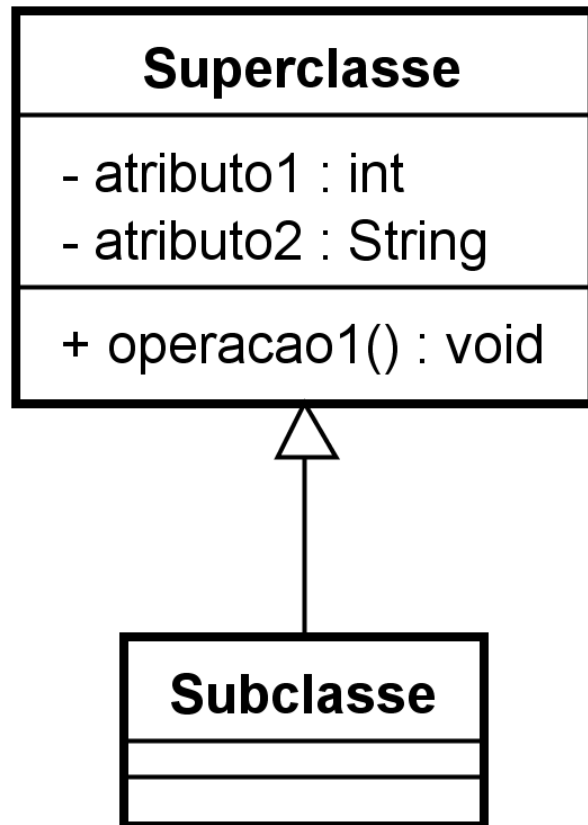


# Herança

# Herança

- Uma forma de reuso de software em que uma nova classe é criada absorvendo os membros (atributos e métodos) de uma classe já existente.
- Ao criar uma classe, ao invés de declarar membros completamente novos, é possível definir que a nova classe deve herdar os membros de uma classe já existente
  - A classe já existente é chamada de *superclasse*, *classe pai* ou *classe base*
  - A nova classe é chamada de *subclasse*, *classe filha* ou *classe derivada*
- A subclasse pode adicionar seus próprio membros
  - A subclasse pode possuir os mesmos comportamentos de sua superclasse mas pode também adicionar comportamento específico.

# Herança – Representação em UML



- A herança é uma forma de reuso
- A subclasse é mais específica que a sua superclasse, representando um grupo de objetos mais especializado
  - A herança também é conhecida como “especialização”

```

public class Calculadora {
    private double memoria;

    public double getMemoria() {
        return memoria;
    }

    public void setMemoria(double mem) {
        this.memoria = mem;
    }

    public double somar(double op1, double op2) {
        return op1+op2;
    }

    public double subtrair(double op1, double op2) {
        return op1-op2;
    }

    public double multiplicar(double op1, double op2) {
        return op1*op2;
    }

    public double dividir(double op1, double op2) {
        return op1/op2;
    }
}

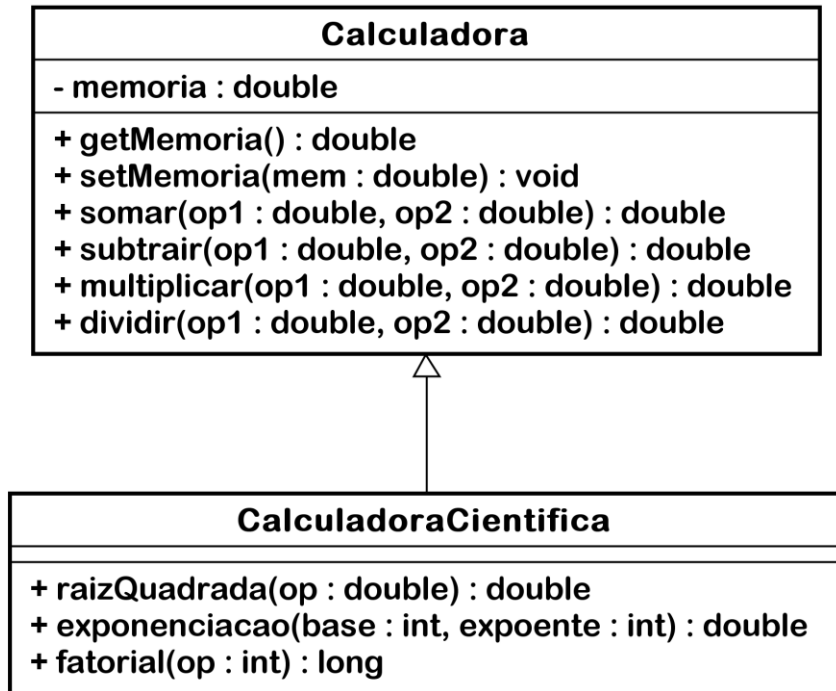
```

### Calculadora

- memoria : double

+ getMemoria() : double  
 + setMemoria(mem : double) : void  
 + somar(op1 : double, op2 : double) : double  
 + subtrair(op1 : double, op2 : double) : double  
 + multiplicar(op1 : double, op2 : double) : double  
 + dividir(op1 : double, op2 : double) : double

# Exemplo



- A subclasse é uma versão especializada da superclasse
- Dizemos que **CalculadoraCientifica estende** a classe **Calculadora**

Ou

- **CalculadoraCientifica herda** a classe **Calculadora**

# Exemplo

```
public class CalculadoraCientifica extends Calculadora {  
    public double raizQuadrada(double op) {  
        // ...  
    }  
  
    public double exponenciacao(int base, int expoente) {  
        // ...  
    }  
  
    public long fatorial(int op) {  
        // ...  
    }  
}
```

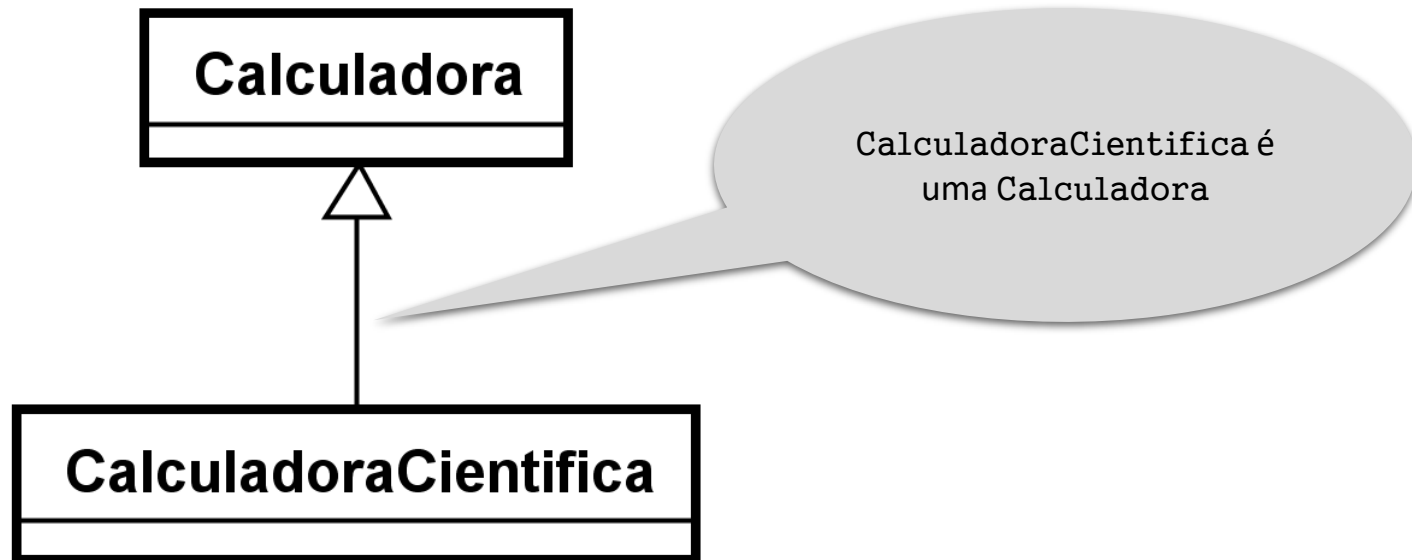
# Exemplo – Programa Java

- Exemplo de programa principal.

```
public static void main(String[] args) {  
    CalculadoraCientifica cs = new CalculadoraCientifica();  
    System.out.println(cs.somar(12,13));  
    System.out.println(cs.fatorial(5));  
}
```

# Herança é um relacionamento

- A herança é um relacionamento entre classes
- Lemos o relacionamento com a expressão “é um(a)”



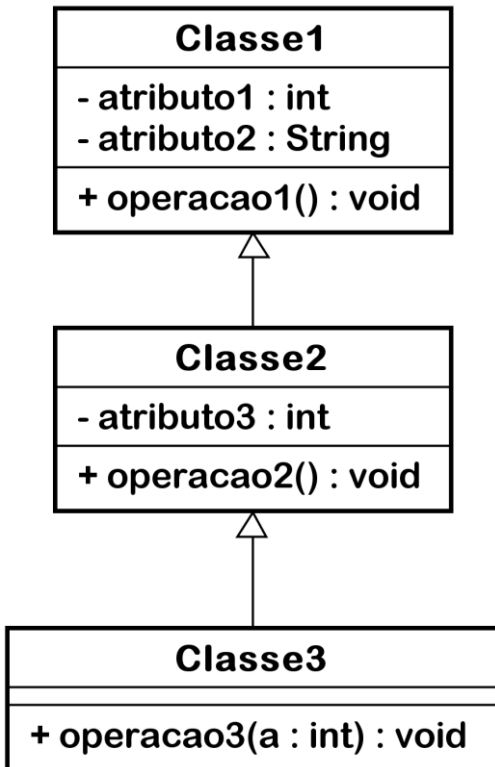


# Herança

- É possível especializar classes construídas pelo próprio programador, bem como especializar classes de terceiros, como as classes da própria linguagem Java.

# Herança

Cada subclasse pode ser uma superclasse de futuras subclASSES.



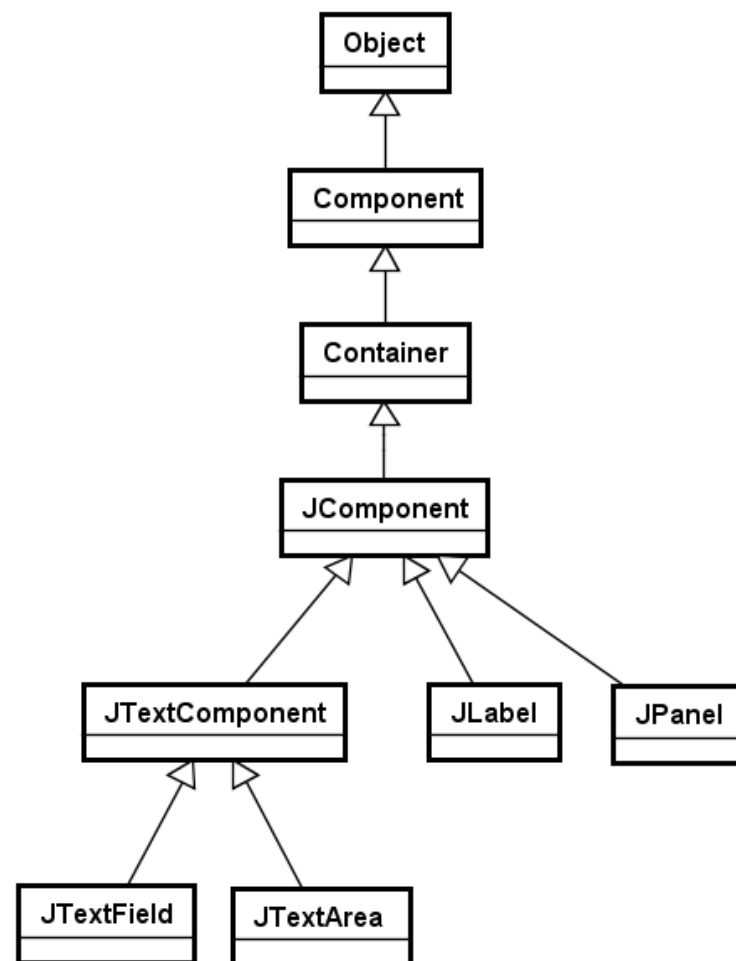
A **superclasse direta** é a superclasse herdada diretamente por uma subclasse

A **superclasse indireta** é a superclasse herdada indiretamente

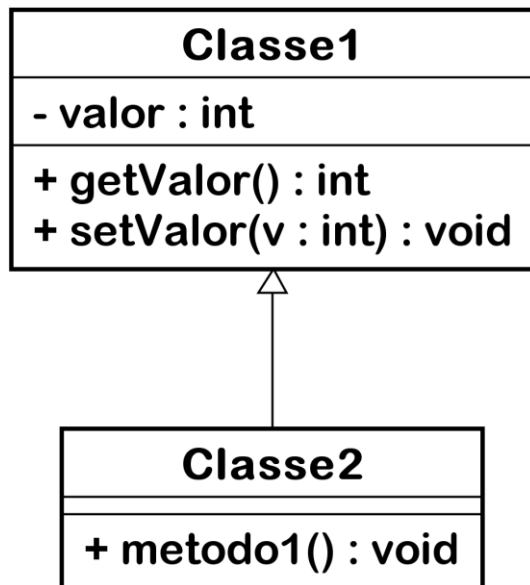
# Estrutura hierárquica de herança

As relações de herança formam estruturas hierárquicas parecidas com uma árvore. Esta hierarquia de classes também é conhecida como **hierarquia de herança**

<a href="#">Overview</a>	<a href="#">Package</a>	<a href="#">Class</a>	<a href="#">Use</a>	<a href="#">Tree</a>	<a href="#">Deprecated</a>	<a href="#">Index</a>	<a href="#">Help</a>
<a href="#">Prev Class</a>	<a href="#">Next Class</a>	<a href="#">Frames</a>	<a href="#">No Frames</a>	<a href="#">All Classes</a>			
Summary: <a href="#">Nested</a>   <a href="#">Field</a>   <a href="#">Constr</a>   <a href="#">Method</a>				Detail: <a href="#">Field</a>   <a href="#">Constr</a>   <a href="#">Method</a>			
javax.swing							
<h2>Class JTextArea</h2>							
java.lang.Object							
java.awt.Component							
java.awt.Container							
javax.swing.JComponent							
javax.swing.text.JTextComponent							
javax.swing.JTextArea							



# Herança X membros privados



```
public class Classe1 {  
  
    private int valor;  
  
    public void setValor(int valor) {  
        this.valor = valor;  
    }  
  
    public int getValor() {  
        return valor;  
    }  
  
}
```

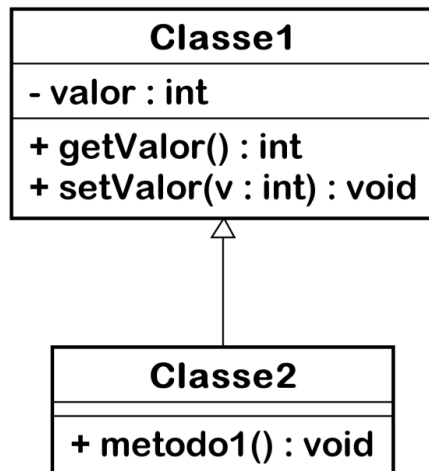
Uma subclasse não  
pode acessar membros  
privados de sua  
superclasse.

```
public class Classe2 extends Classe1 {  
  
    private void metodo1() {  
        valor = 10;  
    }  
  
}
```

Erro de compilação

# Herança X membros privados

- Porém, a subclasse pode alterar o valor de variáveis privadas da superclasse através de métodos **não privados** da superclasse.



```
public class Classe1 {

    private int valor;

    public void setValor(int valor) {
        this.valor = valor;
    }

    public int getValor() {
        return valor;
    }

}

public class Classe2 extends Classe1 {

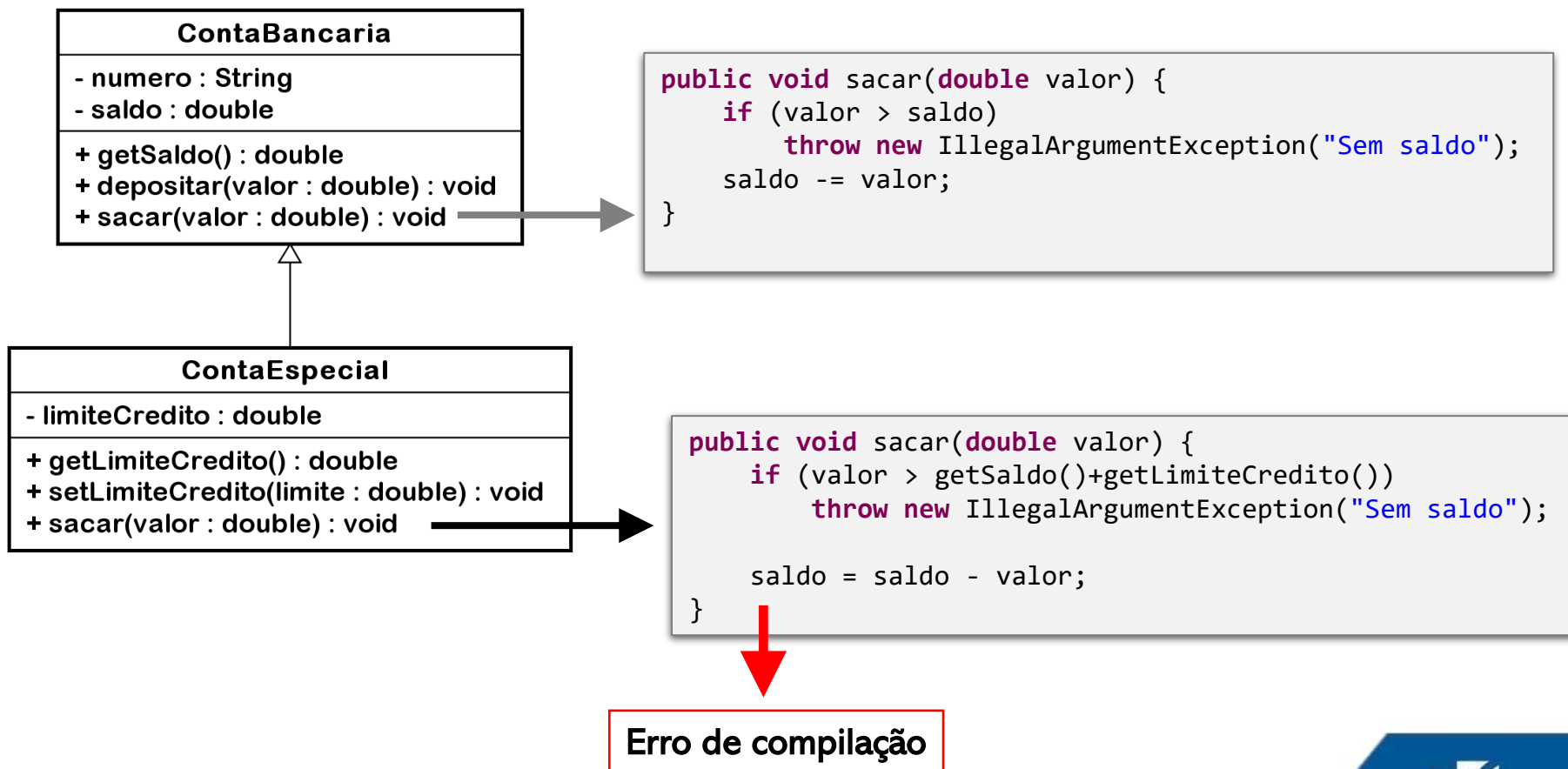
    private void metodo1() {
        setValor(10);
    }

}
```

# Herança X membros privados

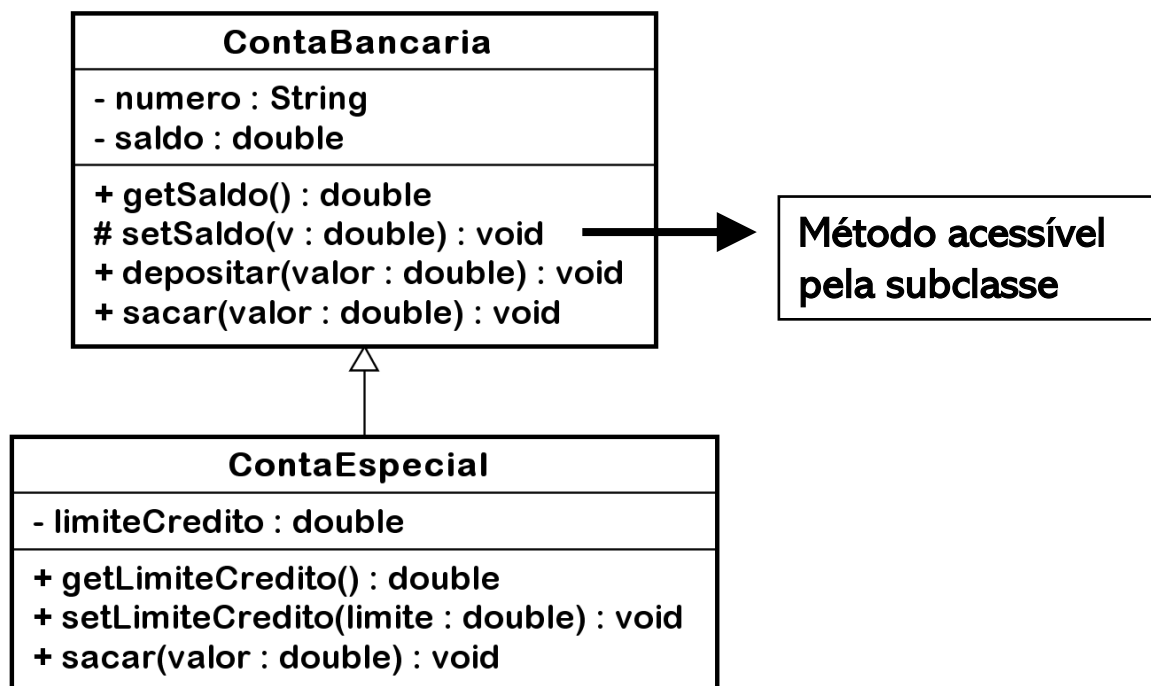
- Se a variável é privada e o *setter* não é público (ou não há *setter*), para acessar a variável na subclasse, recomenda-se:
  - Manter privada a variável de instância
  - Tornar (ou criar) o *setter* com modificador de acesso *protected* (protegido)
- O modificador de acesso *protected* torna o membro acessível:
  - pela própria classe,
  - classes do mesmo pacote e
  - Subclasses
- Em UML, o membro protegido utiliza o símbolo #

# Exemplo de sobrescrita de método



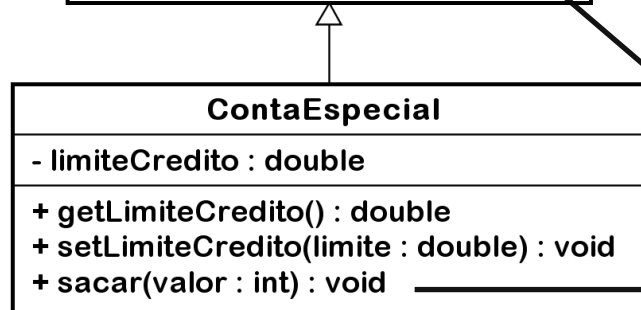
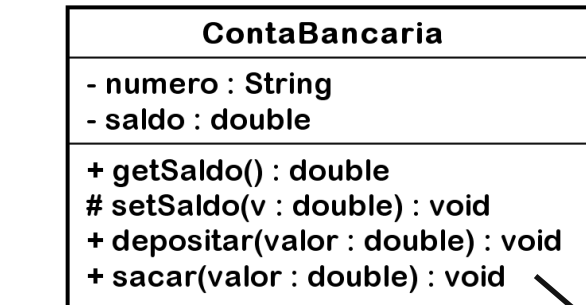
# Exemplo de sobrescrita de método

Como a subclasse precisa alterar o saldo, é preciso implementar o método `setSaldo()` na superclasse, porém de forma protegida.





# Sobrescrita de método



Na tentativa de sobrescrever um método, o programador pode escrever o nome ou parâmetros incorretos, e ao invés de sobrescrever o método, introduz um novo método sobrecarregado.

A classe ContaEspecial agora tem dois métodos sacar(), um para sacar valores com decimais e outro para sacar valores inteiros.

Para evitar este problema, na subclasse, recomenda-se utilizar a anotação `@Override` imediatamente antes do método para que o compilador confira se o método existe na superclasse.

# Sobrescrita de método

- É frequente o método sobrescrito de uma subclasse querer reutilizar o método da superclasse para executar parte do trabalho. Para isso, utilizar a palavra **super** para referenciar o método da superclasse.
- **Exemplo:**

```
@Override  
public void calcularTotal() {  
    setDesconto(10);  
    setIcms(7);  
    super.calcularTotal();  
}
```

# Classe Object

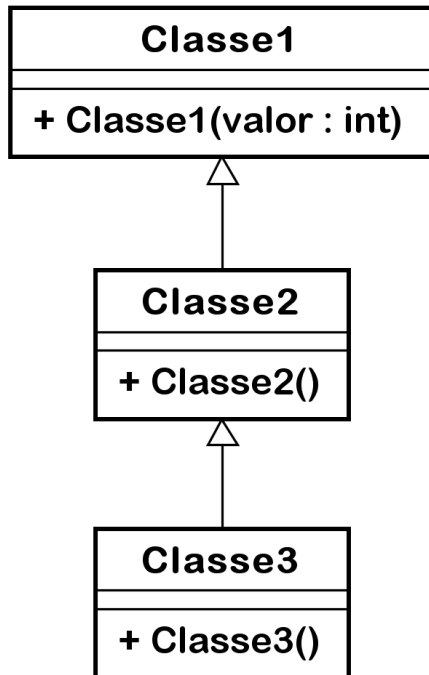
- Em Java, todas as classes herdam a classe Object (direta ou indiretamente), que está no pacote `java.lang`.

Object
+ Object() + hashCode() : int + equals(obj : Object) : boolean + toString() : String

Método	Descrição
equals(Object)	Compara o objeto atual e o objeto recebido como parâmetro, para conferir se são iguais
toString()	Retorna uma representação textual do objeto

# Herança X Construtores

Em Java, os construtores da superclasse não são herdados nas subclasses.



- Ao criar um objeto de uma classe, todos os seus construtores devem chamar algum construtor da superclasse imediata
- Para chamar o construtor da superclasse:  
`super();`

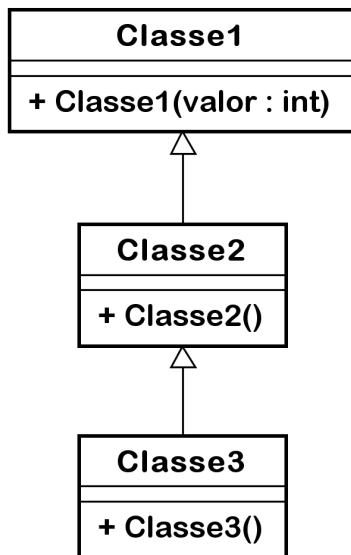
- Exemplo:

```
public class Classe2 extends Classe1 {  
  
    public Classe2() {  
        super(30);  
        System.out.println("Classe2");  
    }  
  
}
```

- O comando `super()` deve ser o primeiro comando do construtor.

# Herança X Construtores

A chamada obrigatória de um construtor da superclasse leva a uma execução em cascata do construtor de todas as classes da hierarquia de herança



```
public class Classe1 {
    public Classe1(int valor) {
        System.out.println(valor);
    }
}

public class Classe2 extends Classe1 {
    public Classe2() {
        super(30);
        System.out.println("Classe2");
    }
}

public class Classe3 extends Classe2 {
    public Classe3() {
        super();
        System.out.println("Classe3");
    }
}
```

```
void executar() {
    new Classe3();
}
```



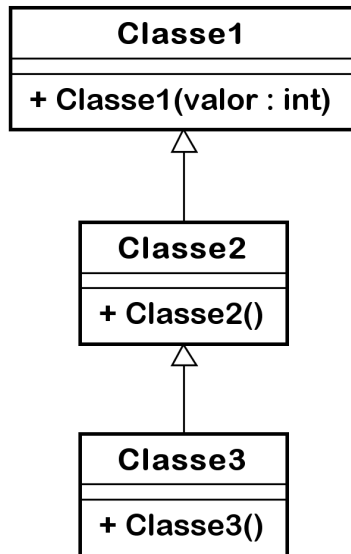
```
30
Classe2
Classe3
```

A execução do construtor ocorre da classe de mais alto nível até atingir a classe que está sendo instanciada.

# Construtores X Herança

- Se a superclasse imediata possuir um construtor padrão, este é chamado automaticamente em todos os construtores da subclasse
  - O compilador introduz o comando `super()` como sendo a primeira linha de código de cada construtor
  - Se a superclasse imediata não possuir um construtor padrão, o programador deverá chamar o construtor da superclasse de forma explícita.

# Construtores X Herança



```
public class Classe1 {  
  
    public Classe1(int valor) {  
        System.out.println(valor);  
    }  
}
```

```
public class Classe2 extends Classe1 {  
  
    public Classe2() {  
        super(30);  
        System.out.println("Classe2");  
    }  
}
```


```
public class Classe3 extends Classe2 {  
  
    public Classe3() {  
        super();  
        System.out.println("Classe3");  
    }  
}
```

O construtor da superclasse deve ser chamado explicitamente, já que a superclasse não tem construtor padrão

O compilador adiciona esta chamada automaticamente

# Construtores X Herança

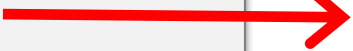
```
public class Classe2 extends Classe1 {  
    public Classe2() {  
        super(30);  
        System.out.println("Classe2");  
    }  
}  
  
public class Classe3 extends Classe2 {  
}
```



Se nenhum construtor é declarado, Java introduz o construtor padrão, fazendo uma chamada ao construtor padrão da superclasse.

```
public class Classe3 extends Classe2 {  
    public Classe3() {  
        super();  
    }  
}
```

```
public class Classe1 {  
    public Classe1(int valor) {  
        System.out.println(valor);  
    }  
}  
  
public class Classe2 extends Classe1 {  
}
```



Se nenhum construtor é declarado e a superclasse não tem construtor padrão, o programador deve criar explicitamente um construtor

**Erro de compilação:** a introdução automática do construtor padrão aqui não compilaria também



# Diversos

Java é uma linguagem de *herança simples*, ao contrário de algumas linguagens que são de *herança múltipla*, permitindo várias superclasses diretas para uma mesma classe