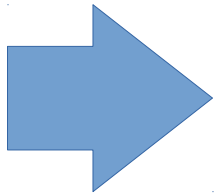
Observações

- Na etapa do design de um sistema orientado a objetos, você frequentemente descobrirá que certas classes são intimamente relacionadas
- Deve-se observar as variáveis de instância e métodos comuns e colocá-los em uma superclasse
- A partir daí, deve-se utilizar a herança para desenvolver subclasses, especializando-as com capacidades além daquelas herdadas da superclasse

Princípio da Substituição

- Se um tipo de uma subclasse também é um tipo de uma superclasse, **podemos atribuir um subtipo ao seu supertipo**
- Este princípio é chamado de **substituição**





```
Vehicle v1 = new Vehicle();
Vehicle v2 = new Car();
Vehicle v3 = new Bicycle();
```

```
Car c1 = new Vehicle();
```

```
Car c2 = new Bicycle();
```

Princípio da Substituição

- Um subtipo que é atribuído a um supertipo pode ser atribuído ao seu subtipo de origem por meio de *casting*

```
1   public class Teste {
2
3       public int teste;
4
5       public static void main(String args[]){
6
7           Aluno aluno1 = new Aluno("Rafael", "346.XXX.XXX-XX", "46547456", "Sistemas de Informação");
8
9           Pessoa pessoa = aluno1;
10
11          Aluno aluno2 = (Aluno)pessoa;
12
13          System.out.println(aluno2.toString());
14      }
15
16  }
```

```
Saída ×
Teste (run) × | Teste (run) #2 × | Teste (run) #3 ×
run:
Nome: Rafael     CPF: 346.XXX.XXX-XX     RGA: 46547456     Curso: Sistemas de Informação
```

Classe Object

- Todas as classes no Java herdam, ou estendem, direta ou indiretamente da classe `Object` (pacote `java.lang`)

- Portanto, seus 11 métodos são herdados, ou estendidos, por todas as outras classes

Métodos da classe `Object`

Método	**Descrição**
`clone`	É um método `protected`, que não possui argumentos e retorna uma referência `Object`. Faz uma "cópia" do objeto em que é chamado. A implementação padrão realiza uma **cópia superficial**, isto é, os valores de variáveis primitivas são copiados enquanto que das variáveis por referência são criadas cópias dos ponteiros de tais variáveis. Uma típica implementação do método `clone` sobrescrito realizaria uma **cópia em profundidade** que cria um novo objeto para cada variável de instancia de tipo por referência

Métodos da classe Object

Método	Descrição
equals	Esse método compara dois objetos quanto à igualdade e retorna true se eles forem iguais ou false caso contrário, false. O método aceita qualquer Object como argumento. A implementação padrão do método equals usa o operador == para determinar se duas referências referenciam o mesmo objeto na memória. Quando os objetos de uma classe particular precisam ser comparados quanto à igualdade, a classe deve sobrescrever o método equals para comparar o conteúdo de dois objetos

Métodos da classe Object

Método	**Descrição**
`finalize`	Esse método é chamado pelo coletor de lixo para realizar a limpeza de término em um objeto antes do **coletor de lixo**. Não é claro se, ou quando, o método `finalize` será chamado. Por essa razão, a maioria dos programadores deve evitar o método `finalize`

Métodos da classe Object

Método	**Descrição**
`getClass`	Todo objeto Java conhece seu próprio tipo em tempo de execução. O método `getClass` retorna um objeto `Class` (pacote `java.lang`) que contém as informações sobre o tipo de objeto, como seu nome de classe (retornado pelo método `getName`)

Métodos da classe Object

Método	**Descrição**
`getClass`	Todo objeto Java conhece seu próprio tipo em tempo de execução. O método `getClass` retorna um objeto Class (pacote `java.lang`) que contém as informações sobre o tipo de objeto, como seu nome de classe (retornado pelo método `getName`)

Métodos da classe `Object`

Método	**Descrição**
`hashCode`	Códigos de *hash* são valores int que são úteis para armazenamento e recuperação de alta velocidade das informações armazenadas em uma estrutura de dados que é conhecida como uma tabela de *hash*.
`wait`, `notify`, `notifyAll`	Os métodos `notify`, `notifyAll` e três versões sobrecarregadas de `wait` estão relacionados à *threads*

Métodos da classe Object

Método	**Descrição**
`toString`	Esse método retorna uma representação `String` do um objeto. A implementação padrão desse método retorna o nome de pacote e o nome de da classe do objeto seguido por uma representação hexadecimal do valor retornado pelo método `hashCode` do objeto

Material Complementar

- Inheritance (Herança)

  `https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html`

- Universidade XTI - JAVA - 047 - Herança extends

  `https://www.youtube.com/watch?v=F1DaEM056H0`

- Introdução ao Garbage Collection

  `http://javafree.uol.com.br/artigo/1386/`

  `Introducao-ao-Garbage-Collection.html`

Material Complementar

- Curso POO Teoria #10a - Herança (Parte 1)

  `https://www.youtube.com/watch?v=_PZldwo0vVo&index=19&list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY`

- Curso POO Java #10b - Herança (Parte 1)

  `https://www.youtube.com/watch?v=19IGAeoFKlU&list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY&index=20`

Material Complementar

- Curso POO Teoria #11a - Herança (Parte 2)

  `https://www.youtube.com/watch?v=He887D2WGVw&index=21&list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY`

- Curso POO Java #11b - Herança (Parte 2)

  `https://www.youtube.com/watch?v=5pwV2WdD-_Y&list=PLHz_AreHm4dkqe2aR0tQK74m8SFe-aGsY&index=22`

- Herança, reescrita e polimorfismo

  `https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/`

Imagem do dia

Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I


Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.


Wikipedia (2015).
Herança,
https://pt.wikipedia.org/wiki/Heran%C3%A7a. Último acesso em 5 de junho de 2017.


Wikipedia (2017a).
Herança genética, https://pt.wikipedia.org/wiki/Heran%C3%A7a_gen%C3%A9tica. Último acesso em 5 de junho de 2017.

Referências Bibliográficas II

Wikipedia (2017b).
Herança (programação), https://pt.wikipedia.org/wiki/Heran%C3%A7a_(programa%C3%A7%C3%A3o). Último acesso em 5 de junho de 2017.