

PROGRAMAÇÃO ORIENTADA A OBJECTOS

CLASSES E OBJECTOS

Vantagens da orientação a objectos

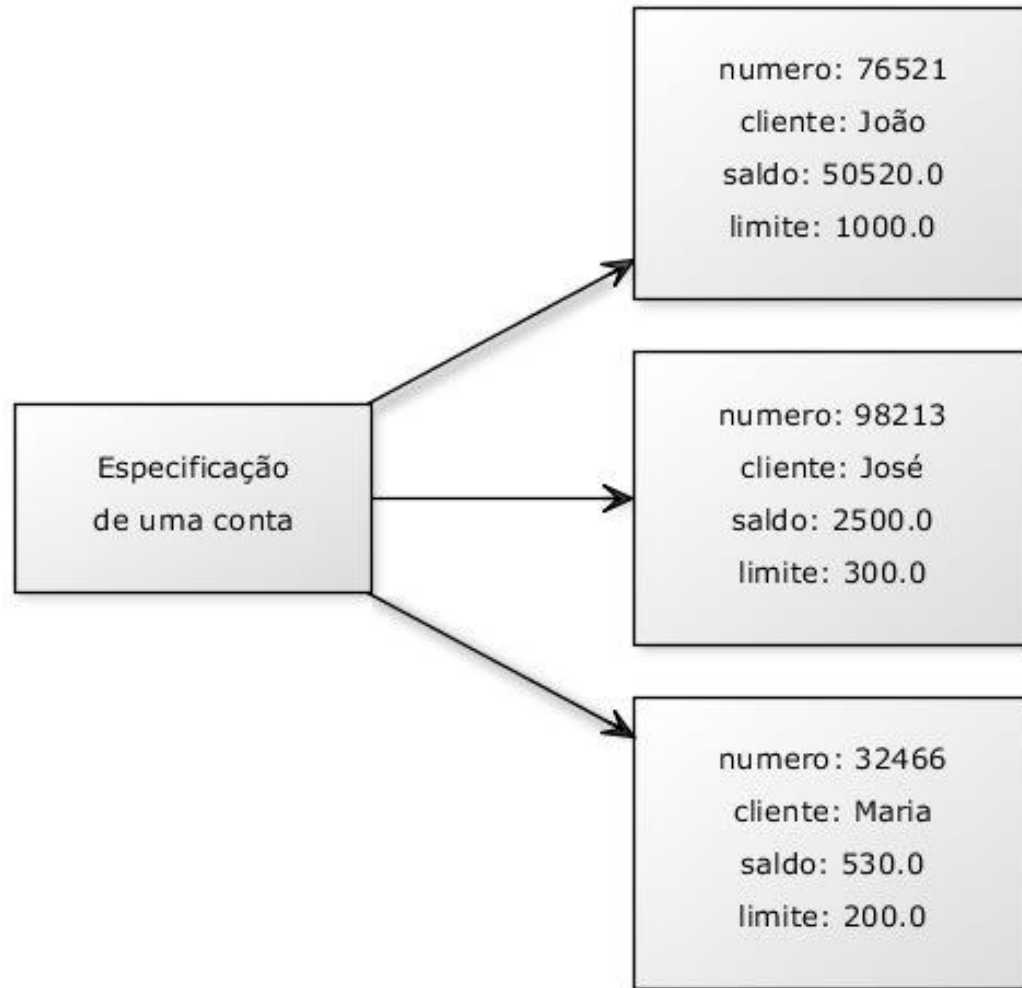
- Ajudar em se organizar e escrever menos
- Concentrar as responsabilidades nos pontos certos
- Flexibilizar o desenvolvimento da sua aplicação
- Encapsulando a lógica de negócios
- Polimorfismo das referências.

Classes e objectos

- Considere uma conta bancária
- O que toda conta tem e é importante para nós?
 - saldo
 - nome do titular da conta
 - número da conta
- O que toda conta faz e é importante para nós ? Isto é, o que gostaríamos de "pedir à conta"?
 - devolve o tipo de conta
 - transfere uma quantidade x para uma outra conta y
 - devolve o saldo actual
 - imprime o nome do titular da conta
 - deposita uma quantidade x
 - levanta uma quantidade x

O que temos aqui é um projecto (especificação) de uma conta bancária

Classes e objectos



- Apesar do papel do lado esquerdo especificar uma Conta, essa especificação é uma Conta?
- Depositamos e levantamos dinheiro desse papel?
- Utilizamos a especificação da Conta para poder criar instâncias que realmente são contas, onde podemos realizar as operações que criamos
- Apesar de declararmos que toda conta tem um saldo, um número e uma agência no pedaço de papel (como à esquerda na figura), são nas instâncias desse projecto que realmente há espaço para armazenar esses valores.
- Ao projecto da conta, isto é, a definição da conta, damos o nome de **classe**. Ao que podemos construir a partir desse projecto, as contas de verdade, damos o nome de **objectos**.

Classes e objectos

Objectos:

Um objecto corresponde à representação de uma entidade sob a forma de:

- um identificador único
- um conjunto de atributos privados (estado do objecto)
- um conjunto de operações
(únicas a aceder ao estado do objecto, constituindo o comportamento do objecto)

Classes e objectos

Objectos ...

Desse conjunto de operações,

- umas são invocáveis a partir do exterior (operações públicas)

outras

- são apenas acessíveis a partir de outras internas ao objecto
(operações privadas)

Classes e objectos

- O conjunto de operações públicas do objecto define o conjunto de serviços que o objecto é capaz de realizar e constitui a:
 - interface do objecto

também designada por:

- API (Application Programmer's Interface) do objecto

Classes e objectos

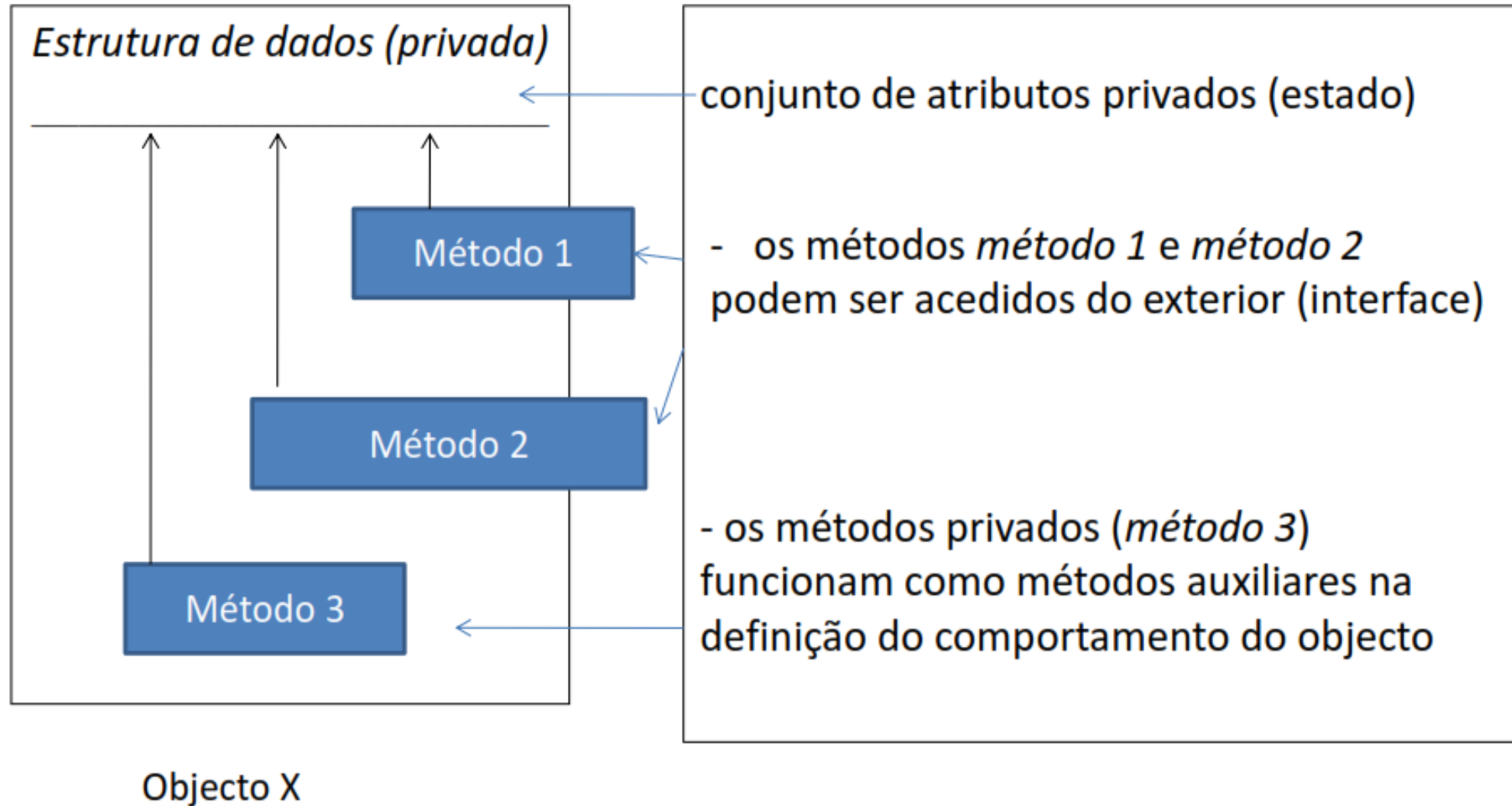
- Aos identificadores que guardam os valores dos atributos
chamam-se

variáveis (de instância)

- Às operações que representam o comportamento do objecto,
chamam-se

métodos (de instância)

Classes e objectos



Classes e objectos

Mensagens:

Os objectos vão interactuar entre si através de um mecanismo de envio de mensagens.

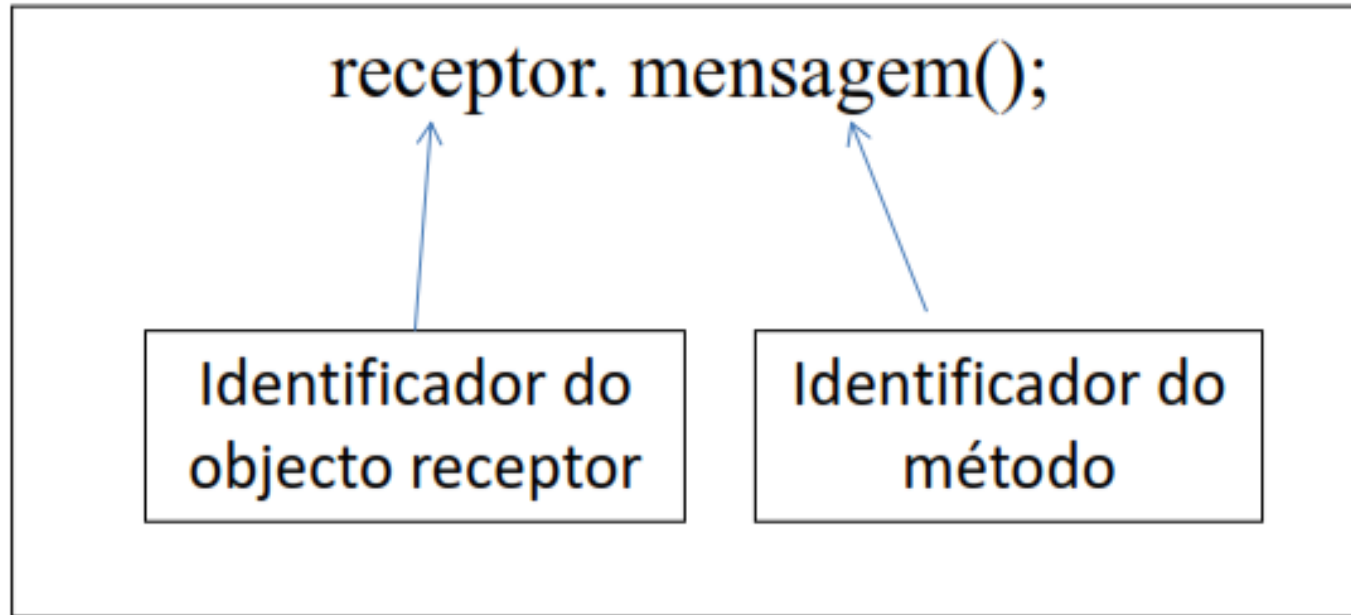
Quando um objecto pretende invocar um método de um outro objecto, envia uma mensagem para que tal método seja executado.

Em cada computação, isto é, em cada alteração do estado do programa existe um objecto que é emissor de uma mensagem e um outro que é o receptor dessa mensagem.

Em geral a sintaxe para o envio de uma mensagem a um objecto tem uma das seguintes formas:

Classes e objectos

a)



- envio de uma mensagem sem argumentos a um objecto, sem retorno de resultado pelo método correspondente.

Classes e objectos

b) `receptor.mensagem(arg1, arg2, ... argn);`

- envio de uma mensagem com argumentos, sem retorno de resultado

c) `resultado = receptor.mensagem();`

- envio de uma mensagem sem argumentos, com retorno de resultado

d) `resultado = receptor.mensagem(arg1, arg2, ... argn);`

- envio de uma mensagem com argumentos e com retorno de resultado

Classes e objectos

Exemplo:

Objecto do tipo contador

int n

void incrementar()

void decrementar()

int consultar()

Contador c

int x

....

c.incrementar();

...

c.decrementar();

...

x = c.consultar();

...

- Os argumentos e o resultado de uma mensagem têm que ser compatíveis com os tipos dos parâmetros e do valor devolvido pelo método correspondente.

Classes e objectos

Instâncias Vs. Classes:

Um mecanismo de classificação permite, na generalidade das linguagens, criar vários objectos do mesmo tipo, isto é, objectos que possuam exactamente a mesma estrutura e o mesmo comportamento.

“Uma classe é um modelo, a partir do qual podem ser criados objectos. Contém a definição dos atributos e dos métodos do objecto”

Uma classe serve para:

- conter a descrição da estrutura e do comportamento de objectos do mesmo tipo,
- criar objectos particulares possuindo tal estrutura e comportamento.

Classes e objectos

Definida uma classe, os seus objectos são criados através de um mecanismo de instanciação, pelo qual o objecto é inicializado, passando a existir e podendo receber mensagens de outros objectos.

Cada objecto vai ter as suas próprias instâncias das variáveis de estado, enquanto que o código que implementa os métodos permanece armazenado na classe.

Apesar de todas as instâncias da classe exibirem um comportamento comum, uma vez que partilham as mesmas operações, elas não são iguais. Cada objecto possui o seu próprio estado que pode variar ao longo do tempo.

Classes e objectos

Quando um objecto é criado (instanciado) a inicialização do seu estado é geralmente feita por invocação automática de um método de inicialização (o construtor da classe).

Se, num dado programa, necessitarmos de vários objectos do tipo Contador, definimos a classe Contador e a partir dela criamos os objectos desse tipo, isto é, criamos instâncias da classe Contador:

Em Java:

```
Contador conta_1 = new Contador();  
Contador conta_2 = new Contador();
```


Classes e objectos

Os objectos `conta_1` e `conta_2` possuem ambos uma variável de instância, `n`, que apenas poderá conter valores inteiros. Estes dois objectos, tal como qualquer outra instância de `Contador`, serão capazes de responder às mensagens `incrementar()`, `decrementar()` e `consultar()`:

(com `x` e `y` variáveis inteiras)

```
conta_1.incrementar();  
        conta_2.decrementar();  
x = conta_1.consultar();  
y = conta_2.consultar();
```

- supondo que cada contador é inicializado com o valor 0, qual será o valor de `x` e `y`?

Classes e objectos

Numa linguagem de programação orientada a objectos “pura” todas as entidades deveriam ser objectos.

A linguagem Smalltalk é a que mais se aproxima desse modelo (classes, instâncias e mensagens são objectos).


Em Java existem tipos de dados que não são objectos (tipos primitivos e arrays).

Classes e objectos

Objectos são acessados por referência:

- Quando declaramos uma variável para associar a um objecto, na verdade, essa variável não guarda o objecto, e sim uma maneira de acessá-lo, chamada de **referência**.
- Diferente dos *tipos primitivos como* int e long, precisamos dar **new** depois de declarada a variável:

```
public static void main (String[]args) {  
    Conta c1;  
    c1=new Conta();  
    Conta c2;  
    c2=new Conta();  
}
```



O correcto aqui, é dizer que c1 se refere a um objecto. **Não é correcto dizer que c1 é um objecto, pois c1 é uma variável referência.**

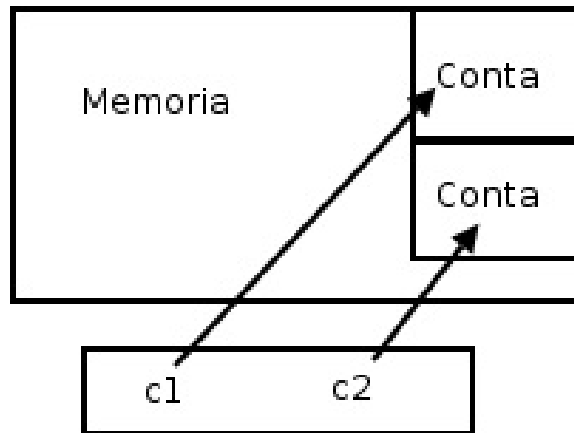
Em Java, **uma variável nunca é um objecto.**

Todo objecto em Java, sem excepção, é acessado por uma variável referência.

Classes e objectos

Objectos são acessados por referência:

```
Conta c1;  
c1=new Conta();  
Conta c2;  
c2=new Conta();
```



Internamente, c1 e c2 vão guardar um número que identifica em que posição da memória aquela Conta se encontra. Dessa maneira, ao utilizarmos o "." para navegar, o Java vai acessar a Conta que se encontra naquela posição de memória, e não uma outra.

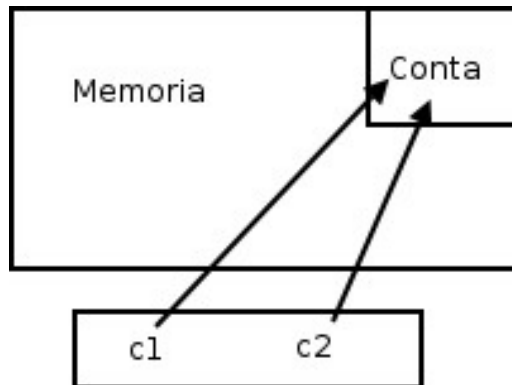
Não pode manipulá-lo como um número e nem utilizá-lo para aritmética.

Classes e objectos

Objectos são acessados por referência:

Outro exemplo:

```
Class TestaReferencias {  
    public static void main(String[] args){  
        Conta c1=new Conta();  
        c1.deposita(100);  
        Conta c2=c1; //linha importante!  
        c2.deposita(200);  
        System.out.println(c1.saldo);  
        System.out.println(c2.saldo);  
    }  
}
```



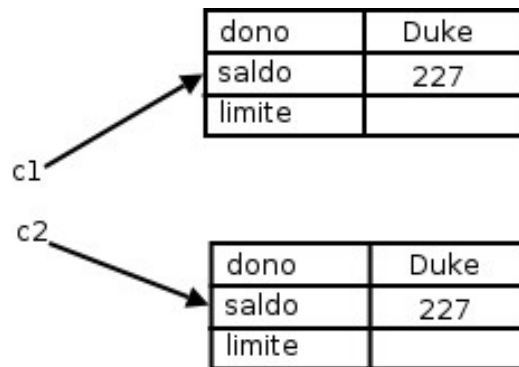
Quando fizemos `c2 = c1`, `c2` passa a fazer referência para o mesmo objecto que `c1` referencia nesse instante. Nesse código em específico, quando utilizamos `c1` ou `c2` estamos nos referindo exactamente ao **mesmo** objecto! Elas são duas referências distintas, porém apontam para o **mesmo** objecto! Compará-las com `"=="` vai nos retornar `true`, pois o valor que elas carregam é o mesmo!

Classes e objectos

Objectos são acessados por referência:

Mais um exemplo:

```
public static void main(String[]args){  
    Conta c1=new Conta();  
    c1.titular="Duke";  
    c1.saldo=227;  
    Conta c2=new Conta();  
    c2.titular="Duke";  
    c2.saldo=227;  
    If (c1==c2) {  
        System.out.println("Contas iguais");  
    }  
}
```



O operador == compara o conteúdo das variáveis, mas essas variáveis não guardam o objecto, e sim o endereço em que ele se encontra. Como em cada uma dessas variáveis guardamos duas contas criadas diferentemente, elas estão em espaços diferentes da memória, o que faz o teste no if valer false.

As contas podem ser equivalentes no nosso critério de igualdade, porém elas não são o mesmo objecto.

Classes e objectos

Packages em Java:

As classes são agrupadas de acordo com a sua funcionalidade.

Cada classe de um package tem acesso às outras classes do mesmo package.

Exemplos:

package java.io

- conjunto de classes que implementam funcionalidades relacionadas com input/output

Classes e objectos

package java.util

Funcionalidades de uso geral

inclui as classes:

Date, GregorianCalendar, EventListener, Vector, ...

package java.lang

Classes fundamentais à execução de programas

inclui as classes:

String, Boolean, Character, Float, Integer, System, ...

Classes e objectos

O nome absoluto de uma classe ou método tem como prefixo o nome do package:

```
java.lang.String.length();  
java.lang.System.out.println();
```

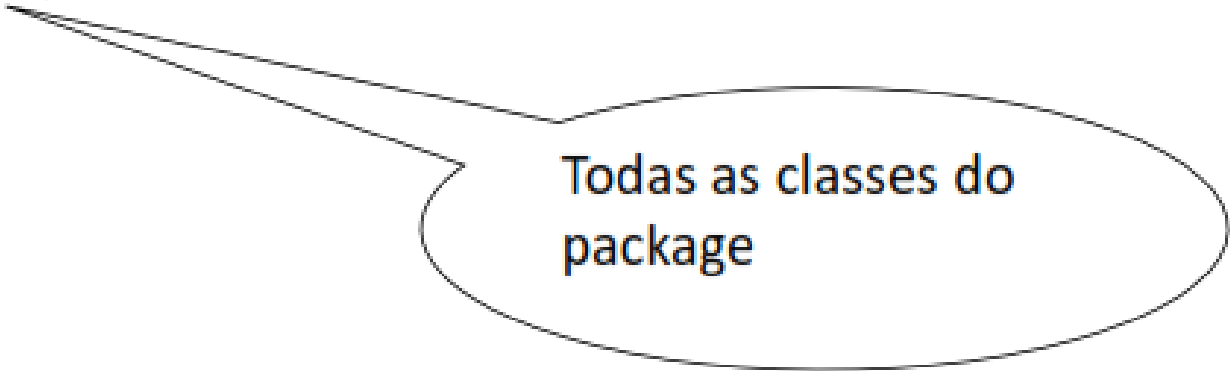
Se queremos aceder a classes de outros packages:

colocamos o nome completo (absoluto)
ou
usamos uma cláusula de importação

```
import java.util.Vector;
```

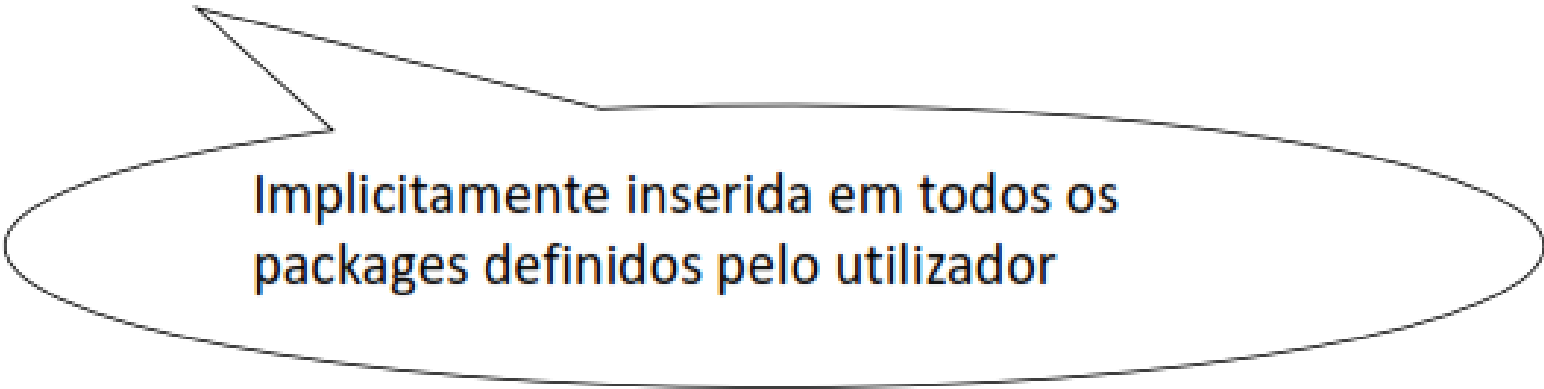
Classes e objectos

```
import java.util.*;
```



Todas as classes do
package

```
import java.lang.*;
```



Implicitamente inserida em todos os
packages definidos pelo utilizador

Classes e objectos

- Usar uma cláusula de importação (“import”) não significa que as classes do package vão ser copiadas para o package do nosso programa.
- Serve apenas para podermos referir o nome de uma classe ou método na sua forma abreviada omitindo o nome do package a que pertence.

(Para usarmos classes de outros packages definidos por nós será necessário ou incluir esses packages numa biblioteca ou definir a variável de ambiente CLASSPATH com a directoria onde estão as classes).

Classes e objectos

Mecanismos de controlo de acesso

Especificam “quem” tem acesso a cada entidade, isto é, quem tem acesso a cada classe e cada membro da classe (dados e métodos)

Modificadores de acesso:

public

protected

private

“por omissão”

Classes e objectos

Regras de acesso a **classes**:

R1: Uma classe é sempre acessível a todas as outras classes do mesmo package
(qualquer que seja o modificador de acesso).

R2: Se nenhum modificador de acesso é usado, a classe apenas pode ser acedida dentro do seu package.

R3: Quando uma classe é declarada como “public” pode ser acedida por qualquer classe que tenha acesso ao seu package.

R4: Quando uma classe é não pública apenas é acessível dentro do seu package.

Classes e objectos

Regras de acesso a **variáveis** e **métodos**:

“norma:”

*variáveis são privadas,
métodos de interface são públicos,
métodos auxiliares são privados*

R1: Um método declarado como “public” é acessível de qualquer ponto de qualquer programa.

Designa-se por API (“Application Programming Interface”) de uma classe, o conjunto de métodos de instância que não forem declarados como “private”.

Classes e objectos

R2: Um método sem modificador de acesso é acessível a qualquer classe do mesmo package.

R3: Métodos ou variáveis declarados como “private” são apenas acessíveis dentro da própria classe.

R4: Métodos ou variáveis declarados como “protected” são acessíveis na própria classe, de outra classe dentro do mesmo package e nas subclasses da classe.

Classes e objectos

Definição da classe Contador

```
public class Contador {  
  
    // variáveis de instância  
    private int conta;  
  
    // construtores  
    public Contador () {  
        conta = 0;  
    }  
    public Contador ( int conta) {  
        this.conta = conta;  
    }  
}
```


Classes e objectos

// métodos de instância

```
public int getConta(){  
    return conta;  
}  
public void incConta () {  
    conta ++;  
}  
public void incConta (int inc) {  
    conta = conta + inc;  
}  
public void decConta () {  
    conta --;  
}  
public void decConta (int dec) {  
    conta = conta - dec;  
}  
public String toString () {  
    return ("Contador: " + conta );  
}}
```

Classes e objectos

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	<i>Sim</i>	<i>Sim</i>	<i>Sim</i>	<i>Sim</i>
Protected	<i>Sim</i>	<i>Sim</i>	<i>Sim</i>	<i>Não</i>
Sem Modificador (Padrão)	<i>Sim</i>	<i>Sim</i>	<i>Não</i>	<i>Não</i>
Private	<i>Sim</i>	<i>Não</i>	<i>Não</i>	<i>Não</i>

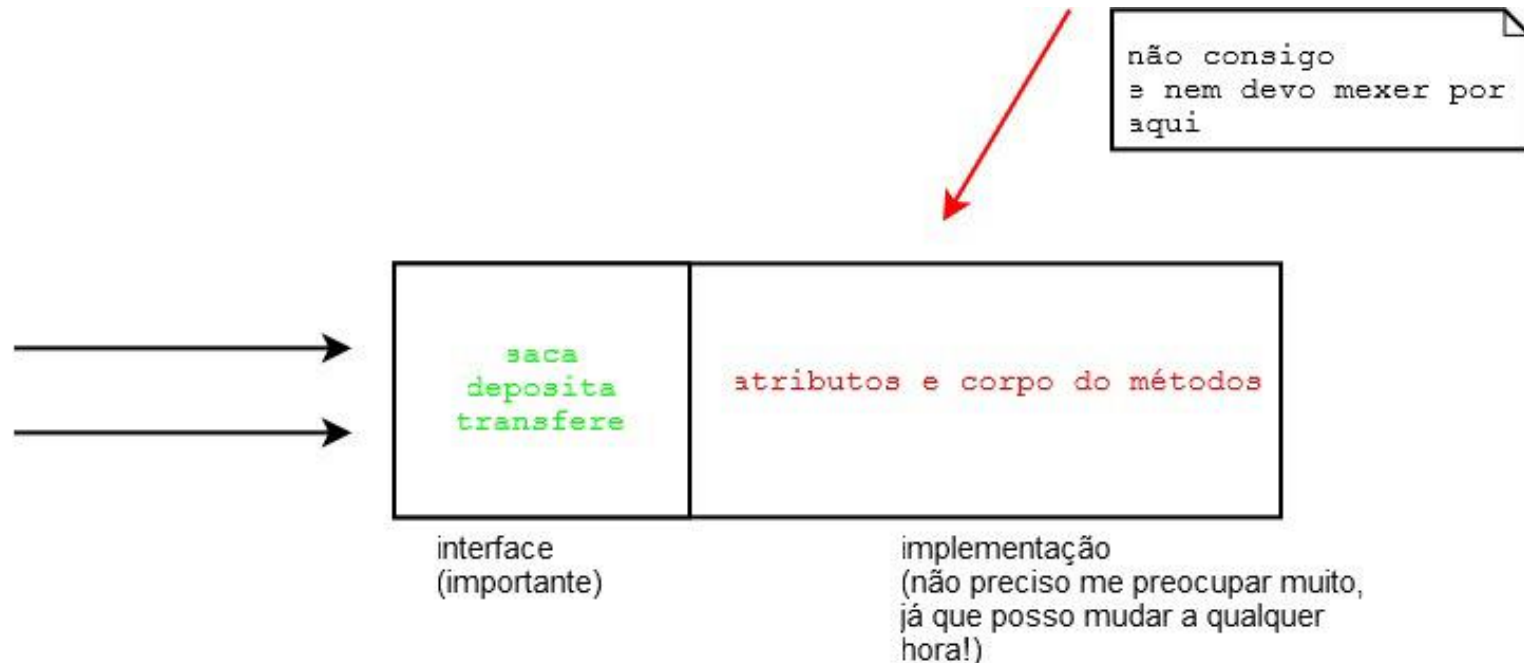
Exemplo:

```
public class MinhaClasse { //classe public
    private int inteiro; //atributo inteiro private
    protected float decimal; //atributo float protected
    boolean ativado; //atributo booleano package-private
}
```

Classes e objectos

Encapsulamento

- Esconder todos os membros de uma classe, além de esconder como funcionam as rotinas (no caso métodos) do nosso sistema.
- Encapsular é **fundamental** para que seu sistema seja susceptível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está **encapsulada**



Classes e objectos

Encapsulamento...

- O conjunto de métodos públicos de uma classe é também chamado de **interface da classe**, pois esta é a única maneira a qual se comunica com objectos dessa classe.
- É sempre bom programar pensando na interface da sua classe, como seus usuários a estarão utilizando, e não somente em como ela vai funcionar.
- A implementação em si, o conteúdo dos métodos, não tem tanta importância para o usuário dessa classe, uma vez que ele só precisa saber o que cada método pretende fazer, e não como ele faz, pois isto pode mudar com o tempo.

Classes e objectos

GETTERS E SETTERS

- Para permitir o acesso aos atributos (já que eles são private) de uma maneira controlada, a prática mais comum é criar dois métodos, um que retorna o valor e outro que muda o valor.
- A convenção para esses métodos é de colocar a palavra get ou set antes do nome do atributo.

```
class Conta {  
    private String titular;  
    private double saldo;  
  
    public double getSaldo() {  
        return this.saldo;  
    }  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
  
    public String getTitular() {  
        return this.titular;  
    }  
  
    public void setTitular(String titular) {  
        this.titular = titular;  
    }  
}
```

É uma má prática criar uma classe e, logo em seguida, criar getters e setters para todos seus atributos. Só deve criar um getter ou setter se tiver a real necessidade. Repare que nesse exemplo setSaldo não deveria ter sido criado, já que queremos que todos usem deposita() e levanta().

Classes e objectos

Construtores

- Quando usamos a palavra chave new, estamos construindo um objecto. Sempre quando o new é chamado, ele executa o **construtor da classe**. O construtor da classe é um bloco declarado com o **mesmo nome** que a classe:

```
class Conta {  
    String titular;  
    int numero;  
    double saldo;  
  
    // construtor  
    Conta() {  
        System.out.println("Construindo uma conta.");  
    }  
  
    // ..  
}
```

Classes e objectos

Construtores

- Um construtor pode receber um argumento, podendo assim inicializar algum tipo de informação.

```
class Conta {  
    String titular;  
    int numero;  
    double saldo;  
  
    // construtor  
    Conta(String titular) {  
        this.titular = titular;  
    }  
  
    // ..  
}
```

Nota: Um construtor **não é** um método. Algumas pessoas o chamam de um método especial, mas definitivamente não é, já que não possui retorno e só é chamado durante a construção do objecto.

Classes e objectos

Chamando outro construtor

- Um construtor só executa durante a construção do objecto, isto é, você nunca conseguirá chamar o construtor em um objecto já construído.
- Porém, durante a construção de um objecto, você pode fazer com que um construtor chame outro, para não ter de ficar copiando e colando:

```
class Conta {  
    String titular;  
    int numero;  
    double saldo;  
  
    // construtor  
    Conta (String titular) {  
        // faz mais uma série de inicializações e configurações  
        this.titular = titular;  
    }  
  
    Conta (int numero, String titular) {  
        this(titular); // chama o construtor que foi declarado acima  
        this.numero = numero;  
    }  
  
    //..  
}
```


Classes e objectos

Construtor default

- Quando você não declara nenhum construtor na sua classe, o Java cria um para você. Esse construtor é o **construtor default**, ele não recebe nenhum argumento e o corpo dele é vazio.
- A partir do momento que você declara um construtor, o construtor default não é mais fornecido.

Classes e objectos

Atributos de classe

- Suponhamos que queremos controlar a quantidade de contas no sistema do banco. Tentamos então, a seguinte proposta:

```
class Conta {  
    private int totalDeContas;  
    //...  
  
    Conta() {  
        this.totalDeContas = this.totalDeContas + 1;  
    }  
}
```

Quando criarmos duas contas, qual será o valor do totalDeContas de cada uma delas? Vai ser 1. Pois cada uma tem essa variável. **O atributo é de cada objecto.**

Seria interessante então, que essa variável fosse **única, compartilhada por todos os objectos dessa classe**. Dessa maneira, quando mudasse através de um objecto, o outro veria o mesmo valor.

Para fazer isso em java, declaramos a variável como **static**.

- Private **static** int totalDeContas;

Classes e objectos

Atributos de classe...

- Quando declaramos um atributo como static, ele passa a não ser mais um tributo de cada objecto, e sim um **atributo da classe**, a informação fica guardada pela classe, não é mais individual para cada objecto.
- Para acessarmos um atributo estático, não usamos a palavra chave this, mas sim o nome da classe:

```
class Conta{  
    Private static int totalDeContas;  
    //...  
    Conta(){  
        Conta.totalDeContas=Conta.totalDeContas+1;  
    }  
    public int getTotalDeContas(){  
        return Conta.totalDeContas;  
    }  
}
```

Como fazemos então para saber quantas contas foram criadas?

Transformar esse método que todo objecto conta tem em um método de toda a classe. Usamos a palavra static de novo, mudando o método getTotalDeContas().

Classes e objectos

Atributos de classe...

```
public static int getTotalDeContas(){  
    return Conta.totalDeContas;  
}
```

- Para acessar esse novo método:
int total=Conta.getTotalDeContas();

Métodos e atributos estáticos só podem acessar outros métodos e atributos estáticos da mesma classe, o que faz todo sentido já que dentro de um método estático não temos acesso à referência `this`, pois um método estático é chamado através da classe, e não de um objecto.

Classes e objectos

Em resumo:

O modelo mais simples de linguagem Orientada a Objectos é:

“Uma linguagem é Orientada a Objectos, se suporta Objectos, se esses objectos pertencem a Classes e se hierarquias de classes podem ser definidas, incrementalmente, por um mecanismo de Herança.”

O Objecto constitui a unidade fundamental de encapsulamento, enquanto Classes e Herança realizam os princípios de partilha de comportamento e evolução.