Hierarquia de Classes e Herança

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3

Generalização e Especialização

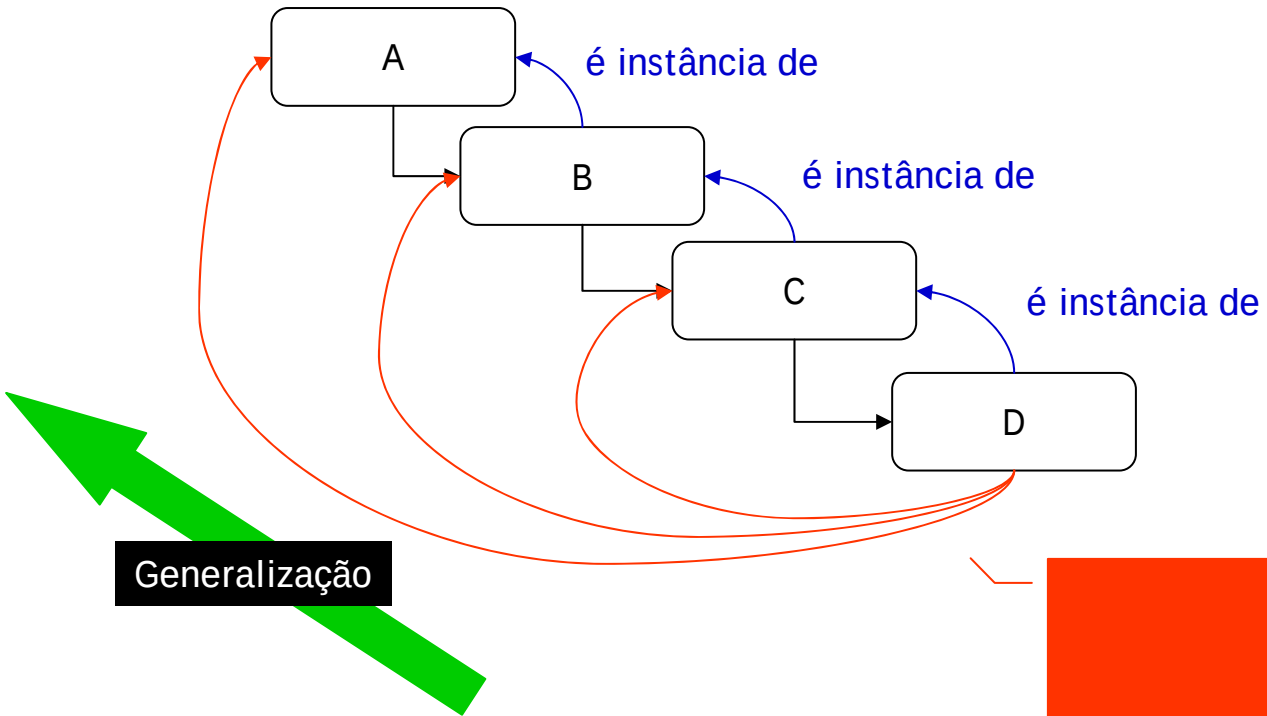
Generalização

- Generalização
  - Relacionamento entre uma classe (superclasse) e uma ou mais variações da mesma (subclasse). É o resultado de distinguir uma classe como sendo mais geral ou inclusiva do que outra
  - Superclasse
    - Mantém atributos, operações e associações comuns
  - Subclasses
    - Adicionam atributos, operações e associações específicos

Generalização

- Generalização
  - Pode ter múltiplos níveis de relacionamento
  - Uma instância de uma subclasse é uma instância das suas superclasses

Generalização



D é instância de C
D é instância de B
D é instância de A

Da mesma forma:

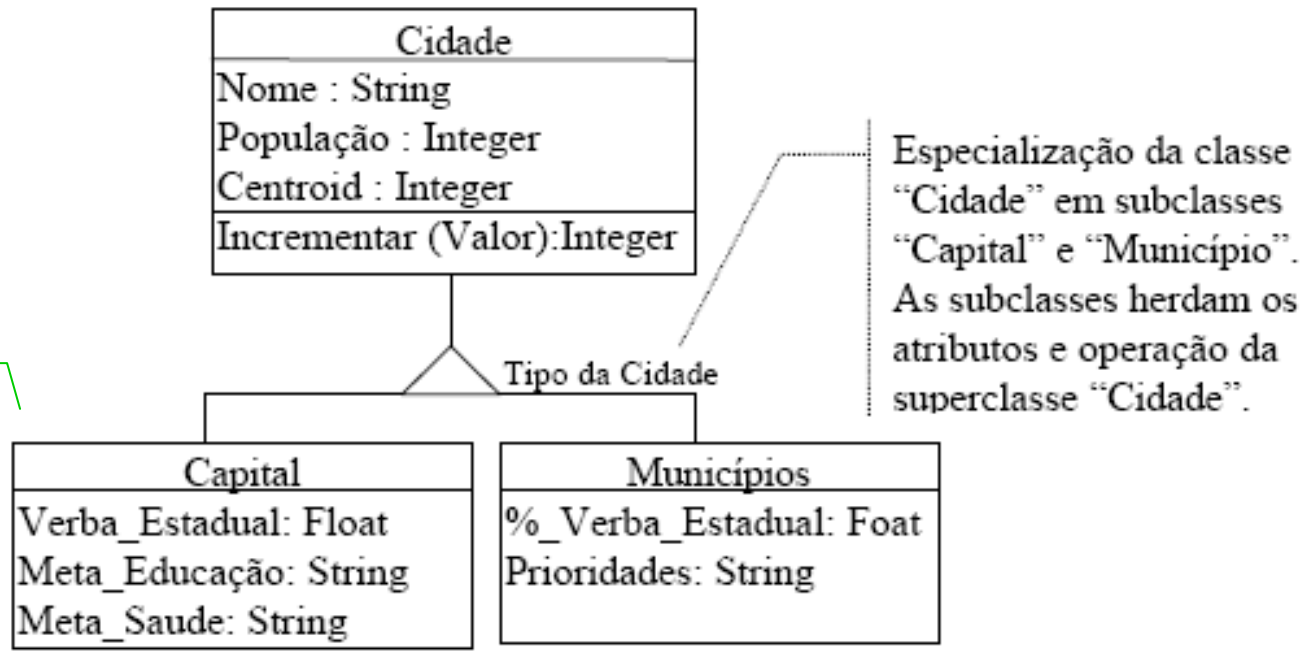
C é instância de B e A
B é instância de A

Especialização

- Especialização
  - Inverso da generalização. subcategorias satisfazem todas as propriedades das categorias de que elas constituem especializações. Além disso, deve existir pelo menos uma propriedade que distingue duas categorias especializadas

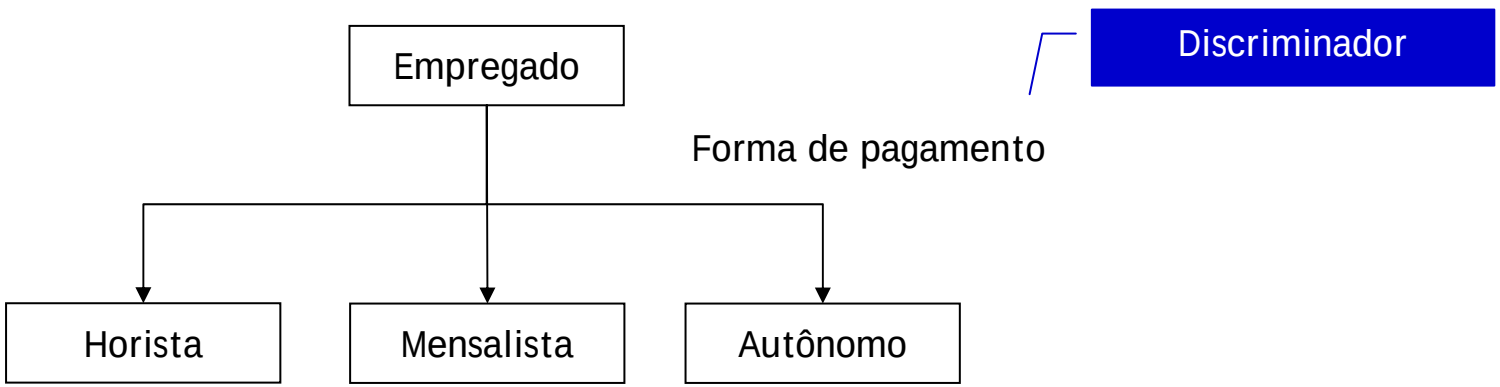
Especialização

Quais os atributos da classe Capital ?

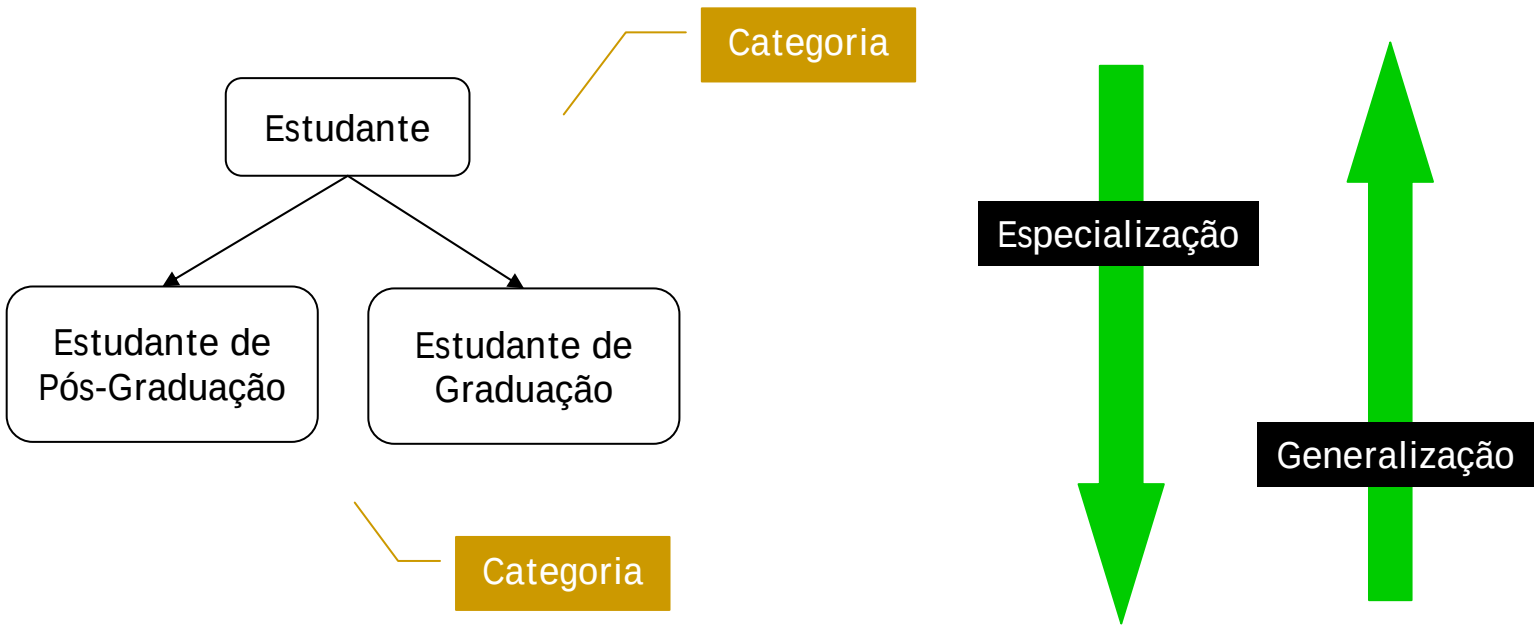


Especialização

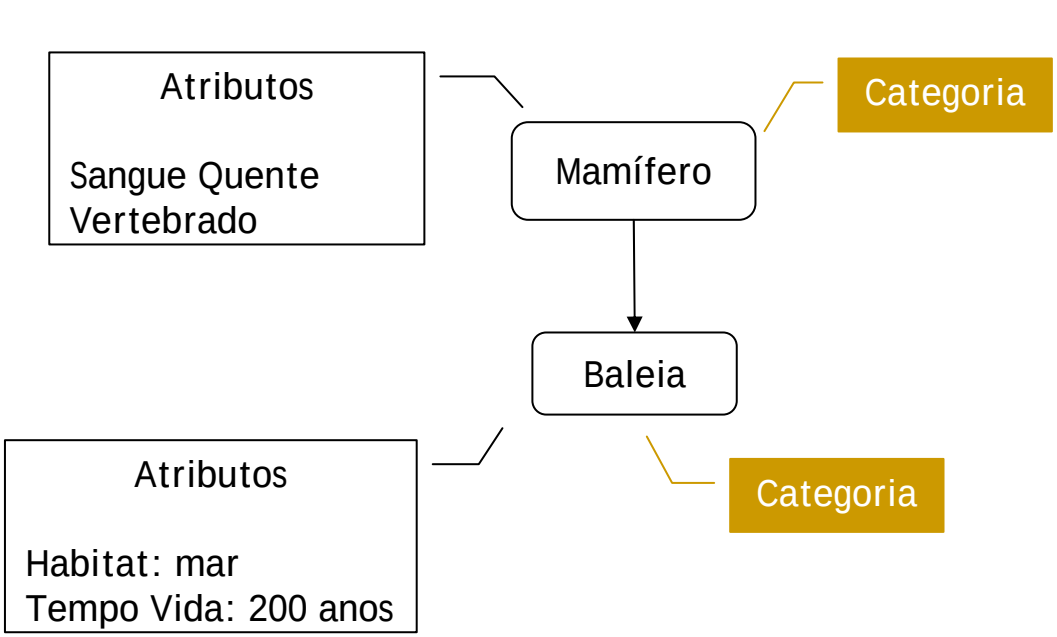
- Discriminador
  - Critério utilizado na partição das subclasses
  - Atributo que tem valor diferente para cada subclasse
  - Pode ser omitido na modelagem

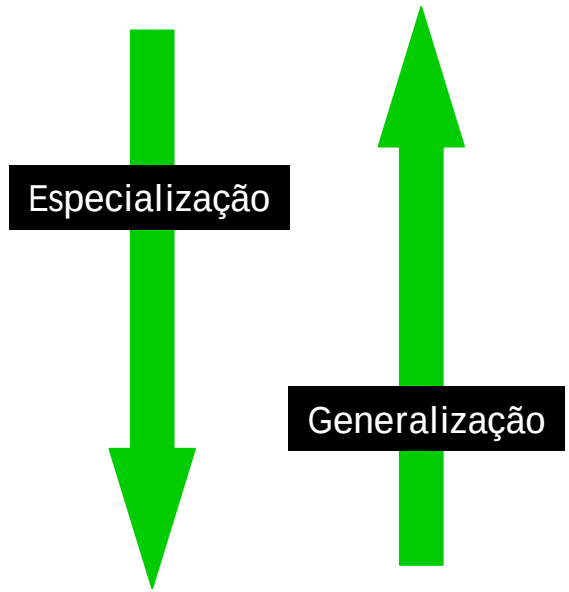


Generalização/Especialização



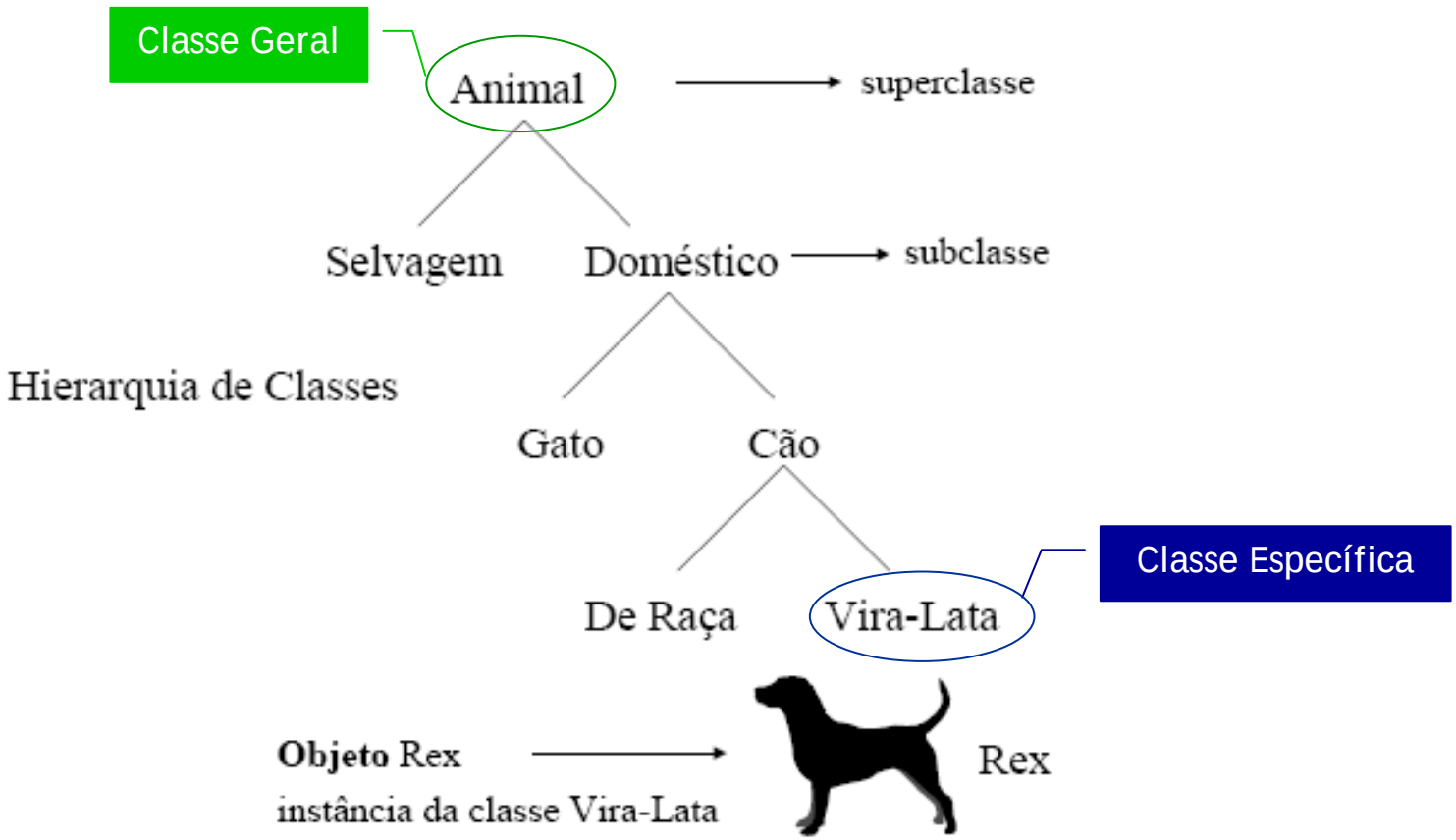
Generalização/Especialização



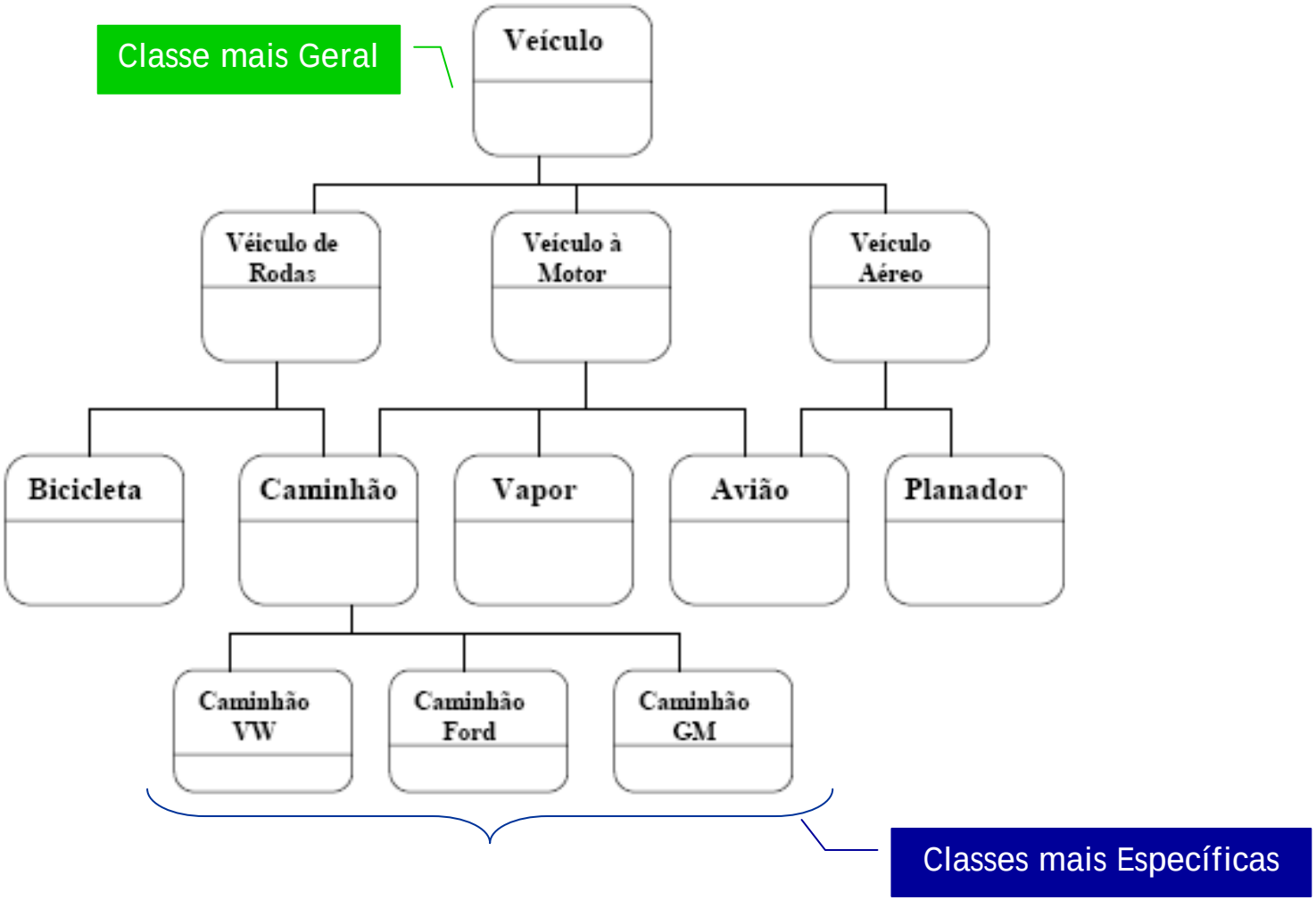


Hierarquia de Classes

Hierarquia de Classes



Hierarquia de Classes

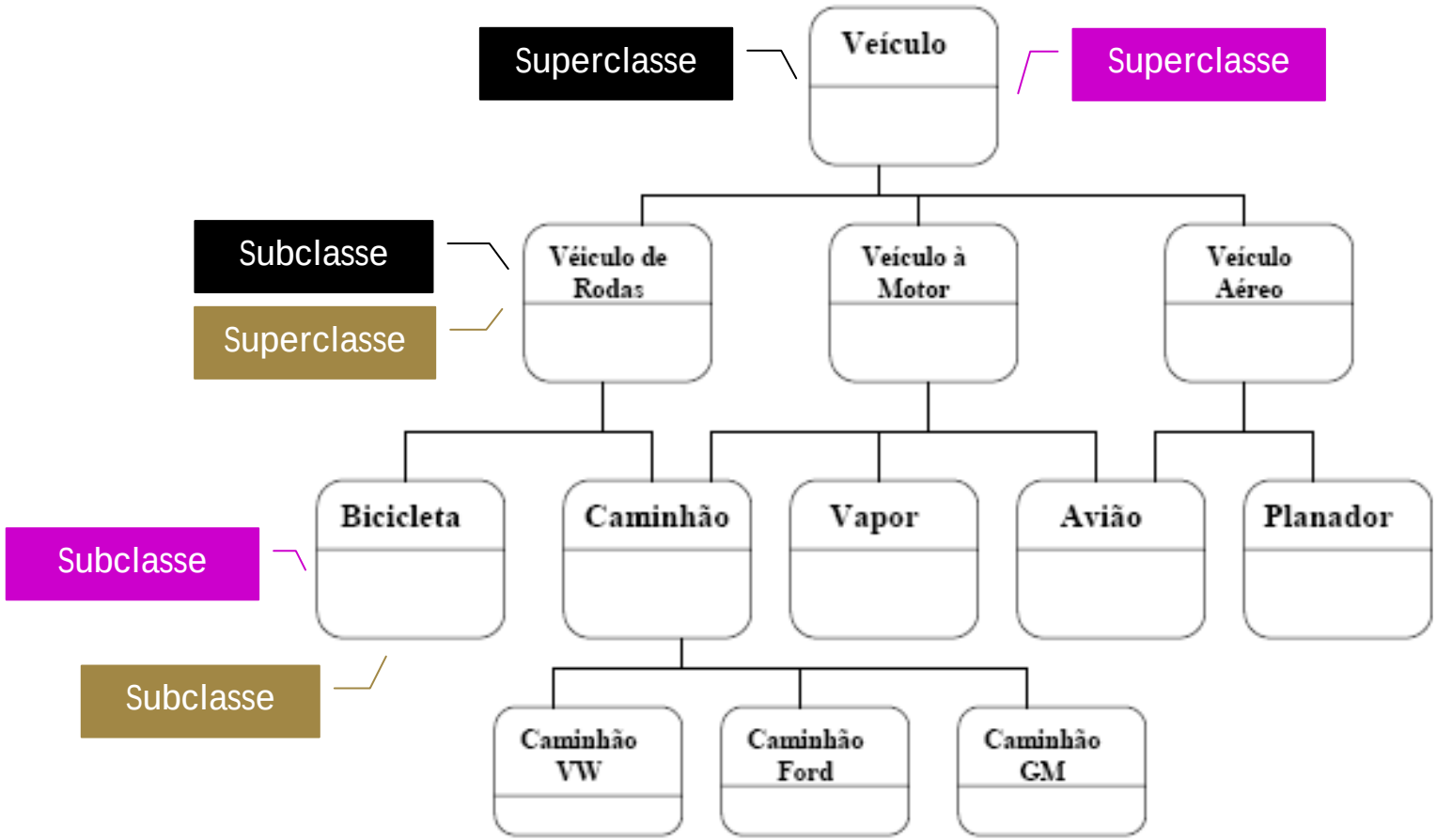


Herança

Herança

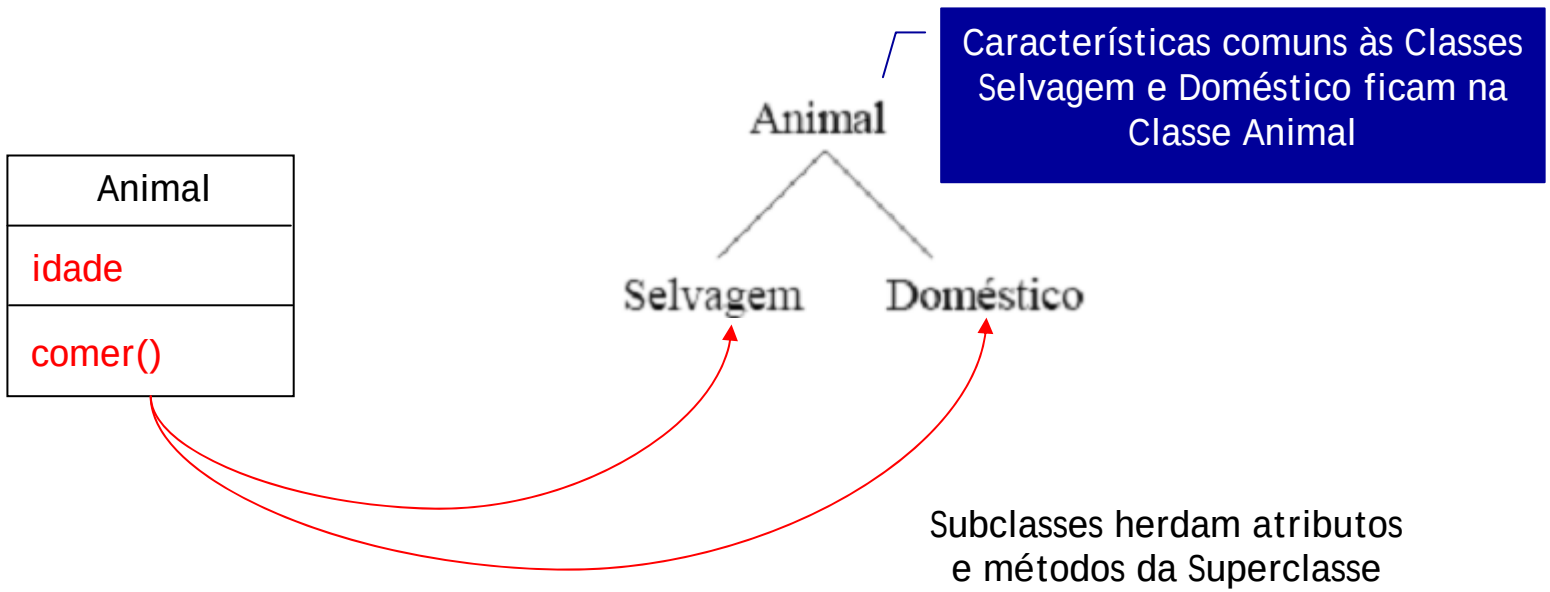
- Definição
  - Herança é um tipo de associação entre classes, onde os atributos e métodos definidos na **superclasse** (a classe mais genericamente definida) são compartilhados pela classe mais especializada, a **subclasse**.

Herança – Super/SubClasse



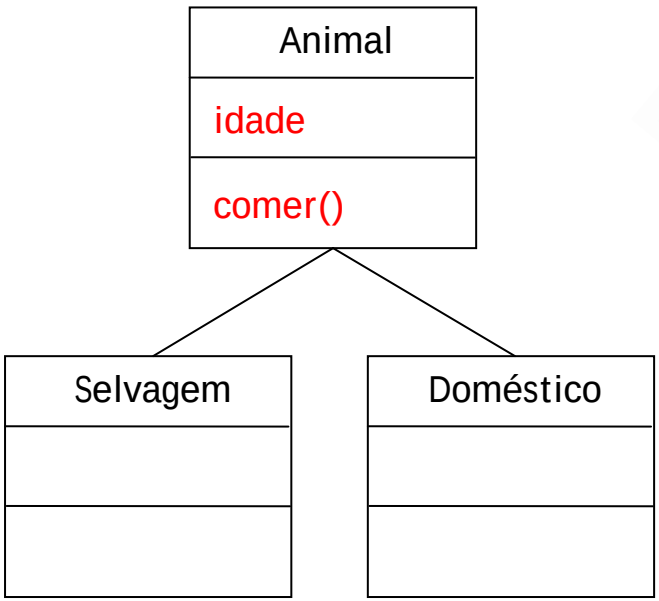
Herança

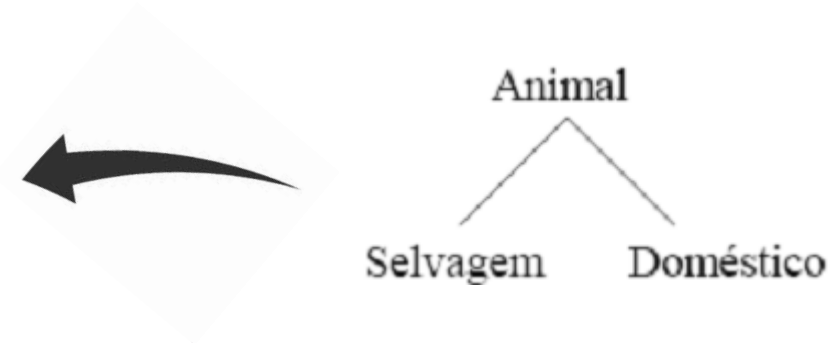
- Pontos Importantes
  - Características comuns são colocadas em uma classe base (superclasse)
  - As subclasses herdam atributos e métodos da superclasse



Herança

- Pontos Importantes
  - As propriedades das subclasses não precisam ser repetidas nas subclasses
  - Isso se reflete em código

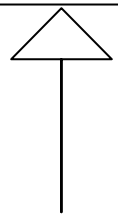




Não é preciso repetir os atributos e métodos de Animal nas Classes Doméstico e Selvagem. Por herança, ambas possuem o atributo idade e o método comer()

Herança

```
1  public class Classe1 {
2
3      protected int atributo1;
4      protected int atributo2;
5
6  }
```

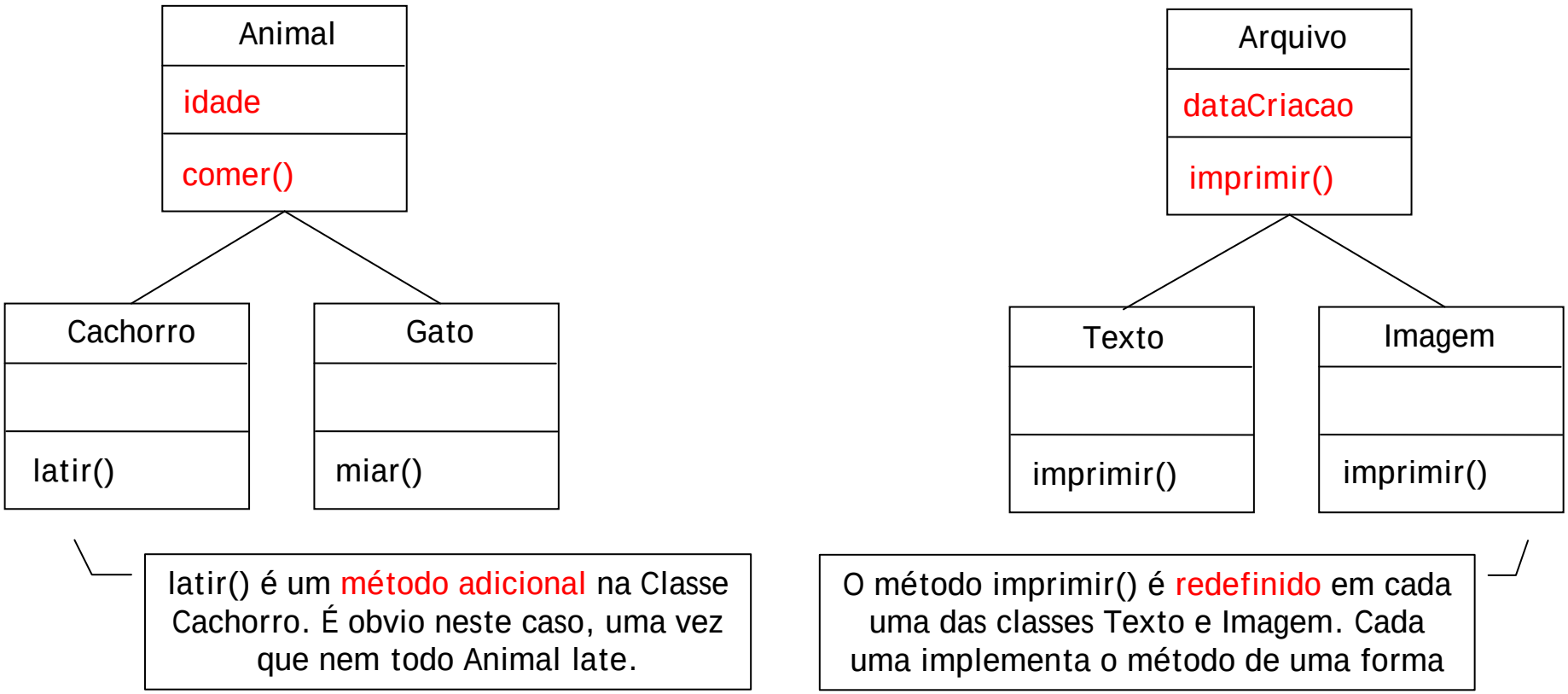


```
1  public class Classe2 extends Classe1{
2
3
4
5  }
```

Herança

- Pontos Importantes
  - As subclasses podem “implementar” suas especificidades
    - Podem conter atributos e métodos adicionais - ou seja, na subclasse é possível ser feita a definição de mais atributos e métodos
    - Pode ser realizada a redefinição dos métodos que foram herdados pela subclasse. É o aproveitamento e extensão das características de uma classe existente.

Herança

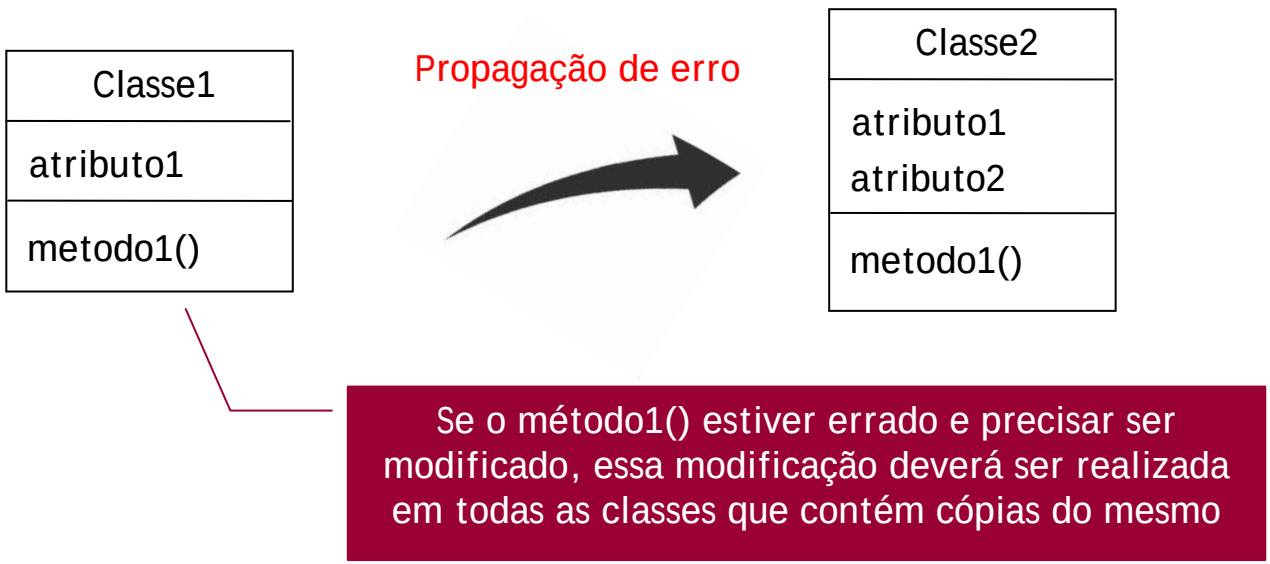

Animal
idade
comer()
Cachorro
latir()
Gato
miar()
Arquivo
dataCriacao
imprimir()
Texto
imprimir()
Imagem
imprimir()
latir() é um método adicional na Classe Cachorro. É obvio neste caso, uma vez que nem todo Animal late.
O método imprimir() é redefinido em cada uma das classes Texto e Imagem. Cada uma implementa o método de uma forma

Herança

- Reforçando
  - Mecanismo que permite construir classes a partir de classes já existentes
  - Facilita a extensibilidade do código
  - Facilita reuso de código
    - Código definido na superclasse pode ser utilizado automaticamente na subclasse – não é preciso copiar o código da superclasse
    - Porque copiar código não é interessante?

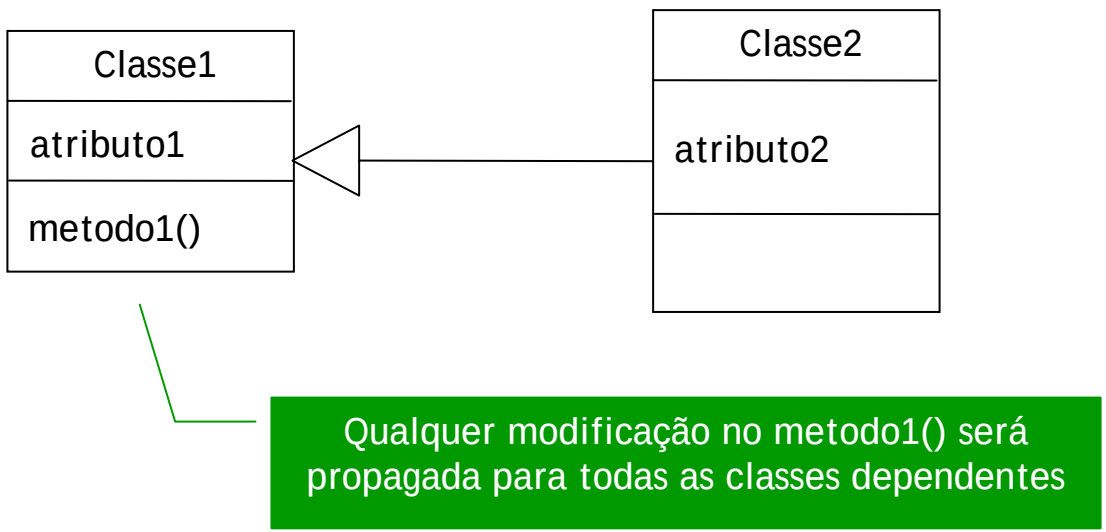
Herança

- Observação
  - Copiar e colar código de uma classe para outra pode espalhar erros por múltiplos arquivos de código-fonte. Para evitar a duplicação de código (e, possivelmente, erros), utilize a herança (em situações que se deseja que uma classe “absorva”as variáveis de instância e métodos de outra classe). Contudo, cuidado, pois nem toda classe pode fzer parte de uma certa hierarquia (nem toda classe pode ser filha/mãe de outra)



Herança

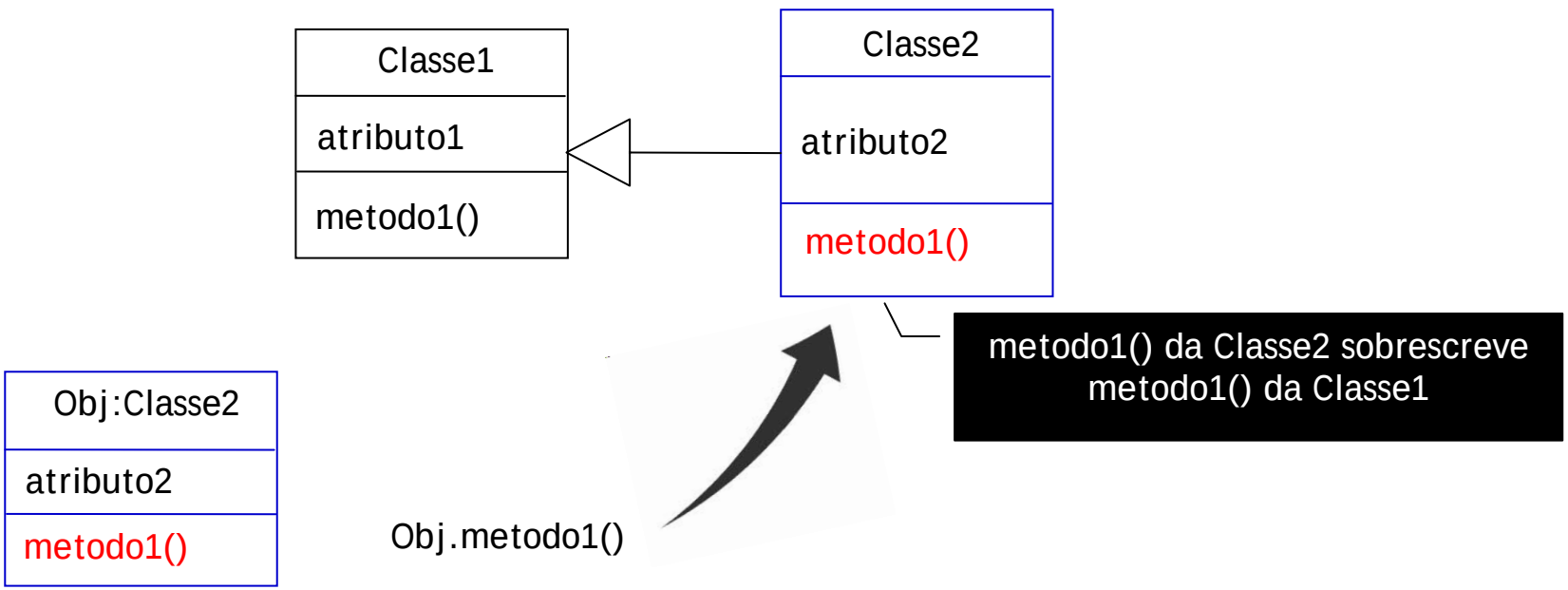
- Observação
  - Quando alterações são requeridas para recursos comuns (declarados em uma superclasse), os desenvolvedors só precisam fazê-las na superclasse – as subclasses herdam as alterações. Sem a herança, as alterações precisariam ser feitas em todos os arquivos de código-fonte que contém uma cópia do código em questão. Óbvio que, se uma classe não “suportar” a alteração, sua inclusão na hierarquia deve ser revista



Herança – Chamada de Métodos

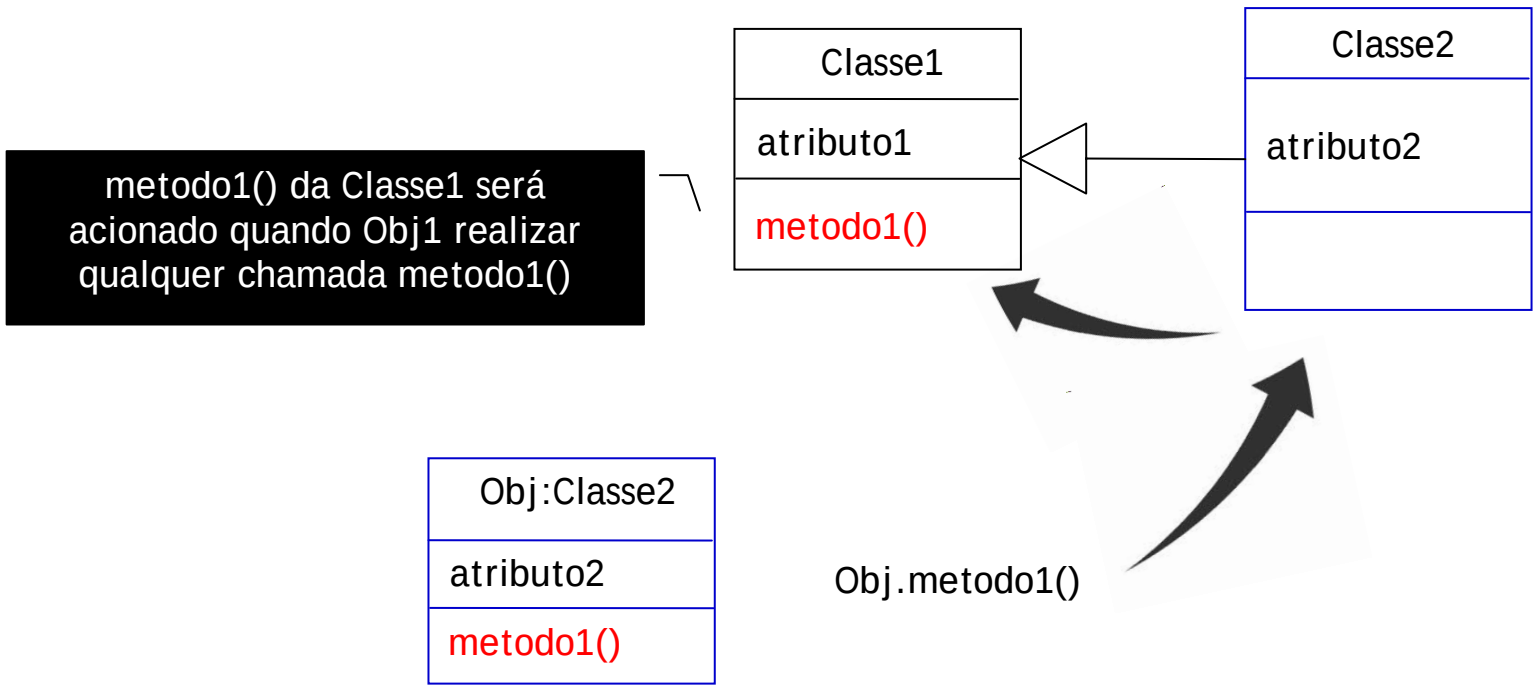
- Chamada de Métodos
  - Quando um objeto da subclasse chama um método, temos:

- Cenário 1
  - Se é um método da subclasse que sobrescreve um método da superclasse, o da subclasse é acionado (ele esconde o da superclasse)



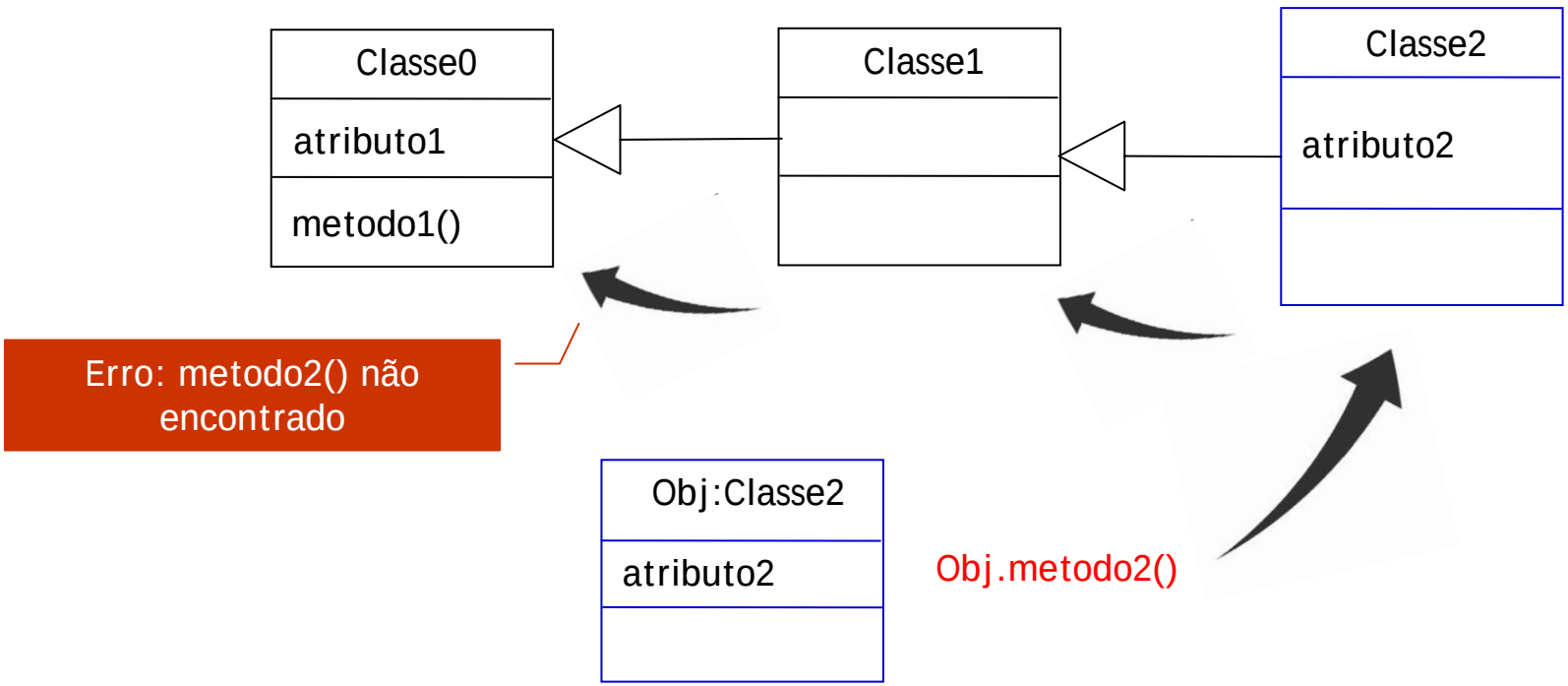
Herança – Chamada de Métodos

- Cenário 2
  - Se o método chamado só existe na superclasse, ele será acionado (a subclasse herda métodos da superclasse que ela não sobrescreve)



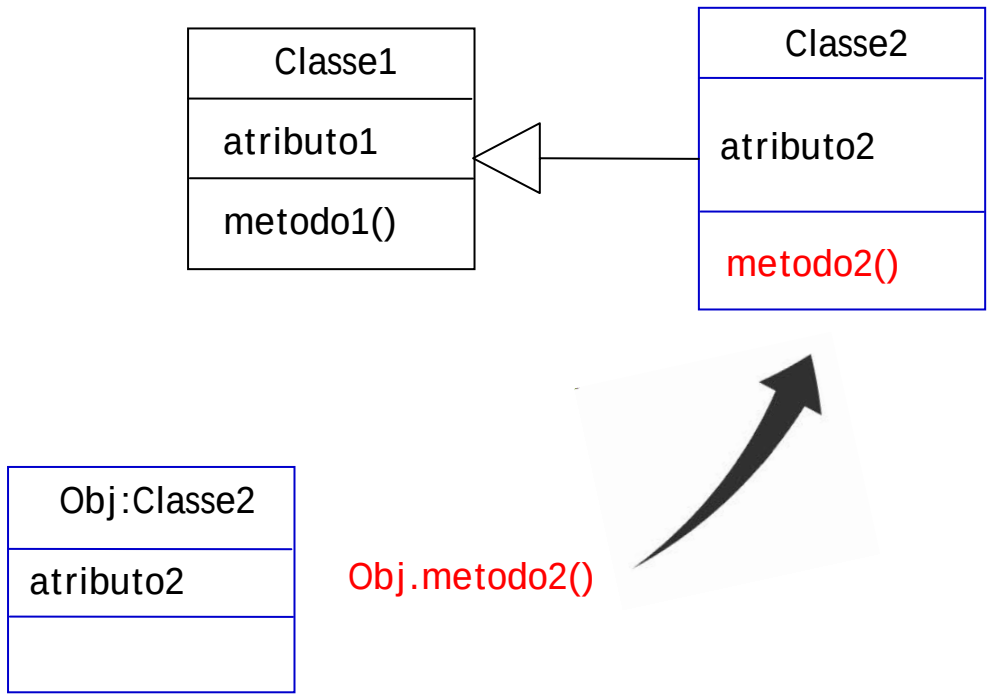
Herança – Chamada de Métodos

- Cenário 3
  - Se o método chamado não existe na subclasse, nem na superclasse, nem na superclasse da superclasse, e em nenhuma superclasse hierarquia acima, então acontece um erro



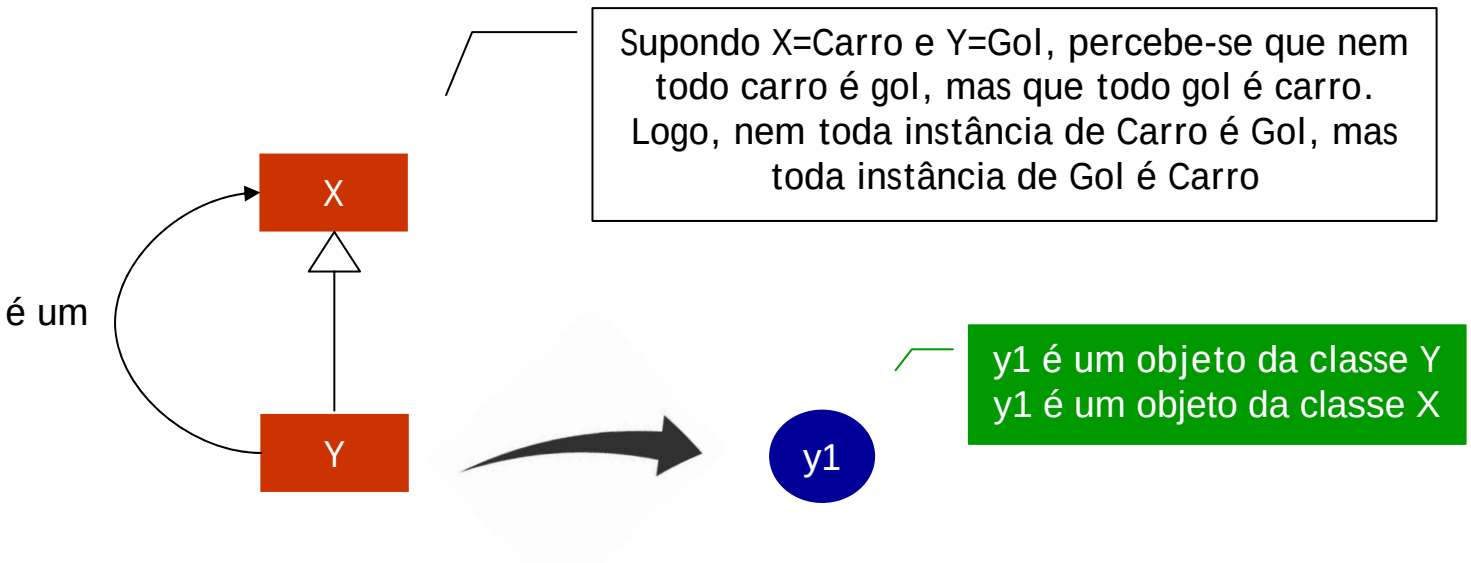
Herança – Chamada de Métodos

- Cenário 4
  - Se o método é exclusivo da subclasse, ele será acionado



Herança

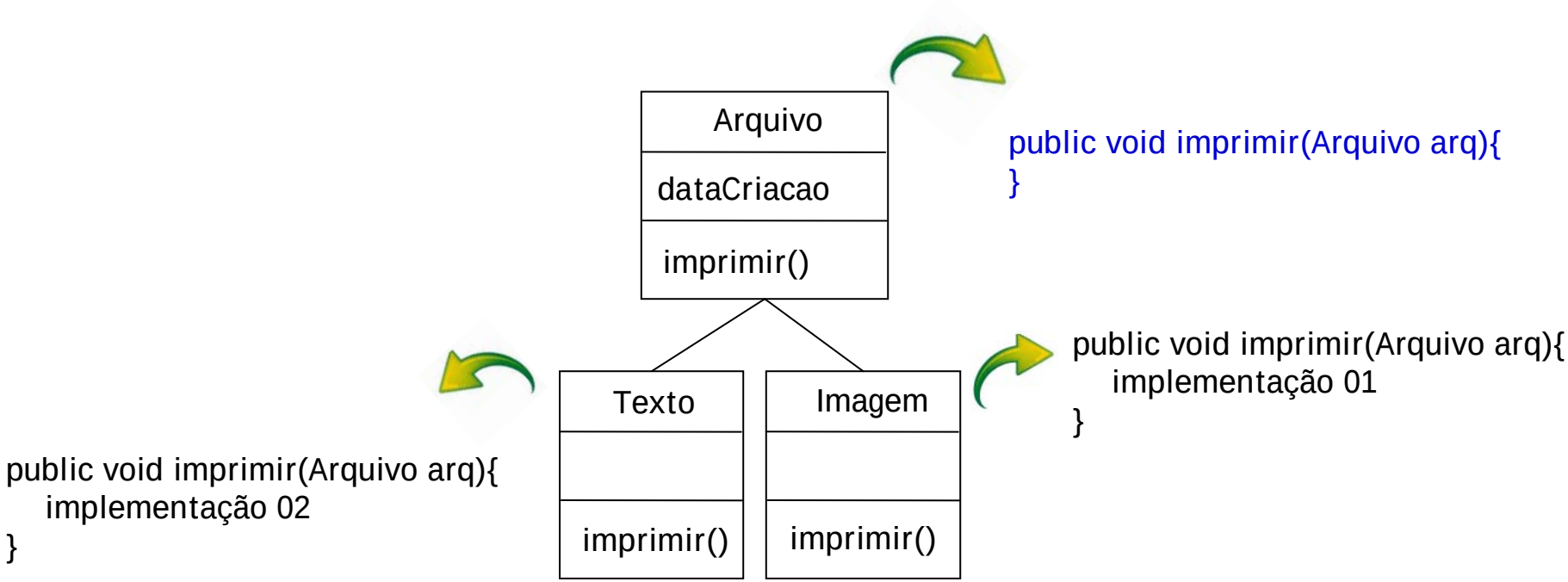
- Observação
  - A herança entre duas classes define uma relação de é um. Ou seja, considere que a classe Y seja subclasse da classe X. Caso seja criado um objeto y1 da classe Y, este objeto é um objeto da classe Y e também é um objeto da classe X. O inverso, no entanto, não é verdadeiro: um objeto criado da classe X não é necessariamente um objeto da classe Y.



Y1: Objeto Instanciado da classe Y

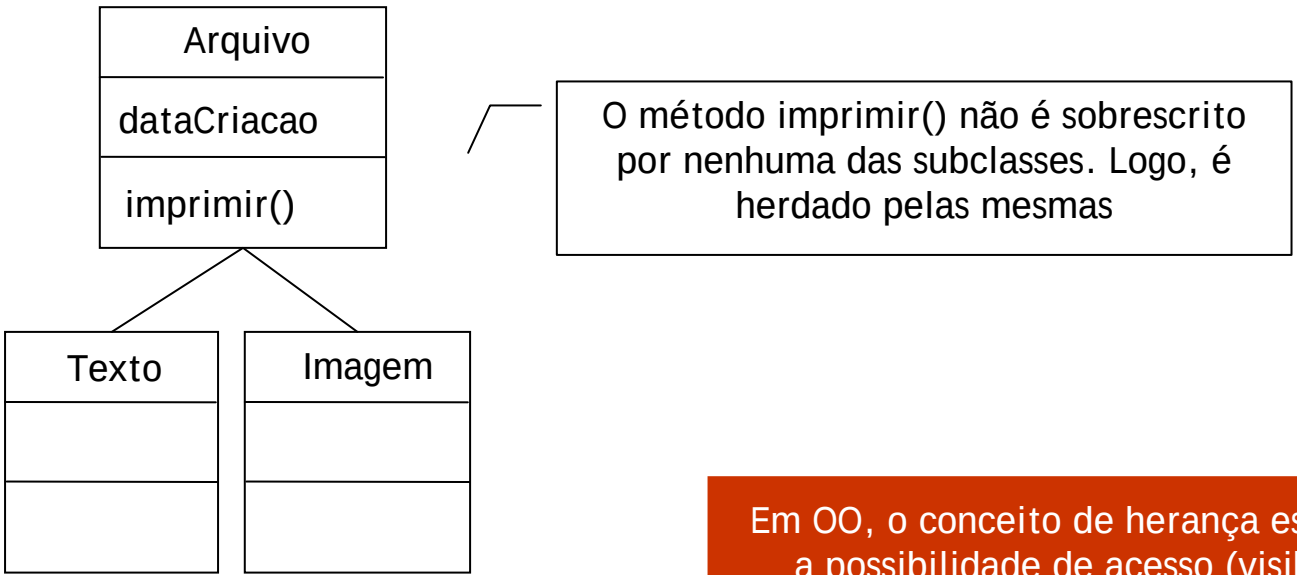
Herança

- Ao escrever métodos na subclasse pode-se:
  - Sobrescrever (ou sobrepor) métodos da superclasse. Um método da subclasse sobrescreve um da superclasse se ele tem a mesma assinatura. Não há o conceito de herdar neste caso.



Herança

- Ao escrever métodos na subclasse pode-se:
  - Herdar métodos da superclasse. Todo método da superclasse que não é sobrescrito na subclasse é herdado por esta. O método da superclasse pode ser aplicado a um objeto da subclasse.



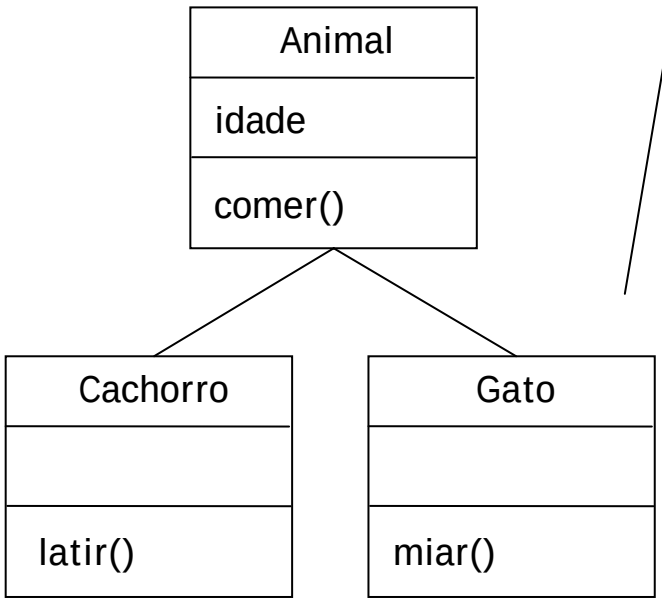
Em OO, o conceito de herança está relacionada a possibilidade de acesso (visibilidade) e a não sobrescrita de métodos

Herança

- Sobrescrita de Métodos
  - Caso 1: subclasse sobrescreve métodos da superclasse
    - Subclasse não herda tais métodos
  - Caso 2: subclasse não sobrescreve métodos da superclasse
    - Subclasse herda tais métodos

Herança

- Ao escrever métodos na subclasse pode-se:
  - Definir novos métodos para a subclasse que não aparecem na superclasse. Esses métodos só poderão ser aplicados a objetos da subclasse.



Os métodos latir() e miar() são específicos, respectivamente, das classes Cachorro e Gato

```
Gato gato1 = new Gato();
gato1.miar();
```

Correto!!

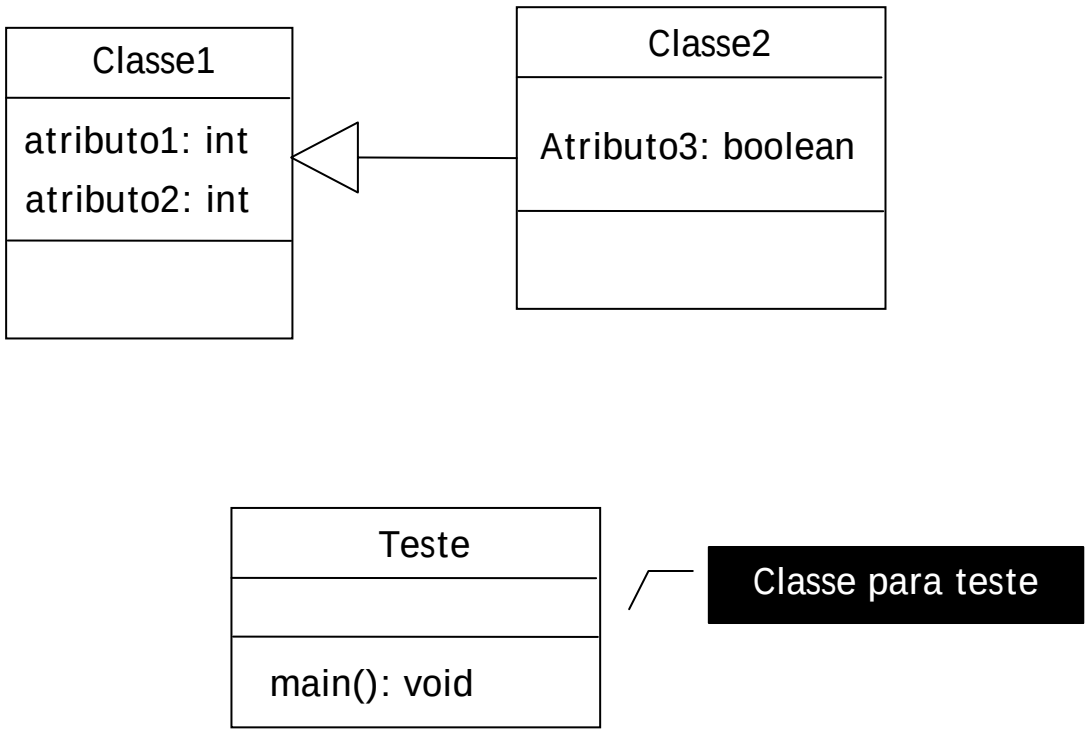
```
Animal animal1 = new Aminal()
animal1.miar();
```

Errado!!

Herança

- Para os atributos
  - Não se pode sobrescrever nenhum atributo da superclasse. Pode-se apenas:
  - Herdar os atributos da superclasse (**"todos"** os atributos da superclasse são automaticamente herdados pela subclasse) e definir novos atributos (Só objetos da subclasse terão esses atributos)

Sobrescrever Atributo



Sobrescrever Atributo

```
1  public class Classe1 {
2
3      protected int atributo1;
4      protected int atributo2;
5
6  }
```

Classe1

Palavra reservada para herança

"extends"

```
1  public class Classe2 extends Classe1{
2
3      protected boolean atributo3;
4
5  }
```

Classe2

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9  }
10
```

atributo1 : int - Classe1
atributo2 : int - Classe1
atributo3 : boolean - Classe2
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Sobrescrever Atributo

```
1  public class Classe1 {
2
3      protected int atributo1;
4      protected int atributo2;
5
6  }
```

Classe1

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9  }
10
```

atributo1 : boolean - Classe2
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

Onde está o atributo1 do tipo "int" da Classe1 ?

```
1  public class Classe2 extends Classe1{
2
3      protected boolean atributo1;
4
5  }
```

Classe2

ClasseTeste

Herança

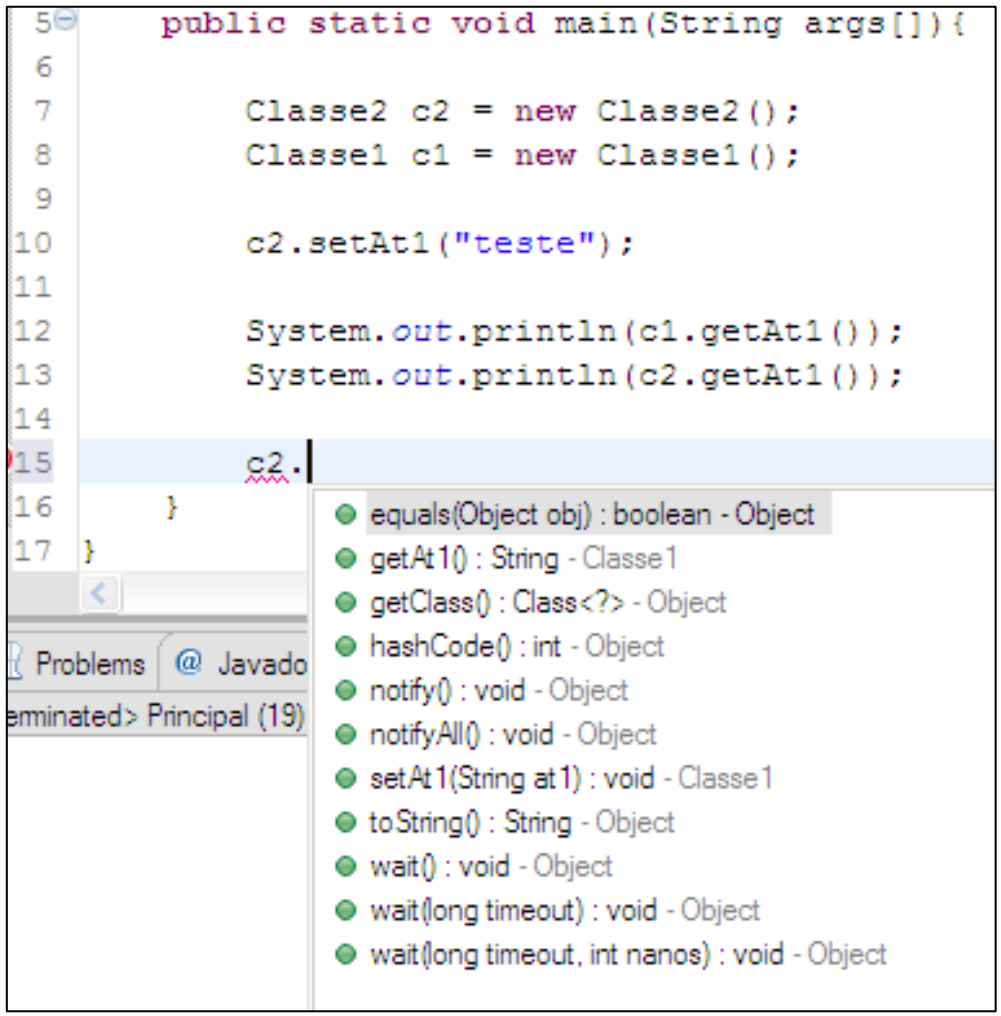
- Herança - Private
  - Uma subclasse não pode acessar diretamente os membros (métodos e atributos) private de uma superclasse. Se isso fosse possível acabaria com os benefícios do ocultamento de informações.
  - O atributo definido como private é acessível diretamente somente pela classe que o declarou. Qualquer outra classe (inclusive a subclasse), para ter acesso ao atributo, terá de fazê-lo através da interface pública da classe, isto é, através de um de métodos públicos da classe.
  - OBS
    - Métodos privados não são herdados
    - Atributos privados são herdados (mas não são acessíveis diretamente)

Exemplificação

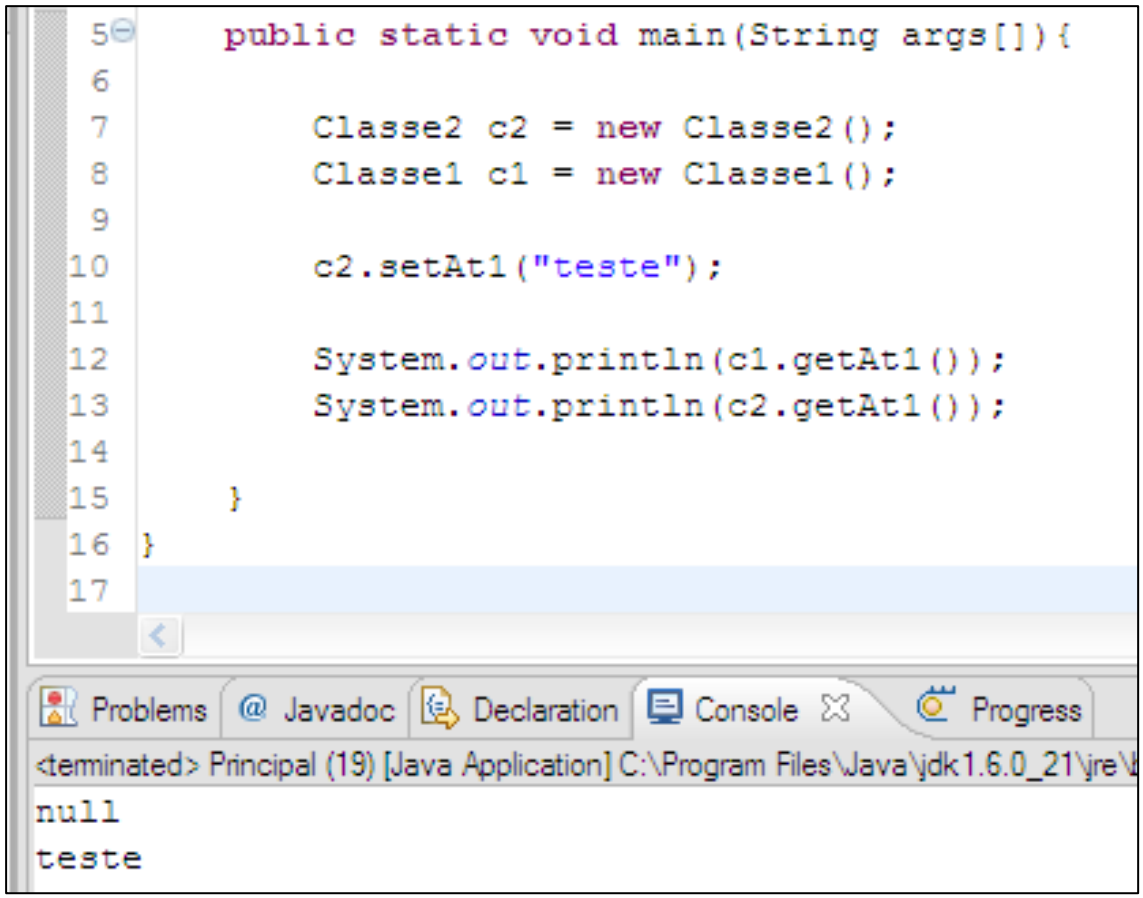
```
3  public class Classe1 {
4
5      private String at1;
6
7      public String getAt1() {
8          return at1;
9      }
10
11     public void setAt1(String at1) {
12         this.at1 = at1;
13     }
14
15     private void teste(){
16     }
17 }
```

```
3  public class Classe2 extends Classe1{
4
5  }
```

Exemplificação



Exemplificação

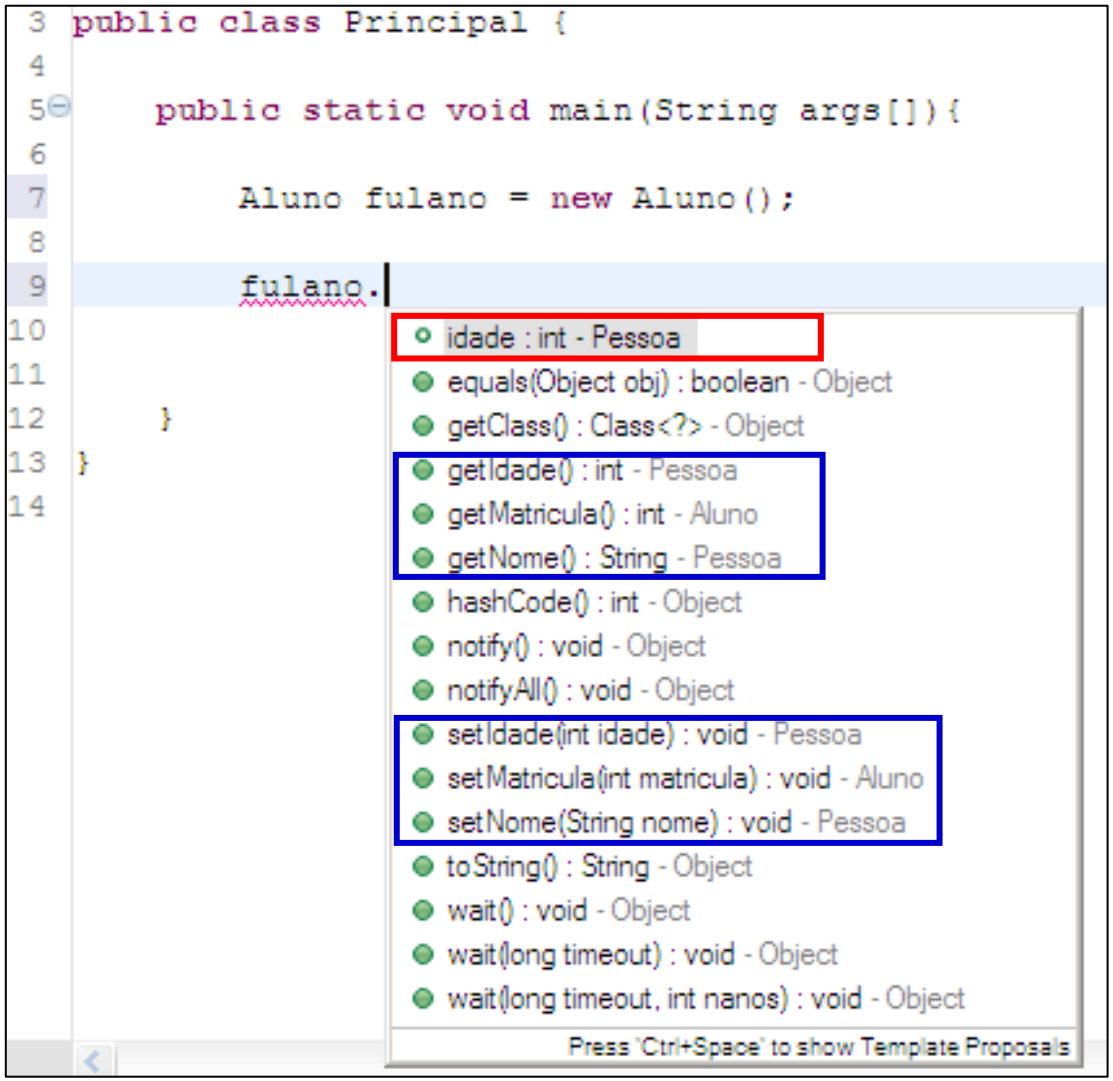


Outro Exemplo

```
3  public class Pessoa {
4
5      private String nome;
6      public int idade;
7
8      public String getNome() {
9          return nome;
10     }
11
12     public void setNome(String nome) {
13         this.nome = nome;
14     }
15
16     public int getIdade() {
17         return idade;
18     }
19
20     public void setIdade(int idade) {
21         this.idade = idade;
22     }
23 }
```

```
3  public class Aluno extends Pessoa{
4
5      private int matricula;
6
7      public int getMatricula() {
8          return matricula;
9      }
10
11     public void setMatricula(int matricula) {
12         this.matricula = matricula;
13     }
14 }
15
```

Outro Exemplo



Outro Exemplo

```
3  public class Pessoa {
4
5      protected String nome;
6      protected int idade;
7
```

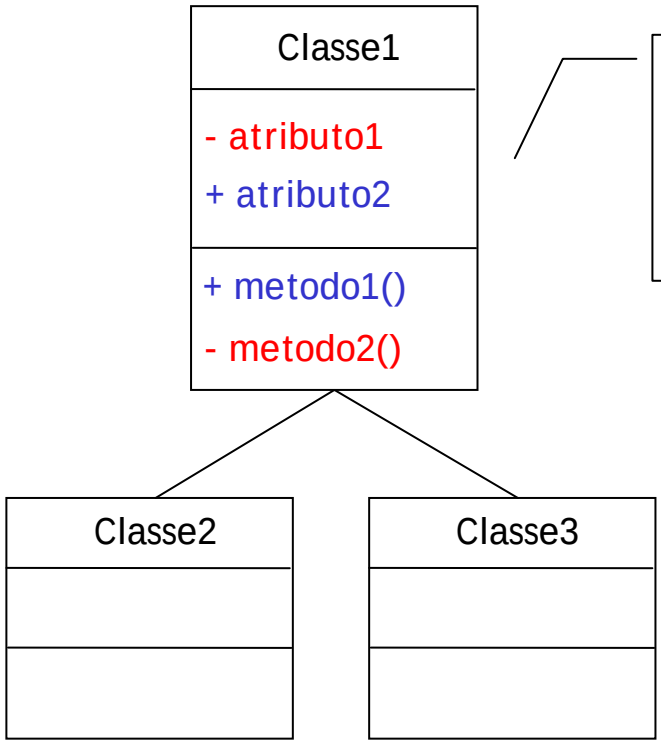
```
3  public class Aluno extends Pessoa{
4
5      private int matricula;
```

```
3  public class Principal {
4
5      public static void main(String args[]){
6
7          Aluno fulano = new Aluno();
8
9          fulano.
10
11
```

idade : int - Pessoa

nome : String - Pessoa

Herança



Ambas as classes 1 e 2 podem acessar diretamente o atributo2 e o método1(), pois são públicos. Contudo, não podem acessar o método2() e não podem acessar diretamente o atributo1

O Atributo atributo1 pode ser acessa pelas classes 1 e 2, mas de forma Indireta, ou seja, através de algum método publico da classe1

Exemplificação

Classe1

```
1  public class Classe1 {
2
3      private int atributo1;
4      public int atributo2;
5
6      public void metodo1(){
7      }
8
9      private void metodo2(){
10
11     }
12 }
13
```

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

```
1  public class Classe2 extends Classe1{
2
3
4  }
```

Classe2

Exemplificação

Classe1

```
1  public class Classe1 {
2
3      private int atributo1;
4      public int atributo2;
5
6      public void metodo1(){
7      }
8
9      private void metodo2(){
10     }
11
12     public int getAtributo(){
13         return atributo1;
14     }
15
16     public void setAtributo(int atributo){
17         atributo1 = atributo;
18     }
19 }
```

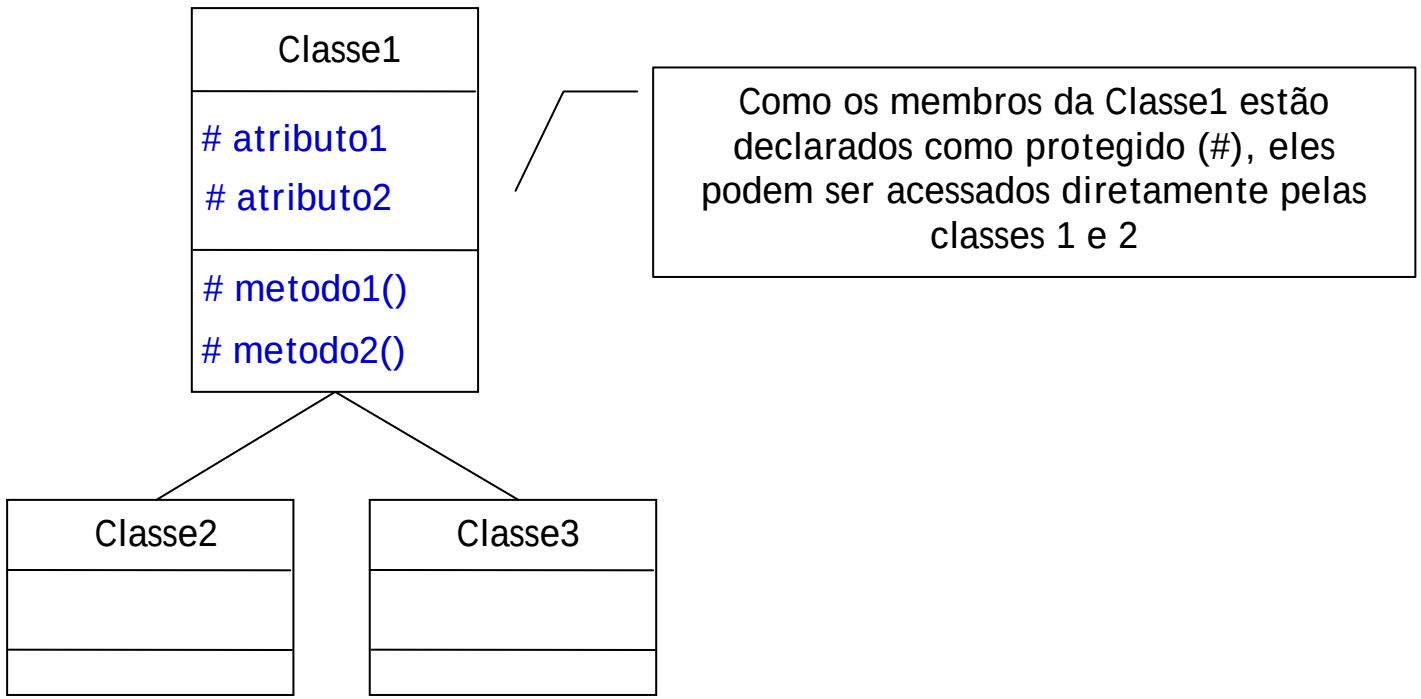
```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getAtributo() : int - Classe1
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
setAtributo(int atributo) : void - Classe1
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Herança

- Rótulo Protected
  - Permite que os membros das subclasses, ou de outras classes de um mesmo pacote, acessem os membros protected da superclasse diretamente – sem a necessidade de usar um método publico da superclasse



Exemplificação - Protected

Classe1

```
1  public class Classe1 {
2
3      protected int atributo1;
4      protected int atributo2;
5
6      protected void metodo1(){
7      }
8
9      protected void metodo2(){
10     }
11 }
```

```
1  public class Classe2 extends Classe1{
2
3
4  }
```

Classe2

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

atributo1 : int - Classe1
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
metodo2() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Exemplificação - Public

Classe1

```
1  public class Classe1 {
2
3      public int atributo1;
4      public int atributo2;
5
6      public void metodo1(){
7      }
8
9      public void metodo2(){
10     }
11 }
```

```
1  public class Classe2 extends Classe1{
2
3
4  }
```

Classe2

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

atributo1 : int - Classe1
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
metodo2() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Exemplificação - Private

Classe1

```
1  public class Classe1 {
2
3      private int atributo1;
4      private int atributo2;
5
6      private void metodo1(){
7      }
8
9      private void metodo2(){
10     }
11 }
```

Classe2

```
1  public class Classe2 extends Classe1{
2
3
4  }
```

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Exemplificação - Default

Classe1

```
1  public class Classe1 {
2
3      int atributo1;
4      int atributo2;
5
6      void metodo1(){
7      }
8
9      void metodo2(){
10     }
11 }
```

Classe2

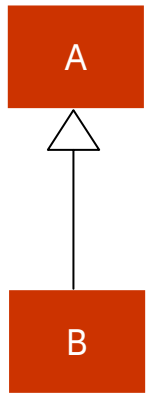
```
1  public class Classe2 extends Classe1{
2
3
4  }
```

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9  }
10
```

atributo1 : int - Classe1
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
metodo2() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

ClasseTeste

Herança



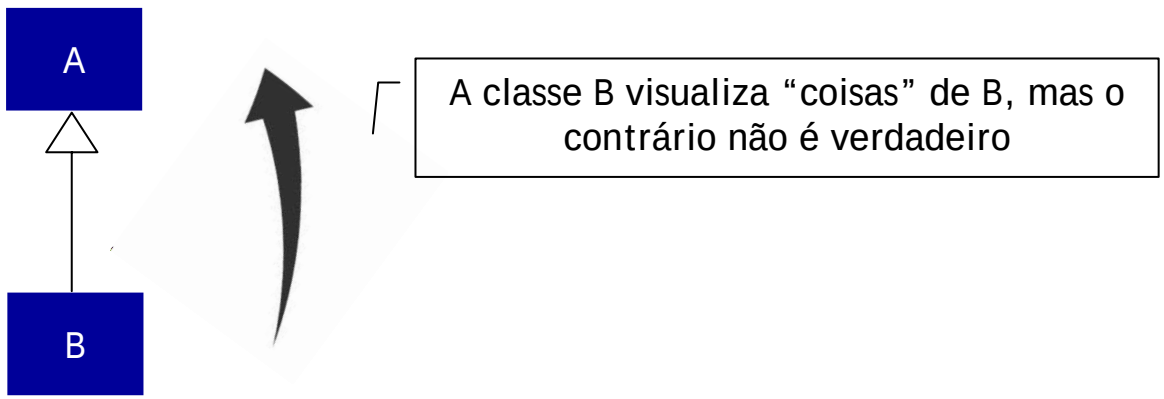
B tem acesso direto a todas as variáveis e métodos de **A** que não sejam declarados como private.

Uma instância de **B** pode responder a mensagens que correspondam a todos os métodos públicos de B e de A.

B pode definir novas variáveis e métodos.
B pode redefinir variáveis e métodos herdados.

Herança

- Observação
  - Também deve se observar que os membros (privados ou públicos) da classe derivada não são conhecidos pela sua superclasse. Isto é fácil de compreender, tendo em vista a organização hierárquica imposta pelo mecanismo de herança.



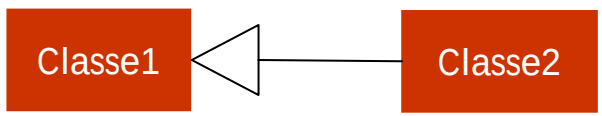
Herança

Classe1

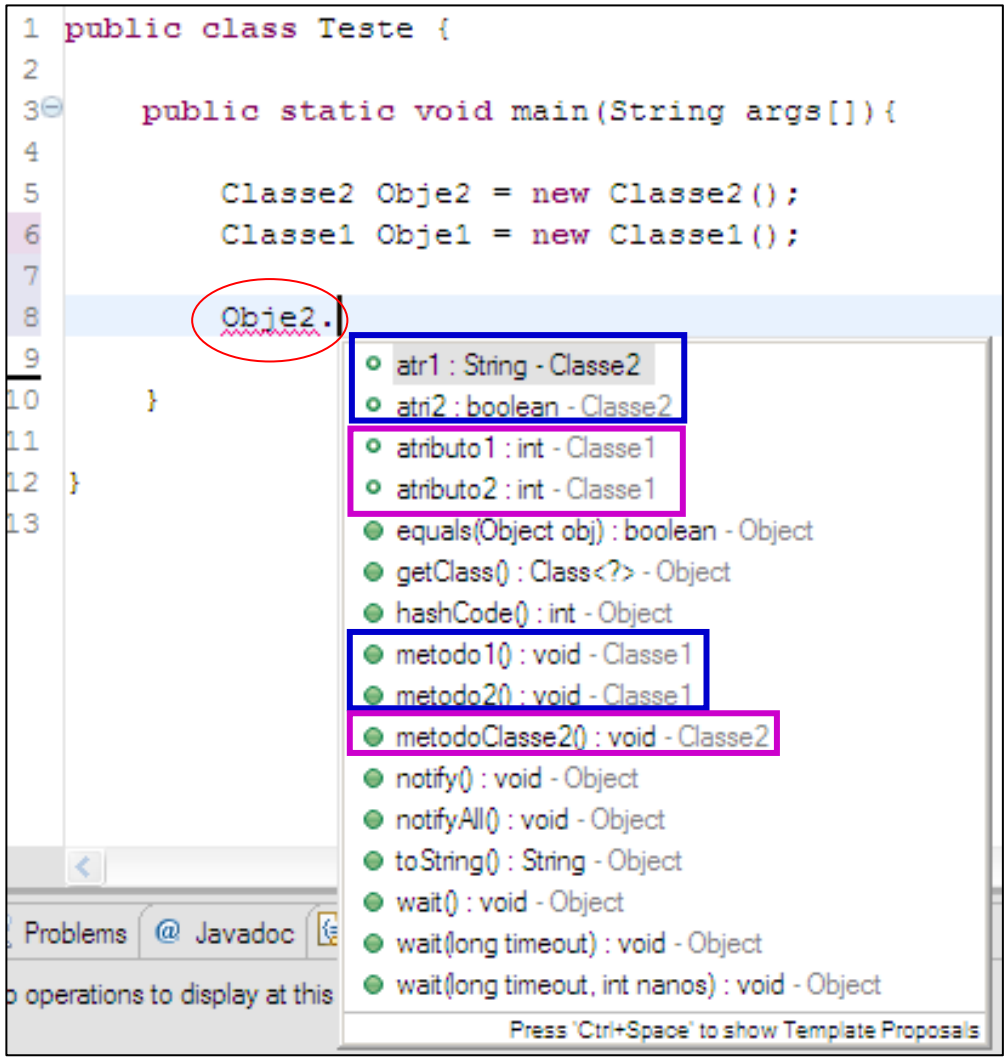
```
1  public class Classe1 {
2
3      public int atributo1;
4      public int atributo2;
5
6      public void metodo1(){
7      }
8
9      public void metodo2(){
10     }
11 }
```

```
1  public class Classe2 extends Classe1{
2
3      public String atr1;
4      public boolean atri2;
5
6      public void metodoClasse2(){
7
8      }
9  }
```

Classe2



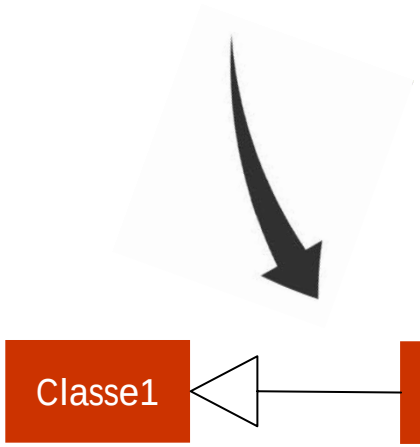
Herança


1 public class Teste {
2
3 public static void main(String args[]){
4
5 Classe2 Obje2 = new Classe2();
6 Classe1 Obje1 = new Classe1();
7
8 Obje2.
9
10 }
11
12 }
13
atr1 : String - Classe2
atri2 : boolean - Classe2
atributo1 : int - Classe1
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
metodo2() : void - Classe1
metodoClasse2() : void - Classe2
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object
Press 'Ctrl+Space' to show Template Proposals
Problems
@ Javadoc
operations to display at this

Herança

Classe1 não consegue ver métodos e atributos da classe2

Óbvio, uma vez que Classe1 não é "filho" da Classe 2, ou seja Classe1 não extends de Classe2




```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6          Classe1 Obje1 = new Classe1();
7
8          Obje1.
9
10     }
11
12 }
13
```

atributo1 : int - Classe1
atributo2 : int - Classe1
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
metodo1() : void - Classe1
metodo2() : void - Classe1
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

Problems | @ Javadoc

lo operations to display at this

Press 'Ctrl+Space' to show Template Proposals

Herança

- Reforçando
  - Atributos declarados como privados na superclasse são herdados pelas subclasses. Contudo, subclasses ficam sem acesso direto aos mesmos. Métodos privados não são herdados pelas subclasses (são métodos exlusivos das classes que os declaram).
    - Uma subclasse pode ter acesso aos atributos privados da superclasse. Contudo, isso só pode ser feito através dos métodos públicos da superclasse
    - Se não existir um método público que permita acesso aos atributos da superclasse, as subclasses não poderão acessá-los.

Exemplificação - Private

Classe1

```
1  public class Classe1 {
2
3      private int atributo1;
4      private int atributo2;
5
6      private void metodo1(){
7      }
8
9      private void metodo2(){
10     }
11 }
```

Classe2

```
1  public class Classe2 extends Classe1{
2
3
4  }
```

ClasseTeste

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe2 Obje2 = new Classe2();
6
7          Obje2.
8      }
9
10 }
11
```

equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

Os membros private da Classe 1 NÃO fazem parte do escopo dos objetos da Classe2

Herança – Acesso a Métodos

Objetos da Classe A podem acessar os métodos públicos, mas não podem acessar o método descontaTarifa() – este é um método específico da classe B

Classe A

Classe B

```
class Conta {
    private double saldo;

    void deposita(double valor) {
        this.saldo += valor;
        this.descontaTarifa();
    }

    void saca(double valor) {
        this.saldo -= valor;
        this.descontaTarifa();
    }

    private descontaTarifa(){
        this.saldo -= 0.1;
    }
}
```

O que aconteceria se objetos da Classe A Pudessem acessar diretamente o método privado descontaTarifa da classe B

Acesso Direto (Público)

```java
public class Classe1 {

    public int atributo1 = 10;
    public int atributo2;

    public void metodo1(){
    }

    public void metodo2(){
    }
```

Classe1

Classe3

```java
public class Classe3{

    public int calculo;
    int atrClasse3;

    public int calculo(){

        Classe1 obj1 = new Classe1();

        atrClasse3 = obj1.atributo1;
        calculo = atrClasse3 * atrClasse3;
        return calculo;
    }
}
```

Acesso Direto (Público)

```
1  public class Teste {
2
3      static int resultado;
4
5      public static void main(String args[]){
6
7          Classe3 Obje3 = new Classe3();
8
9          resultado = Obje3.calculo();
10         System.out.println(resultado);
11     }
12 }
```

Classe Teste

Acesso Direto – Alteração de Valor

```java
public class Classe1 {

    public int senha = 123456;

}
```

Classe1

```java
public class Classe3{

    public void alteraValor(){

        Classe1 obj1 = new Classe1();
        System.out.println(obj1.senha);

        obj1.senha = 111111;
        System.out.println(obj1.senha);
    }
}
```

Classe3

```java
public class Teste {

    public static void main(String args[]){

        Classe3 Obje3 = new Classe3();
        Obje3.alteraValor();
    }
}
```

Classe Teste

Resultado da Execução

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe3 Obje3 = new Classe3();
6          Obje3.alteraValor();
7      }
8  }
9
```

Problems | Javadoc | Declaration | Console | Progress

<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\javaw.exe (27/03/2011 14:28:09)

```
123456
111111
```

Resultado da Execução

Valor de senha antigo: 123456
Novo valor de senha: 111111

Acesso Direto – Alteração de Valor

```
1  public class Classe1 {
2
3      private int senha = 123456;
4
5  }
```

Classe1

Classe3

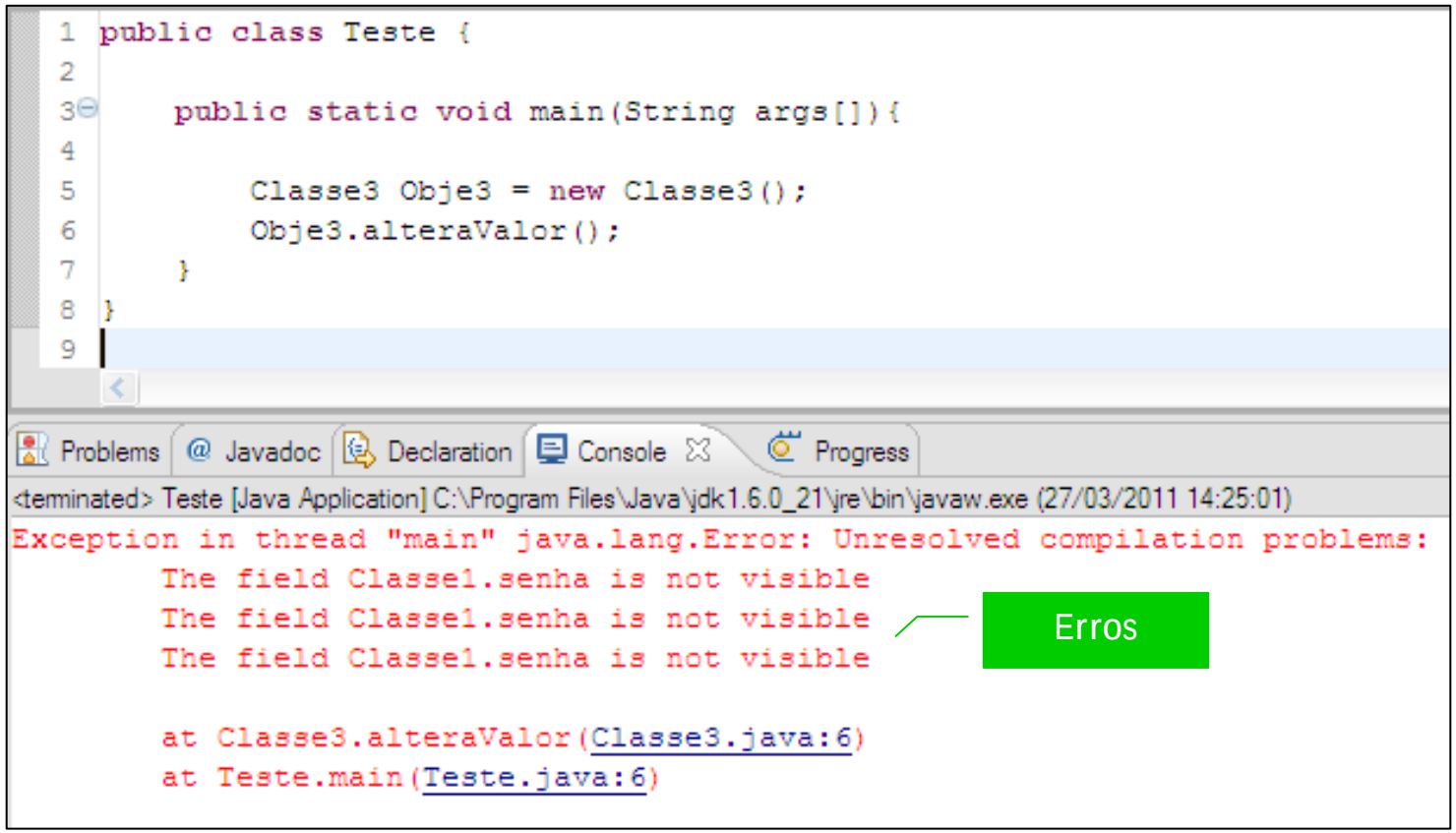
```
1  public class Classe3{
2
3      public void alteraValor(){
4
5          Classe1 obj1 = new Classe1();
6          System.out.println(obj1.senha);
7
8          obj1.senha = 111111;
9          System.out.println(obj1.
10     }
11 }
12
```

The field Classe1.senha is not visible

2 quick fixes available:

- Change visibility of 'senha' to 'default'
- Create getter and setter for 'senha'

Resultado da Execução



Visibilidade de Métodos

o método m() de A é herdado pelas as classes B e C

o método a() de A não é herdado pois é privado

o método z() de B é herdado pela classe C

```
class A {
    protected int x, y ;
    private int z ;
    public void m() { }
    private void a() { }
    public void p() { System.out.println(1) ; }
}

class B extends A {
    public float y ;
    public void z() { }
    public void p() { System.out.println(2) ; }
}

class C extends B {
    private double y ;
    public void p() { System.out.println(3) ; }
     public void a() { }
}
```

Visibilidade de Métodos

Métodos da Classe A

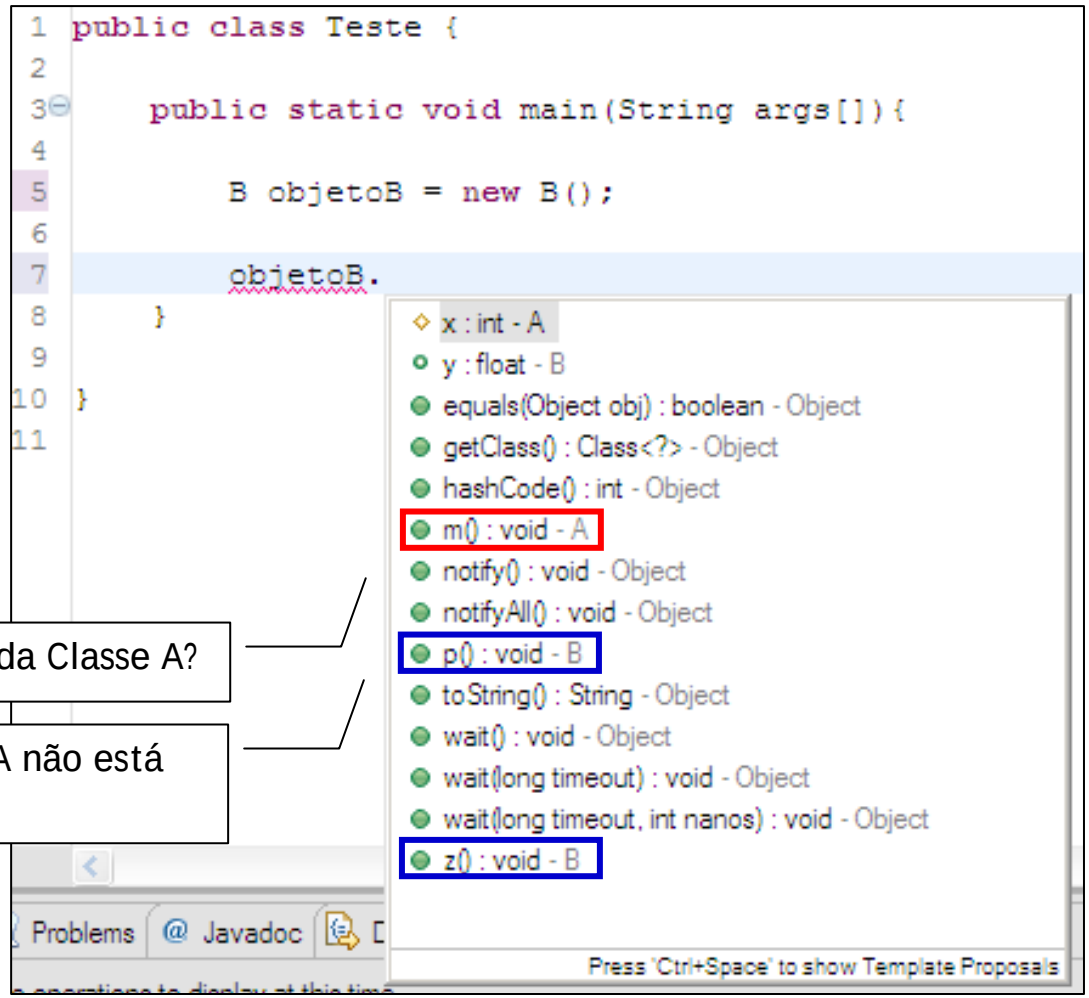
Públicos: m() e p()
Privados: a()

Métodos da Classe B

Públicos: z() e p()
Privados: -

Onde está o método p() herdada da Classe A?

Porque o método a() da classe A não está presente na lista?



Visibilidade de Métodos

o método p() de A não é herdado pela Classe B, pois é sobrescrito

o método p() de B não é herdado pela Classe C, pois é sobrescrito

```
class A {
    protected int x, y ;
    private int z ;
    public void m() { }
    private void a() { }
    public void p() { System.out.println(1) ; }
}

class B extends A {
    public float y ;
    public void z() { }
    public void p() { System.out.println(2) ; }
}

class C extends B {
    private double y ;
    public void p() { System.out.println(3) ; }
     public void a() { }
}
```

Visibilidade de Métodos

```
class A {
    protected int x, y ;
    private int z ;
    public void m() { }
    private void a() { }
    public void p() { System.out.println(1) ; }
}

class B extends A {
    public float y ;
    public void z() { }
    public void p() { System.out.println(2) ; }
}

class C extends B {
    private double y ;
    public void p() { System.out.println(3) ; }
     public void a() { }
}
```

o método a() de C não é considerado uma redefinição do método a() de A. Porque?

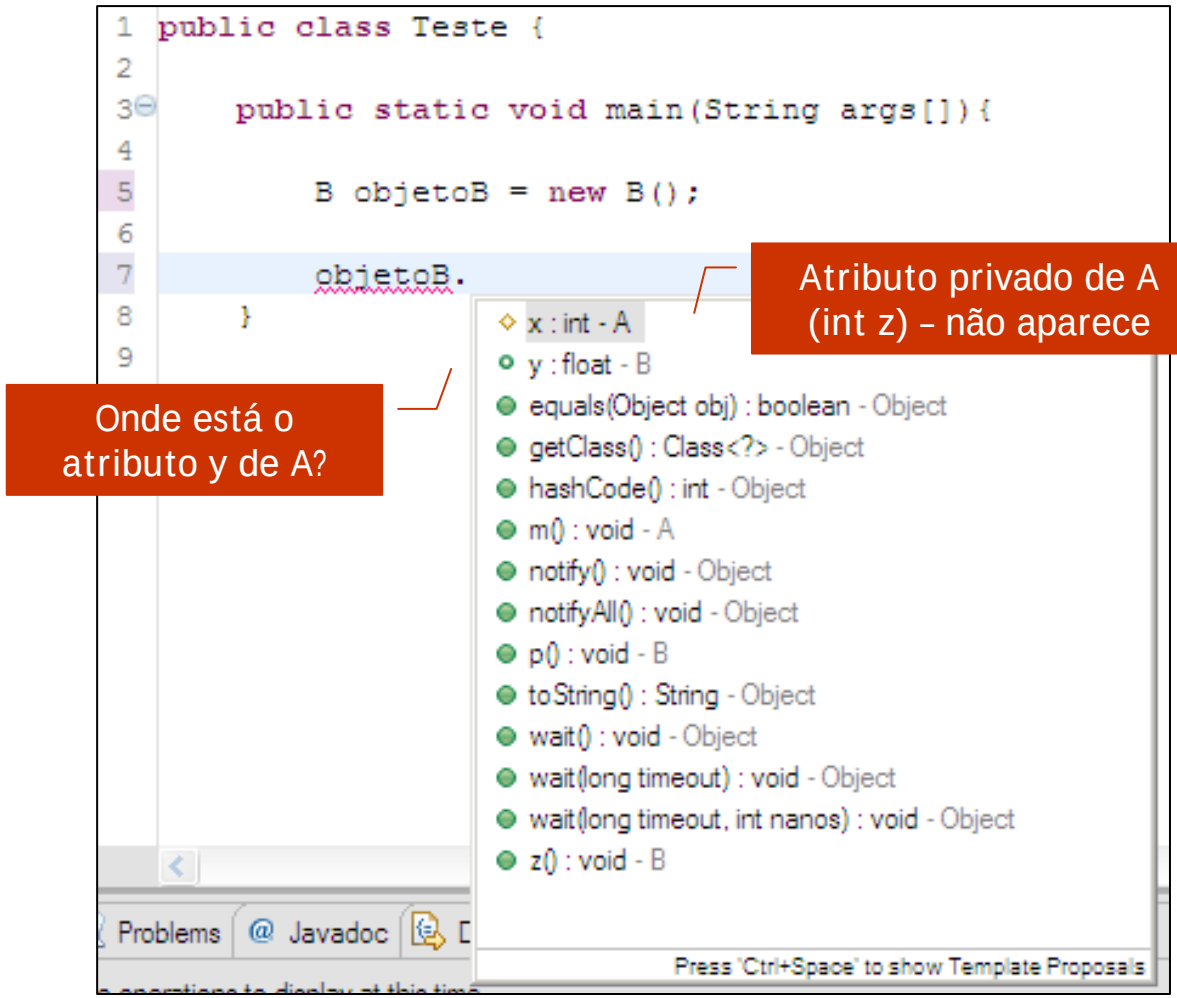
Visibilidade de Atributos

Atributos da Classe A

Protegidos: int x e int y
Privados: int z

Atributos da Classe B

Públicos: float y



Visibilidade de Atributos

Atributos da Classe1

Protegidos: int x e int y
Privados: int z

Atributos da Classe2

Públicos: float y

Atributos da Classe3

Private: double y

```java
public class Classe1 {

    protected int x;
    protected int y;
    private int z;

}
```

```java
public class Classe2 extends Classe1{

    protected float y;

}
```

```java
public class Classe3 extends Classe2{

    private double y;

}
```

Visibilidade de Atributos

```java
public class Classe1 {

    protected int x;
    protected int y;
    private int z;

}
```

```java
public class Classe2 extends Classe1{

    protected float y;

}
```

```java
public class Classe3 extends Classe2{

    private double y;

}
```

```java
public class Teste {

    public static void main(String args[]){

        Classe3 Obje3 = new Classe3();

        Obje3.
    }
}
```

x : int - Classe1
y : float - Classe2
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeout) : void - Object
wait(long timeout, int nanos) : void - Object

Visibilidade de Atributos

```java
public class Classe1 {

    protected int x;
    protected int y;
    private int z;

}
```

```java
public class Classe2 extends Classe1{

    private float y;

}
```

```java
public class Classe3 extends Classe2{

    private double y;

}
```

```java
public class Teste {

    public static void main(String args[]){

        Classe3 Obje3 = new Classe3();

        Obje3.
    }
}
```

- x : int - Classe1
- y : int - Classe1
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object
- wait(long timeout) : void - Object
- wait(long timeout, int nanos) : void - Object

Visibilidade de Atributos

Atributos da Classe1

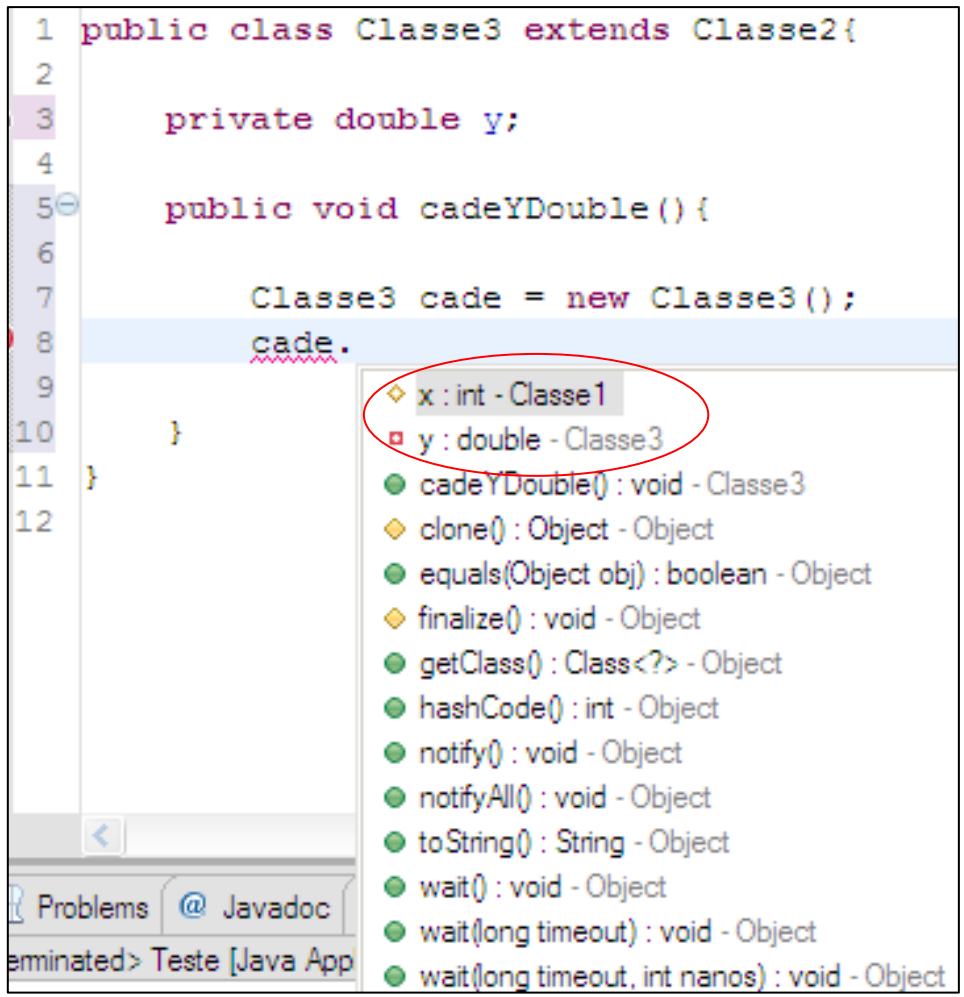
Protegidos: int x e int y
Privados: int z

Atributos da Classe2

Públicos: float y

Atributos da Classe3

Private: double y



Visibilidade de Atributos

Atributos da Classe1

Protegidos: int x e int y
Privados: int z

Atributos da Classe2

Públicos: float m

Atributos da Classe3

Private: double t

```
1  public class Teste {
2
3      public static void main(String args[]){
4
5          Classe3 Obje3 = new Classe3();
6
7          Obje3.
8      }
9  }
10
```

- m : float - Classe2
- x : int - Classe1
- y : int - Classe1
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object
- wait() : void - Object

Visibilidade de Atributos

Atributos da Classe1

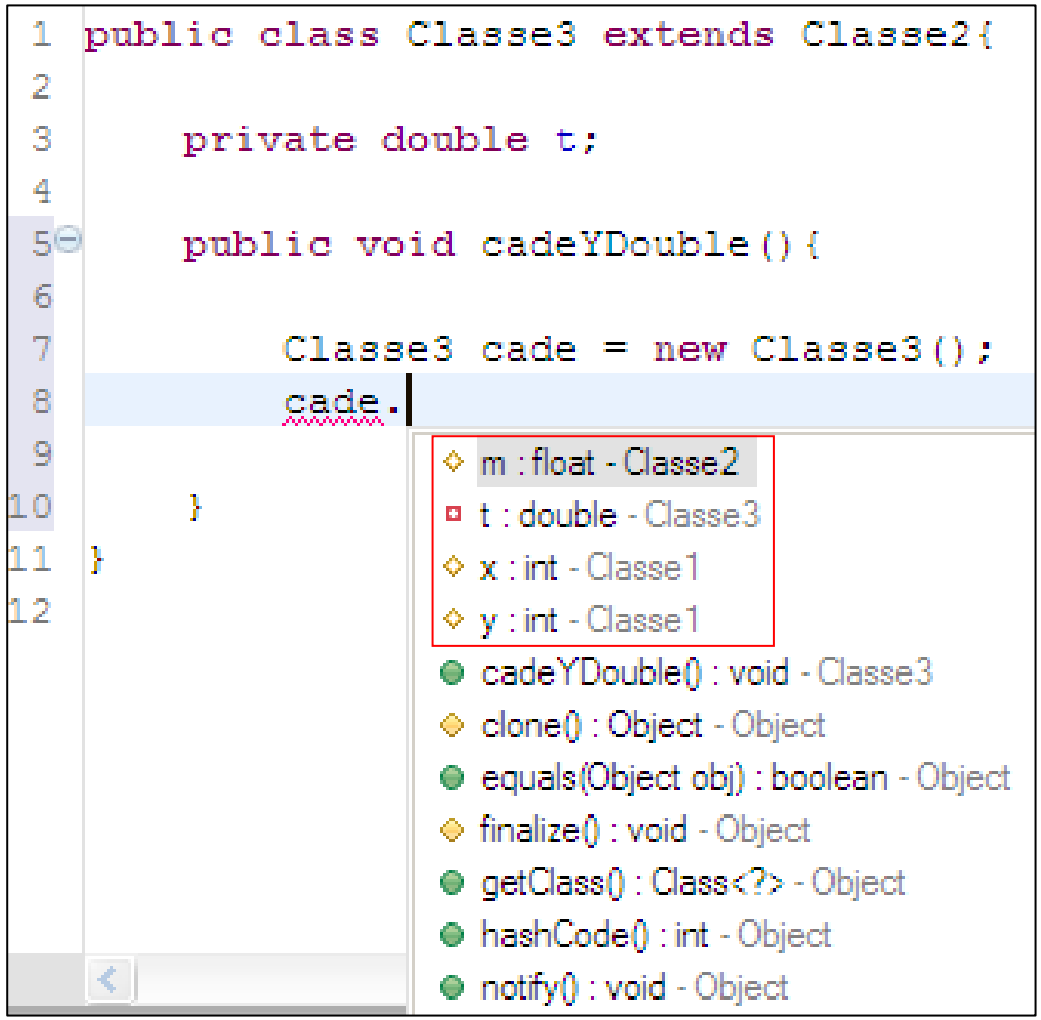
Protegidos: int x e int y
Privados: int z

Atributos da Classe2

Públicos: float m

Atributos da Classe3

Private: double t



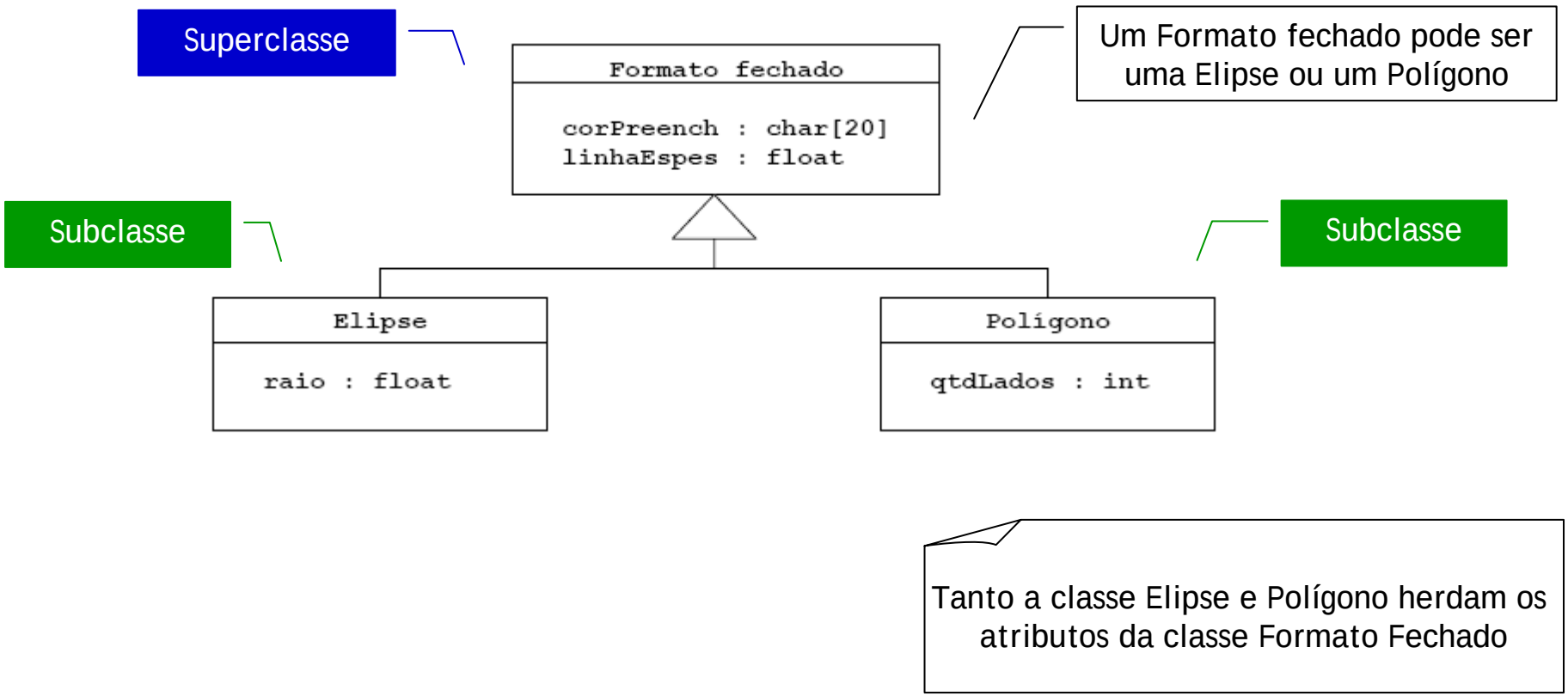
Tipos de Herança

Herança - Tipos

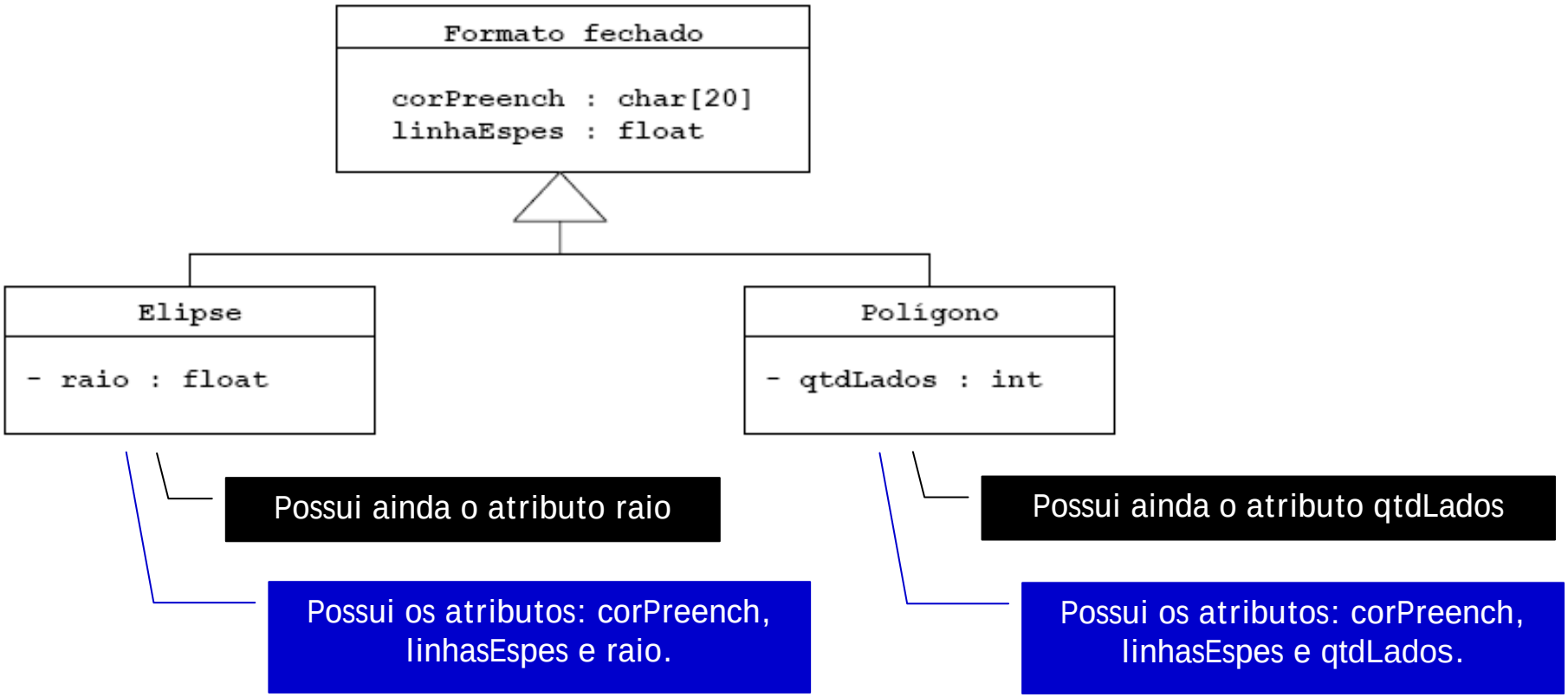
- Tipos de Herança
  - Herança Simples
  - Herança Múltipla

Herança Simples

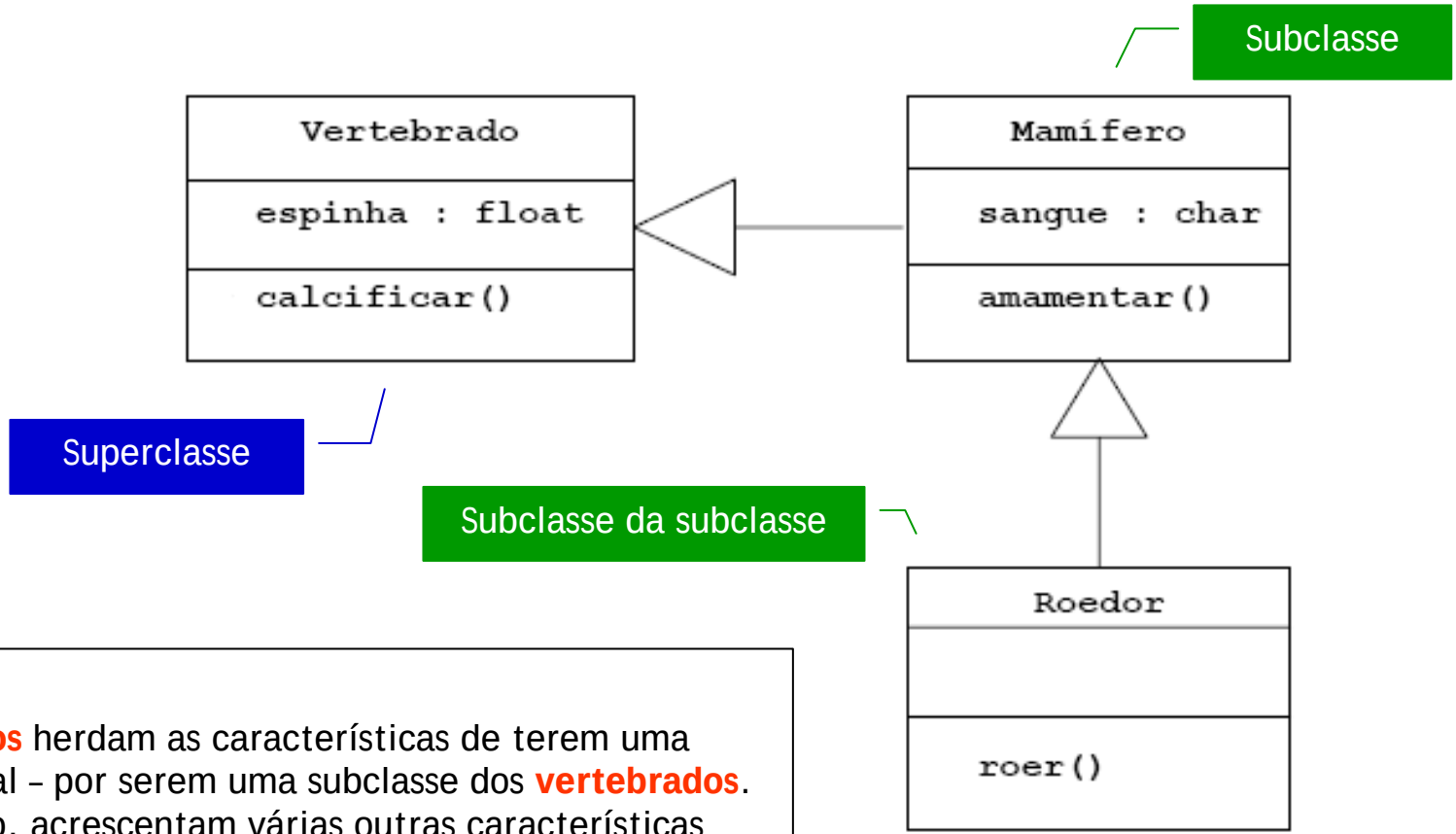
- Característica
  - A subclasse herda os atributos e métodos de uma única superclasse



Herança Simples



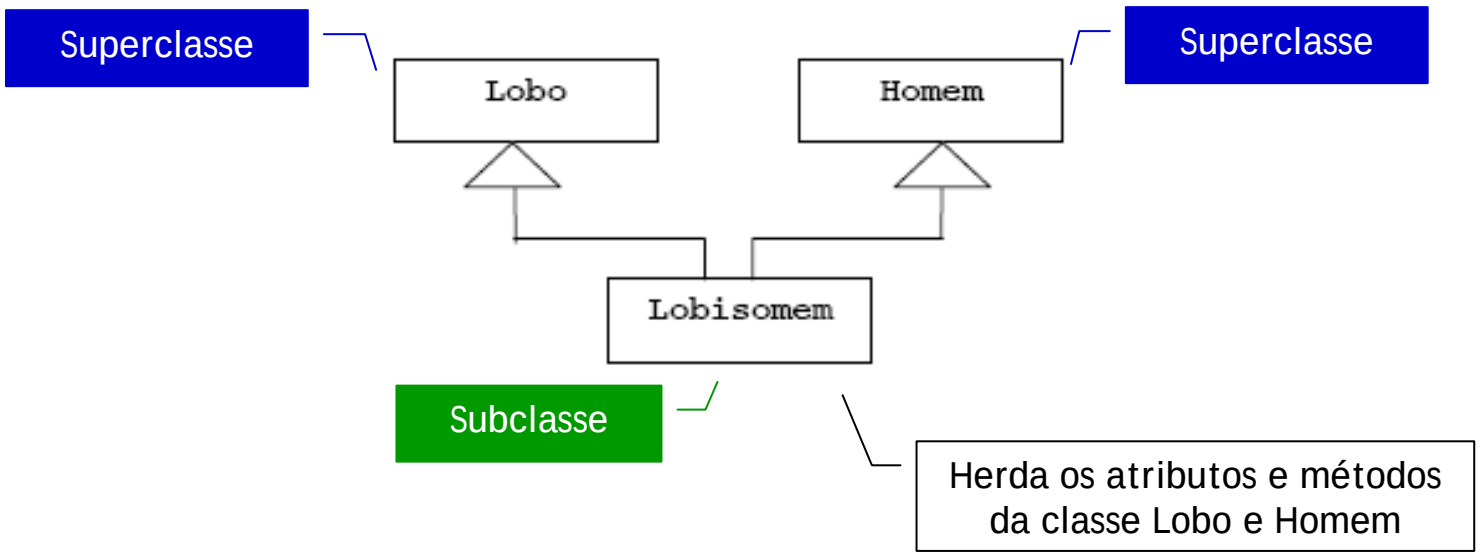
Herança Simples



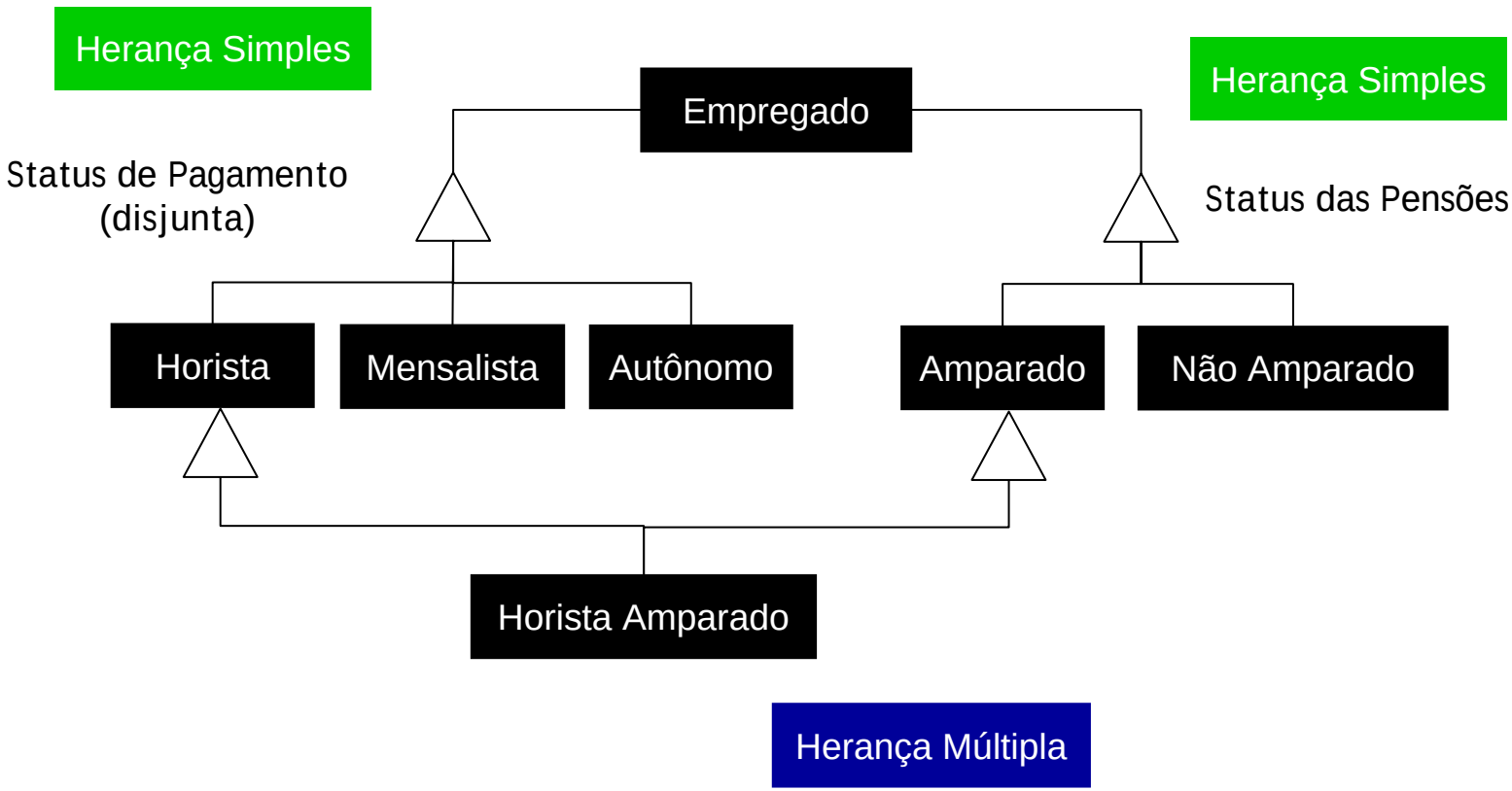
Mamíferos herdam as características de terem uma espinha dorsal – por serem uma subclasse dos **vertebrados**. Além disso, acrescentam várias outras características (sangue quente, amamentar, etc.). Os **roedores**, por sua vez, herdam todas as características dos **vertebrados** e **mamíferos** e acrescentam outras

Herança Múltipla

- Definição
  - Pela herança múltipla, a subclasse (também chamada de classe de junção) herda os atributos e métodos de duas ou mais superclasses.



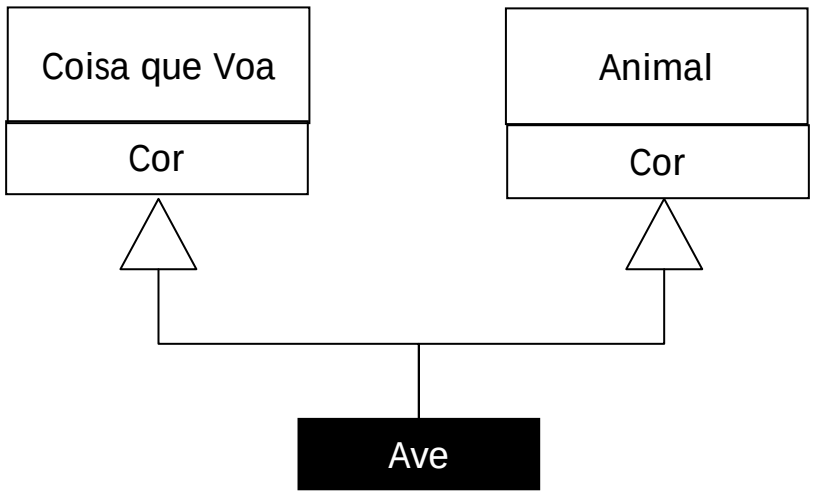
Herança Simples/Múltipla



Problemas – Herança Múltipla

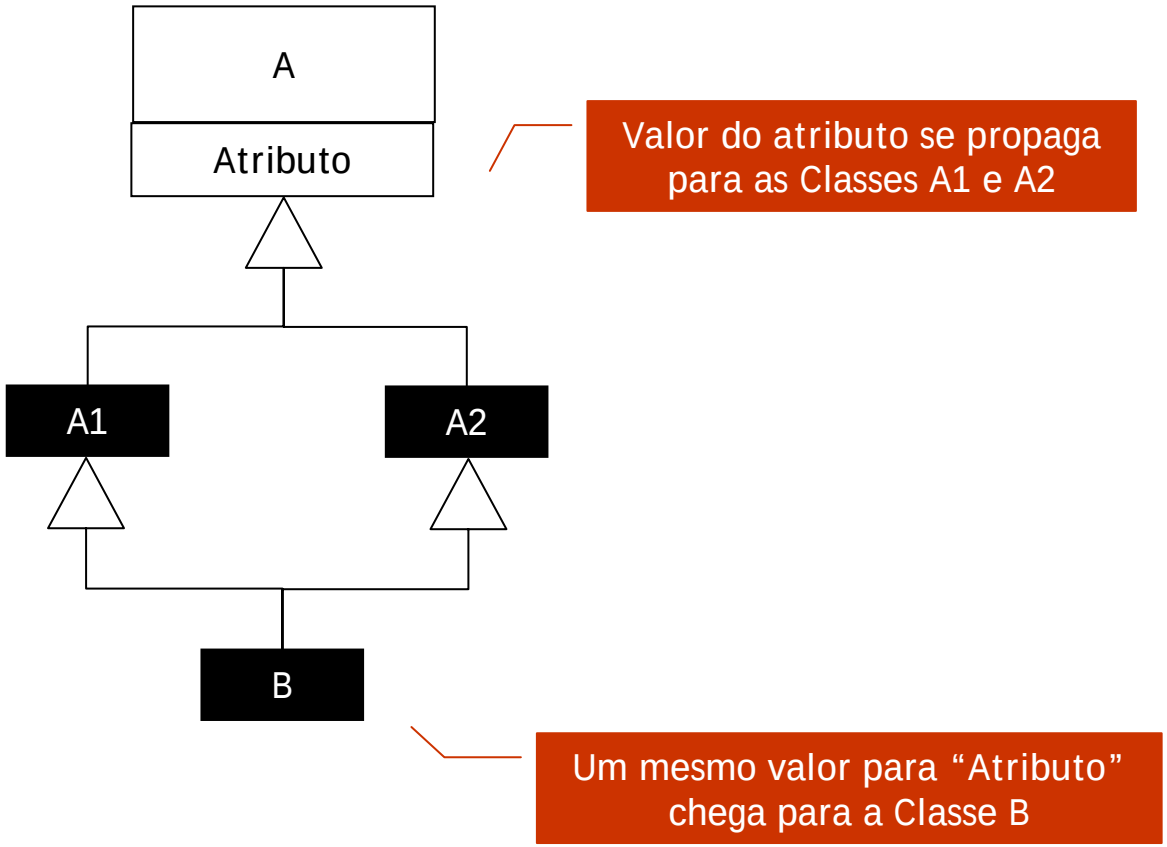
- Conceitualmente
  - Importante para modelar o mundo real de forma mais precisa
  - Contudo, pode levar a erros de implementação
  - Nem todas as linguagens OO suportam herança múltipla
    - Java não suporta herança múltipla
    - Pode-se marcarar herança múltipla usando interfaces

Problemas – Herança Múltipla

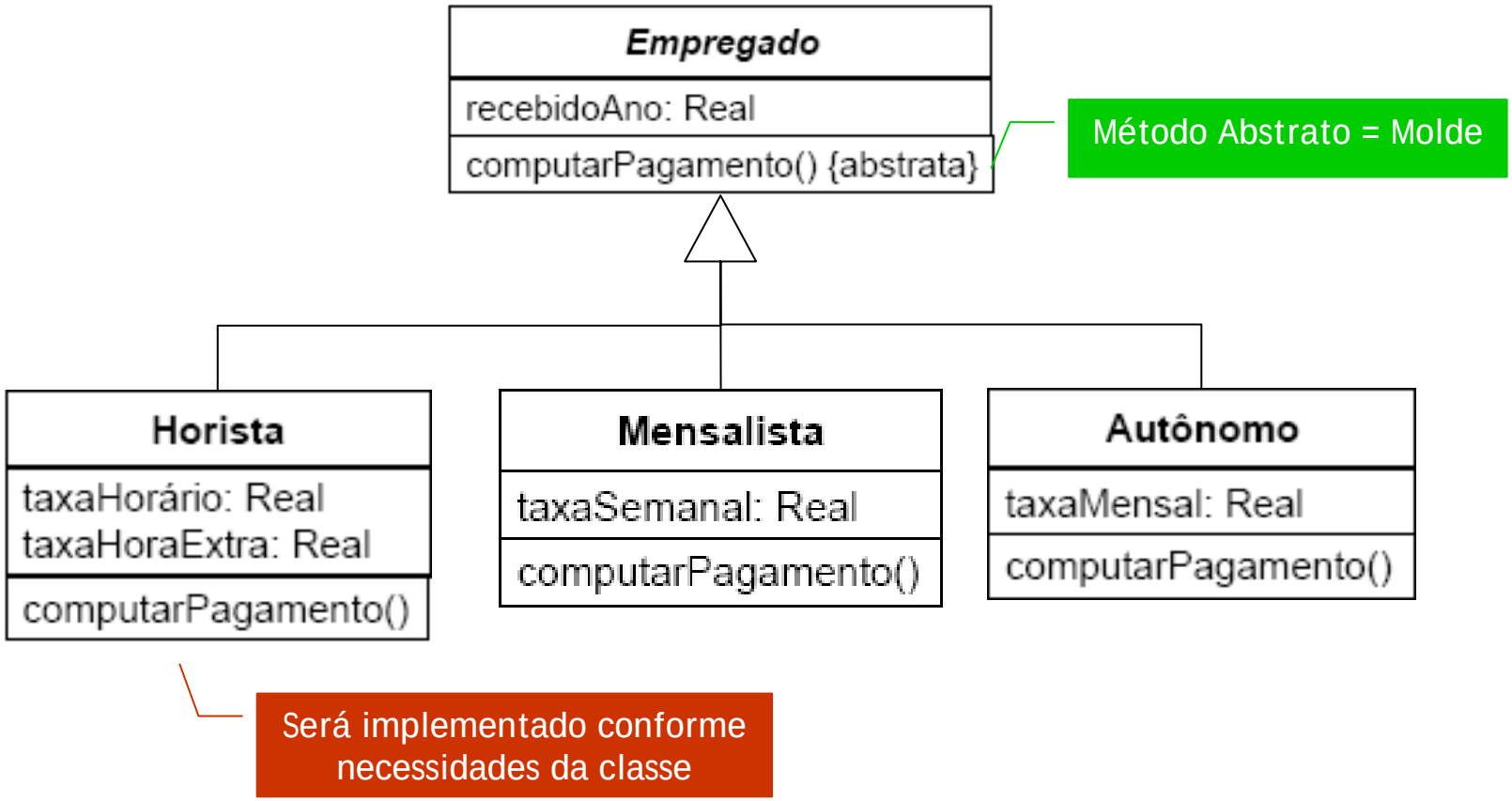


Qual a cor da ave?

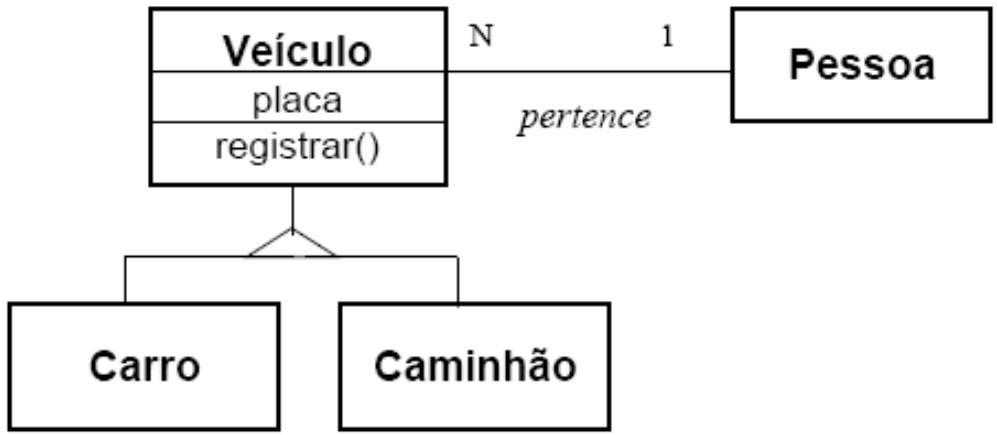
Problemas – Herança Múltipla



Herança


Empregado
recebidoAno: Real
computarPagamento() {abstrata}
Método Abstrato = Molde
Horista
taxaHorário: Real
taxaHoraExtra: Real
computarPagamento()
Mensalista
taxaSemanal: Real
computarPagamento()
Autônomo
taxaMensal: Real
computarPagamento()
Será implementado conforme necessidades da classe

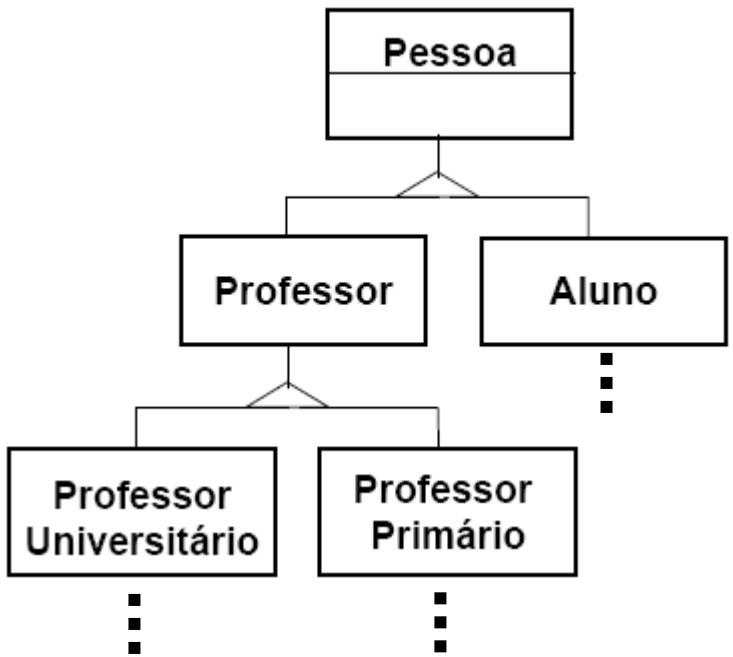
Herança e Relacionamentos



Todo carro e caminhão pertence a uma pessoa

Herança

- Não há limites para o número de níveis na hierarquia de herança



Hierarquia de Classes e Herança

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3