Programação de Computadores 2

Programação Orientada a Objetos

Herança

Material adaptado das aulas da Prof.ª. Kécia Ferreira (Cefet-MG) e Prof. Moisés Ramos (IFMG)

Daniel Hasan Dalip
hasan@decom.cefetmg.br

Na Aula Anterior...

Coleções

- Lista
  - ArrayList
  - Stack
  - LinkedList
- Conjuntos
  - HashSet
- Mapas
  - HashMap

Na Aula de Hoje...
Herança

- Reuso

- Herança
  - Redefinição de métodos
  - Visibilidade protected de atributos/métodos ao utilizar herança
- Exceções e Herança

Reuso

- Reuso de software: na construção de software, é importante a existência de recursos que permitam a reutilização de software no todo ou em parte.

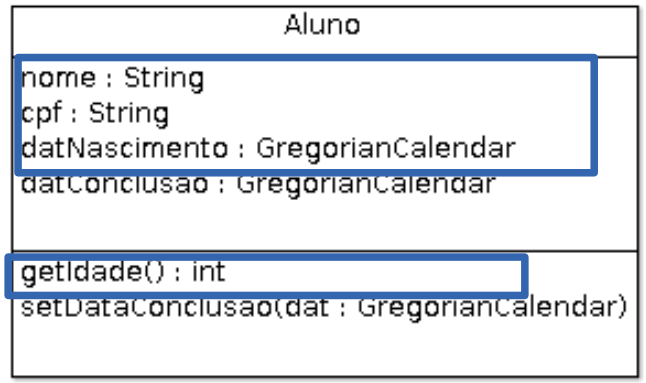
- Razões:
  - Evita-se “reinventar a roda”;
  - Economiza-se tempo de desenvolvimento;
  - Aumenta-se a qualidade do software, pois partes reutilizadas já foram testadas e depuradas;
  - Diminui-se o custo do software.

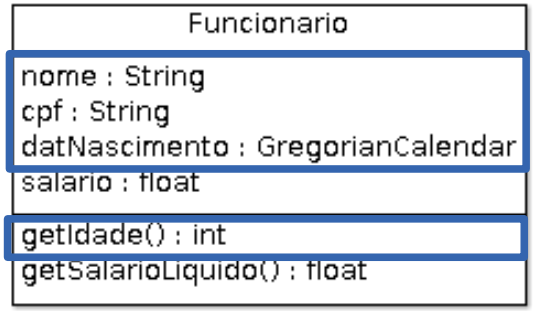
Reuso

- O paradigma estruturado é pobre em recursos que propiciem o reuso de software.
- Por exemplo, em um paradigma estruturado se tivéssemos que armazenar o nome, cpf e data de nascimento dos alunos e funcionários.
- Além disso, de um método para calcular a idade do aluno e funcionário
- Seria necessário duas funções: uma para exibir a idade do funcionário e outra do aluno
  - Pois estariam em estrutura de dados diferentes

Reuso

- Em programação orientada a objetos é possível fazer o uso da herança (generalização):

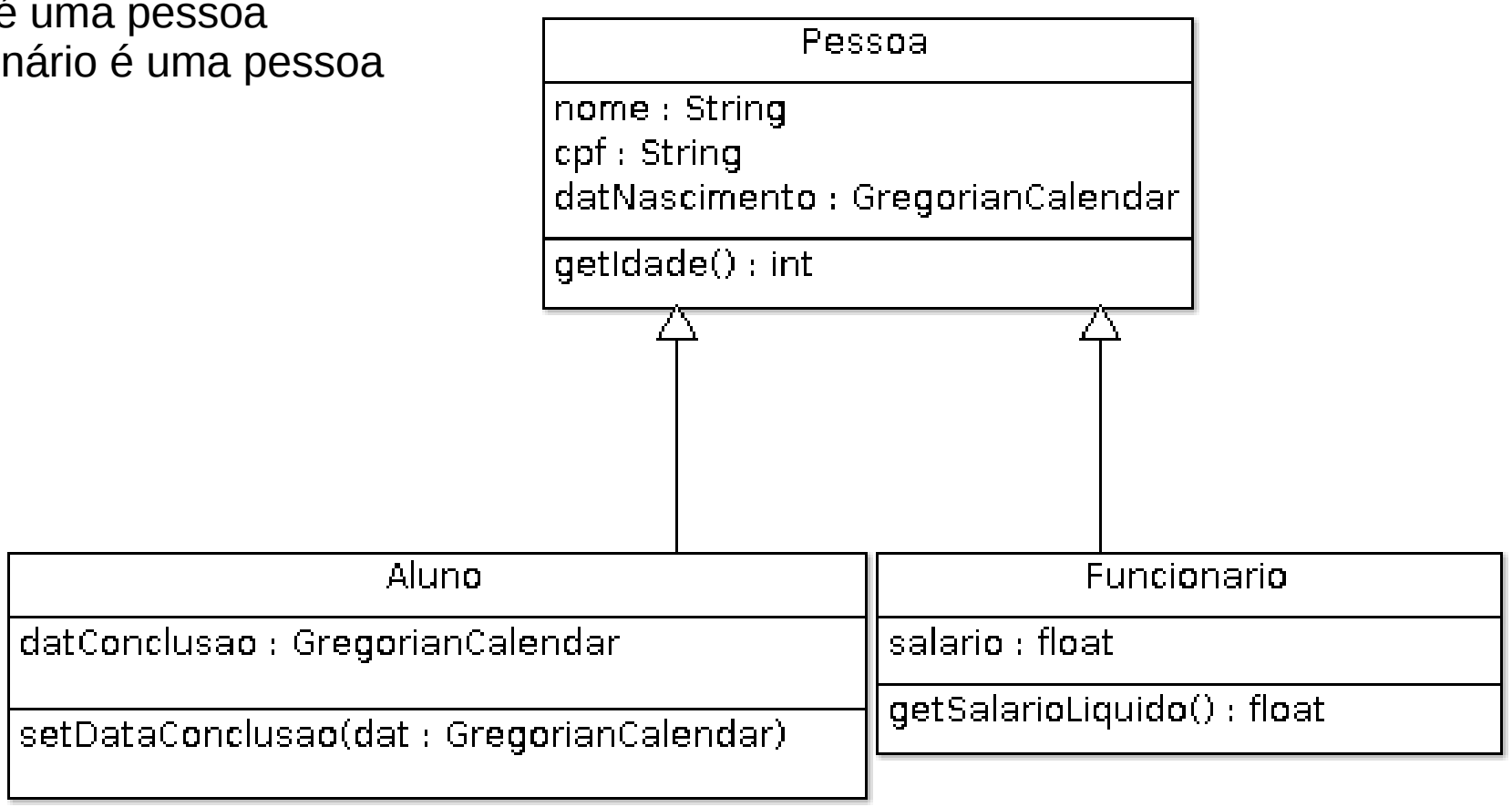




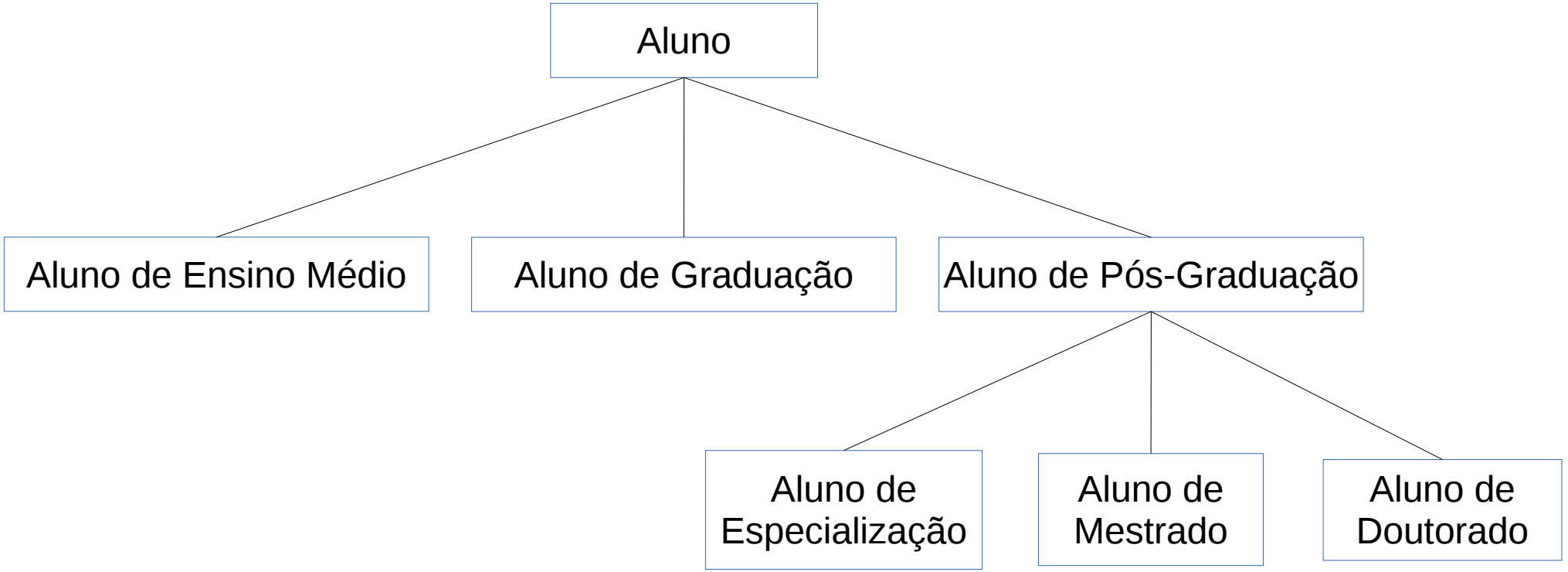
Reuso

- Em programação orientada a objetos é possível fazer o uso da herança (generalização):

Aluno é uma pessoa
Funcionário é uma pessoa



Reuso


Aluno
Aluno de Ensino Médio
Aluno de Graduação
Aluno de Pós-Graduação
Aluno de Especialização
Aluno de Mestrado
Aluno de Doutorado

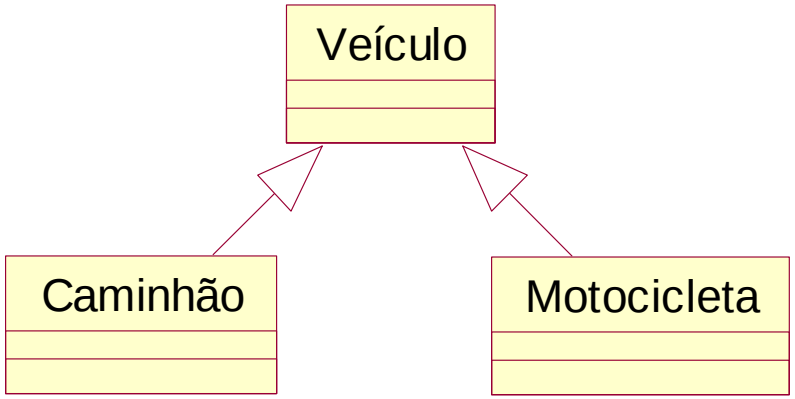
Herança

- Estes relacionamentos são do tipo “é um”.

- Um relacionamento do tipo “é um” indica que uma entidade possui (herda) características (atributos) e comportamento (métodos) da outra.

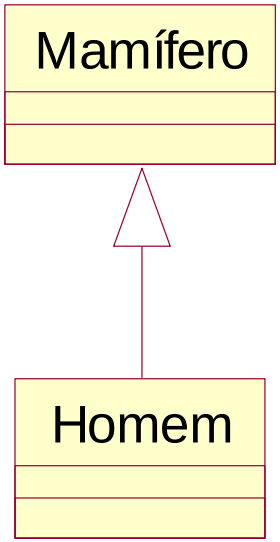
- A programação orientada por objetos possui um recurso poderoso para implementar este tipo de reuso: herança.

Herança
Exemplos



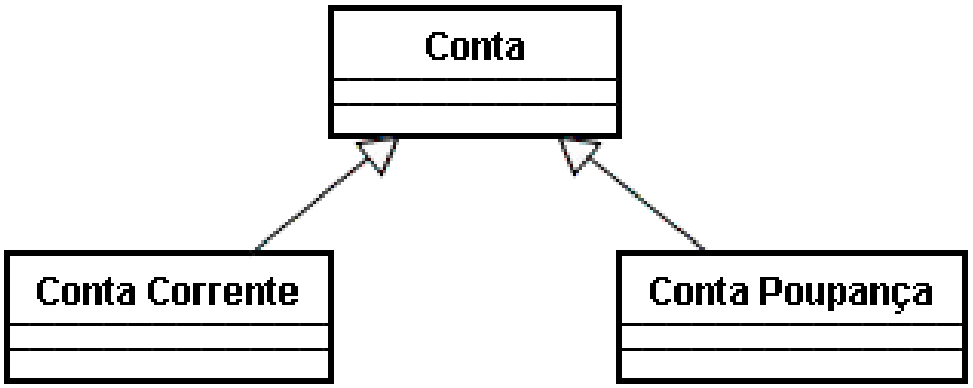
- Veículo é superclasse de Caminhão e de Motocicleta.
- Caminhão e Motocicleta são subclasses de Veículo.
- Caminhão e Motocicleta herdam as definições da classe Veículo.

Herança
Exemplos



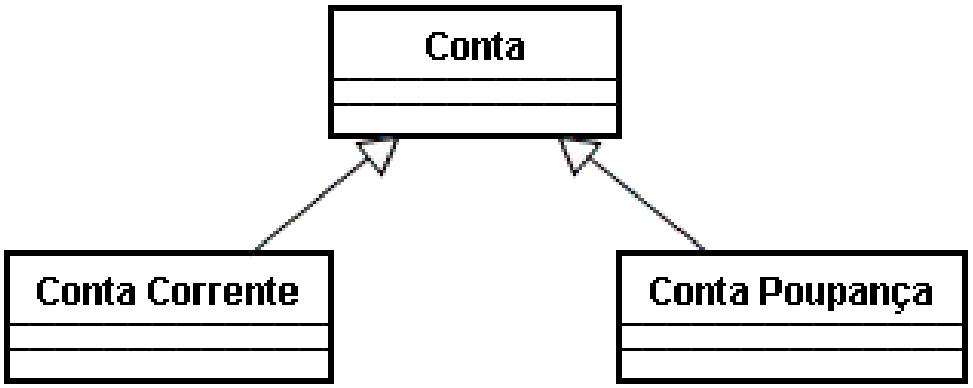
- Mamífero é superclasse de Homem.
- Homem é subclasse de Mamífero.
- Homem herda definições da classe Mamífero.

Herança
Exemplo – Conta Corrente



- *Conta* é **superclasse** de *Conta Corrente* e de *Conta Poupança.*
- *Conta Corrente* e *Conta Poupança* são **subclasses** de *Conta.*

Herança
Exemplo – Conta Corrente



- Ou seja:
  - Conta Corrente e Conta Poupança **herda características e comportamentos** de Conta.
    - Caracteristicas: atributos
    - Comportamentods: métodos

Herança
Implementação

A palavra chave **extends** indica herança em Java.

- Ao criar a classe B que será subclasse de A:

```
public class B extends A
```

Indica que a classe **B** herda os métodos e atributos da classe **A**

```
public class A {
(...)
}
```

```
public class B extends A {
(...)
}
```

Herança
Visibilidade: Protected

- Nos relacionamentos de herança, o modificador de acesso **protected** indica que o atributo ou método é visível nas subclasses.

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

```
public class B extends A {
   private int k;
}
```

- A classe **B** tem acesso e pode modificar os atributos x e y.
- A classe **B** não pode modificar/acessar (diretamente) o atributo z

Herança
Construtor

- Para invocar o método construtor da superclasse, utiliza-se a instrução **super** no construtor da subclasse :

```
public class A {
   protected int x,y;
   private int z;
   public A(int x, int y,
            int z){
      this.x = x;
      this.y = y;
      this.z = z;
   }
}
```

```
public class B extends A {
   private int k;
   public B(int a, int b,
            int c, int d){
      super(a,b,c);
      this.k = d;
   }
}
```

Herança - Redefinição de Método

Se um método é implementado na subclasse B com a mesma assinatura de um método da superclasse A, diz-se que o método foi **redefinido**. Neste caso, o método que será executado para um objeto da classe B será aquele definido na classe B.

```
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```
public class B extends A {
    private int k;
    public B(int a, int b,
             int c, int d){
        super(a,b,c);
        this.k = d;
    }
    public String toString(){
        return "x: "+x+" y: "+y+
            "k: "+k+"\n"
            "B não tem acesso à z";
    }
}
```

```java
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```java
public class B extends A {
    private int k;
    public B(int a, int b,
             int c, int d){
        super(a,b,c);
        this.k = d;
    }
    public String toString(){
        return "x: "+x+" y: "+y+
            "k: "+k+"\n"
            "B não tem acesso à z";
    }
}
```

```java
public class TesteHeranca {
    public static void main(String[] args){
      B obj1 = new B(10,20,30,40);
        System.out.println(obj1);
    }
}
```

```java
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```java
public class B extends A {
    private int k;
    public B(int a, int b,
             int c, int d){
        super(a,b,c);
        this.k = d;
    }
    public String toString(){
        return "x: "+x+" y: "+y+
            "k: "+k+"\n"
            "B não tem acesso à z";
    }
}
```

```java
public class TesteHeranca {
    public static void main(String[] args){
        B obj1 = new B(10,20,30,40);
        System.out.println(obj1);
    }
}
```

X: 10 y:20 k:40
B não tem acesso à z

Herança
Invocando um método da superclasse

- Na classe B, se usamos super.nome_do_metodo(), podemos chamar ao método definido na superclasse.

```
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```
public class B extends A {
    private int k;
    public B(int a, int b,
             int c, int d){
        super(a,b,c);
        this.k = d;
    }
    public String toString(){
        return super.toString()+
               "k: "+k;
    }
}
```

Herança
Invocando um método da superclasse

- Na classe B, se usamos super.nome_do_metodo(), podemos chamar ao método definido na superclasse.

```
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```
        this.k = d;
    }
    public String toString(){
        return super.toString()+
               "k: "+k;
    }
}
```

Só é necessário utilizar a cláusula super
Quando há um método na subclasse
Com o mesmo nome.

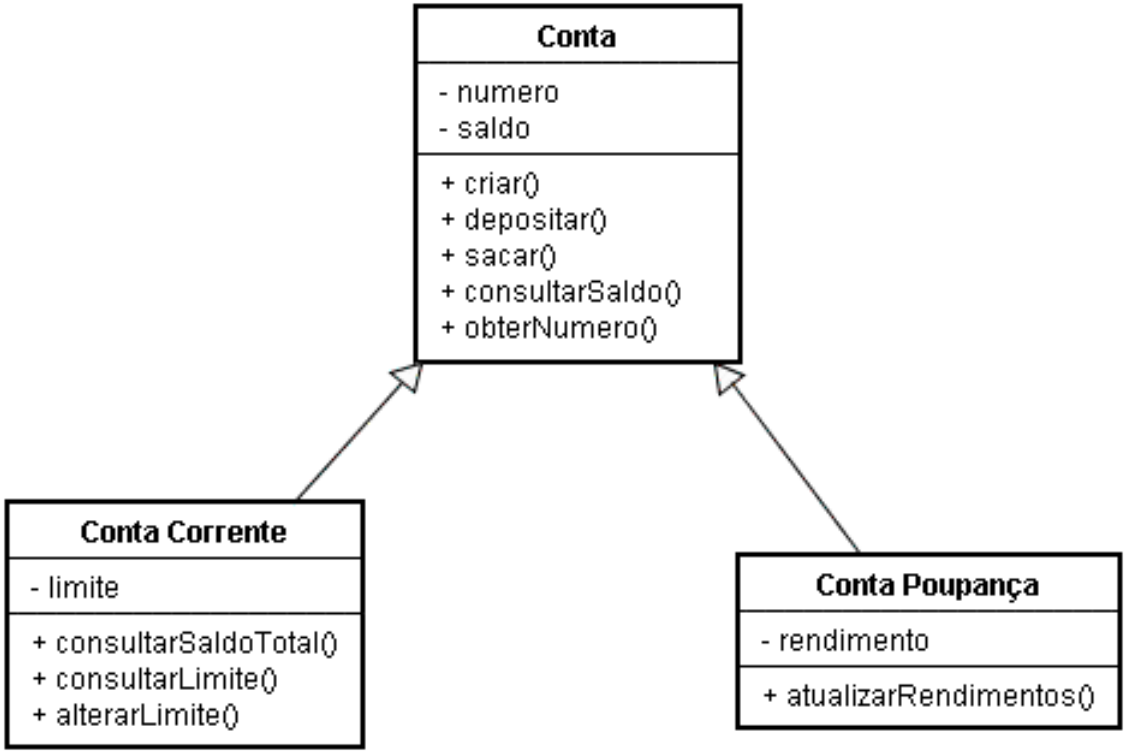
```java
public class A {
    protected int x,y;
    private int z;
    public A(int x, int y,
             int z){
        this.x = x;
        this.y = y;
        this.z = z;
    }
    public String toString(){
        return "x: "+x+" y: "+
               y+" z:"+z;
    }
}
```

```java
public class B extends A {
    private int k;
    public B(int a, int b,
             int c, int d){
        super(a,b,c);
        this.k = d;
    }
    public String toString(){
        return super.toString()+
               "k: "+k;
    }
}
```

```java
public class TesteHeranca {
    public static void main(String[] args){
        B obj1 = new B(10,20,30,40);
        System.out.println(obj1);
    }
}
```

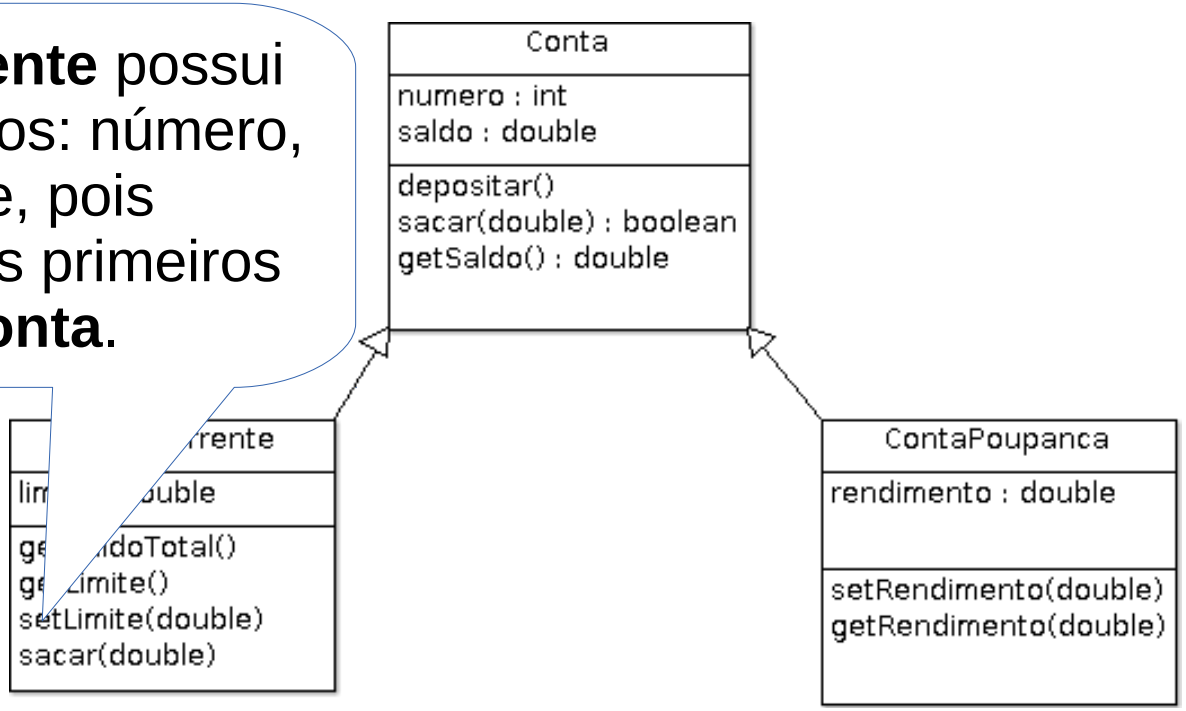
```
x: 10 y:20 z:30 k:40
```

Herança
Exemplo – Conta Corrente



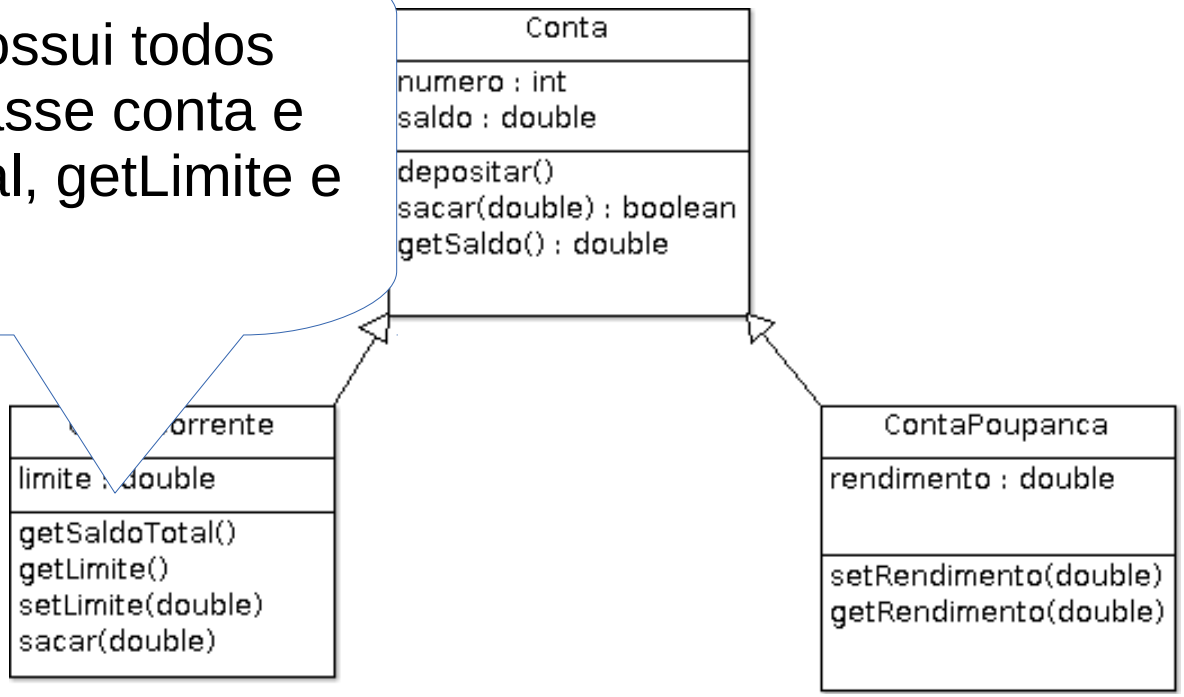
Herança
Exemplo – Conta Corrente

Conta Corrente possui como atributos: número, saldo e limite, pois herda os dois primeiros da classe **Conta**.



Herança
Exemplo – Conta Corrente

Conta Corrente possui todos os métodos da classe conta e mais getSaldoTotal, getLimite e setLimite



Exemplo – Conta Corrente

```
public class Conta {
    protected long numero;
    protected double saldo;
    public Conta(long num) {
           numero = num;
        saldo = 0;
    }
    public void depositar(double v){
        if (v>0)
            saldo = saldo + v;
    }
    public boolean sacar(double v){
        if( v>0 && ((saldo-v) >= 0) ){
          saldo = saldo - v;
          return true;
        }
        else
            return false;
     }
     /**implementação do getSaldo e getNumero**/
    (...)
}
```

Exemplo – Conta Corrente

```
public class Conta {
    (…)
    public String toString(){
        return "Número: "+numero+" saldo: "+saldo;
    }
}
```

```
public class ContaCorrente extends Conta{
      private double limite;
      public ContaCorrente(long num, double limite) {
         super(n);
         if (limite > 0)
            this.limite = limite;
    }
    public void setLimite(double l){
       if (l>0)
           limite = l;
    }
    (...)
}
```

```
public class ContaCorrente  extends Conta{
    (...)
     public boolean sacar(double v){
        if (v>0 && ((this.getSaldoTotal() - v ) >= 0) ){
            saldo = saldo - v;
            return true;
        }
        else
            return false;
     }
     public double getSaldoTotal(){
         return(saldo + limite);
     }
    public String toString(){
        return super.toString()+
                "\nlimite: "+limite+" Saldo Total:"+getSaldoTotal();
    }
}
```

```
public class MainBanco {
     public static void main (String[] args){
        ContaCorrente minhaConta;
        minhaConta = new ContaCorrente(12345, 300);
        System.out.println(minhaConta);
        minhaConta.setLimite(200);
        System.out.println(minhaConta);
        minhaConta.depositar(300);
        if (minhaConta.sacar(200)) {
              System.out.println("** após saque **");
              System.out.println(minhaConta);
        }
        else
           System.err.println("Não foi possível realizar a operação."+
                              "Saldo total disponível é de " +
                              minhaConta.getSaldoTotal());

    }
}
```

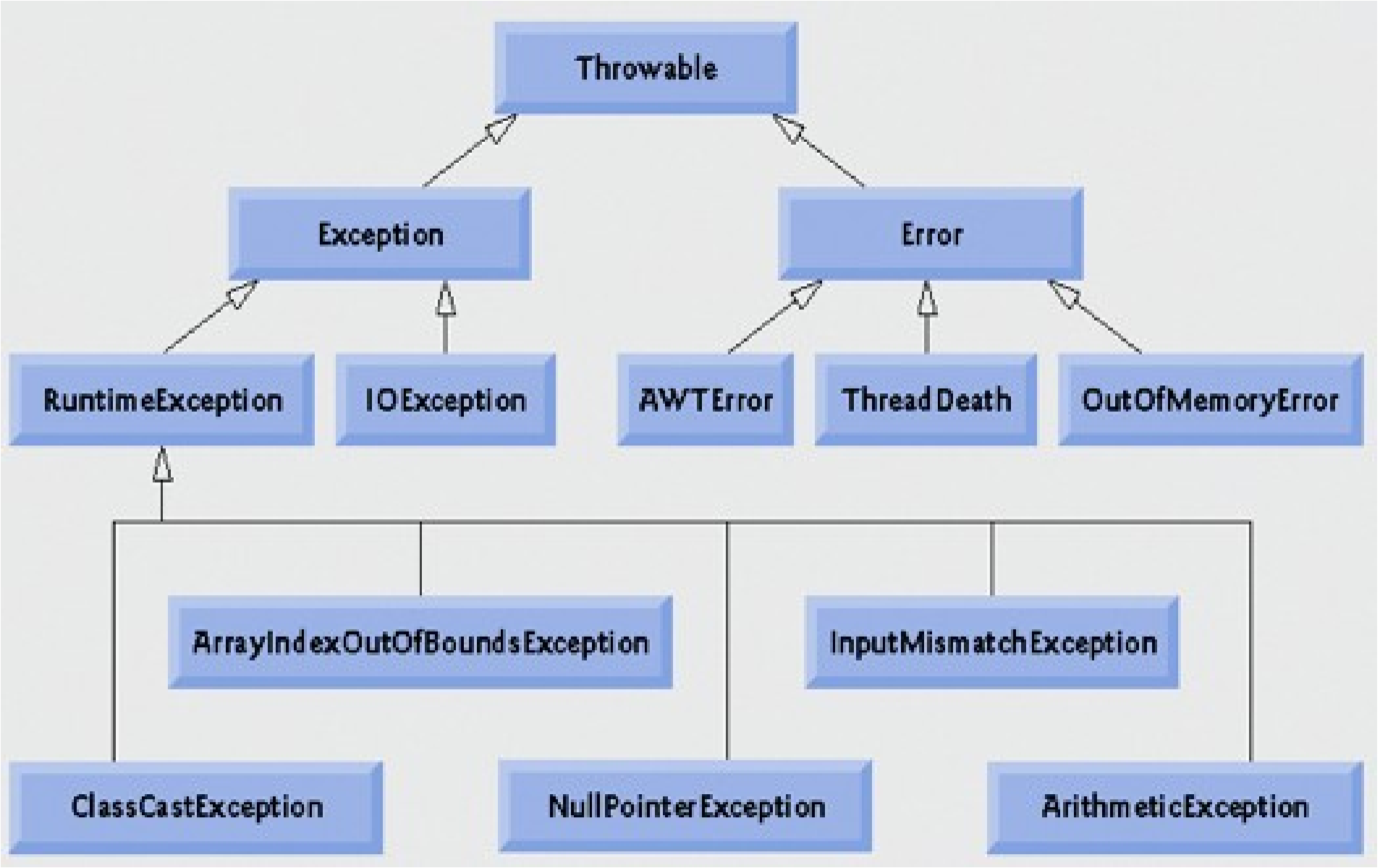
Classe Object

- Em Java, toda classe é implicitamente herdeira da classe `Object`.

- Desta forma, os métodos definidos nesta classe são herdados por toda classe que é criada.

- Dentre os métodos da classe Object, os mais usados são:
  - equals(Object o): verifica se um objeto é igual ao outro
  - toString(): exibe o conteúdo do objeto no formato de uma String

Hierarquia de Exceções



Tratamento de Exceções
Sintaxe – Try catch e finally

```
double[] a = new double[6];
try{
      a[5] = 10;
      a[3] = 50/2;
      System.out.println("Hi!");
  }
  catch(Exception e){
      System.out.println("Exceção!");
  }
  finally {
      System.out.println("Goodbye!");
  }
```

Funcionará:

ArrayIndexOutOfBoundsException e **ArithmeticException** são subclasses de **Exception**

Tratamento entre classes e subclasses de Exception

```
try{
      (...)
  }
  catch(Exception e1){
      System.out.println("Exceção!");
  }
  catch(ArithmeticException e2){
      System.out.println("Arithmetic Exception!");
  }
  finally {
      System.out.println("Goodbye!");
  }
```

Erro:
Este código nunca será executado!

Tratamento entre classes e subclasses de Exception

```
try{
      (...)
  }
  catch(ArithmeticException e2){
      System.out.println("Arithmetic Exception!");
  }
  catch(Exception e1){
      System.out.println("Exceção!");
  }

  finally {
      System.out.println("Goodbye!");
  }
```

Funcionará:

Nesta ordem, inicialmente serão tratadas as exceções **ArtimeticException** e, logo após, as demais exceções que são **Exception**

Tratamento entre classes e subclasses de Exception

```
try{
      (...)
  }
  catch(ArithmeticException e2){
      System.out.println("Arithmetic Exception!");
  }
  catch(Exception e1){
      System.out.println("Exceção!");
  }

  finally {
      System.out.println("Goodbye!");
  }
```

Funcionará:

Nesta ordem, inicialmente serão tratadas as exceções **ArtimeticException** e, logo após, as demais exceções que são **Exception**

Bibliografia

BARNES, David; KÖLLING, Michael.. *Programação Orientada a Objetos com Java: Uma introdução prática usando o BlueJ*. São Paulo: Pearson Prentice Hall, 4ª ed., 2009.

Deitel, H. M.; Deitel, P. J. *Java - Como Programar*. 8 ed. Prentice-Hall, 2011. Capítulos 7 e 8.