

# Programação 1

## Aula 9

Valeri Skliarov, Prof. Catedrático

Email: [skl@ua.pt](mailto:skl@ua.pt)

URL: <http://sweet.ua.pt/skl/>

Departamento de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

<http://elearning.ua.pt/>

- Estruturas de Dados (Tipos Compostos)
- Introdução
- Criação de novos tipos de dados
- Declaração de variáveis de novos tipos
- Cópia de variáveis tipo referência
- Exemplos

- Em muitas situações práticas, precisamos de armazenar informação relacionada entre si, eventualmente de tipos diferentes, na mesma variável.
- As linguagens de programação permitem que o programador defina tipos de dados particulares para adequar a representação da informação às condições concretas do problema.
- Estes tipos de dados são designados normalmente por **Estruturas de Dados, Tipos Compostos** ou **Registos**.
- Na linguagem JAVA podemos utilizar classes (`class`) para a construção de registos.
- Um registo é então um tipo de dados que pode conter campos de cada um dos tipos básicos (`int`, `double`, `char`, `boolean`, ...), ou outros tipos compostos.

# Tipos de dados

- Tipos primitivos:
  - aritméticos:
    - inteiros:
      - `byte`, `short`, `int`, `long`
    - reais:
      - `float`, `double`
    - caracter:
      - `char`
  - booleanos:
    - `boolean`
- Tipos referência:
  - `class` (registros), `array`, ...

a declaração de variável reserva automaticamente um espaço na memória.

a variável de tipos primitivos é sempre passada por valor às funções como argumento

# Criação de um novo tipo de dados

- Estrutura de um programa (relembrar):

```
// inclusão de classes externas
public class Programa{
    public static void main (String[] args){
        // declaração de constantes e variáveis
        // sequências de instruções
    }
    // funções desenvolvidas pelo programador
    // definição de tipos de dados (registos) locais
}
// definição de tipos de dados (registos) globais
```

- Os novos tipos de dados são criados **no alcance global (i.e. antes ou depois da definição da classe do programa) ou dentro da definição da classe do programa**, neste momento no mesmo ficheiro.

# Criação de um novo tipo de dados

```
class nomeDoTipo
{
    //tipo1 nomeDoCampo1;
    //tipo2 nomeDoCampo2;
    //...
    //tipon nomeDoCampoN;
}
```

## Exemplos:

```
class data
{
    int dia;
    int mês;
    int ano;
}
```

```
class data
{ int dia, mês, ano; }
```

```
class data
{ int dia=1;
  int mês=6;
  int ano=2014; }
```

A `class` define um novo tipo de dados constituído por vários campos.

A partir desta definição passa a existir um novo tipo de dados, sendo possível declarar variáveis deste novo tipo.

O acesso a cada um dos campos faz-se através do nome do campo correspondente.

# Utilização de um novo tipo de dados

```
class data
{
    int dia;
    int mês;
    int ano;
}
```

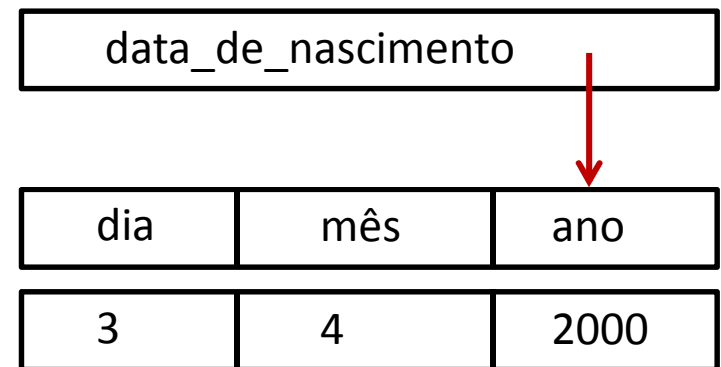
```
class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
```

```
class aluno
{
    pessoa p;
    int n_mec;
    double nota_média;
}
```

Vamos pensar que o objeto p da classe pessoa contém informação adicional sobre aluno

```
data data_de_nascimento = new data();
```

```
data_de_nascimento.dia = 3;
data_de_nascimento.mês = 4;
data_de_nascimento.ano = 2000;
```



Para representar a relação entre a classe aluno e a classe pessoa existe outra possibilidade melhor que se chama herança. Vamos estudar esta possibilidade no segundo semestre

## Exemplo:

```
public class first_class
{
    public static void main(String[] args)
    {
        data data_de_nascimento = new data();
        data_de_nascimento.dia = 3;
        data_de_nascimento.mes = 4;
        data_de_nascimento.ano = 2000;

        System.out.printf("%d : %d : %d\n",
                           data_de_nascimento.dia,
                           data_de_nascimento.mes,
                           data_de_nascimento.ano);
    }
}

class data
{
    int dia;
    int mes;
    int ano;
}
```



3 : 4 : 2000



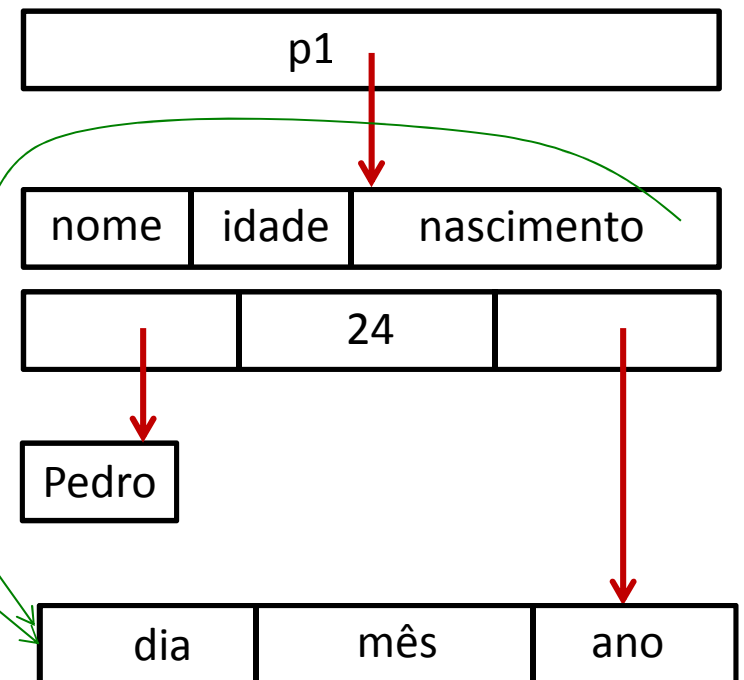
# Utilização de um novo tipo de dados

```
class data
{
    int dia;
    int mês;
    int ano;
}
```

```
class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
```

```
pessoa p1 = new pessoa();
pessoa p1.nascimento = new data();
```

```
p1.nome = "Pedro";
p1.idade = 24;
p1.nascimento.dia = 5;
p1.nascimento.mês = 10;
p1.nascimento.ano = 1991;
```



## Exemplo:

```
public class second_class
{
    public static void main(String[] args)
    {
        pessoa p1 = new pessoa();
        p1.nome = "Pedro";
        p1.idade = 24;
        p1.nascimento.dia = 5;
        p1.nascimento.mes = 10;
        p1.nascimento.ano = 1990;
        System.out.printf("%d : %d : %d\n",
                           p1.nascimento.dia,
                           p1.nascimento.mes,
                           p1.nascimento.ano);
    }
}

class data
{
    int dia;
    int mes;
    int ano;
}

class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
```

**p1.nascimento = new data();**

*java.lang.NullPointerException*  
at second\_class.main(second\_class.java:10)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)  
at java.lang.reflect.Method.invoke(Unknown Source)  
at edu.rice.cs.drjava.model.compiler.JavacCompiler.runCommand(JavacCompiler.java:271)

Exception in thread "main" java.lang.NullPointerException  
at second\_class.main(second\_class.java:?)  
Press any key to continue . . .

## Exemplo:

```
public class second_class
{
    public static void main(String[] args)
    {
        pessoa p1 = new pessoa();
        p1.nascimento = new data();
        p1.nome = "Pedro";
        p1.idade = 24;
        p1.nascimento.dia = 5;
        p1.nascimento.mes = 10;
        p1.nascimento.ano = 1990;
        System.out.printf("%d : %d : %d\n",
                           p1.nascimento.dia,
                           p1.nascimento.mes,
                           p1.nascimento.ano);
    }
}

class data
{
    int dia;
    int mes;
    int ano;
}

class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
```



5 : 10 : 1990

```
5 : 10 : 1990
Press any key to continue . . . _
```

# Utilização de um novo tipo de dados

## Exemplos:

```
class data
{
    int dia;
    int mês;
    int ano;
}
```

```
class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
```

```
class aluno
{
    pessoa p;
    int n_mec;
    double nota_média;
}
```

```
data d1 = new data();
data d2 = new data();
```

```
pessoa p1 = new pessoa();
pessoa p2 = new pessoa();
```

```
aluno a1 = new aluno();
aluno a2 = new aluno();
```

```
p1.nascimento = d1;
p2.nascimento = d2;
```

```
a1.p = p1;
a2.p = p2;
```

```
a1.p.nascimento.dia = 20;
```

```
d2.ano = 2012;
```

```
p1.nascimento.mês = 5;
```

```
d1.dia = 30;
```

```
p2.idade = 35;
```

```
p2.nome = "Nuno";
```

```
a1.nota_média = 17.3;
```

```
a2.n_mec = 3456;
```

```
a2.p.nome = "Joana";
```

```
p2.nascimento.dia = 10;
```

```

public class third_class
{
    public static void main(String[] args)
    {
        data d1 = new data();
        data d2 = new data();
        pessoa p1 = new pessoa();
        pessoa p2 = new pessoa();
        aluno a1 = new aluno();
        aluno a2 = new aluno();
        p1.nascimento = d1;
        p2.nascimento = d2;
        a1.p = p1;
        a2.p = p2;
        a1.p.nascimento.dia = 20;
        d2.ano = 2012;
        p1.nascimento.mes = 5;
        d1.dia = 30;
        p2.idade = 35;
        p2.nome = "Nuno";
        a1.nota_media = 17.3;
        a2.n_mec = 3456;
        a2.p.nome = "Joana";
        p2.nascimento.dia = 10;
        p1.nascimento.ano = 2010;

        System.out.printf("%d : %d : %d\n",
            p1.nascimento.dia,
            p1.nascimento.mes,
            p1.nascimento.ano);
    }
}

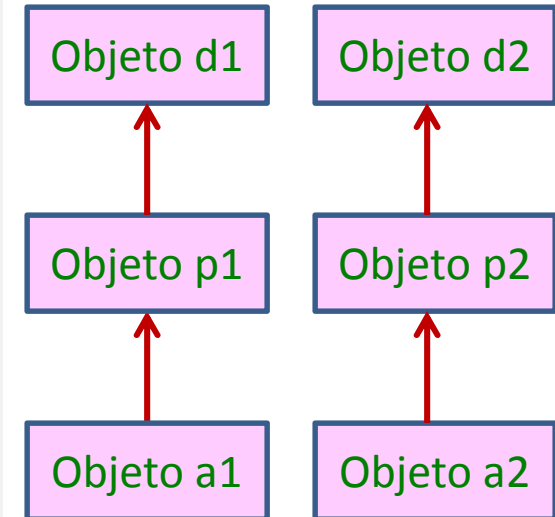
```

```

class data
{
    int dia;
    int mes;
    int ano;
}
class pessoa
{
    String nome;
    int idade;
    data nascimento;
}
class aluno
{
    pessoa p;
    int n_mec;
    double nota_media;
}

```

## Exemplo:



**Campos do objeto d1  
podem ser alterados  
através do objeto p1  
e através do objeto a1**

```

System.out.printf("%d : %d : %d\n",
    d1.dia,
    d1.mes,
    d1.ano);

```

30 : 5 : 2010

## Exemplo:

```
public class fourth_class
{
    public static void main(String[] args)
    {
        ➡ aluno a1 = new aluno();
        ➡ a1.p = new pessoa();
        ➡ a1.p.nascimento = new data();
        a1.p.nascimento.dia = 20;
        a1.p.nascimento.mes = 5;
        a1.p.nascimento.ano = 1992;
        a1.p.nome = "Nuno";
        a1.p.idade = 21;
        a1.n_mec = 1234;
        a1.nota_media = 17.3;
        System.out.printf("%d : %d : %d\n",
            a1.p.nascimento.dia,
            a1.p.nascimento.mes,
            a1.p.nascimento.ano);
        System.out.printf("%s : %d\n",a1.p.nome,a1.p.idade);
        System.out.printf("%d : %4.1f\n",a1.n_mec,a1.nota_media);
    }
}
class data
{   int dia, mes, ano;  }
class pessoa
{   String nome;
    int idade;
    data nascimento;  }
class aluno
{   pessoa p;
    int n_mec;
    double nota_media;  }
```

Objeto a1 do tipo aluno

Objeto do tipo pessoa

Objeto do tipo data

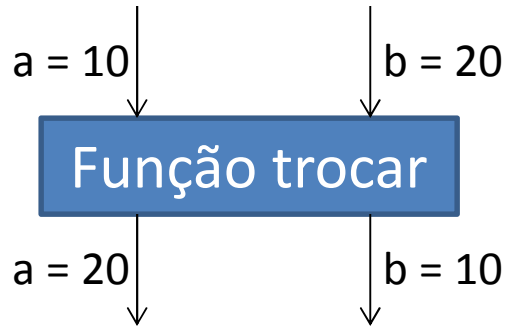
20 : 5 : 1992

Nuno : 21

1234 : 17.3

## Exemplo:

Descrever uma função que recebe dois argumentos inteiros e troca valores destes argumentos



**a = 10; b = 20**  
**a = 10; b = 20**

a variável de tipos primitivos é sempre passada por valor às funções como argumento

O código seguinte não permite trocar valores de a e b

```
import java.util.*;
public class trocar
{
    public static void trocar(int a, int b)    {
        int tmp;
        tmp = a; a = b; b = tmp;
    }

    public static void main(String[] args)
    {
        int a = 10, b = 20;
        System.out.printf("a = %d; b= %d\n", a, b);
        trocar(a,b);
        System.out.printf("a = %d; b= %d\n", a, b);
    }
}
```

```

import java.util.*;
public class trocar
{
    public static void trocar(int a, int b)    {
        int tmp;
        tmp = a; a = b; b = tmp;
    }

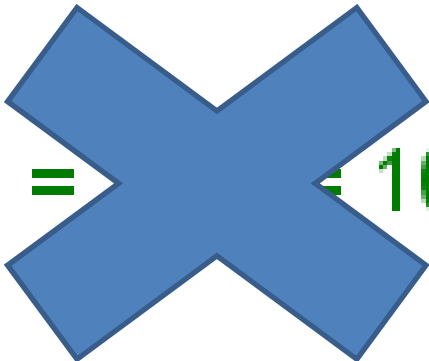
    public static void main(String[] args)
    {
        int a = 10, b = 20;
        System.out.printf("a = %d; b= %d\n", a, b);
        trocar(a,b);
        System.out.printf("a = %d; b= %d\n", a, b);
    }
}

```

**a = 10; b= 20**

**public static void trocar(int a, int b)**

**a = X = 10**



a variável de tipos primitivos é sempre passada por valor às funções como argumento

Uma função só pode devolver um valor

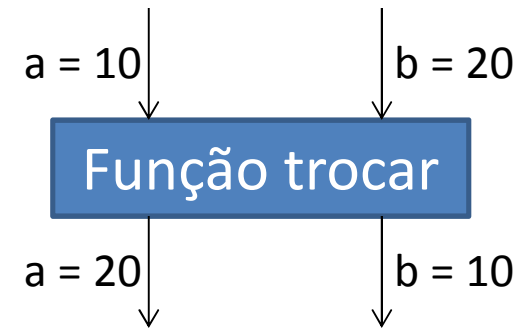


O código correto:

```
import java.util.*;
public class trocar
{
    public static void trocar(dados in)    {
        int tmp;
        tmp = in.a; in.a = in.b; in.b = tmp;
    }

    public static void main(String[] args)
    {
        dados d = new dados();
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
        trocar(d);
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
    }
}

class dados
{
    int a = 10;
    int b = 20;
}
```



`a = 10; b = 20`  
`a = 20; b = 10`

`dados d; // = new dados();`

Error: variable d might not have been initialized

```

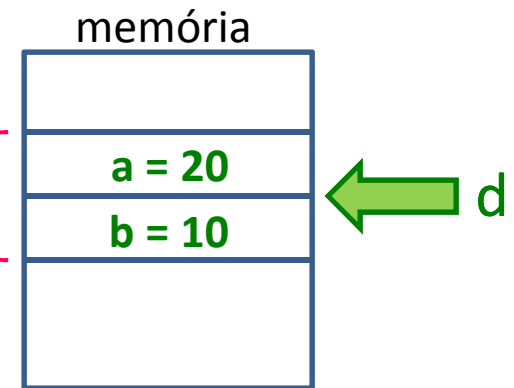
import java.util.*;
public class trocar
{
    public static void trocar(dados in)    {
        int tmp;
        tmp = in.a; in.a = in.b; in.b = tmp;
    }

    public static void main(String[] args)
    {
        dados d = new dados();
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
        trocar(d);
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
    }
}

class dados
{
    int a = 10;
    int b = 20;
}

```

Valeri Skliarov  
2016/2017



Troca vai ser feita em função  
main através de referência d

trocar(d)



a variável de tipo novo é  
passada por referência às  
funções como argumento

# O que é uma referência

```
dados d = new dados();  
//.....  
class dados  
{ int a = 10, int b = 20; }
```

1. Para os tipos novos (para registos) memória deve ser reservada, por exemplo, `new dados();`
2. Na linha `dados d = new dados();` `d` é uma referência que significa **onde fica na memória o objeto novo que foi criado**

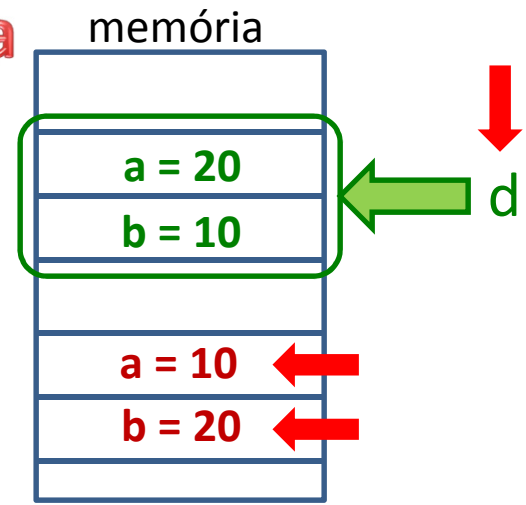
3. De notar que memória foi reservada fora de funções
4. Vamos chamar uma função `f` e passar `d` como argumento `f(d);`
5. Agora o argumento `d` dentro da função `f` pode utilizar dados na memória fora da função `f`
6. Quando a função `f` terminar a memória da função vai ser destruída mas a memória fora da função não vai ser destruída. Por isso todas as alterações do objeto feitas pela função `f` são válidas depois de terminação da função.
7. Para aceder um campo do objeto é necessário utilizar: *referencia.nome\_do\_campo*, por exemplo: `d.a`

# O que é uma referência

```
import java.util.*;
public class trocar
{
    public static void trocar(dados d, int a, int b)    {
        int tmp;
        tmp = d.a; d.a = d.b; d.b = tmp;
        tmp = a; a = b; b = tmp;
    }

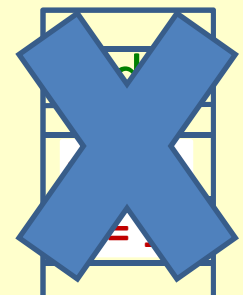
    public static void main(String[] args)
    {
        dados d = new dados();
        int a=10, b=20;
        System.out.printf("d.a = %d; d.b= %d\n", d.a, d.b);
        System.out.printf("a = %d; b= %d\n", a, b);
        trocar(d,a,b);
        System.out.printf("d.a = %d; d.b= %d\n", d.a, d.b);
        System.out.printf("a = %d; b= %d\n", a, b);
    }
}

class dados
{
    int a = 10;
    int b = 20; }
```



Troca de valores foi feita através de referências e não foi feita através de valores

```
public static void trocar(dados d, int a, int b)    {
    int tmp;
    tmp = d.a; d.a = d.b; d.b = tmp;
    tmp = a; a = b; b = tmp;
}
```



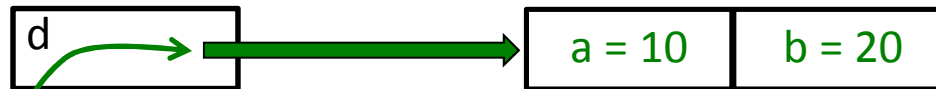
dados d;



d = **new** dados();

```
class dados  
{ int a = 10;  
  int b = 20; }
```

dados d = **new** dados();

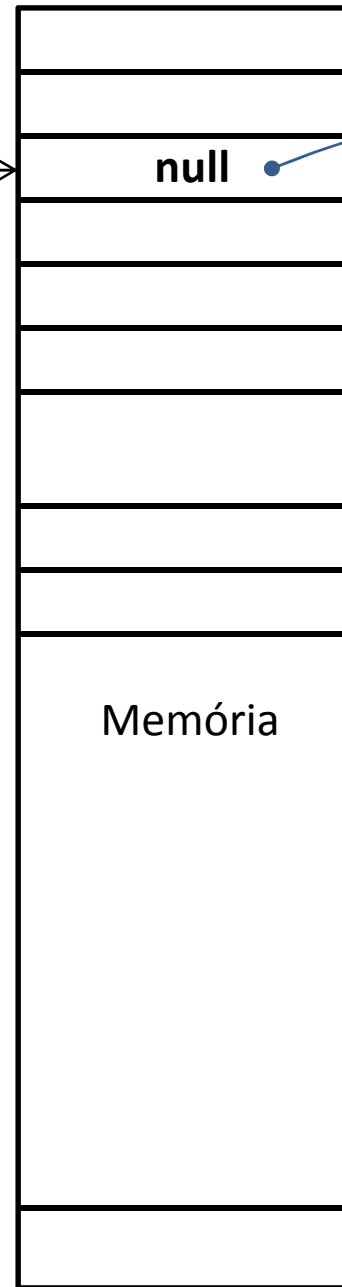


trocar(d);

- A declaração da variável `d` cria apenas uma referência para o que será mais tarde um endereço da memória com variáveis `a` e `b`.
- A invocação do operador **new** vai reservar espaço na memória do computador para objeto do tipo `d`, ficando a variável `d` com o endereço onde esse "espaço" se encontra na memória.
- O operador **new** inicializa todos os campos da estrutura com valores indicados.
- A partir deste momento, o objeto `d` pode ser manipulado através da variável `d`.

```
class dados
{
    int a = 10;
    int b = 20;
}
```

d é endereço  
(referência)



dado d;

d = **new** dados();

Pretendemos criar a tabela em baixo

n – nome de aluno	n_mc - número mecanográfico de aluno	n_m - nota média de aluno
-------------------	--------------------------------------	---------------------------

---

```
class aluno
{
    String n;      // nome de aluno
    int n_mc;      // número mecanográfico de aluno
    double n_m;    // nota média de aluno
}
```

A tabela está vazia e não tem linhas

```
aluno a1 = new aluno(); aluno a2 = new aluno(); aluno a3 = new aluno();
```

Agora três linhas vazias foram adicionadas

a1

a2

a3

	n – nome de aluno	n_mc - número mecanográfico de aluno	n_m - nota média de aluno
a1	Pedro	1624	14.6
a2	Cláudia	3726	18.1
a3	Carla	1926	10.1

Agora vamos preencher a tabela

```
a1.n = "Pedro"; a1.n_mc = 1624; a1.n_m = 14.6;
```

```
a2.n = "Claudia"; a2.n_mc = 3726; a2.n_m = 18.1;
```

```
a3.n = "Carla"; a3.n_mc = 1926; a3.n_m = 10.1;
```

```
class aluno
{
    String n;      // nome de aluno
    int n_mc;      // número mecanográfico de aluno
    double n_m;    // nota média de aluno
}
```



# Construir um novo objeto

```
public class trocar_const
{
    public static void trocar(ob d)    {
        int tmp;
        tmp = d.a; d.a = d.b; d.b = tmp;
    }

    public static void main(String[] args)
    {
        ob d = new ob(30,40);
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
        trocar(d);
        System.out.printf("a = %d; b= %d\n", d.a, d.b);
    }

    static class ob
    {
        ob(int x, int y)
        { a = x; b = y; }
        int a;
        int b;
    }
}
```

Reserva de memória e chamada do construtor do objeto (construção do objeto d)

Construtor do objeto

Construtores vão ser usados no segundo semestre

## **Exemplo 1:** Criar a classe aluno

```
public class aluno_class
{

    public static void main(String[] args)
    {
        aluno a1 = new aluno("Pedro",1624,14.6);
        aluno a2 = new aluno("Claudia",3726,18.1);
        aluno a3 = new aluno("Carla",1926,10.1);
        System.out.printf("%s  %d  %f\n", a1.n, a1.n_mc, a1.n_m);
        System.out.printf("%s  %d  %f\n", a2.n, a2.n_mc, a2.n_m);
        System.out.printf("%s  %d  %f\n", a3.n, a3.n_mc, a3.n_m);
    }
}

class aluno
{
    aluno(String nome, int n_mec, double nota_media)
    { n = nome; n_mc = n_mec; n_m = nota_media; }
    String n;
    int n_mc;
    double n_m;
}
```

## **Exemplo 1:** Criar a classe aluno sem construtor

```
public class aluno_class
{

    public static void main(String[] args)
    {
        aluno a1 = new aluno(); a1.n = "Pedro"; a1.n_mc = 1624; a1.n_m = 14.6;
        aluno a2 = new aluno(); a2.n = "Claudia"; a2.n_mc = 3726; a2.n_m = 18.1;
        aluno a3 = new aluno(); a3.n = "Carla"; a3.n_mc = 1926; a3.n_m = 10.1;
        System.out.printf("%s  %d  %f\n", a1.n, a1.n_mc, a1.n_m);
        System.out.printf("%s  %d  %f\n", a2.n, a2.n_mc, a2.n_m);
        System.out.printf("%s  %d  %f\n", a3.n, a3.n_mc, a3.n_m);
    }
}

class aluno
{
    String n;
    int n_mc;
    double n_m;
}
```

# Tipos de dados primitivos e referência

- Tipos de **dados primitivos**:
  - a declaração da variável cria automaticamente a variável, reservando espaço em memória;
  - a variável é sempre passada por valor às funções como argumento.
- Tipos de **dados referência**:
  - a declaração da variável não cria de fato uma variável desse tipo, cria apenas uma referência;
  - a criação do objeto correspondente é feita com o operador `new`;
  - o objeto é sempre passado por referência como argumento às funções.

# Cópia de variáveis tipo referência

- Atenção à cópia de uma variável tipo referência: é necessário distinguir a cópia do objeto da cópia da referência propriamente dita.
- Este é um dos erros frequentemente cometido pelos programadores.

```
Complexo x = new Complexo();  
Complexo y = new Complexo();  
x.real = 10;  
x.imag = 20;
```

```
class Complexo{  
    double real, imag;  
}
```

```
y = x; // estamos a copiar a referência e não o conteúdo  
// Para copiar o conteúdo:  
y.real = x.real; // cópia do campo real  
y.imag = x.imag; // cópia do campo imag
```

# Conclusão

Declaração de referência permite só reservar memória para esta referência más não reserva memória para o conteúdo

A cópia de referências não permite copiar os conteúdos

A cópia de referência cria nomes que podem ser usados para aceder o mesmo registo

# **Erros mais comuns na avaliação**

Utilização de registos sem reserva de memória

Copiar referências mas não copiar o conteúdo

# Exemplo mais complicado

Separate slides



## Exemplos adicionais.

```
class docente {  
    String nome;  
    int idade;  
    double salário;  
    int anos_de_serviço;  
    String especialização;  
}
```

## Exemplo da classe 1:

```
class aluno {  
    String nome;  
    int idade;  
    int n_mec;  
    int ano;  
    String curso;  
}
```

```
class docente {  
    String nome;  
    int idade;  
    double salário;  
    int anos_de_serviço;  
    String especialização;  
    aluno início_da_lista_de_alunos;  
}
```

```
class docente {  
    String nome;  
    int idade;  
    double salário;  
    int anos_de_serviço;  
    String especialização;  
    aluno turma[];  
}
```



## Exemplo da classe 2:

```
class computador {  
    processador proc;  
    int tamanho_de_memoria;  
    double frequência;  
    double preço;  
}
```

```
class processador {  
    String nome;  
    double preço;  
    int ano;  
}
```

```
class casa {  
    String endereço;  
    int idade;  
    double preço;  
    double area;  
}
```

```
class pais {  
    String nome;  
    cidade início_da_lista_de_cidades;  
    double área;  
    int população;  
    localização loc;  
    pessoa presidente;  
}
```

```
class pessoa {  
    String nome;  
    int idade;  
    // .....  
}
```

# Exemplo de uma lista ligada

Definição da  
classe **A**

```
class A  
{ int x;  
  A proximo = null; }
```

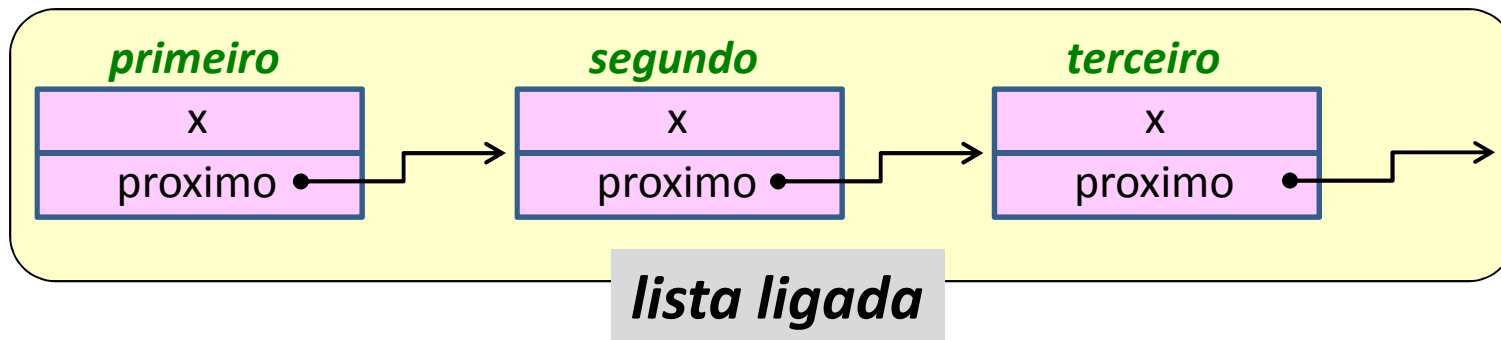
Definição de um  
objeto **first** do tipo **A**

```
A first = new A();
```

é uma referência (endereço) de outro objeto do tipo A

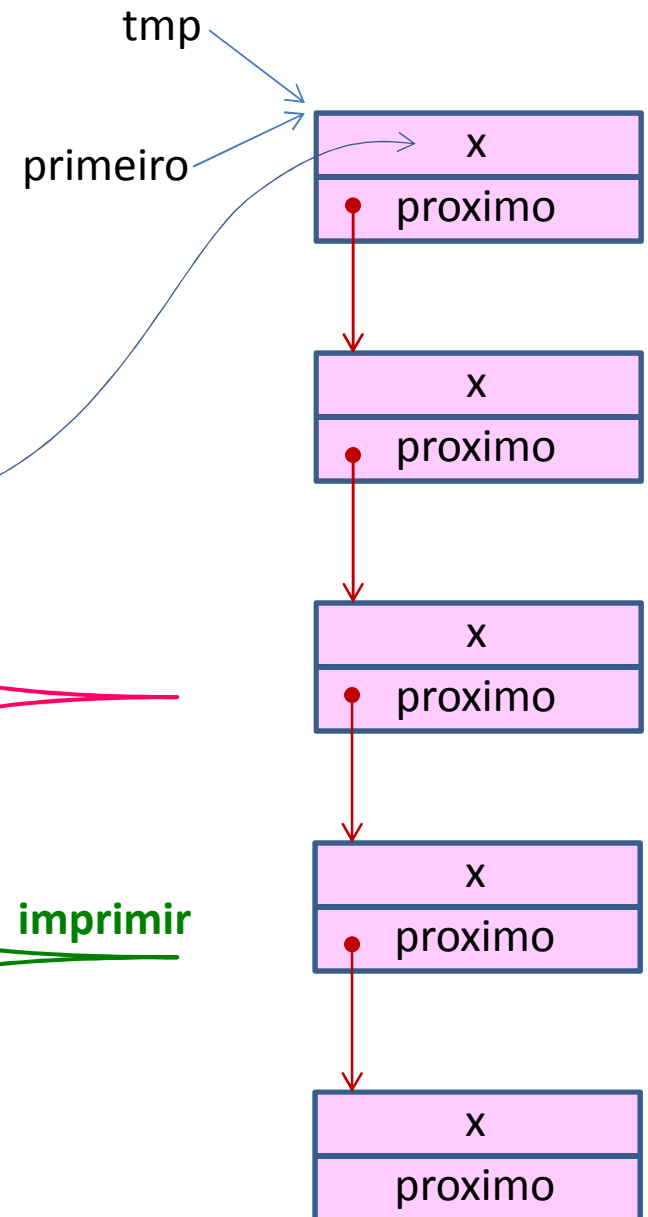
Definição da  
referência **primeiro**  
e de outros objetos  
**segundo, terceiro,**  
etc.

```
A primeiro = first;  
A segundo = new A();  
primeiro.proximo = segundo;  
A terceiro = new A();  
segundo.proximo = terceiro;  
// .....
```



# Exemplo de uma lista ligada

```
import java.util.*;
public class lista_ligada_class1
{   public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    { A tmp = new A();
      A primeiro = tmp;
      System.out.print("Inteiro ? ");
      tmp.x = sc.nextInt();
      for(int i = 1; i < 5; )
      { if (tmp.proximo == null)
        { i++; System.out.print("Inteiro ? ");
          tmp.proximo = new A(); tmp.proximo.x = sc.nextInt(); }
        else tmp = tmp.proximo;
      }
      for(tmp = primeiro; tmp.proximo != null;)
      {   System.out.println(tmp.x);
          tmp = tmp.proximo;      }
      System.out.println(tmp.x);
    }
}
class A
{   int x;
    A proximo = null; }
```



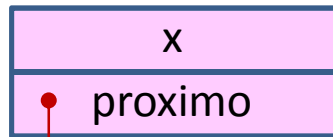
## Exemplo de uma lista ligada

```
import java.util.*;
public class lista_ligada_class1
{   public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    { A tmp = new A();
      A primeiro = tmp;
      System.out.print("Inteiro  ? ");
      tmp.x = sc.nextInt();
      for(int i = 1; i < 5; )
          if (tmp.proximo == null)
              { i++; System.out.print("Inteiro  ? ");
                tmp.proximo = new A(); tmp.proximo.x = sc.nextInt(); }
              else tmp = tmp.proximo;
      for(tmp = primeiro; tmp.proximo != null;)
      {   System.out.println(tmp.x);
          tmp = tmp.proximo;      }
      System.out.println(tmp.x);
      }
    }
class A
{   int x;
    A proximo = null;  }
```

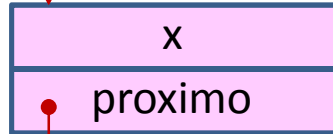
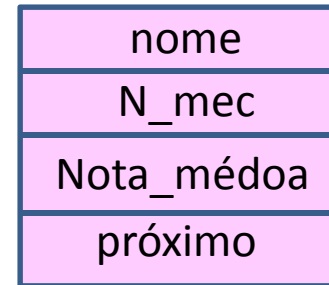
Inteiro	?	1
Inteiro	?	2
Inteiro	?	3
Inteiro	?	4
Inteiro	?	5
Inteiro	?	6
Inteiro	?	7
Inteiro	?	8
Inteiro	?	9
Inteiro	?	10

```
1
2
3
4
5
6
7
8
9
10
-----
(program exited with code: 0)
Press return to continue
```

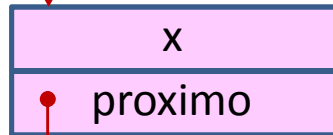
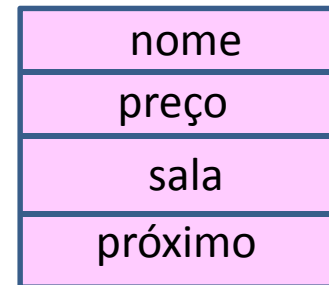
# Exemplo de uma lista ligada



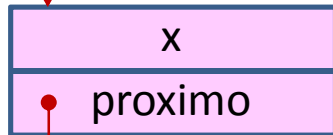
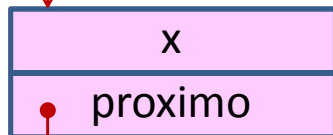
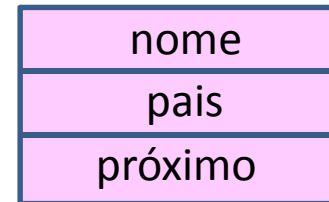
Pode ser uma lista de alunos



Uma lista de equipamentos



Uma lista de cidades

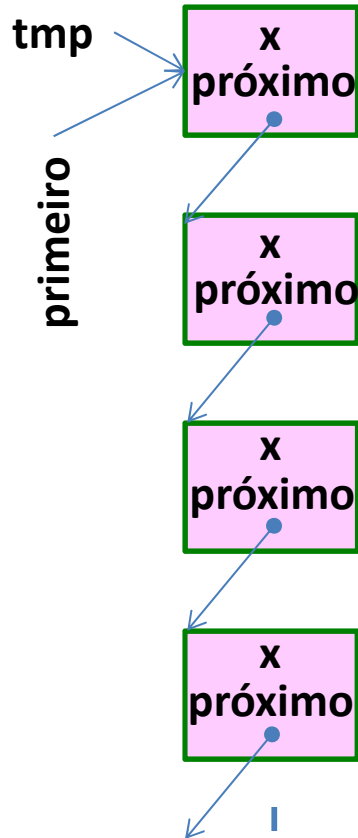


Etc.

É importante:

1. Pode adicionar um elemento novo facilmente.
2. Pode remover elementos facilmente.
3. Só reserva memória para elementos que estão incluídos na lista

## Exemplo:



```
A tmp = new A();  
A primeiro = tmp;
```

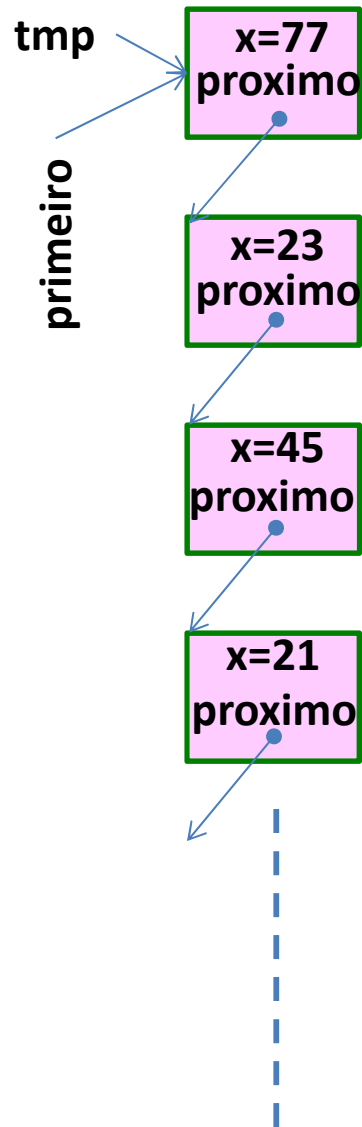
## Criar

```
for(int i = 1; i < 5; )  
  if (tmp.proximo == null)  
    { i++; System.out.print("Inteiro ? ");  
      tmp.proximo = new A(); tmp.proximo.x = sc.nextInt(); }  
  else tmp = tmp.proximo;
```

## Percorrer

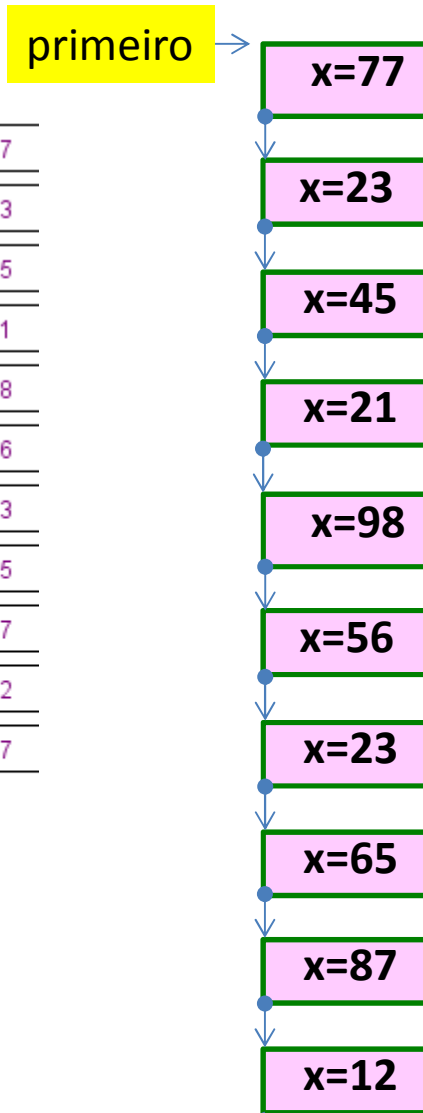
```
for(tmp = primeiro; tmp.proximo != null;)  
{ System.out.println(tmp.x);  
  tmp = tmp.proximo; }  
System.out.println(tmp.x);
```

```
class A  
{ int x;  
  A proximo = null; }
```



Inteiro	?	77
Inteiro	?	23
Inteiro	?	45
Inteiro	?	21
Inteiro	?	98
Inteiro	?	56
Inteiro	?	23
Inteiro	?	65
Inteiro	?	87
Inteiro	?	12
Inteiro	?	37

77  
23  
45  
21  
98  
56  
23  
65  
87  
12



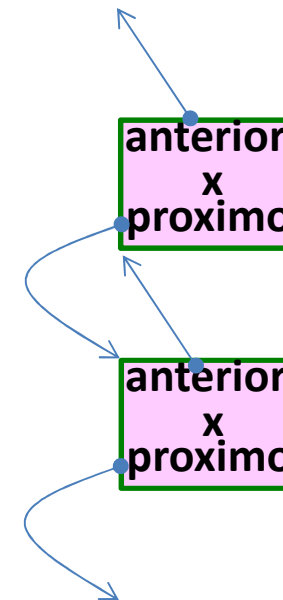
proximo = null



## Exemplo: Criar uma pilha

```
import java.util.*;
public class pilha
{
    public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    {
        A primeiro = new A();
        A last = primeiro, tmp = primeiro;
        System.out.print("Inteiro ? ");
        tmp.x = sc.nextInt();
        for(int i = 1; i < 5; )
            if (tmp.proximo == null)
            {
                i++; System.out.print("Inteiro ? ");
                tmp.proximo = new A();
                tmp.proximo.x = sc.nextInt();
                last = tmp.proximo;
                tmp.proximo.anterior = tmp;
            }
            else tmp = tmp.proximo;
        for(; last.anterior != null;)
        {
            System.out.println(last.x);
            last = last.anterior;
        }
        System.out.println(last.x);
    }
}
```

```
class A
{
    int x;
    anterior= null;
    proximo= null;
}
```

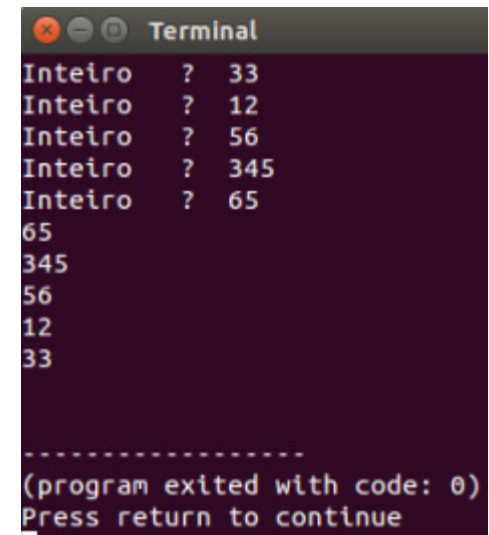


## Exemplo: Criar uma pilha

```
import java.util.*;
public class pilha
{   public static Scanner sc = new Scanner(System.in);
    public static void main(String[] args)
    {
        A primeiro = new A();
        A last = primeiro, tmp = primeiro;
        System.out.print("Inteiro  ? ");
        tmp.x = sc.nextInt();
        for(int i = 1; i < 5; )
            if (tmp.proximo == null)
            { i++; System.out.print("Inteiro  ? ");
              tmp.proximo = new A();
              tmp.proximo.x = sc.nextInt();
              last = tmp.proximo;
              tmp.proximo.anterior = tmp; }
            else tmp = tmp.proximo;
        for(; last.anterior != null;)
        {
            System.out.println(last.x);
            last = last.anterior;
        }
        System.out.println(last.x);
    }
}
```

Inteiro	?	1
Inteiro	?	2
Inteiro	?	3
Inteiro	?	4
Inteiro	?	5

5  
4  
3  
2  
1



```
Terminal
Inteiro  ? 33
Inteiro  ? 12
Inteiro  ? 56
Inteiro  ? 345
Inteiro  ? 65
65
345
56
12
33
-----
(program exited with code: 0)
Press return to continue
```