

Paradigmas de programação

As linguagens podem ser agrupadas consoante a
“filosofia” de programação que defendem

Paradigmas de programação

As linguagens podem ser agrupadas consoante a “filosofia” de programação que defendem

- Imperativo
- Funcional
- Orientado a objectos

Paradigma imperativo

Conjunto sequencial de instruções que afectam variáveis, passível de ser representado por um fluxograma

Paradigma imperativo

Conjunto sequencial de instruções que afectam variáveis, passível de ser representado por um fluxograma

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int i, j, rows;
```

Declara variáveis



```
    printf("Enter the number of rows: ");
```

```
    scanf("%d",&rows);
```

```
    for(i=1; i<=rows; ++i)
```

```
    {
```

```
        for(j=1; j<=i; ++j)
```

```
        {
```

```
            printf("* ");
```

```
        }
```

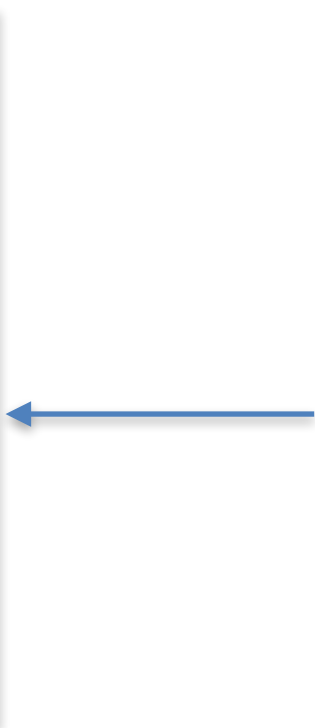
```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

Conjunto de instruções
que afetam essas variáveis



Paradigma funcional

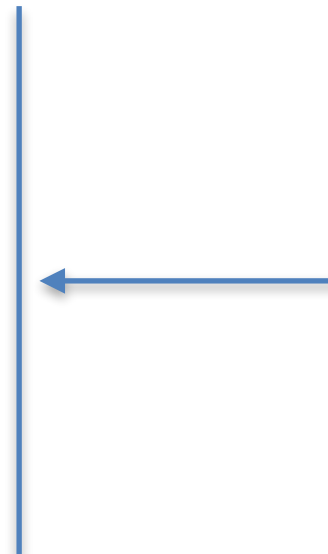
As funções podem ser atribuídas a variáveis e passadas como parâmetro para outras funções. Todas as variáveis são locais à função. Os dados são passados diretamente entre as funções, seja pelos parâmetros seja pelo retorno da função.

Paradigma funcional

As funções podem ser atribuídas a variáveis e passadas como parâmetro para outras funções. Todas as variáveis são locais à função. Os dados são passados diretamente entre as funções, seja pelos parâmetros seja pelo retorno da função.

```
const render = comp(  
  partial(join, '\n'),  
  partial(concat, header()),  
  partial(map, renderResult),  
  partial(zipmap, range(1, 11))  
);
```

(Exemplo em Javascript)



Não há variáveis declaradas, o output de uma função é o input de outra e por aí fora

Paradigma funcional

As funções podem ser atribuídas a variáveis e passadas como parâmetro para outras funções. Todas as variáveis são locais à função. Os dados são passados diretamente entre as funções, seja pelos parâmetros seja pelo retorno da função.

```
const render = comp(  
  partial(join, '\n'),  
  partial(concat, header()),  
  partial(map, renderResult),  
  partial(zipmap, range(1, 11))  
);
```

(Exemplo em Javascript)

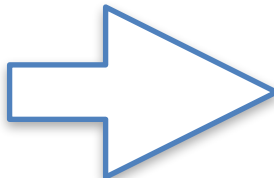
As funções podem ser passadas como parâmetro para outras funções

Paradigma orientado a objectos

Conjunto de entidades (objetos) que agregam estado (variáveis) e comportamento (funções)

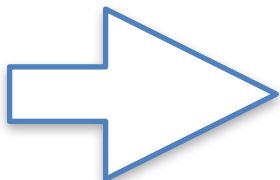
Paradigma orientado a objectos

Conjunto de entidades (objetos) que agregam estado (variáveis) e comportamento (funções)

`move(boneco, 2, 3)`  `boneco.move(2, 3)`

Paradigma orientado a objectos

Conjunto de entidades (objetos) que agregam estado (variáveis) e comportamento (funções)

`move(boneco, 2, 3)`  `boneco.move(2, 3)`

```
class Boneco {
```

```
    int x, y;
```

```
    void move(int x, int y) {
```

```
        this.x = x;
```

```
        this.y = y;
```

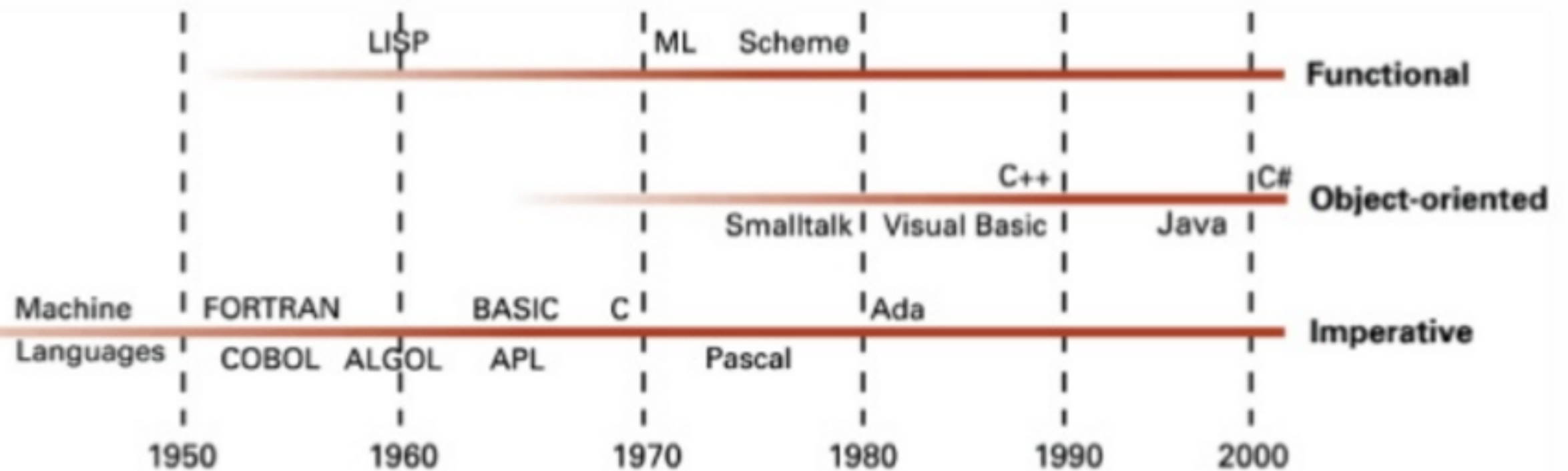
```
    }
```

```
}
```

Estado

Comportamento

Paradigmas de programação



Paradigmas de programação

| Paradigma | Descrição | Bom para... | Linguagens |
|------------|--|---|-------------|
| Imperativo | Conjunto de instruções que afetam variáveis e são executadas de forma sequencial | Problemas algorítmicos simples (ex: ordenar um array) | Assembly, C |

Paradigmas de programação

| Paradigma | Descrição | Bom para... | Linguagens |
|------------|--|---|-----------------------------------|
| Imperativo | Conjunto de instruções que afetam variáveis e são executadas de forma sequencial | Problemas algorítmicos simples (ex: ordenar um array) | Assembly, C |
| Funcional | Conjunto de funções que chamam funções, passando o estado (variáveis) entre elas | Programação paralela (ex: processamento de grandes quantidades de informação usando múltiplos computadores) | Haskell, Erlang, Lisp, Javascript |

Paradigmas de programação

| Paradigma | Descrição | Bom para... | Linguagens |
|----------------------|---|---|-----------------------------------|
| Imperativo | Conjunto de instruções que afetam variáveis e são executadas de forma sequencial | Problemas algorítmicos simples (ex: ordenar um array) | Assembly, C |
| Funcional | Conjunto de funções que chamam funções, passando o estado (variáveis) entre elas | Programação paralela (ex: processamento de grandes quantidades de informação usando múltiplos computadores) | Haskell, Erlang, Lisp, Javascript |
| Orientado a objectos | Conjunto de entidades que comunicam entre si. Cada entidade agrega estado e comportamento | Problemas complexos envolvendo muitas entidades (ex: gestão de funcionários de uma empresa) | C#, Java, Ruby, Python |

Paradigmas de programação

Hoje em dia, as principais linguagens incorporam de alguma forma os 3 paradigmas, embora haja um paradigma dominante

Praticamente todas as linguagens incorporam o paradigma imperativo pois é a forma mais simples de implementar algoritmos

Algumas linguagens orientadas a objetos têm vindo a incorporar mecanismos do paradigma funcional: Java, C#

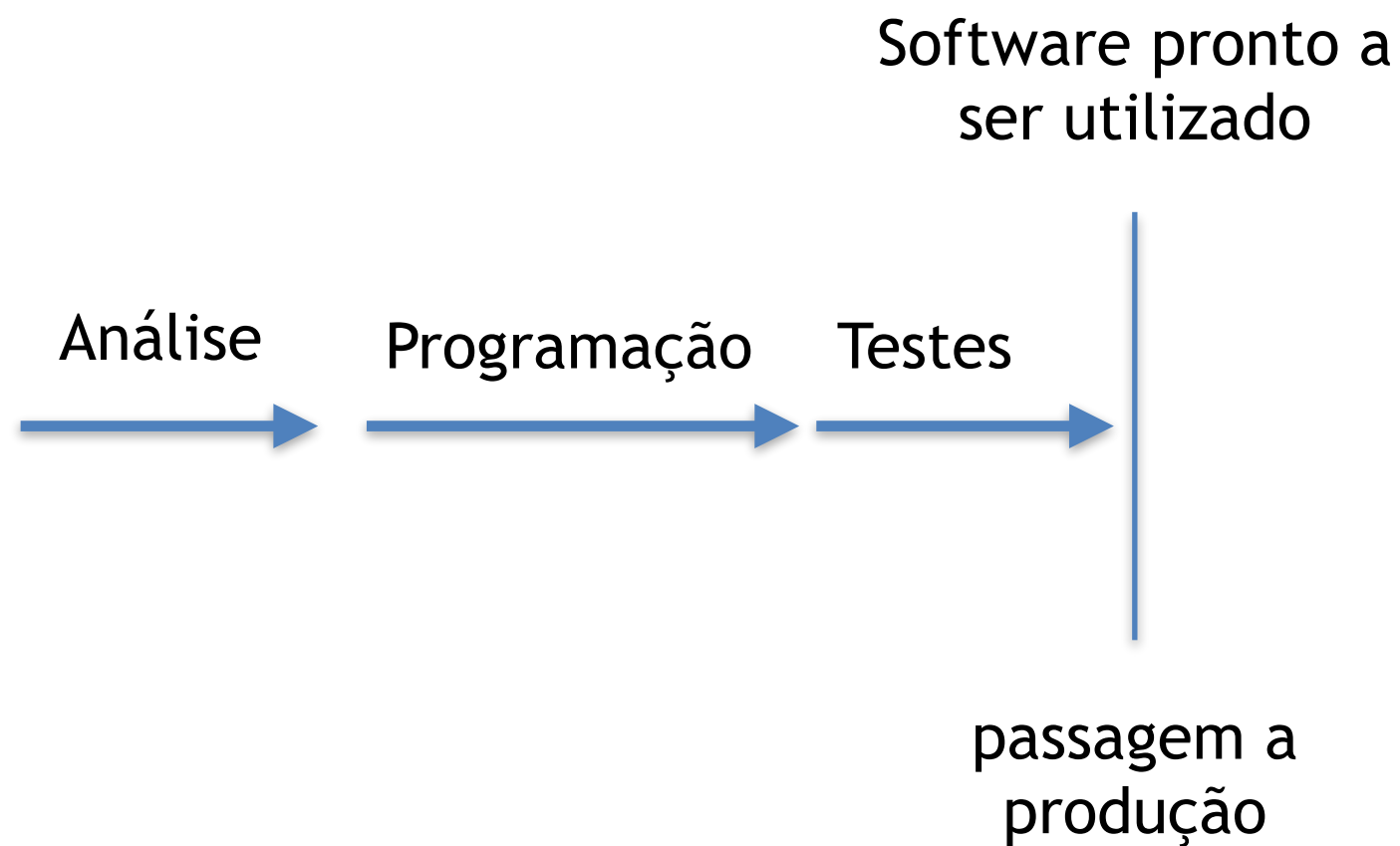
O Javascript começou por ser mais funcional mas a versão mais recente (ES6) já introduz mecanismos orientados a objetos

O Kotlin foi já criado de raiz com um cariz multiparadigma

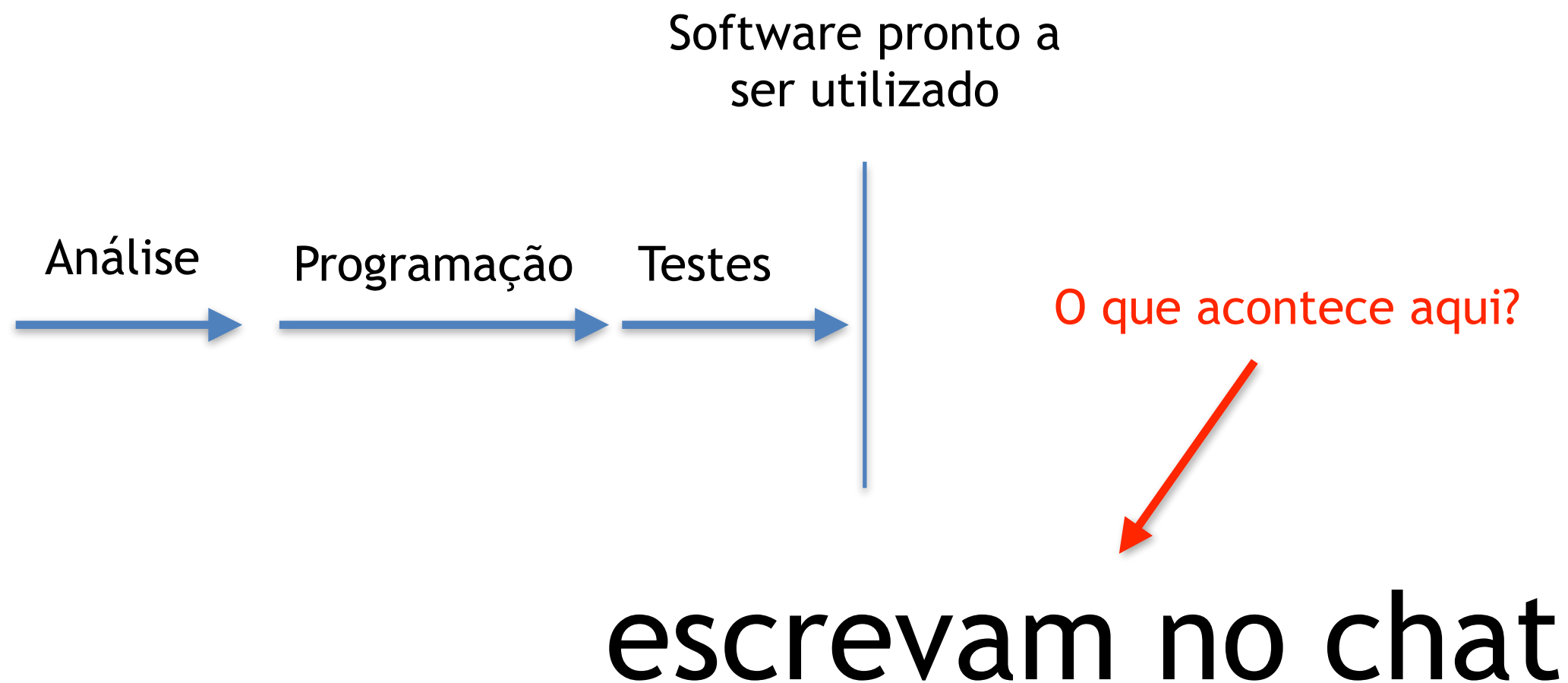
Porquê programação orientada a objetos?



Ciclo de vida do software



Ciclo de vida do software



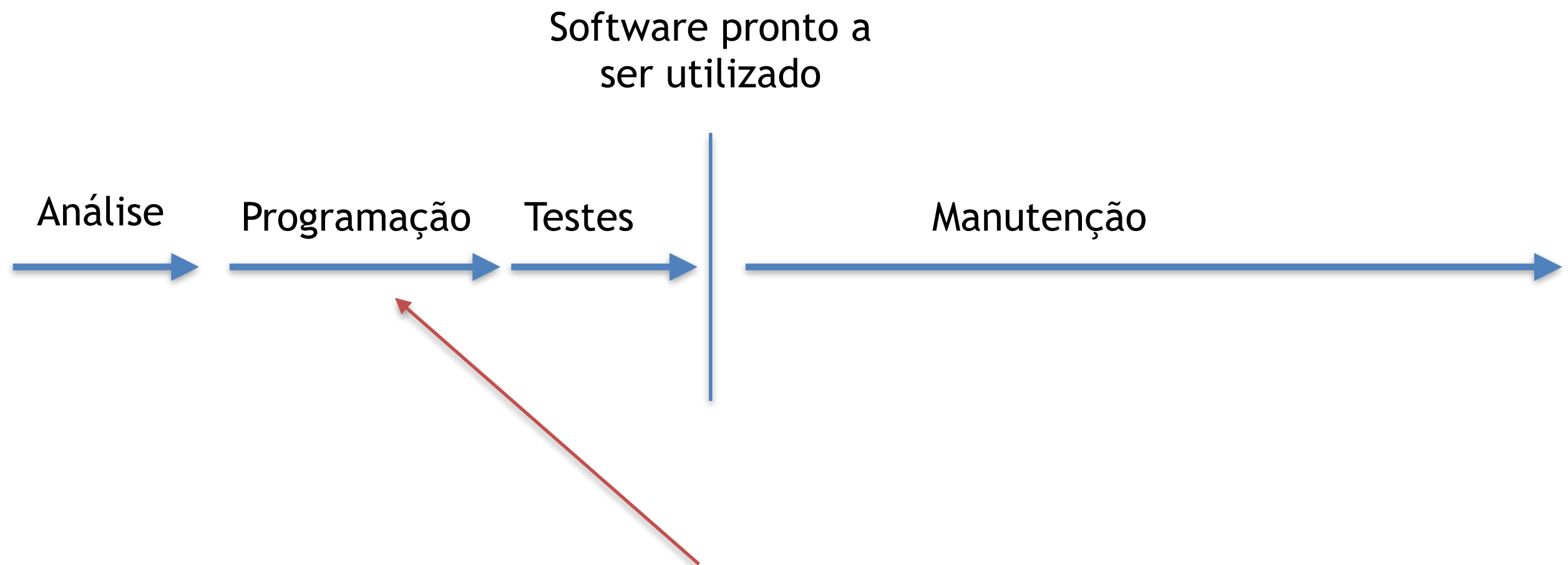
Ciclo de vida do software



Manutenção

- Correção de erros que vão sendo detectados pelos utilizadores
- Melhorias diversas
- Novas funcionalidades
- etc.

Ciclo de vida do software



É aqui que devemos preparar o software para uma fácil manutenção!

Complexidade

Quantas linhas de código tem um sistema operativo?

Complexidade

| Sistema Operativo | LOC (Milhões) |
|---------------------|---------------|
| Windows NT 3.1 | 4-5 |
| Windows 2000 | > 29 |
| Windows XP | 45 |
| Windows Server 2003 | 50 |
| Linux 3.6 | 15.9 |

LOC = Lines of Code

Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
  
    public static void main(String[] args) {  
  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```


Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
  
    public static void main(String[] args) {  
  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```

numero

3

Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
    public static void main(String[] args) {  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```

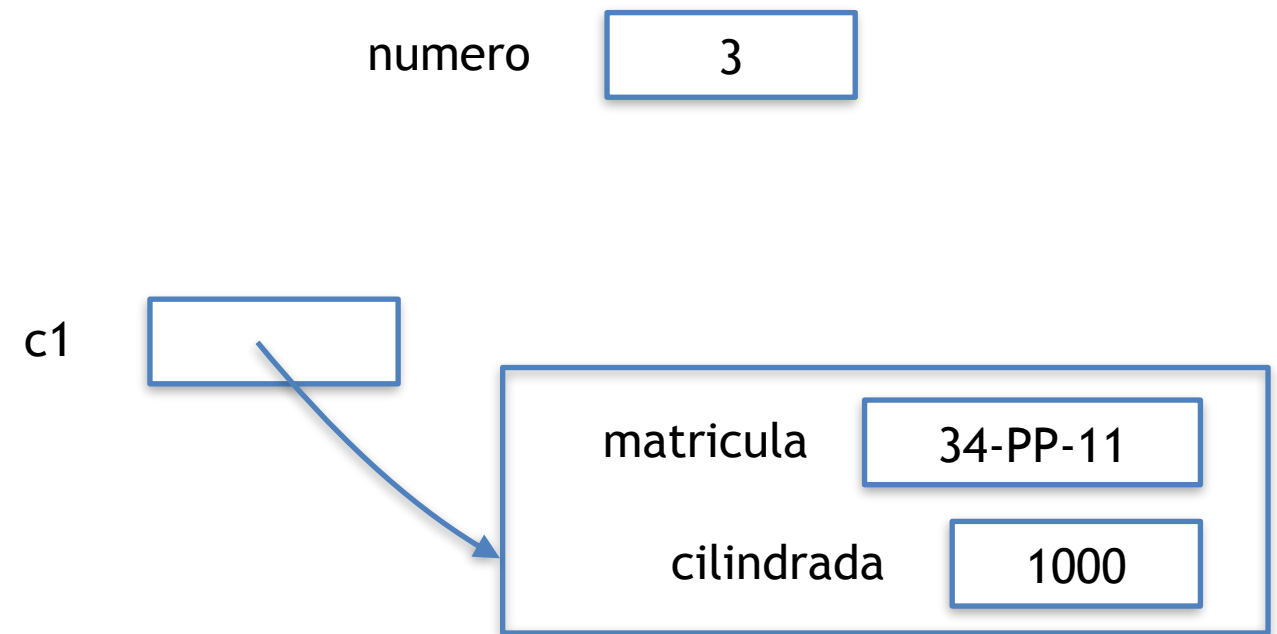


Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
    public static void main(String[] args) {  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```

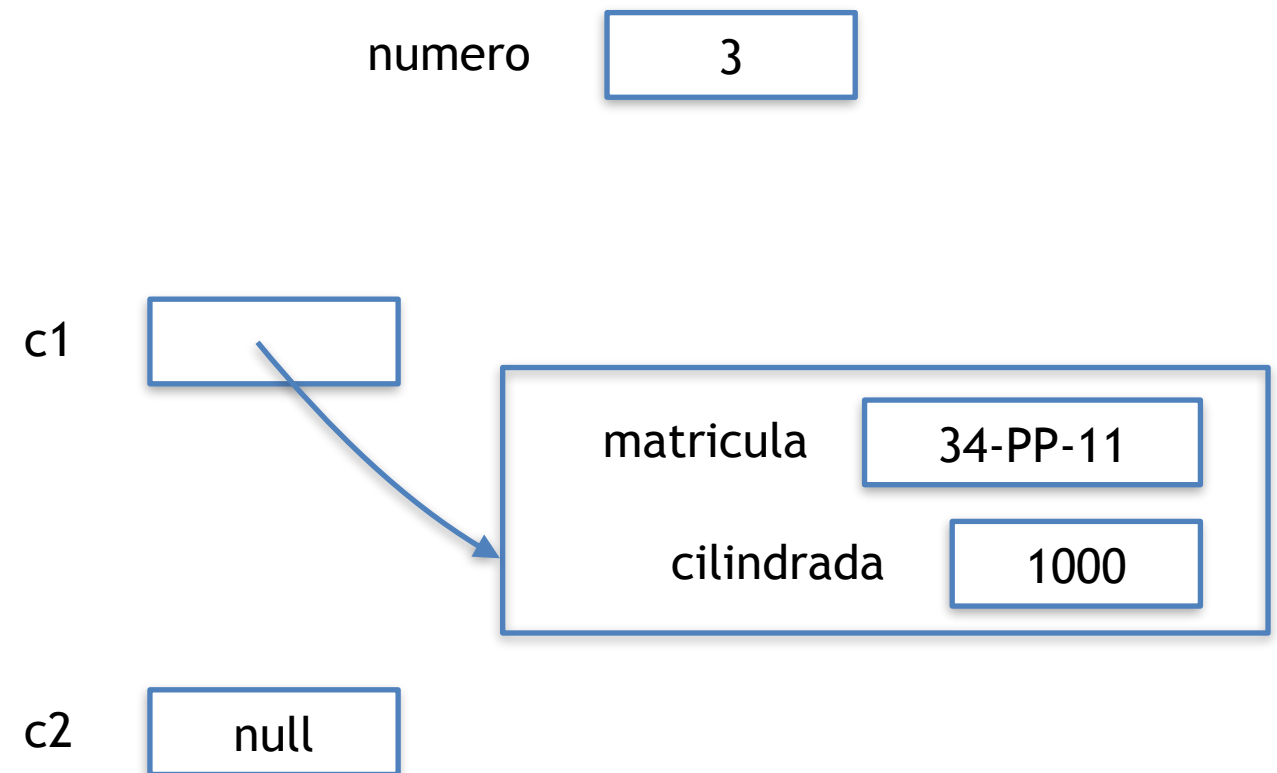


Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
    public static void main(String[] args) {  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```

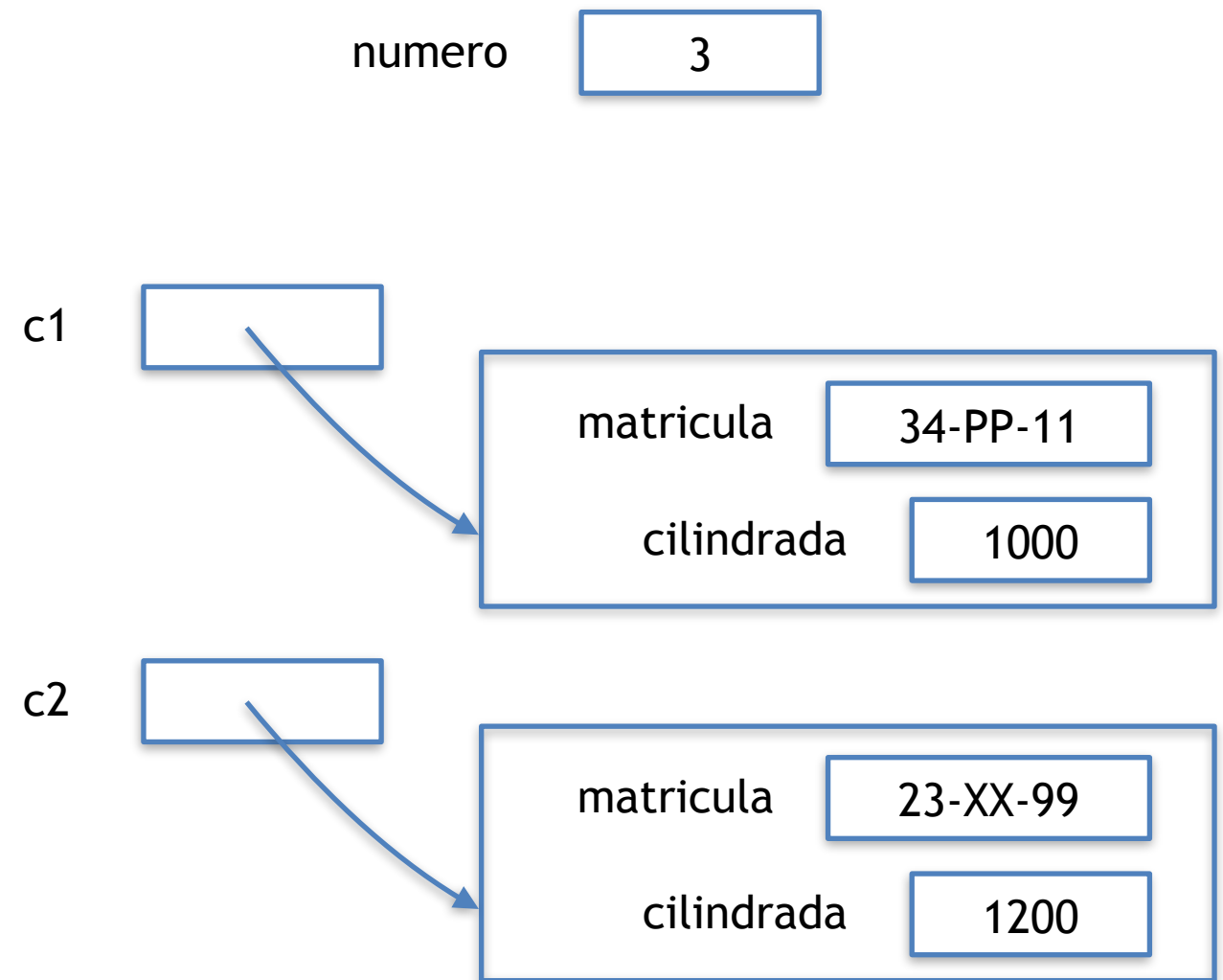


Diagrama de memória (1)

```
class Carro {  
    String matricula;  
    int cilindrada;  
  
    // construtor escondido para simplificar  
    public Carro(String matricula,  
                  int cilindrada) {...}  
}  
  
public class Aplicacao {  
    public static void main(String[] args) {  
        int numero = 3;  
        Carro c1 = new Carro("34-PP-11", 1000);  
        Carro c2;  
  
        c2 = new Carro("23-XX-99", 1200);  
  
        Carro c3;  
    }  
}
```

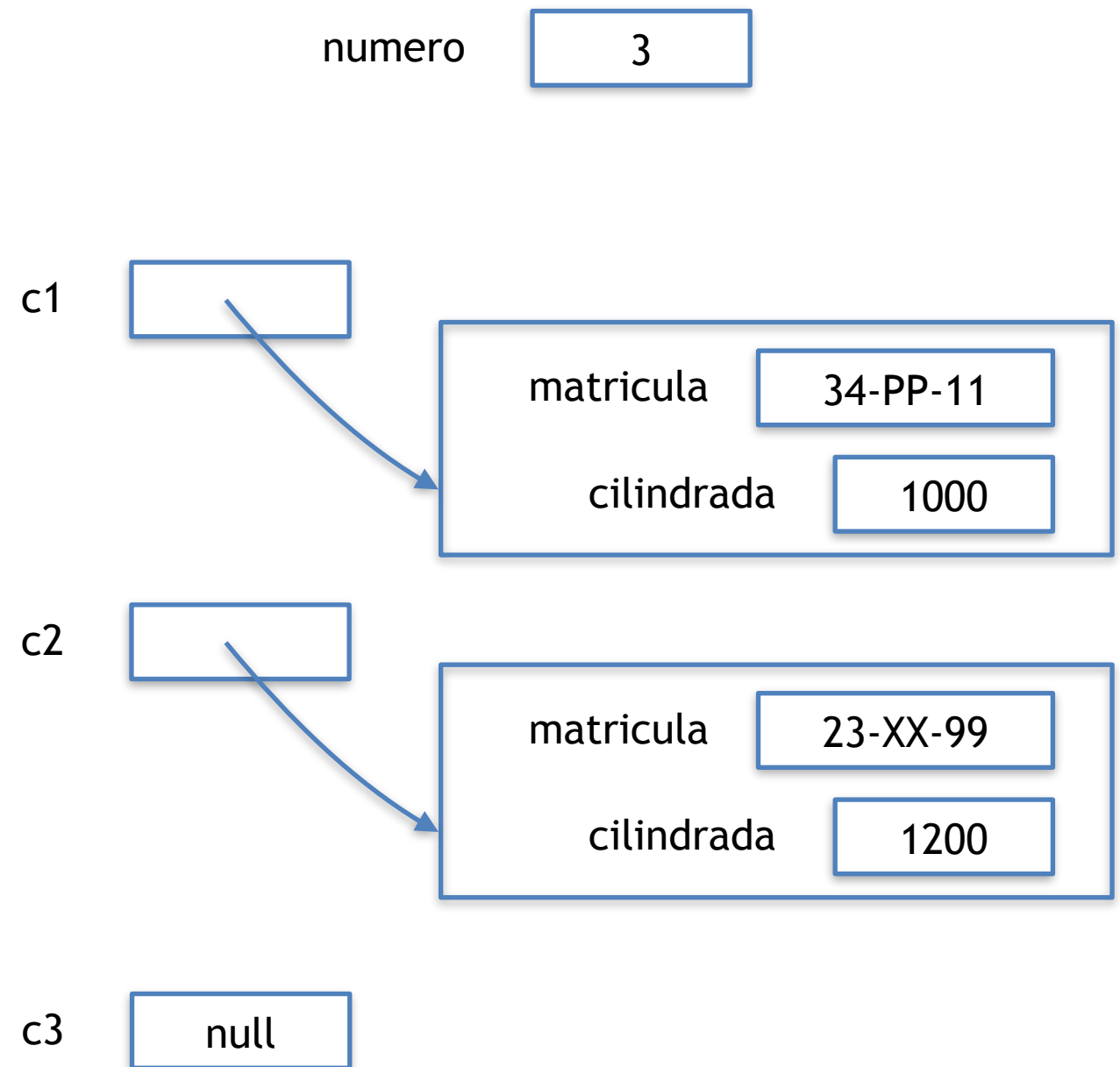


Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;
    }
}
```

(alguns construtores omitidos por simplificação)

Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;
    }
}
```



Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;
    }
}
```



Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;

    }
}
```

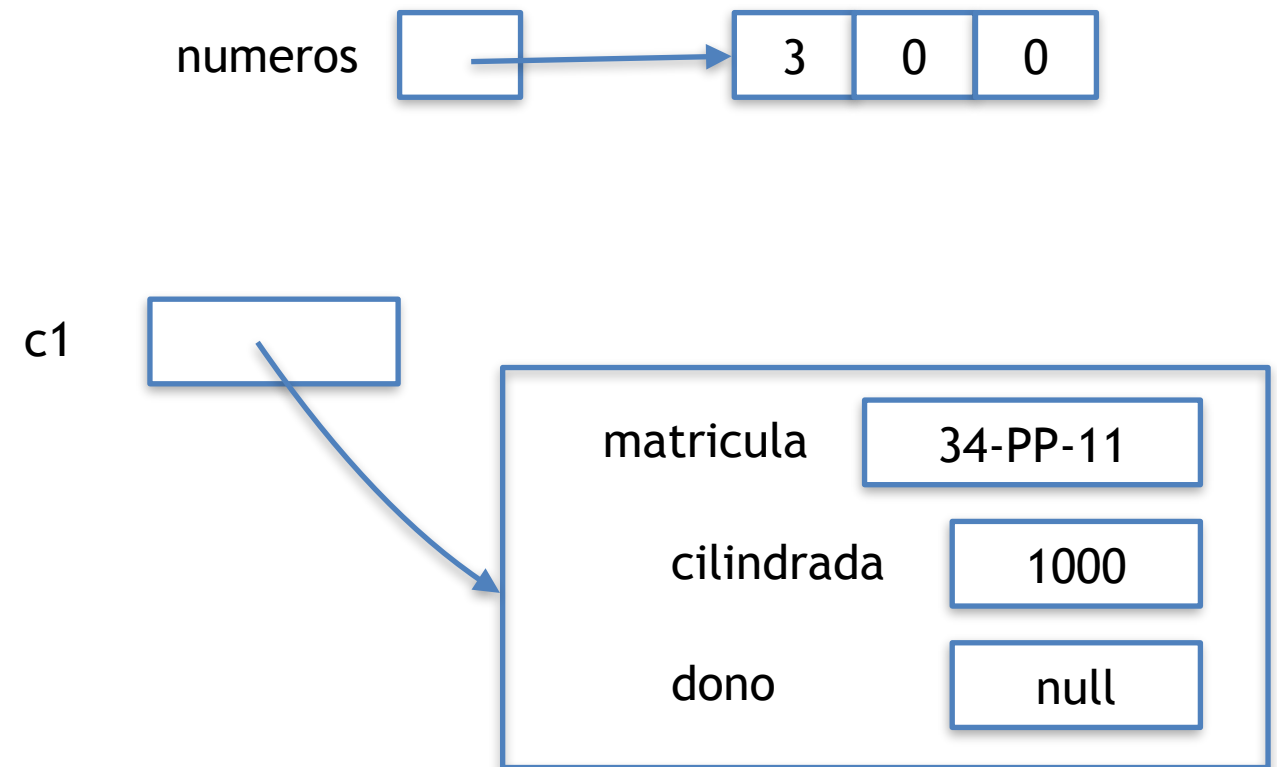


Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;

    }
}
```

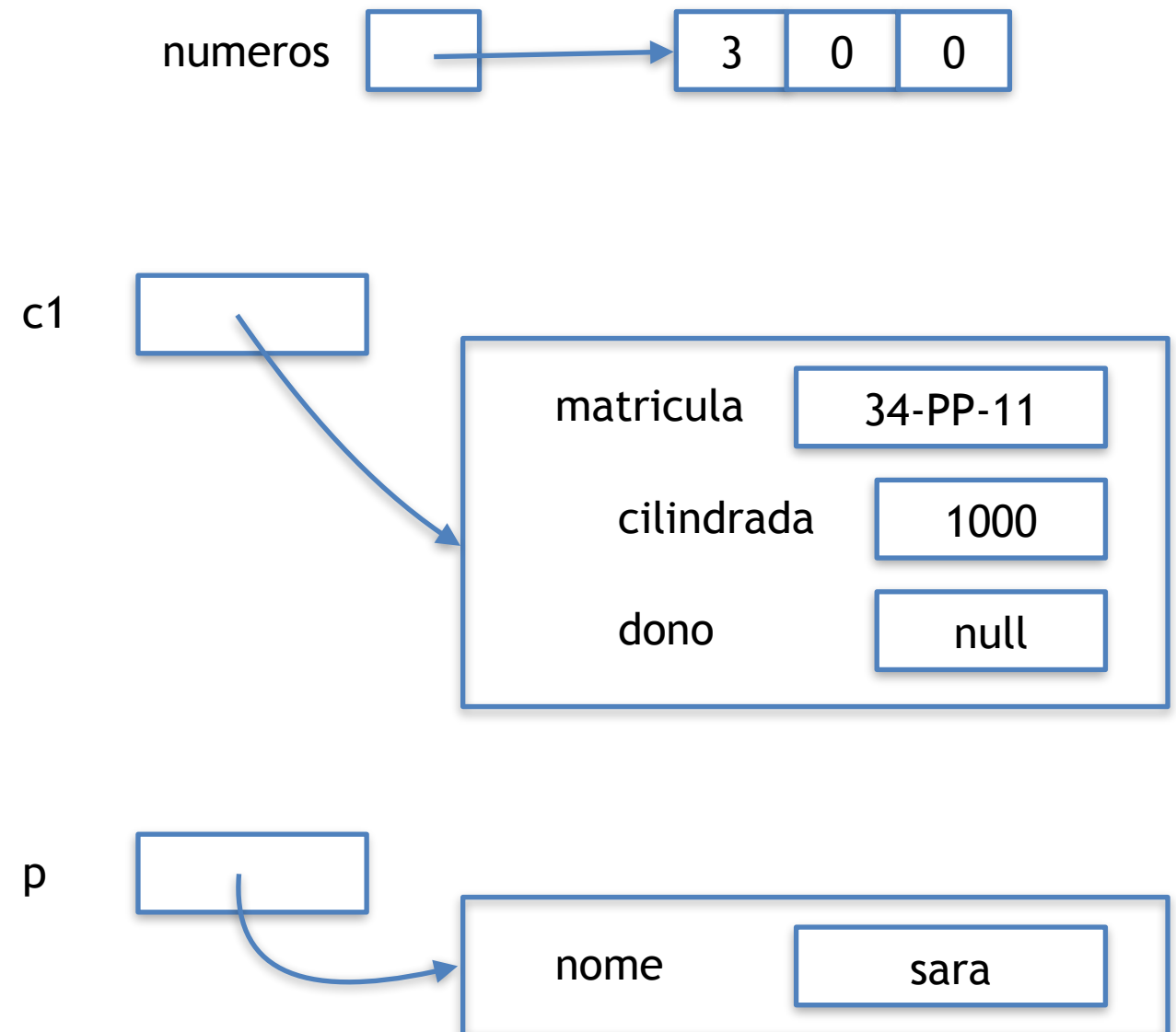


Diagrama de memória (2)

```
class Carro {
    String matricula;
    int cilindrada;
    Pessoa dono;

    public Carro(String matricula, int cilindrada) {
        ...
    }
}

class Pessoa {
    String nome;
}

public class Aplicacao {

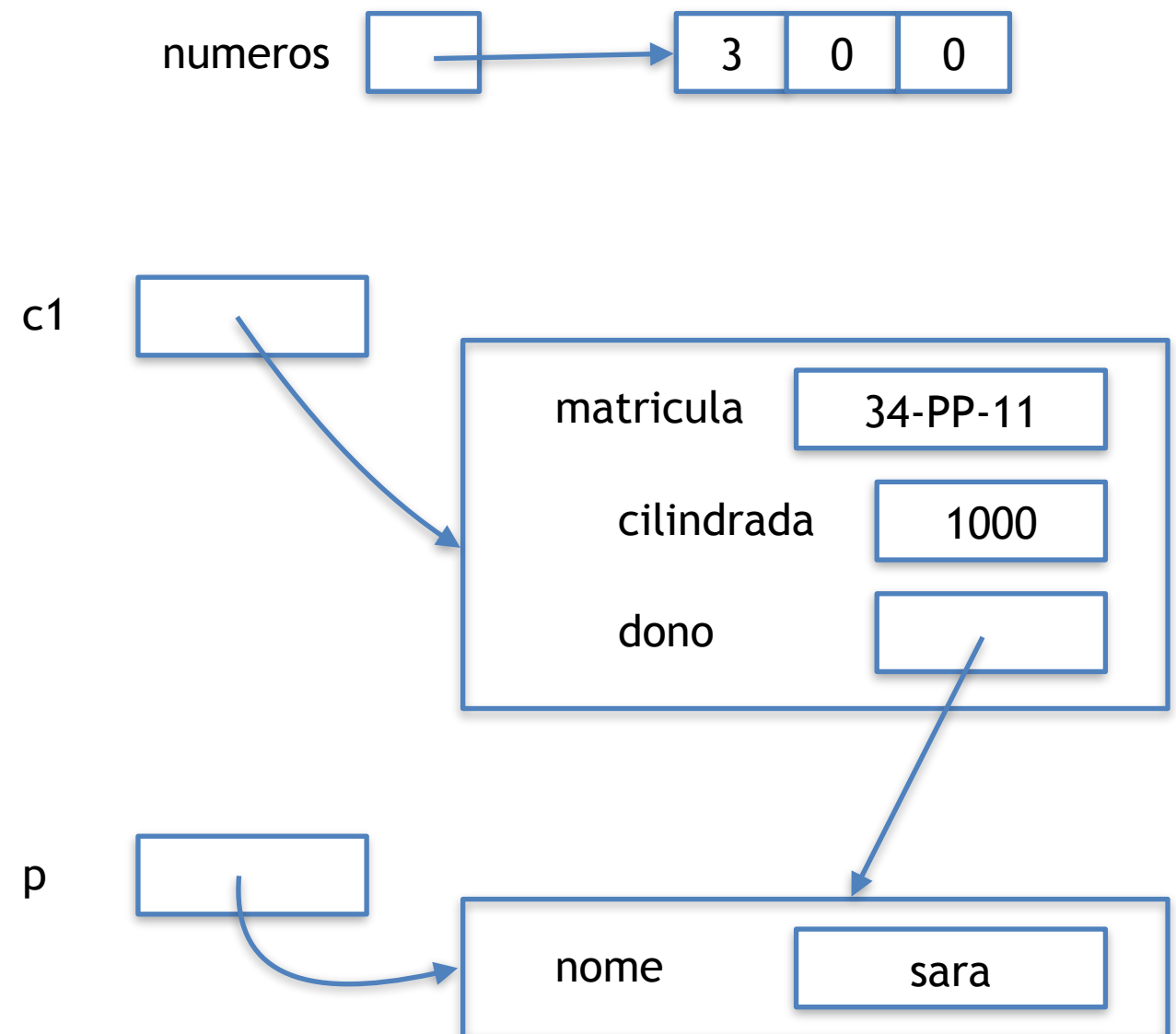
    public static void main(String[] args) {

        int[] numeros = new int[3];
        numeros[0] = 3;

        Carro c1 = new Carro("34-PP-11", 1000);

        Pessoa p = new Pessoa("sara");
        c1.dono = p;

    }
}
```



Exercício

```
class Animal {
    String nome;
    boolean vacinado;
}

class Pessoa {
    String nome;
    Animal[] animais = new Animal[2];
}
(...)
public static void main(String args[]) {

    Pessoa[] pessoas = new Pessoa[4];

    Pessoa p1 = new Pessoa("Ana");
    Animal gato1 = new Animal("Soneca", true);
    p1.animais[0] = new Animal("Pirata", false);

    Pessoa p2 = new Pessoa("Bruno");
    p2.animais[0] = gato1;
    p2.animais[1] = new Animal("Papagaio", false);
    p1.nome = "Soraia";

    pessoas[0] = p2;
    pessoas[1] = p1;
    pessoas[3] = pessoas[0];

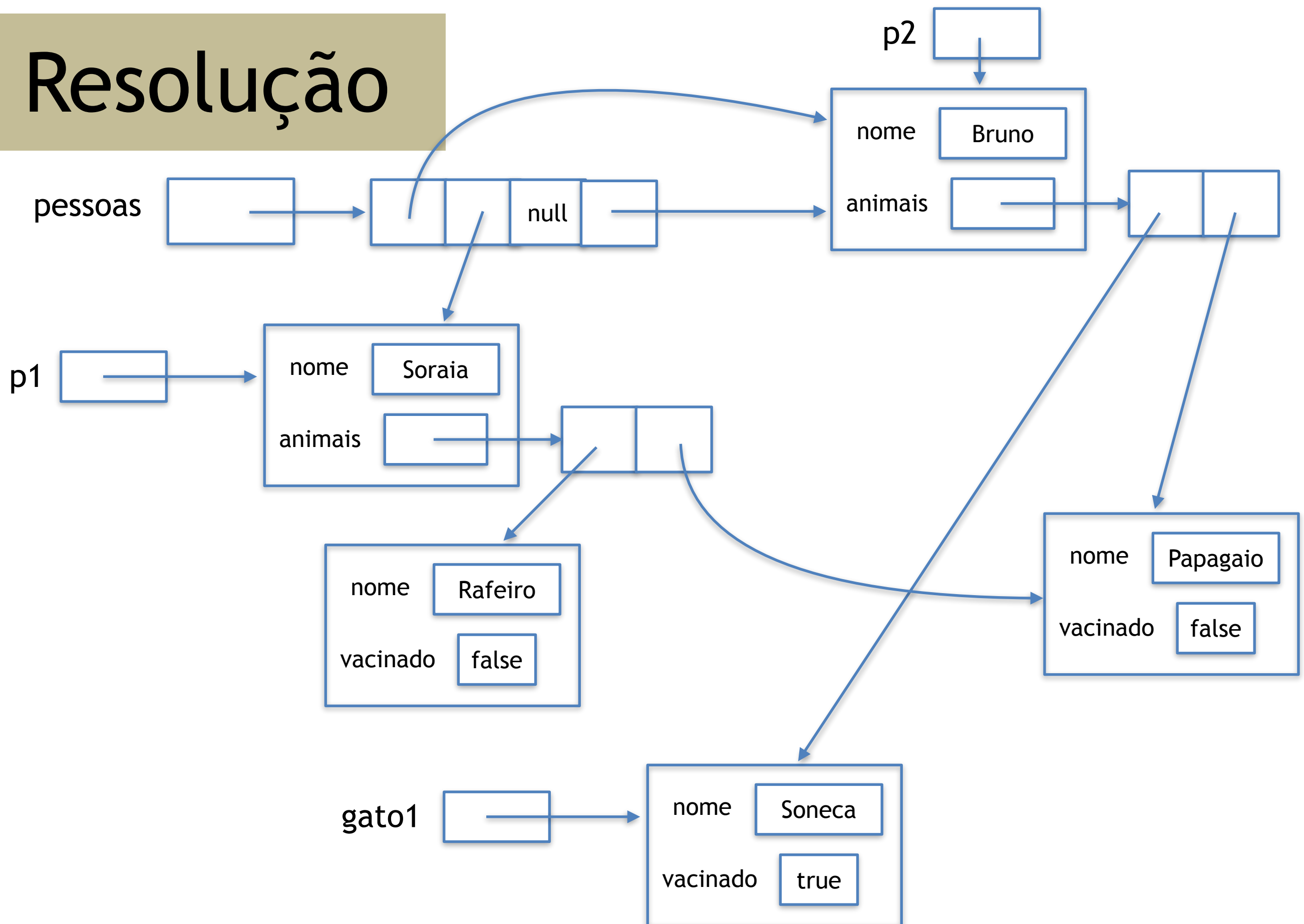
    p1.animais[1] = pessoas[0].animais[1];
    pessoas[1].animais[0].nome = "Rafeiro";
}
```

Nota: Construtores e classe principal omitidos por simplificação

Desenha a estrutura de memória no final da execução deste programa

Enviar via teams para
p4997@ulht.pt

Resolução



Scope das variáveis

(revisões)

Indica qual o espaço do programa dentro do qual as variáveis existem

```
{  
    int numero = 0;  
    ...  
}  
  
{  
    int numero2 = 0;  
    ...  
}
```

A variável numero só existe dentro destas chavetas

A variável numero2 só existe dentro destas chavetas

Scope das variáveis

(revisões)

O scope das variáveis está delimitado pelas chavetas nas quais foram declaradas. São visíveis dentro dessas chavetas e das respectivas subchavetas

```
1 {  
2   int numero;  
3  
4   {  
5       int numero2 = 0;  
6       ...  
7   }  
8 }
```

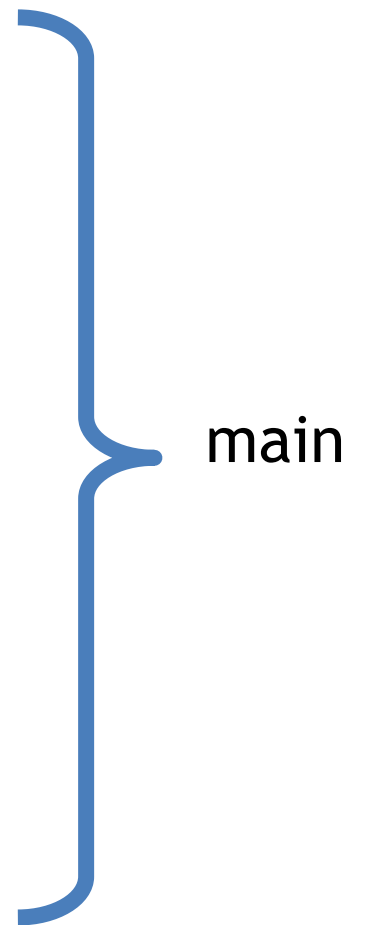
A variável numero é vista entre as linhas 3 e 8 (incluindo as linhas 5 e 6)

A variável numero2 só é vista na linha 6

Estruturar um programa

Para pequenos programas, pode-se colocar todo o código no função *main*

```
public class Aplicacao {  
    public static void main(String args[]) {  
  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        if (numero >= 0) {  
            System.out.println("Positivo");  
        } else {  
            System.out.println("Negativo");  
        }  
    }  
}
```

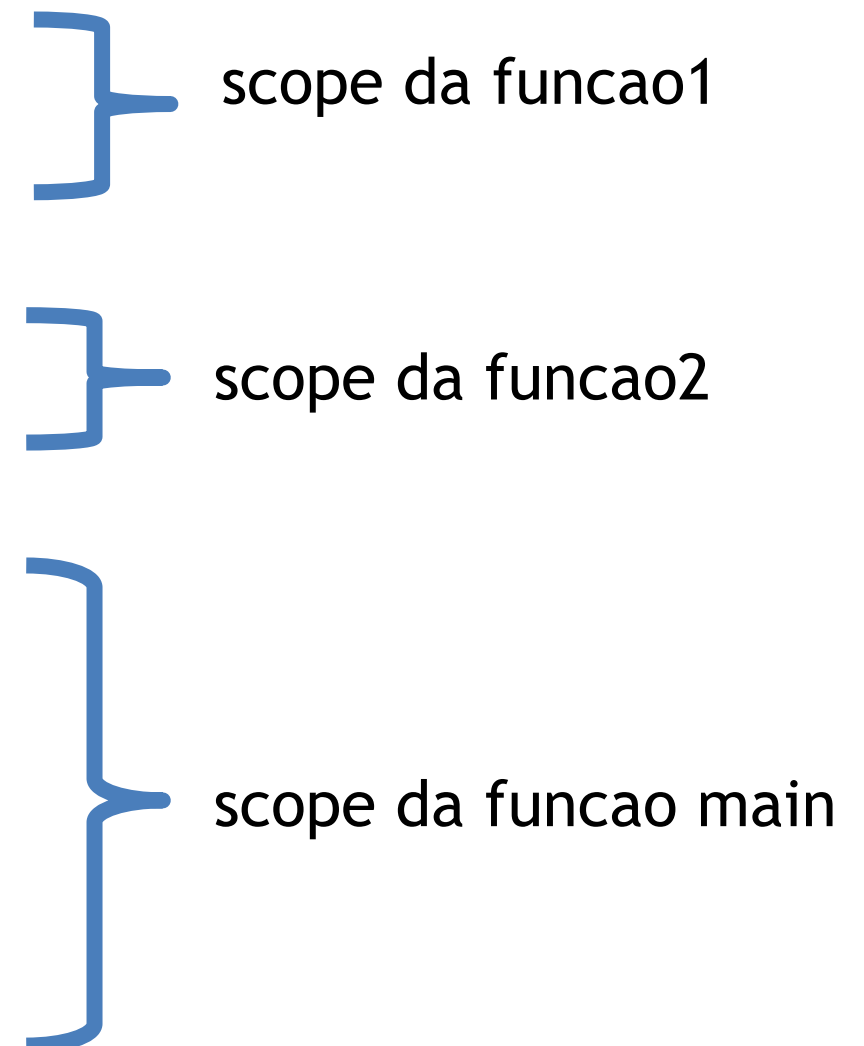


main

Estruturar um programa

Há medida que os programas crescem de complexidade, devemos usar funções para dividir o código

```
public class Aplicacao {  
  
    static int funcao1(int numero) {  
        int resultado;  
        ...  
    }  
  
    static String funcao2(int numero) {  
        ...  
    }  
  
    public static void main(String args[]) {  
  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        int resultado = funcao1(numero);  
        System.out.println(funcao2(resultado));  
    }  
}
```



Scope das variáveis

```
public class Aplicacao {  
  
    static int funcao1(int numero) {  
        int resultado;  
        ...  
    }  
  
    static String funcao2(int numero) {  
        ...  
    }  
  
    public static void main(String args[]) {  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        int resultado = funcao1(numero);  
        System.out.println(funcao2(resultado));  
    }  
}
```

} scope da funcao1

} scope da funcao2

} scope da funcao main

Scope das variáveis

```
public class Aplicacao {  
    static int funcao1(int numero) {  
        int resultado;  
        ...  
    }  
    static String funcao2(int numero) {  
        ...  
    }  
    public static void main(String args[]) {  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        int resultado = funcao1(numero);  
        System.out.println(funcao2(resultado));  
    }  
}
```

Estas duas variáveis são independentes. Cada uma “vive” no seu scope.


Scope das variáveis

```
public class Aplicacao {  
  
    static int funcao1(int numero) {  
        int resultado;  
        ...  
    }  
  
    static String funcao2(int numero) {  
        resultado += 3;  
    }  
  
    public static void main(String args[]) {  
  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        int resultado = funcao1(numero);  
        System.out.println(funcao2(resultado));  
    }  
}
```

Erro: Esta função não consegue aceder à variável resultado (nem da funcao1 nem do main)

Scope das variáveis

```
public class Aplicacao {  
    static int resultado;  
  
    static int funcao1(int numero) {  
        ...  
    }  
  
    static String funcao2(int numero) {  
        ...  
    }  
  
    public static void main(String args[]) {  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        ...  
        System.out.println(funcao2(resultado));  
    }  
}
```



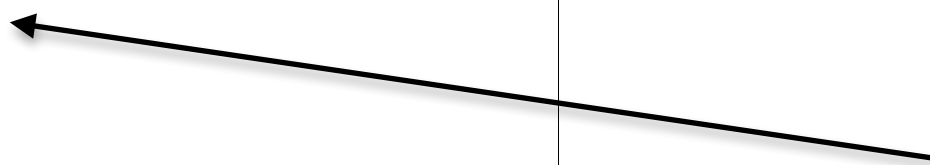
scope da classe Aplicacao

a variável resultado “vive”
neste scope

Scope das variáveis

```
public class Aplicacao {  
  
    static int resultado;  
  
    static int funcao1(int numero) {  
        resultado = numero;  
        ...  
    }  
  
    static String funcao2(int numero) {  
        resultado += 3;  
    }  
  
    public static void main(String args[]) {  
        Scanner teclado = new Scanner(System.in);  
        int numero = teclado.nextInt();  
  
        resultado = ...;  
        System.out.println(funcao2(resultado));  
    }  
}
```

Todas as funções da classe Aplicacao conseguem aceder à variável resultado



Exercício

```
public class Aplicacao {  
    static int numero;  
  
    static void funcao1(int valor) {  
        numero += valor;  
        valor++;  
    }  
  
    public static void main(String args[]) {  
        numero = 3;  
        int valor = 2;  
        funcao1(valor);  
        valor++;  
        funcao1(valor);  
        System.out.println("numero=" + numero);  
    }  
}
```

Qual o output deste programa?

Resolução

```
public class Aplicacao {  
    static int numero;  
  
    static void funcao1(int valor) {  
        numero += valor;  
        valor++;  
    }  
  
    public static void main(String args[]) {  
        numero = 3;  
        int valor = 2;  
        funcao1(valor);  
        valor++;  
        funcao1(valor);  
        System.out.println("numero=" + numero);  
    }  
}
```

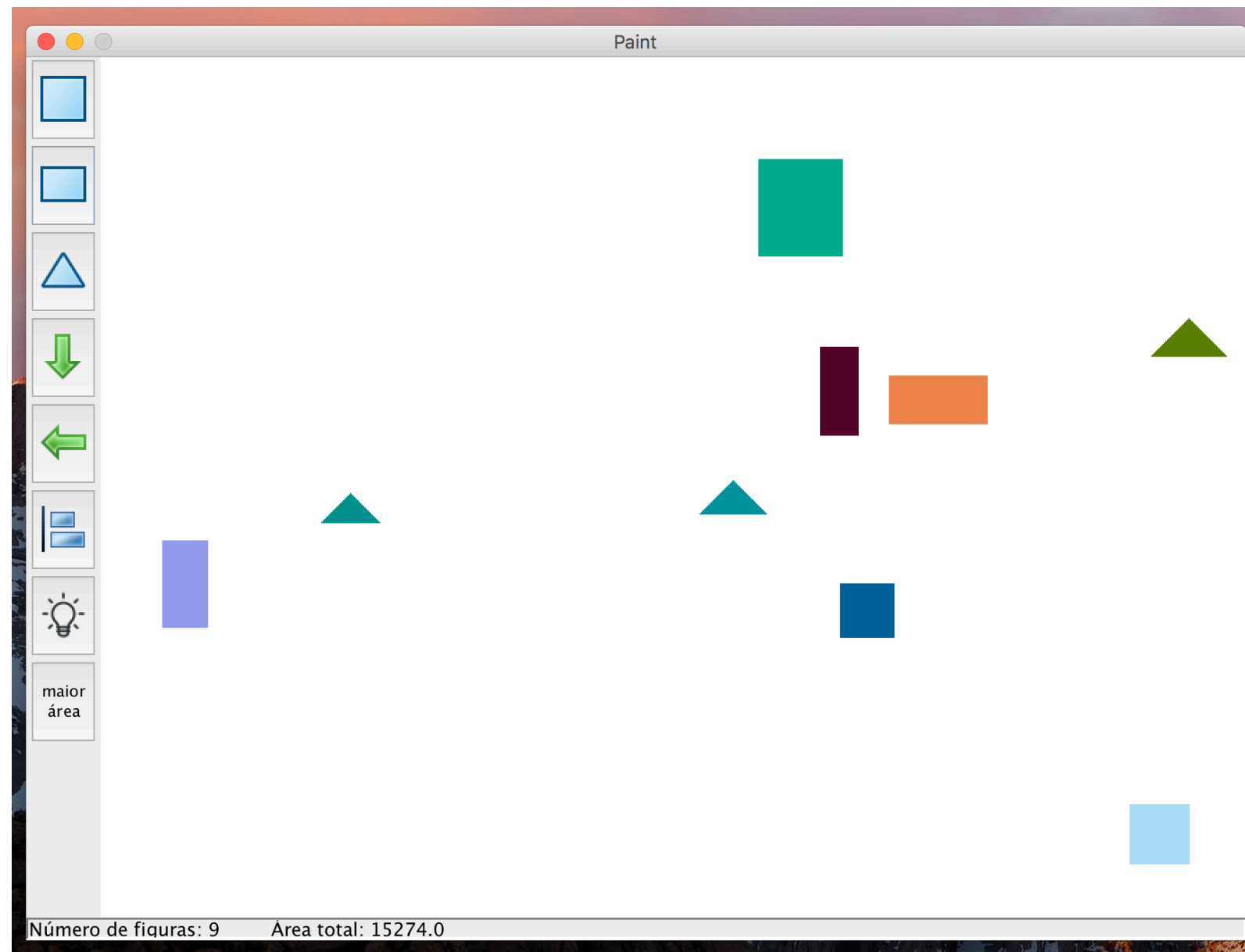
O programa escreve no écran:
numero=8

Quiz

Estará disponível um quiz no moodle, até dia 4 às 20h00

Podem submeter várias vezes, basta responderem uma certa já têm a nota máxima por isso tentem responder por vocês!

Exercício para pontos extra



Exercícios vão sendo publicados no Moodle (com vídeo)