Introdução a Orientação a Objetos em Java (I)

Grupo de Linguagens de Programação

Departamento de Informática
PUC-Rio

Paradigmas de Programação

- Programação Funcional
- Programação Procedural
- Programação Orientada por Objetos

Paradigma OO

Cenário Exemplo

"João deseja enviar flores para Maria mas ela mora em outra cidade. João vai, então, até a floricultura e pede a José, o floricultor, para que ele envie um bouquet de rosas ao endereço de Maria. José, por sua vez, liga para uma outra floricultura, da cidade de Maria, e pede para que as flores sejam entregues."

Nomenclatura

- João precisa resolver um problema;
- Então, ele procura um *agente*, José, e lhe passa uma *mensagem* contendo sua *requisição*: enviar rosas para Maria;
- José tem a *responsabilidade* de, através de algum *método*, cumprir a *requisição*;
- O *método* utilizado por José pode estar *oculto* de João.

Modelo OO

- Uma ação se inicia através do envio de uma *mensagem* para um *agente* (um *objeto*) responsável por essa ação.
- A mensagem carrega uma *requisição*, além de toda a informação necessária (*argumentos*) para que a ação seja executada.
- Se o *agente receptor* da *mensagem* a aceita, ele tem a *responsabilidade* de executar um *método* para cumprir a *requisição*.

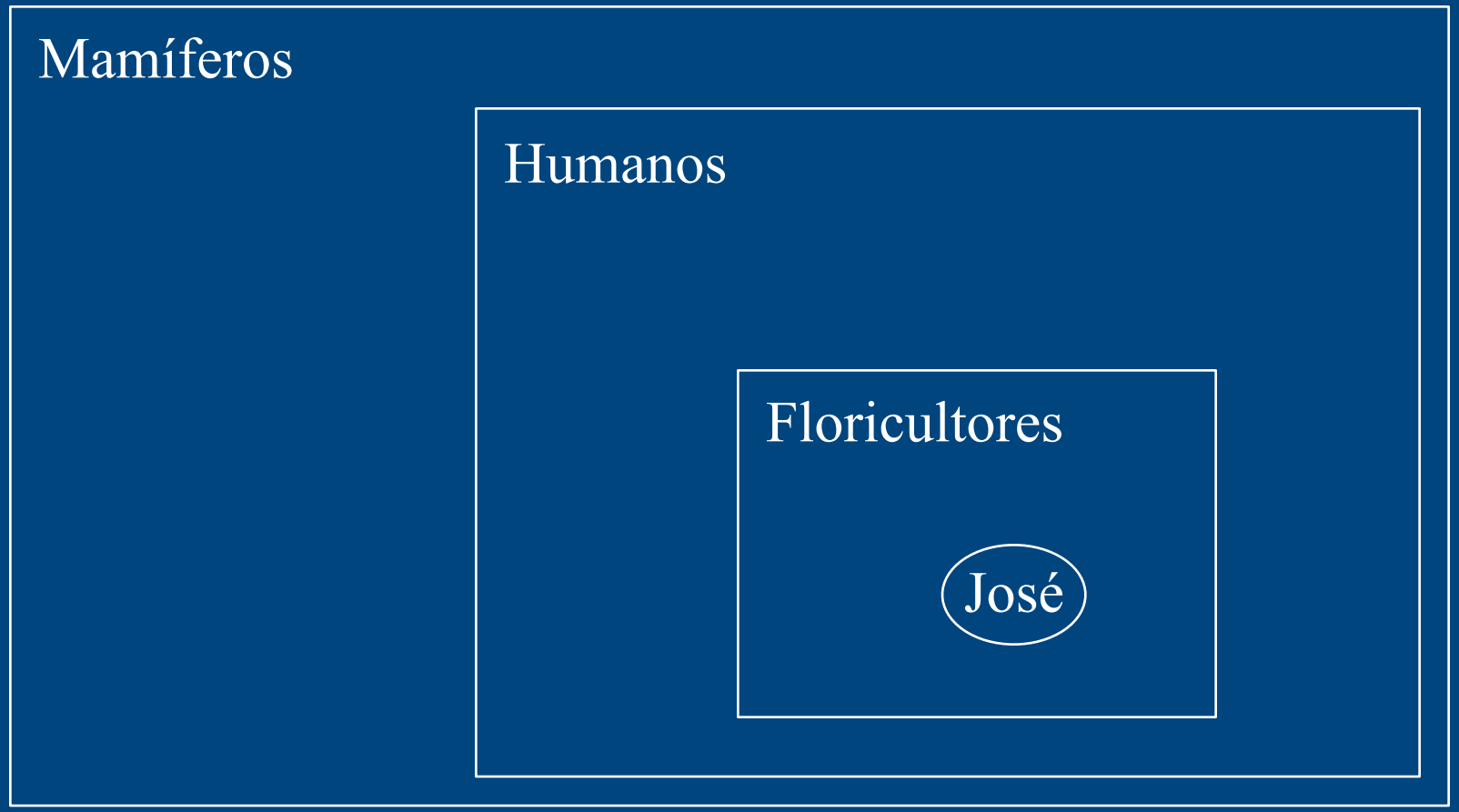
Objetos

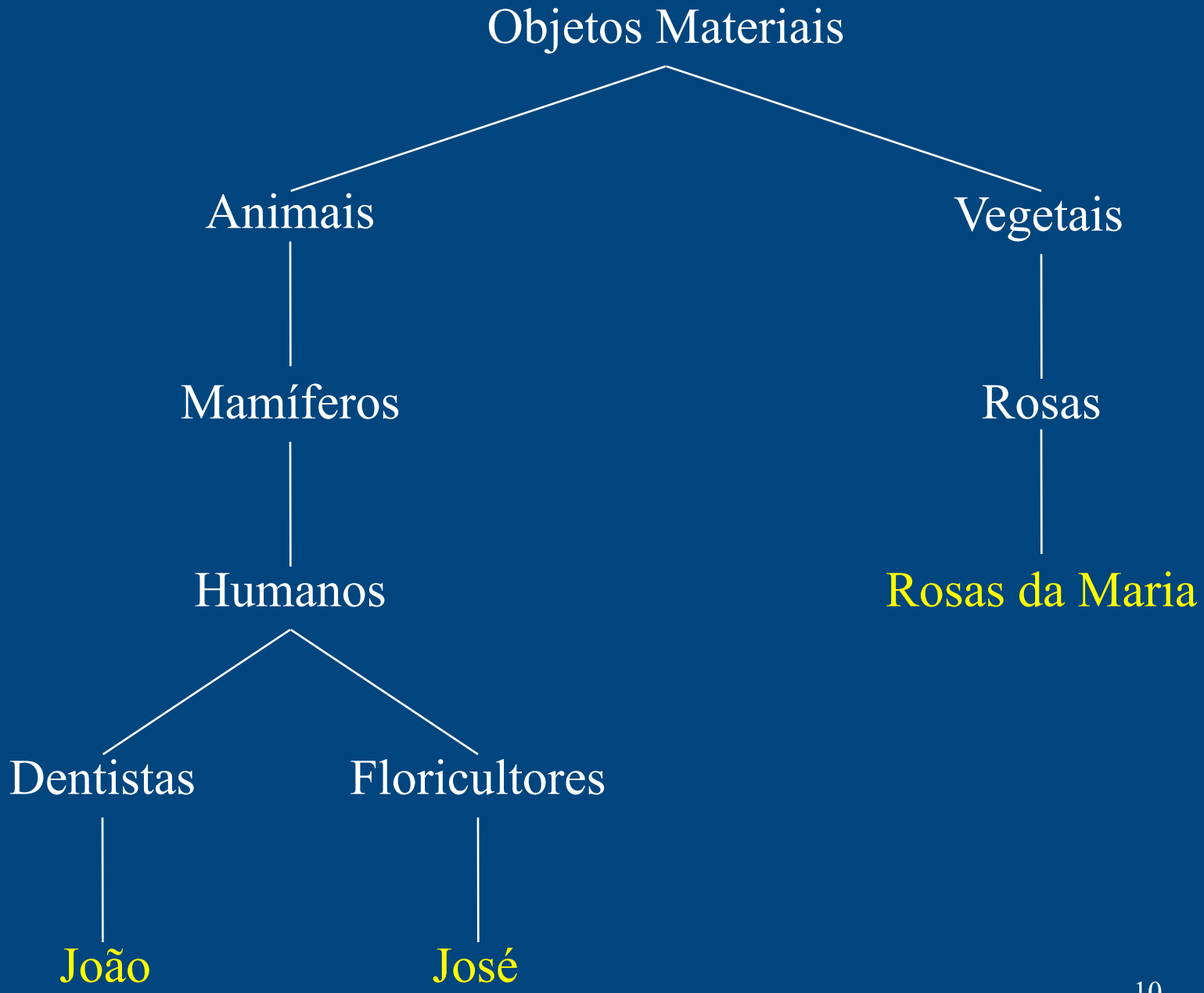
- Estão preparados para cumprir um determinado conjunto de requisições.
- Recebem essas requisições através de mensagens.
- Possuem a responsabilidade de executar um método que cumpra a requisição.
- Possuem um *estado* representado por informações internas.

Classes

- O conjunto de requisições que um objeto pode cumprir é determinado pela sua classe.
- A classe também determina que método será executado para cumprir uma requisição.
- A classe especifica que informações um objeto armazena internamente.
- Objetos são *instâncias* de classes.
- Classes podem ser compostas em hierarquias, através de *herança*.

Hierarquia de Classes (Herança)




Objetos Materiais
Animais
Vegetais
Mamíferos
Rosas
Humanos
Rosas da Maria
Dentistas
Floricultores
João
José

Resumo

- Agentes são objetos;
- Ações (computações) são executadas através da troca de mensagens entre objetos;
- Todo objeto é uma instância de uma classe;
- Uma classe define uma interface e um comportamento;
- Classes podem estender outras classes através de herança.

Tipos Abstratos de Dados

TAD

- Modela uma estrutura de dados através de sua funcionalidade.
- Define a interface de acesso à estrutura.
- Não faz qualquer consideração com relação à implementação.

Exemplo de TAD: Pilha

- Funcionalidade: armazenagem LIFO
- Interface:

```
boolean isEmpty()
```

verifica se a pilha está vazia

```
push(int n)
```

empilha o número fornecido

```
int pop()
```

desempilha o número do topo e o retorna

```
int top()
```

retorna o número do topo

TAD × Classes

- Uma determinada implementação de um TAD pode ser realizada por meio de uma classe.
- A classe deve prover todos os métodos definidos na interface do TAD.
- Um objeto dessa classe implementa uma instância do TAD.

Classes & Objetos

Classes em Java

- Em Java, a declaração de novas classes é feita através da construção **class**.
- Podemos criar uma classe **Point** para representar um ponto (omitindo sua implementação) da seguinte forma:

```
class Point {
   ...
}
```

Campos

- Como dito, classes definem dados que suas instâncias conterão.
- A classe **Point** precisa armazenar as coordenadas do ponto sendo representado de alguma forma.

```
class Point {
  int x, y;
}
```

Instanciação

- Uma vez definida uma classe, uma nova instância (objeto) pode ser criada através do comando **new**.
- Podemos criar uma instância da classe **Point** da seguinte forma:

```
Point p = new Point();
```

Uso de Campos

- Os campos de uma instância de **Point** podem ser manipulados diretamente.

```
Point p1 = new Point();
p1.x = 1;
p1.y = 2;
// p1 representa o ponto (1,2)
Point p2 = new Point();
p2.x = 0;
p2.y = 0;
// e p2 o ponto (0,0)
```

Referências para Objetos

- Em Java, nós sempre fazemos referência ao objeto. Dessa forma, duas variáveis podem se referenciar ao mesmo ponto.

```
Point p1 = new Point();
Point p2 = p1;
p2.x = 2;
p2.y = 3;
// p1 e p2 representam o ponto (2,3)
```

Métodos

- Além de atributos, uma classe deve definir os métodos que irá disponibilizar, isto é, a sua interface.
- A classe **Point** pode, por exemplo, prover um método para mover o ponto de um dado deslocamento.

Declaração de Método

- Para mover um ponto, precisamos saber quanto deslocar em x e em y. Esse método não tem um valor de retorno pois seu efeito é mudar o *estado* do objeto.

```
class Point {
  int x, y;
  void move(int dx, int dy) {
    x += dx;
    y += dy;
  }
}
```

Envio de Mensagens: Chamadas de Método

- Em Java, o envio de uma mensagem é feito através de uma chamada de método com passagem de parâmetros.
- Por exemplo, a mensagem que dispara a ação de deslocar um ponto é a chamada de seu método **move**.

```
p1.move(2,2);
// agora p1 está deslocado de duas unidades,
// no sentido positivo, nos dois eixos.
```

this

- Dentro de um método, o objeto pode precisar de sua própria referência.
- Em Java, a palavra reservada **this** significa essa referência ao próprio objeto.

```
class Point {
  int x, y;
  void move(int dx, int dy) {
    this.x += dx;
    this.y += dy;
  }
}
```

Inicializações

- Em várias circunstâncias, é interessante inicializar um objeto.
- Por exemplo, poderíamos querer que todo ponto recém criado estivesse em (0,0).
- Esse tipo de inicialização se resume a determinar valores iniciais para os campos.

Inicialização de Campos

- Por exemplo, a classe **Point** poderia declarar:

```
class Point {
  int x = 0;
  int y = 0;
  void move(int dx, int dy) {
    this.x += dx;
    this.y += dy;
  }
}
```

Construtores

- Ao invés de criar pontos sempre em (0,0), poderíamos querer especificar a posição do ponto no momento de sua criação.
- O uso de *construtores* permite isso.
- Construtores são mais genéricos do que simples atribuições de valores iniciais aos campos: podem receber parâmetros e fazer um processamento qualquer.

Declaração de Construtores

- O construtor citado para a classe **Point** pode ser definido da seguinte forma:

```
class Point {
  int x, y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
}
```

Usando Construtores

- Como o construtor é um método de inicialização do objeto, devemos utilizá-lo no momento da instanciação.

```
Point p1 = new Point(1,2); // p1 é o ponto (1,2)
Point p2 = new Point(0,0); // p2 é o ponto (0,0)
```

Construtor Padrão

- Quando não especificamos nenhum construtor, a linguagem Java declara, implicitamente, um construtor padrão, vazio, que não recebe parâmetros.
- Se declararmos algum construtor, esse construtor padrão *não* será mais declarado.

Finalizações

- Pode ser necessário executar alguma ação antes que um objeto deixe de existir.
- Para isso são utilizados os *destrutores*.
- Destrutores são métodos que são chamados automaticamente quando um objeto deixa de existir.
- Em Java, destrutores são chamados de *finalizadores*.

Finalizadores

```
class Test {
  ...
 protected void finalize () {
  System.out.println("Objeto coletado");
  }
}
```

- cuidados
  - pode nunca ser chamado
  - ou então ser chamado tarde demais
  - evitar a ressurreição de objetos
  - referências a objetos já coletados

Gerência de Memória

- Java possui uma gerência automática de memória, isto é, quando um objeto não é mais referenciado pelo programa, ele é automaticamente coletado (destruído).
- A esse processo chamamos “coleta de lixo”.
- Nem todas as linguagens OO fazem coleta de lixo e, nesse caso, o programador deve destruir o objeto explicitamente.

Finalizadores em Java

- Quando um objeto Java vai ser coletado, ele tem seu método **finalize** chamado.
- Esse método deve efetuar qualquer procedimento de finalização que seja necessário antes da coleta do objeto.

Membros de Classe

- Classes podem declarar membros (campos e métodos) que sejam comuns a todas as instâncias, ou seja, membros compartilhados por todos os objetos da classe.
- Tais membros são comumente chamados de ‘membros de classe’ (versus ‘de objetos’).
- Em Java, declaramos um membro de classe usando o qualificador **static**. Daí, o nome ‘membros estáticos’ usado em Java.

Membros de Classe: Motivação

- Considere uma classe que precise atribuir identificadores unívocos para cada objeto.
- Cada objeto, ao ser criado, recebe o seu identificador.
- O identificador pode ser um número gerado seqüencialmente, de tal forma que cada objeto guarde o seu mas o próximo número a ser usado deve ser armazenado na *classe*.

Membros de Classe: Um Exemplo

- Podemos criar uma classe que modele produtos que são produzidos em uma fábrica.
- Cada produto deve ter um código único de identificação.

Membros de Classe: Codificação do Exemplo

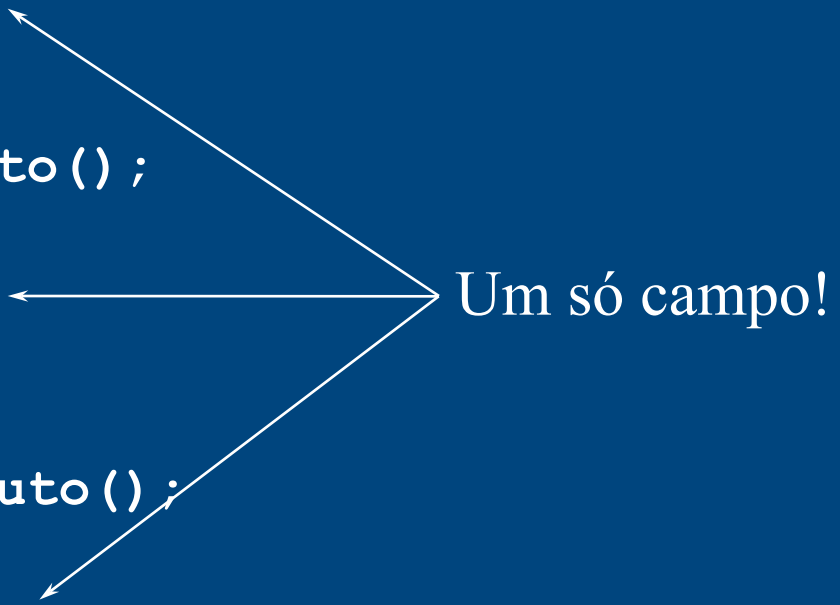
```
class Produto {
  static int próximo_id = 0;
  int id;
  Produto() {
    id = próximo_id;
    próximo_id++;
  }
  ...
}
```

Membros de Classe: Análise do Exemplo

```
// Considere que ainda não há nenhum produto.
// Produto.próximo_id = 0

Produto lápis = new Produto();
  // lápis.id = 0
  // lápis.próximo_id = 1

Produto caneta = new Produto();
  // caneta.id = 1
  // caneta.próximo_id = 2
```



Membros de Classe: Acesso Direto

- Como os membros estáticos são da classe, não precisamos de um objeto para acessá-los: podemos fazê-lo diretamente sobre a classe.

```
Produto.próximo_id = 200;
// O próximo produto criado terá id = 200.
```

Membros de Classe: Outras Considerações

- Java possui apenas declarações de classes: a única forma de escrevermos uma função é como um método em uma classe.

this revisitado

Nós vimos que um método estático pode ser chamado diretamente sobre a classe. Ou seja, não é necessário que haja uma instância para chamarmos um método estático. Dessa forma, não faz sentido que o **this** exista dentro de um método estático.

Noção de Programa

- Uma vez que tudo o que se escreve em Java são declarações de classes, o conceito de programa também está relacionado a classes: a execução de um programa é, na verdade, a execução de uma classe.
- Executar uma classe significa executar seu método estático **main**. Para ser executado, o método **main** deve possuir uma assinatura bem determinada.

Executando uma Classe

- Para que a classe possa ser executada, seu método **main** deve possuir a seguinte assinatura:

```
public static void main(String[] args)
```

Olá Mundo!

- Usando o método **main** e um atributo estático da classe que modela o sistema, podemos escrever nosso primeiro programa:

```
class Mundo {
  public static void main(String[] args) {
    System.out.println("Olá Mundo!");
  }
}
```

Idiossincrasias de Java

- Uma classe deve ser declarada em um arquivo homônimo (case-sensitive) com extensão **.java**.


Tipos
Expressões
Comandos

Tipos Básicos de Java

- **boolean** — **true** ou **false**
- **char** — caracter UNICODE (16 bits)
- **byte** — número inteiro com sinal (8 bits)
- **short** — número inteiro com sinal (16 bits)
- **int** — número inteiro com sinal (32 bits)
- **long** — número inteiro com sinal (64 bits)
- **float** — número em ponto-flutuante (32 bits)
- **double** — número em ponto-flutuante (64 bits)

Classes Pré-definidas

- Textos
- Vetores

```
String texto = "Exemplo";
int[] lista = {1, 2, 3, 4, 5};
int[] v = new int[10];
String[] nomes = {"João", "Maria"};
int i = nomes.length;

System.out.println(nomes[0]); // Imprime "João".
```

Operadores

```
[]   .   (params)   exp++   exp--
++exp   --exp   +exp   -exp   ~exp   !exp
new   (tipo) exp
*   /   %
+   -
<<   >>   >>>
<   >   <=   >=   instanceof
==   !=
&
^
|
&&
||
?:
=   +=   -=   *=   /=   %=   <<=   >>=   >>>=   &=   ^=   |=
```

Expressões

- Tipos de expressões
  - conversões automáticas
  - conversões explícitas

```
byte b = 10;
float f = 0.0F;
```

Comandos

- Comando
  - expressão de atribuição
  - formas pré-fixadas ou pós-fixadas de ++ e --
  - chamada de métodos
  - criação de objetos
  - comandos de controle de fluxo
  - bloco
- Bloco = { <lista de comandos> }

Controle de Fluxo

- **if-else**
- **switch-case-default**
- **while**
- **do-while**
- **for**
- **break**
- **return**

if-else

```
if (a>0 && b>0)
  m = média(a, b);
else
{
  errno = -1;
  m = 0;
}
```

switch-case-default

```
int i = f();
switch (i)
{
  case -1:
    ...
    break;
  case 0:
    ...
    break;
  default:
    ...
}
```

while

```
int i = 0;
while (i<10)
{
  i++;
  System.out.println(i);
}
```

do-while

```
int i = 0;
do
{
  i++;
  System.out.println(i);
}
while (i<10);
```

for

```
static double average (double[] values) {
  double sum = 0.0;
  for (int j = 0; j < values.length; j++)
     sum += values[j];
  return sum/values.length;
}
// o que fazer se values.length for igual a zero?
```

enhanced for

```
static double average (double[] values) {
  double sum = 0.0;
  for (double val : values)
     sum += val;
  return sum/values.length;
}
// o que fazer se values.length for igual a zero?
```

break

```
int i = 0;
while (true)
{
  if (i==10) break;
  i++;
  System.out.println(i);
}
```

label

```
início:
for (int i=0; i<10; i++)
  for (int j=0; j<10; j++)
  {
    if (v[i][j] < 0) break início;
    ...
  }
...
```

return

```
int média(int a, int b)
{
  return (a+b)/2;
}
```