INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Estrutura de Dados

Aula 2

GUSTAVO FORTUNATO PUGA

Estrutura de Dados

Uma estrutura de dados **mantém os dados organizados seguindo alguma lógica** e disponibiliza operações para o usuário manipular os dados.

Instruções condicionais

Nome formal	Palavras-chave	Tipos de expressão	Exemplo
`if`	`if`, `else` (opcional)	`boolean`	`if (value == 0) {}`

`if-then`	`if`,
	`else if`,
	`else if` (opcional)
		`boolean`	`if (value == 0) {}`
			`else if (value == 1) {}`
			`else if (value >= 2) {}`

`if-then-else`	`if`,
	`else if` (opcional),
	`else if` (opcional),
	`else`
		`boolean`	`if (value == 0) {}`
			`else if (value >= 1) {}`
			`else {}`

Operador ternário	`?`, `:` (não é uma palavra-chave Java oficial)	`boolean`	`minVal = a < b ? a : b;`

`switch`	`switch`,
	`case`,
	`default` (opcional),
	`break` (opcional)
		`char`, `byte`, `short`, `int`, `Character`, `Byte`, `Short`, `String`, `Integer`, tipos enumerados	`switch (100) {`
			`case 100: break;`
			`case 200: break;`
			`case 300: break;`
			`default: break;`
			`}`

Instruções condicionais

Você deseja usar um operador AND que avalie o segundo operando não importando se o primeiro operando for igual a `true` ou `false`. Qual usaria?	AND bitwise (`&`)
Você deseja usar um operador OR que avalie o segundo operando não importando se o primeiro operando for igual a `true` ou `false`. Qual usaria?	OR bitwise (`|`)
Você deseja usar um operador AND que avalie o segundo operando somente quando o primeiro operando for igual a `true`. Qual usaria?	AND lógico (`&&`)
Você deseja usar um operador OR que avalie o segundo operando somente quando o primeiro operando for igual a `false`. Qual usaria?	OR lógico (`||`)

Instruções de laço

Nome formal	Palavras-chave	Componentes principais da expressão	Exemplo
Laço `for`	`for`, `break` (opcional), `continue` (opcional)	Inicializador, expressão, mecanismo de atualização	`for (i=0; i<j; i++) {}`
Laço `for` melhorado	`for`, `break` (opcional), `continue` (opcional)	Elemento, array ou coleção	`for (Fish f : listOfFish) {};`
`while`	`while` `break` (opcional), `continue` (opcional)	Expressão booleana	`while (value == 1) { }`
`do-while`	`do`, `while`, `break` (opcional), `continue` (opcional)	Expressão booleana	`do { } while (value == 1);`

Palavras chave

Palavras-chave Java				
break	continue	else	if	throw
case	default	finally	return	try
catch	do	for	switch	while

Tipos primitivos

SUBCONJUNTO ESPECIAL DOS TIPOS DE DADOS

- São a base do armazenamento de dados em um programa
- Há tipos oito primitivos
  - boolean (booleano)
  - char (caractere)
  - byte (byte)
  - short (inteiro curto)
  - int (inteiro)
  - long (inteiro longo)
  - float (ponto flutuante)
  - double (ponto flutuante de dupla precisão)

Tipos primitivos

- Formato de dados mais básico
- Quando declarado reserva um certo número de bits na memória
- Possibilita ler ou atribuir um valor a ela
- Os cálculos executados com primitivos são mais rápidos do que os executados com objetos semelhantes

Tipos primitivos

Tipo de dado	Usado para	Tamanho	Intervalo
`boolean`	`true` ou `false`	1 bit	N/A
`char`	caractere Unicode	16 bits	\u0000 a \uFFFF (0 a 65.535)
`byte`	inteiro	8 bits	–128 a 127
`short`	inteiro	16 bits	–32768 a 32767
`int`	inteiro	32 bits	–2.147.483.648 a 2.147.483.647
`long`	inteiro	64 bits	-2^{63} a $2^{63}-1$
`float`	ponto flutuante	32 bits	$1,4e^{-45}$ positivo a $3,4e^{+38}$
`double`	ponto flutuante	64 bits	$5e^{-324}$ positivo a $1,8e^{+308}$

Tipos primitivos

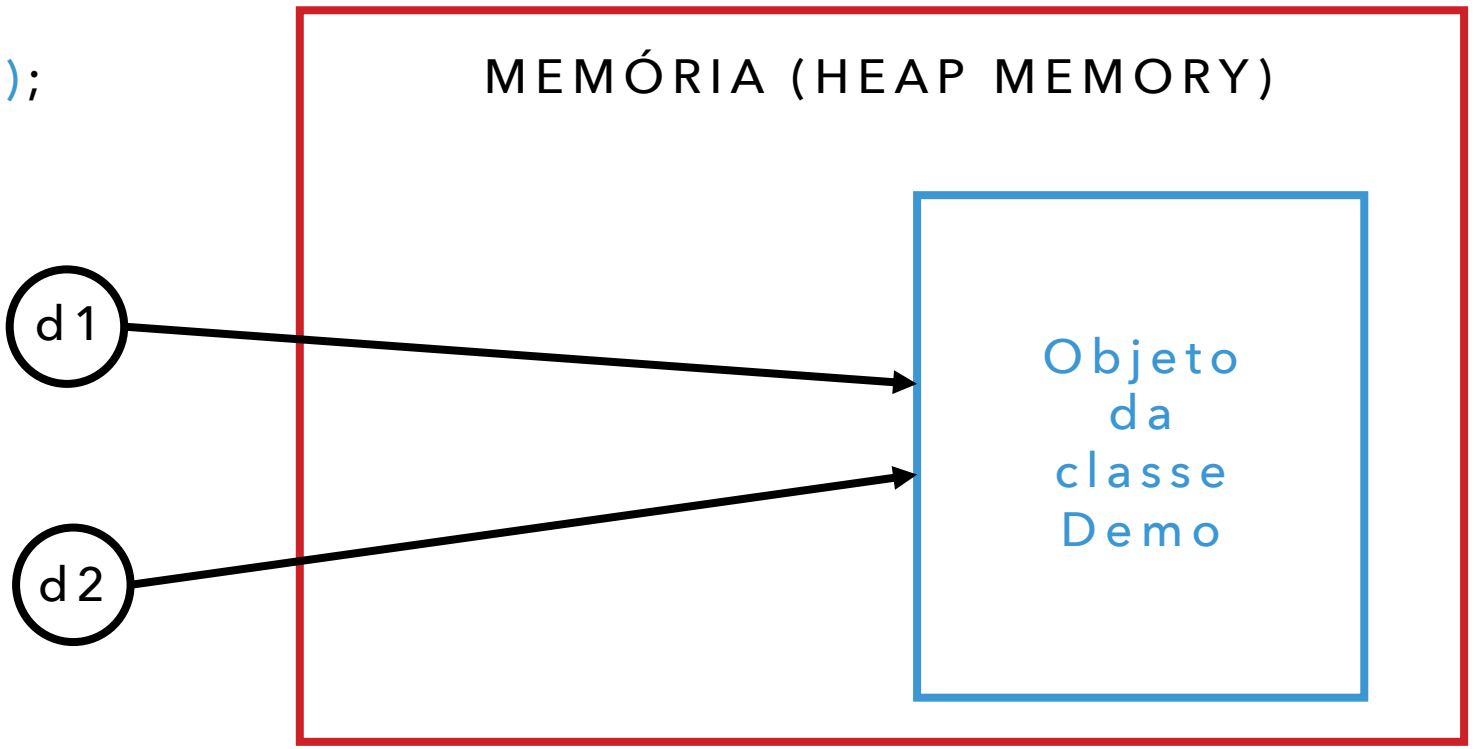
Ao ler código Java que outros provavelmente serão empregados os primitivos **int, double e boolean**. Esses três primitivos são a opção padrão pois **abrangem a maioria dos casos comuns**.

A otimização do tamanho dos primitivos é **relevante** ao usá-lo repetidamente em arrays ou em estruturas de dados que **permaneçam em memória** por longo período de tempo.

Variáveis de Referência

ARMAZENAM AS LOCALIZAÇÕES DE OBJETOS NA MEMÓRIA DO COMPUTADOR

```
Demo d1 = new Demo();

Demo d2 = d1;
```



Variáveis e Atributos

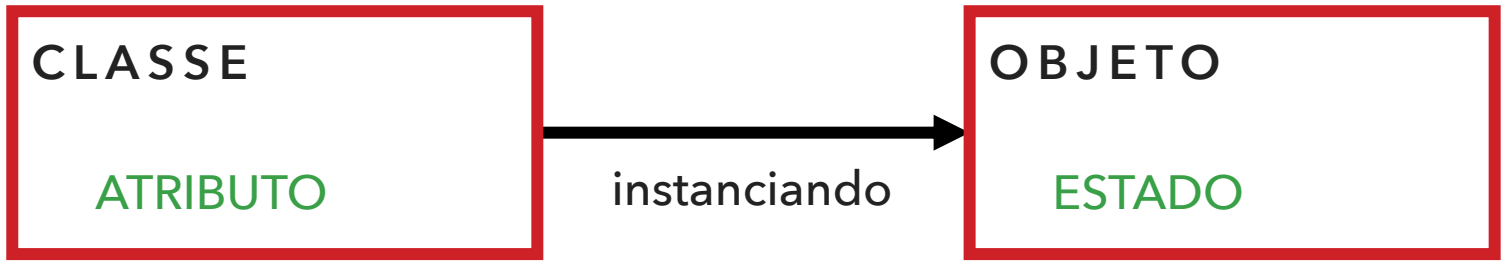
- Atributos e variáveis são a mesma coisa em questão de funcionalidade.
- Ambos são endereços de memória que tem um espaço ou tamanho definido de acordo com o tipo de dado que será guardado
    - Caracter, número inteiro, número decimal, dentre outros.

Em Orientação a objetos, utiliza-se o termo atributo, que consiste numa variável que está dentro de um determinado contexto, que é representada por uma classe. Como tudo que fazemos em Java está contido dentro de uma classe, então usamos o termo atributo ao invés de variável.

Variáveis e Atributos

Variáveis declaradas diretamente dentro do escopo da classe são denominadas como

variável de objeto ou atributo



Objetos

- Compostos por primitivos e outros objetos para armazenar seus dados e incluir o código relacionado.
- Os dados mantem o estado do objeto, enquanto o código é organizado em métodos que executam ações com esses dados.
- Uma classe bem projetada deve ser definida claramente e tem que poder ser reutilizada facilmente em diferentes aplicativos.

Objetos

A abordagem baseada em objetos preocupa-se **primeiro em identificar os objetos** contidos no domínio da aplicação e **depois estabelecer os procedimentos** relativos à eles.

Rumbaugh

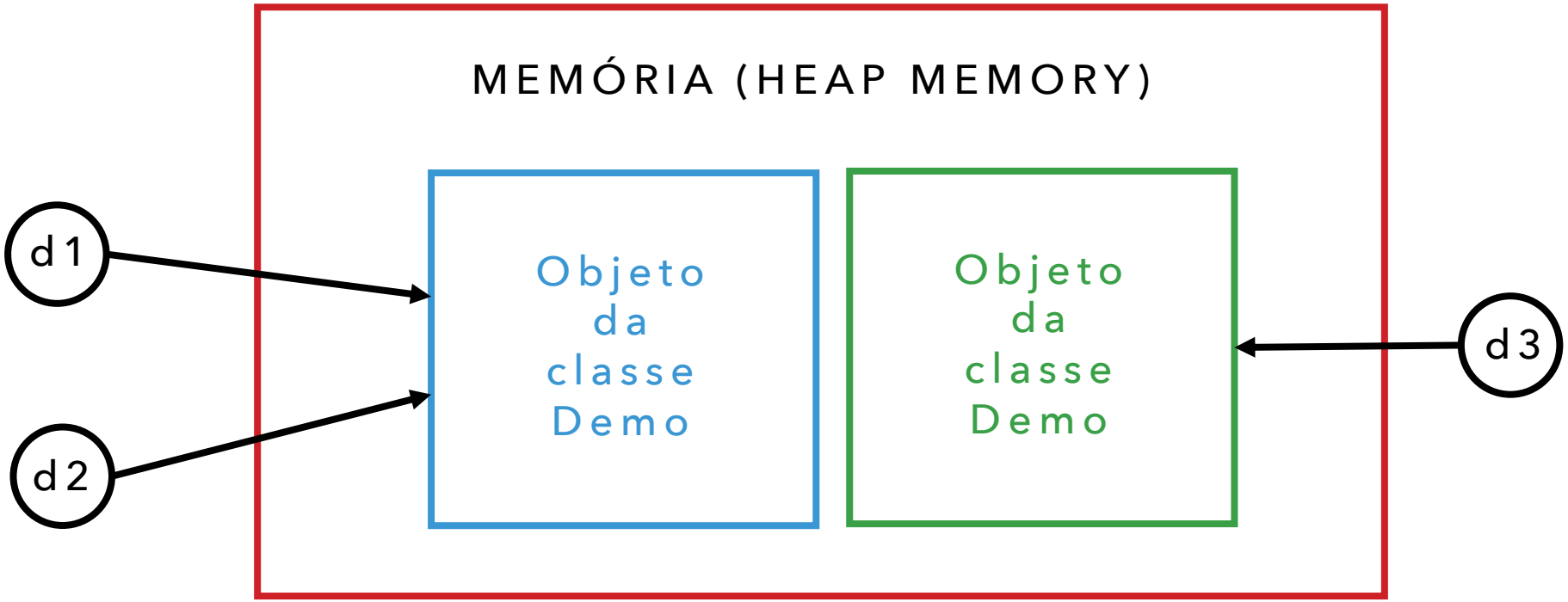
Objetos e Classes

Ao escrever código, é criado ou modificado uma classe. Uma classe é o arquivo contendo o código que foi escrito. É um item tangível. A classe é como um modelo que diz à JVM como criar um objeto em tempo de execução. O operador new solicita à JVM que use o modelo para criar uma nova instância dessa classe, cujo resultado é um objeto.

Muitos objetos podem ser construídos a partir de uma única classe.

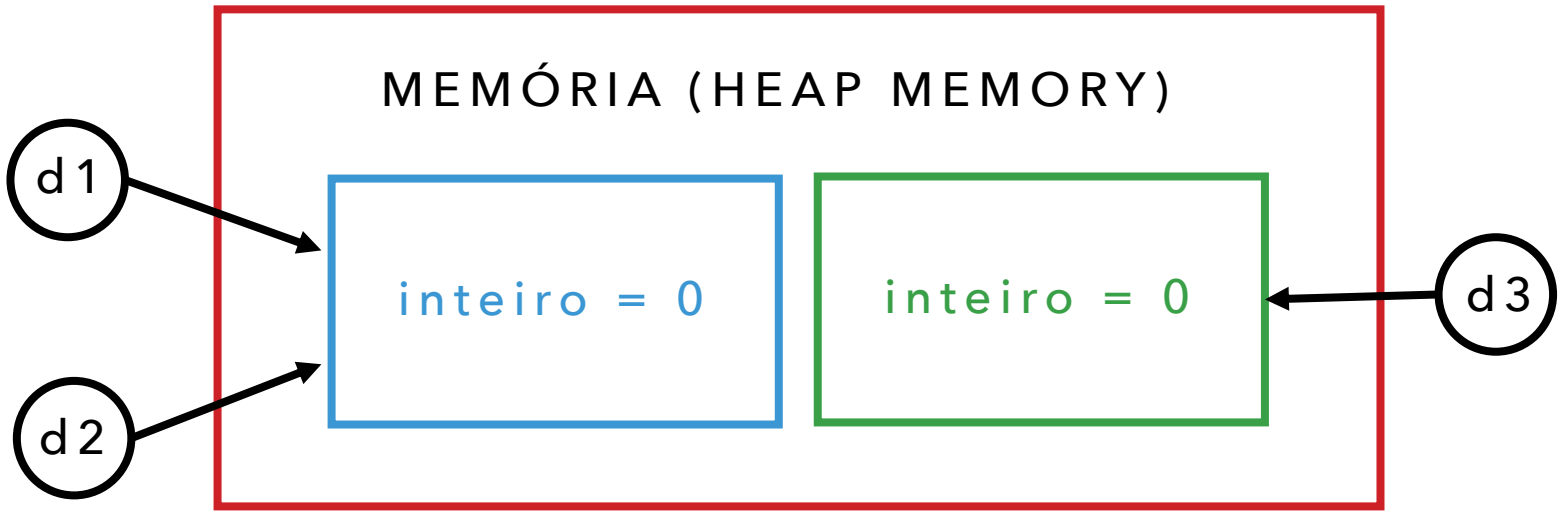
Objetos e Classes

```
Demo d1 = new Demo();
Demo d2 = d1;
Demo d3 = new Demo();
```



Construtor

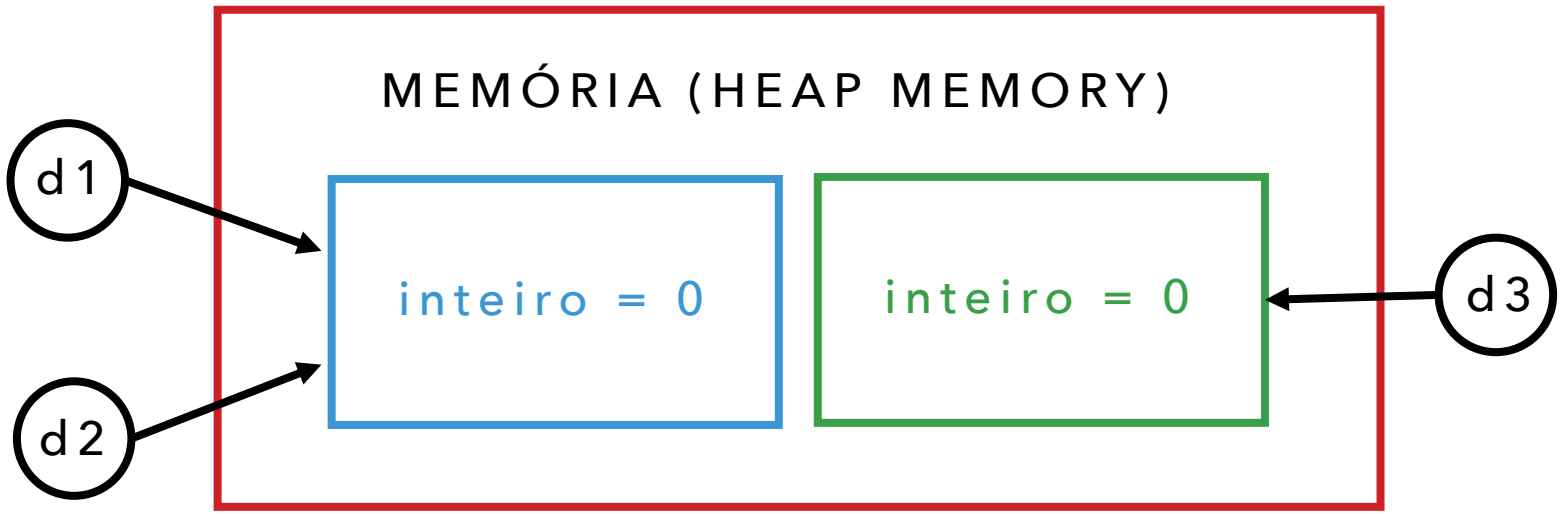
[pseudo-]método que determina que ações devem ser executadas para criar de um objeto. O construtor é definido como um método cujo nome deve ser o mesmo nome da classe e sem indicação do tipo de retorno -- nem mesmo void. O construtor é unicamente invocado no momento da criação do objeto através do operador new.



Objetos e Classes

Se nenhum construtor for explicitamente definido, um construtor padrão, que não recebe argumentos, é incluído para a classe pelo compilador Java. No entanto, se for criado pelo menos um método construtor, o construtor padrão não será criado automaticamente

```
class Demo {

    int inteiro;

     int incrementalInteiro(){
        return ++inteiro;
     }
}
```

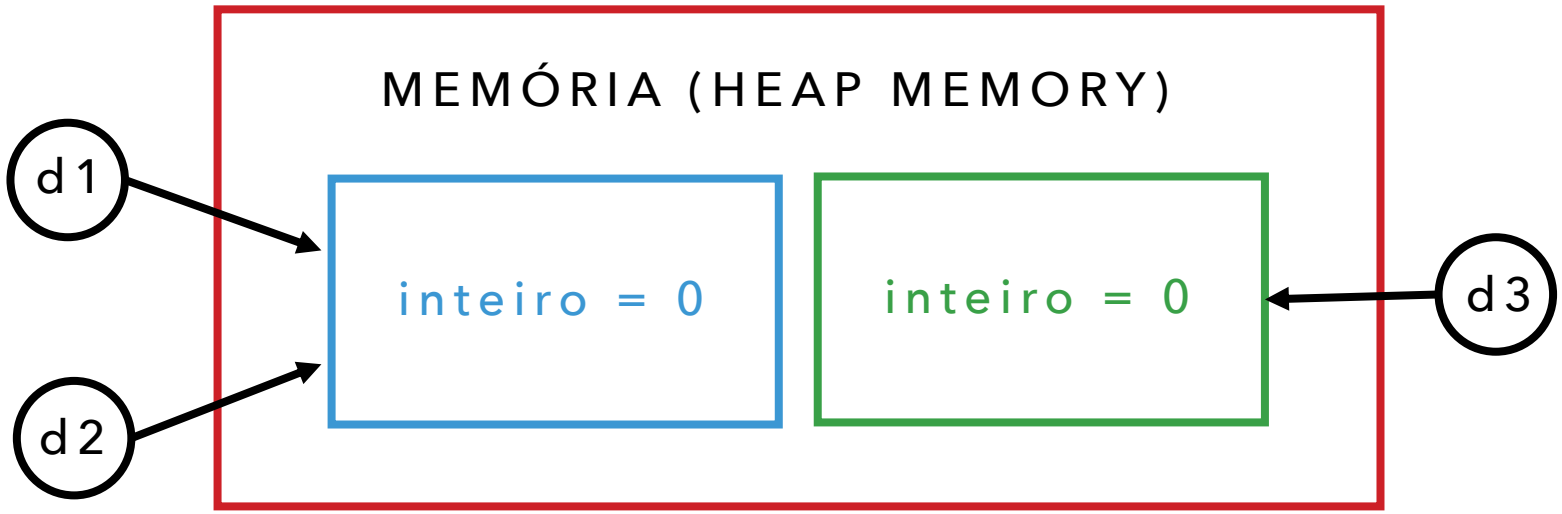


Objetos e Classes

- O espaço para o objeto é alocado e seu conteúdo é inicializado (bitwise) com zeros.
- O construtor da classe base é invocado. Se a classe não tem uma superclasse definida explicitamente, a classe Object é a classe base.
- Os membros da classe são inicializados para o objeto, seguindo a ordem em que foram declarados na classe.
- O restante do corpo do construtor é executado.

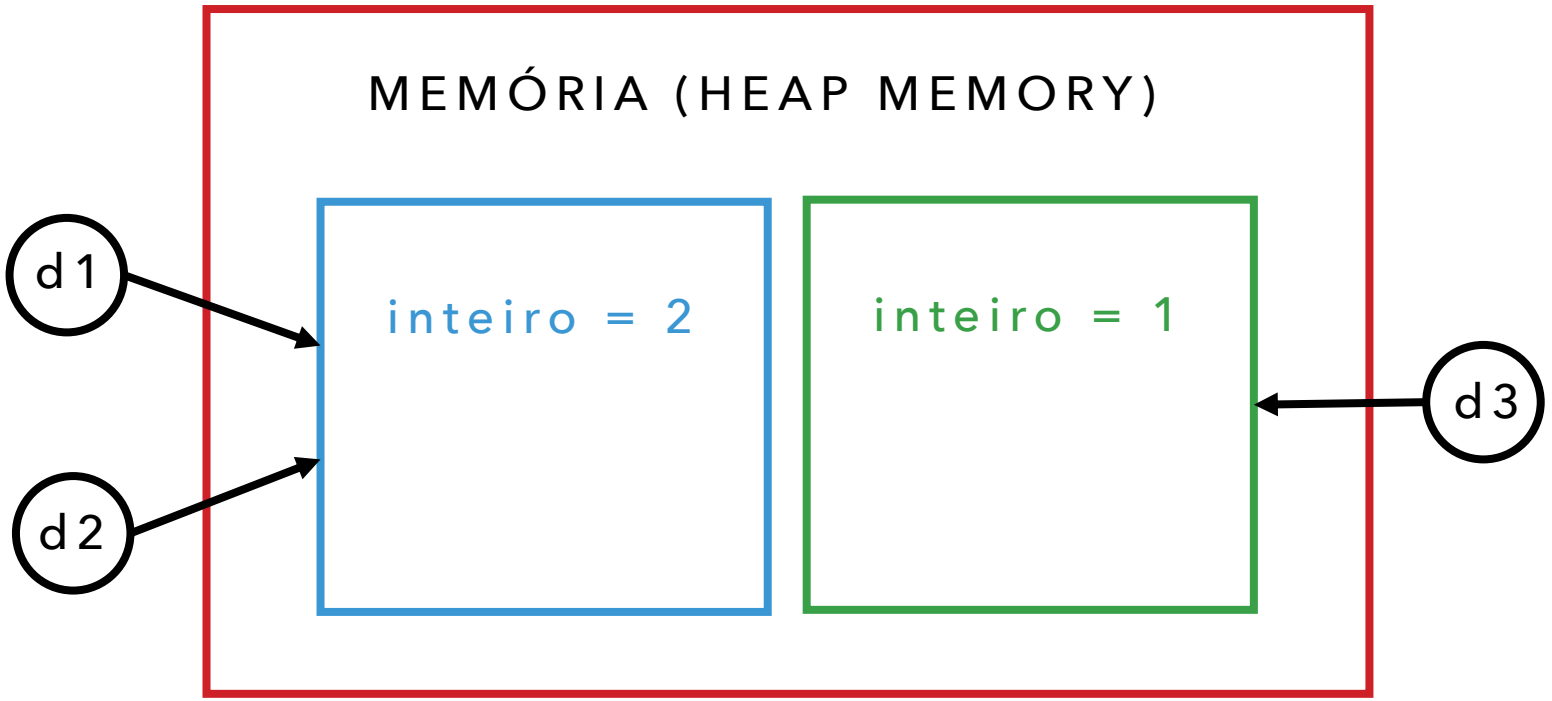
Valores padrão de atributos

Tipo da variável	Valor inicial
numérico	0
boolean	false
referência	null (nulo)



Objetos e Classes

```
Demo d1 = new Demo();
Demo d2 = d1;
Demo d3 = new Demo();

d1.incrementaInteiro();
d3.incrementaInteiro();
d2.incrementaInteiro();
```



String

Usada para representar cadeias de caracteres Unicode de 16 bits

Ao contrário que ocorre em C e C++, strings em Java não são tratadas como sequências de caracteres terminadas por NULL. São objetos ou instâncias da classe **java.lang.String**, portanto, devem ser declarados e instanciados.

String

```
String quote1; // quote1 é uma variável de referência sem objeto string atribuído

quote1 = "Ahoy matey"; // Atribui um objeto string à referência


String quote2a = new String(); // quote2a é uma variável de referência

String quote2b = new String(""); // Instrução equivalente
```

String

```
String quote1 = "Ahoy matey";

String quote2a = new String("Ahoy matey");
```

StringBuilder

Mantém um array mutável de caracteres. Em nosso exemplo de código, declaramos um tamanho inicial de 100 por meio do construtor StringBuilder.

```
StringBuilder corvo = new StringBuilder(100);

stringBuilder.append(”Você");

stringBuilder.append(” disse");

stringBuilder.append(" pipoca?");
```

Operador de Adição

Como o objeto String é imutável, cada chamada para concatenação resultará na criação de um novo objeto String.

```
String gimli = "E " + "meu " + "machado ";
```

String.concat

Esse método retorna um objeto String, portanto, encadear o método é um recurso útil.

```
String gimli = ”Ninguém".concat(" joga") .concat(" um")
.concat(" anão");
```

String.format

Este método contém caracteres '%' para representar onde os vários objetos devem ser colocados dentro dele.

```
String myString = String.format("%s %s %.2f %s %s, %s...",
"E", "comi", 2.5056302, "tortas", "de", "morango");
```

String.join (Java 8+)

Uma grande vantagem desse método é não precisar se preocupar com o delimitador entre nossas strings.

```
String[] strings = {"Eu estou", "correndo", "todo", "dia"};

String myString = String.join(" ", strings);
```

String

Como se verifica igualdade entre Strings?

Arrays ou Matrizes

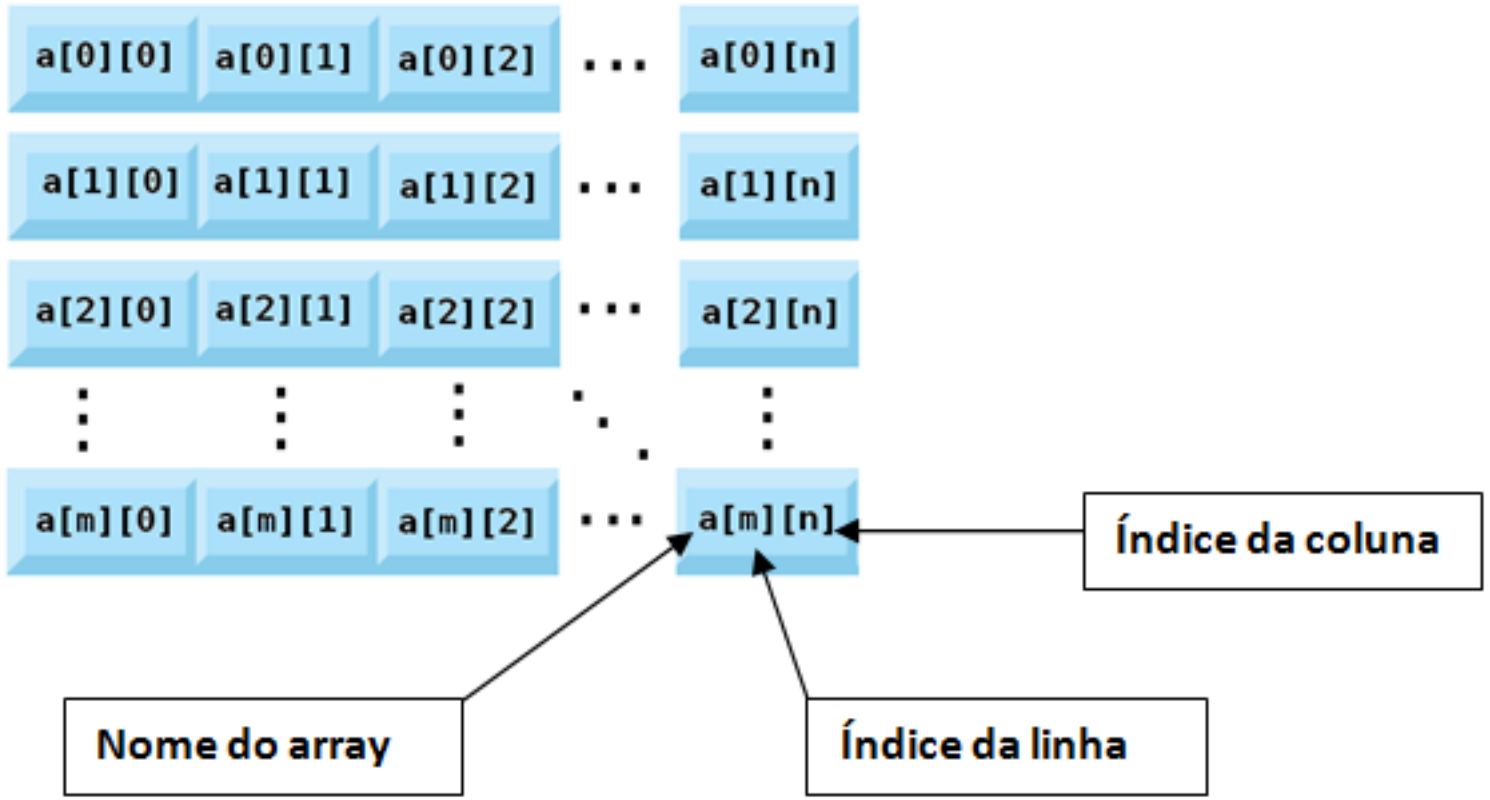
São objetos de “recipientes” que contém um número fixo de valores de um único tipo. O comprimento de um *array* é estabelecido quando criado, sendo que após a criação o seu comprimento fica fixo.

Arrays ou Matrizes

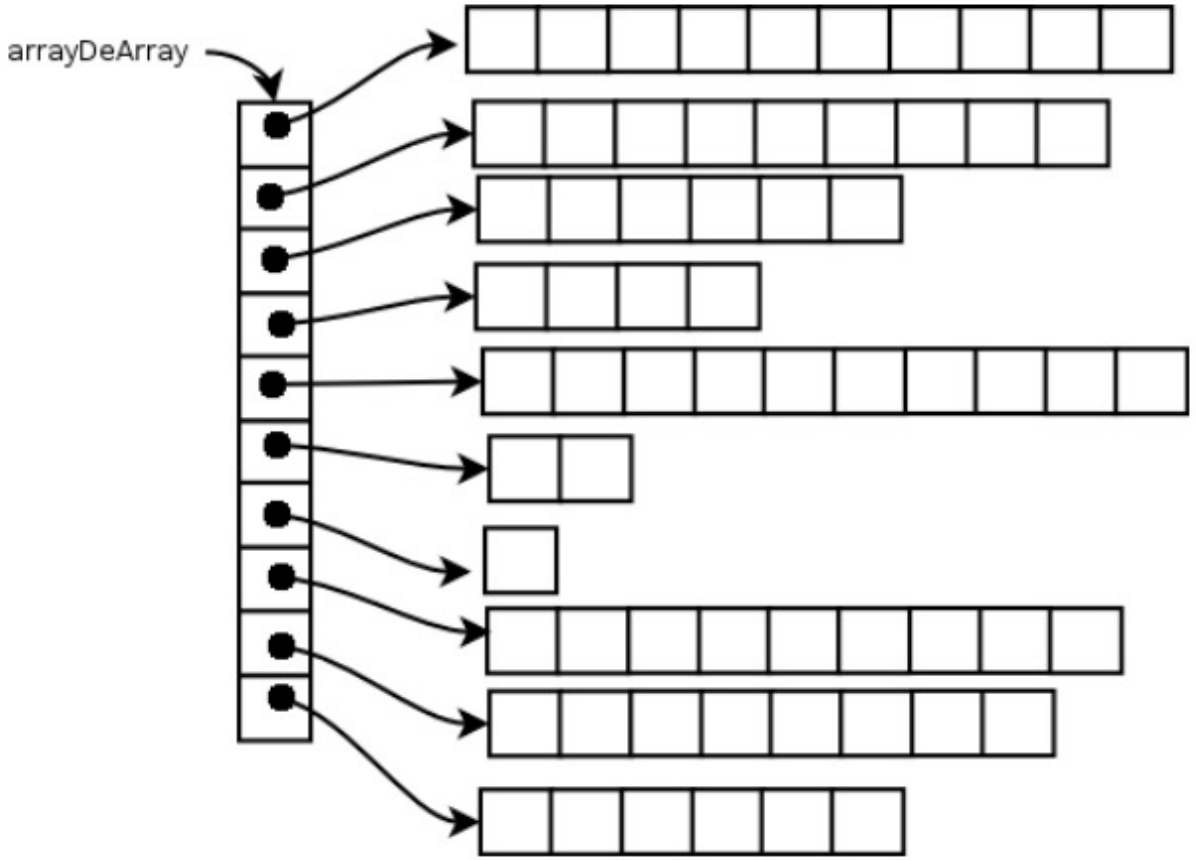
```
        //[] - são inseridos em uma variável que referecia um array
        int[] a = new int[4];
        //OUTRA MANEIRA DE FAZER UMA DECLARAÇÃO DE ARRAY
        int[] b;
        b = new int[10];
        //DECLARANDO VÁRIOS ARRAYS
        int[] r = new int[44], k = new int[23];

//{} - inicializar valores em um array sua declaração
        int[] iniciaValores = {12,32,54,6,8,89,64,64,6};

//DECLARA UM ARRAY DE INTEIROS
        int[] meuArray;

        //ALOCA MEMÓRIA PARA 10 INTEIROS
        meuArray = new int[10];

        //INICIALIZA O PRIMEIRO ELEMENTO
        meuArray [0] = 100;
        meuArray [1] = 85;
        meuArray [2] = 88;
```

Arrays ou Matrizes

```
int [][] a = { { 1, 2 }, { 2, 2 } };
```

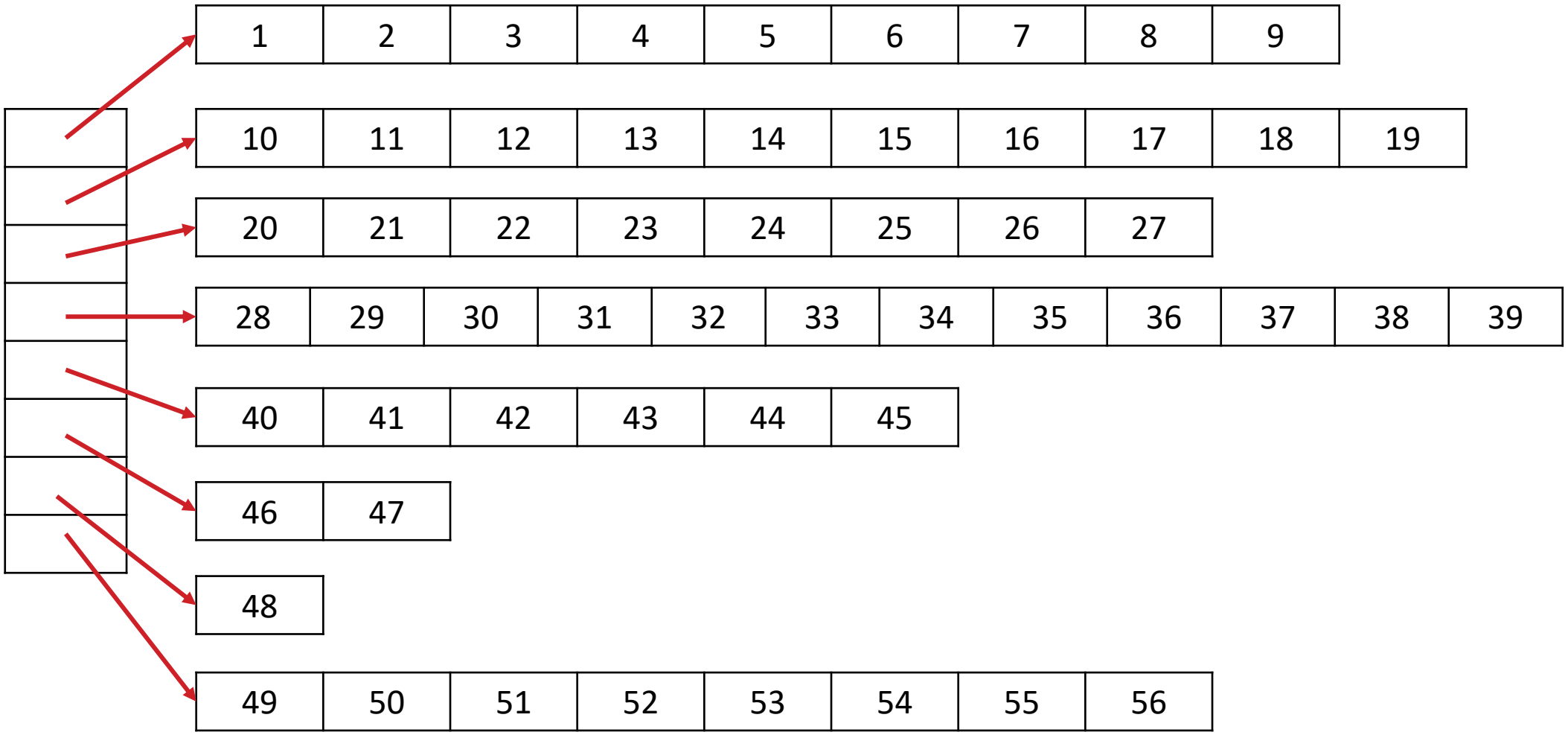


Arrays ou Matrizes

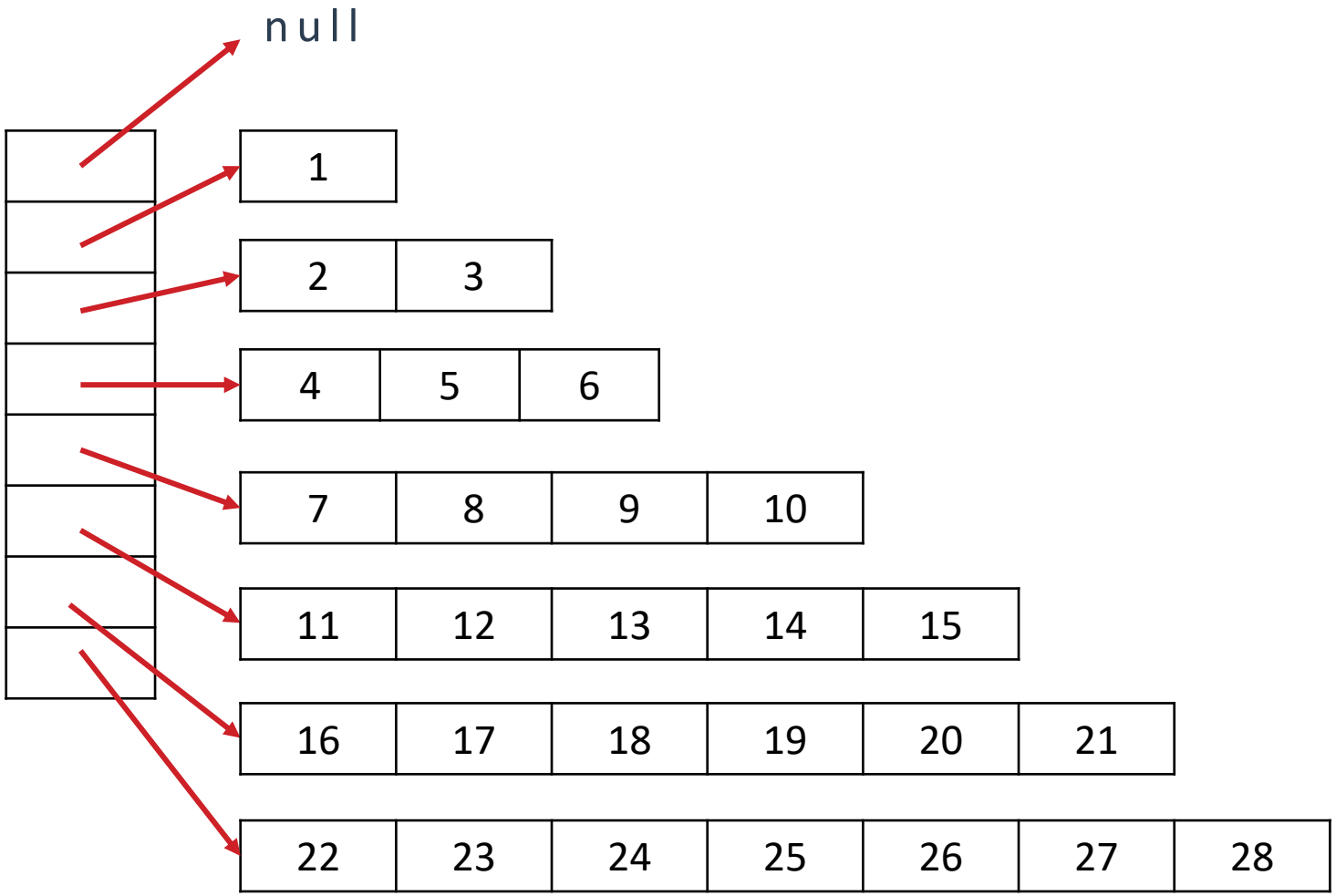


Arrays bidimensional não precisam ser retangulares, ou seja, cada linha pode ter um número diferente de colunas.

Exercício


1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45
46 47
48
49 50 51 52 53 54 55 56

Exercício


null
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

String

Usada para representar cadeias de caracteres Unicode de 16 bits

Ao contrário que ocorre em C e C++, strings em Java não são tratadas como sequências de caracteres terminadas por NULL. São objetos ou instâncias da classe **java.lang.String**, portanto, devem ser declarados e instanciados.

Varargs

Introduzidos no Java 5 e fornecem um atalho para métodos que suportam um número arbitrário de parâmetros de um tipo.

```
public String formatWithVarArgs(String... values) {
        // ...
 }
```

Antes de Varargs

Era necessário passar todos os argumentos em um array ou implementar N métodos (um para cada parâmetro adicional):

```
public String format() { ... }

public String format(String value) { ... }

public String format(String val1, String val2) { ... }
```

Regras Varargs

- Cada método pode ter apenas um parâmetro *varargs*
- O argumento *varargs* deve ser o último parâmetro

```
public static String format(String format, Object... args)

public static String format(Locale l, String format, Object... args)
```

Java é fortemente tipada

Linguagens fortemente tipadas são linguagens que exigem que, na declaração de variáveis/funções, você já forneça o tipo da variável/retorno da função.

Criando um tipo

```
[modificadores] class nomeDaClasse [extends identificadorDaSuperClasse]
    [implements identificadorDaInterface, identificadorDaInterface2,
etc.] {

    [membros de dados]

    [construtores]

    [métodos]

}
```

Métodos

Um método em Java é equivalente a uma função, subrotina ou procedimento em outras linguagens de programação.

Não existe em Java o conceito de métodos globais. Todos os métodos devem sempre ser definidos dentro de uma classe.

Sintaxe do método

```
[modificador] tipoRetorno identificador ([argumentos]) {
        //Corpo do método
}
```

Passando atributos por referência

Como funciona?

Teste

```
class TesteAtributoEstatico {

    static int somaDois(int valor){
        return valor + 2;
    }

    static void somaTres(int valor){
         System.out.println(valor + 3);
    }

    public static void main(String[] args) {

        int valor = 1;
        int valor2 = somaDois(valor);
        soma3(valor);
        System.out.println(valor);
        System.out.println(valor2);

    }
}
```

Teste

```
class TesteAtributoClasse {

    static Integer somaDois(Integer valor){
        return valor + 2;
    }

    static void somaTres(Integer valor){
         System.out.println(valor + 3);
    }

    public static void main(String[] args) {

        Integer valor = 1;
        Integer valor2 = somaDois(valor);
        soma3(valor);
        System.out.println(valor);
        System.out.println(valor2);

    }
}
```

Convenções de nomenclatura

- Use nomes significativos e sem ambiguidade.
- A principal finalidade das convenções é tornar os programas mais legíveis
- Facilita a produção e manutenção de código.

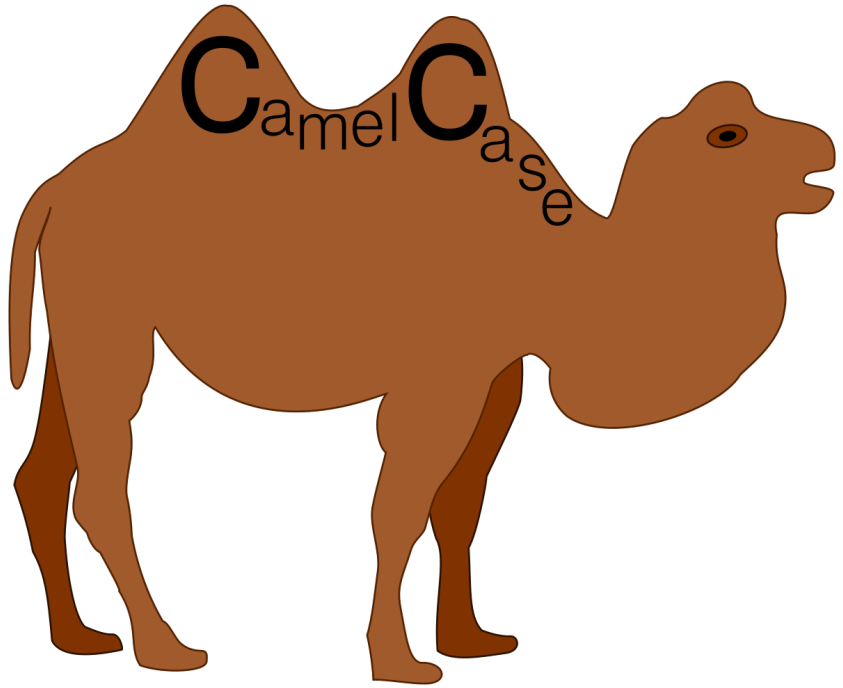
nomenclatura[1]

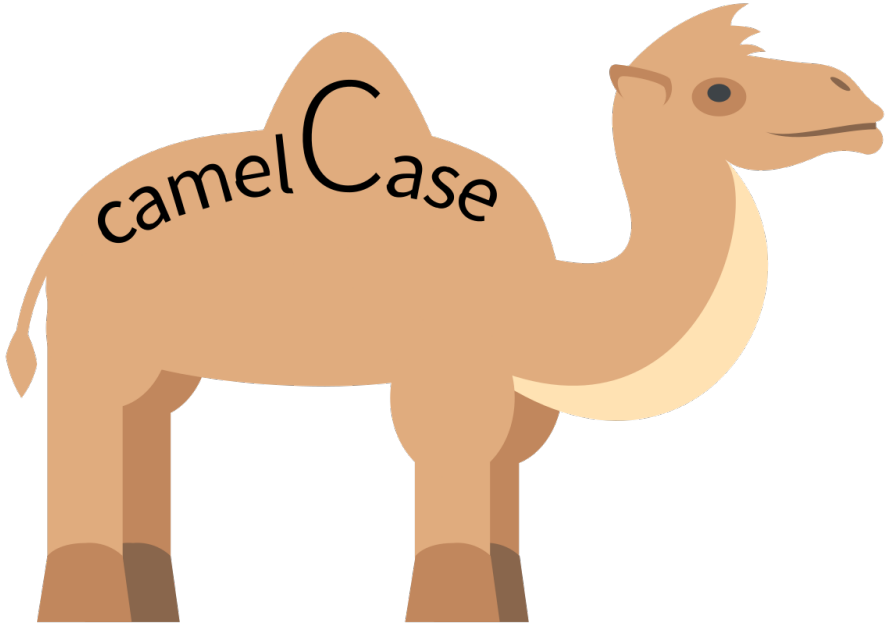
substantivo feminino

1. lista de nomes; nominata, catálogo.
2. m.q. *TERMINOLOGIA* ('conjunto de termos').
   "n. botânica"
3. **LEXICOGRAFIA • LEXICOLOGIA**
   relação de entradas de uma enciclopédia, dicionário, vocabulário, glossário etc.; nominata.

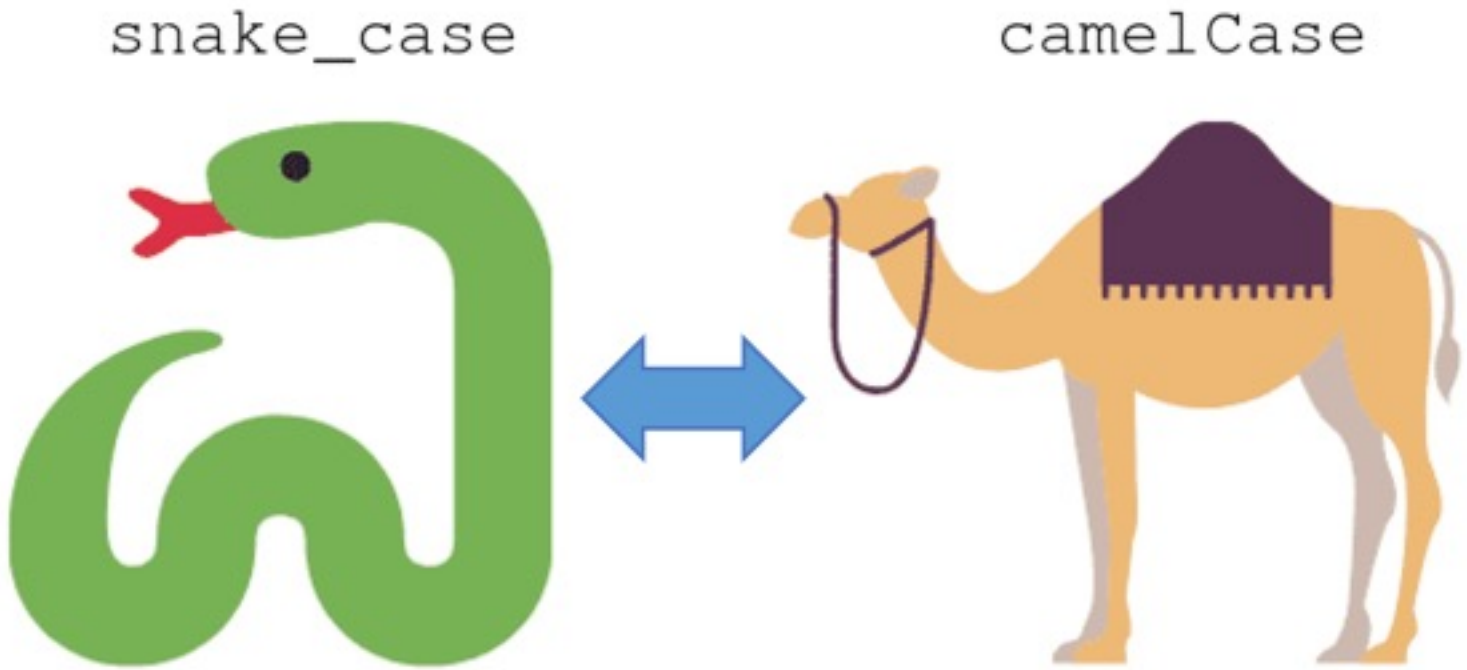
Convenções de nomenclatura

- Use a ''Notação Camelo'' (CamelCase)
  - Letras maiúsculas para os primeiros caracteres em palavras compostas





Convenções de nomenclatura



Convenções de nomenclatura

Camel case

```
{
    "firstName": "John",
    "lastName": "Smith",
    "email": "john.smith@example.com",
    "createdAt": "2021-02-20T07:20:01",
    "updatedAt": "2021-02-20T07:20:01",
    "deletedAt": null
}
```

Snake case

```
{
    "first_name": "John",
    "last_name": "Smith",
    "email": "john.smith@example.com",
    "created_at": "2021-02-20T07:20:01",
    "updated_at": "2021-02-20T07:20:01",
    "deleted_at": null
}
```

Pacotes

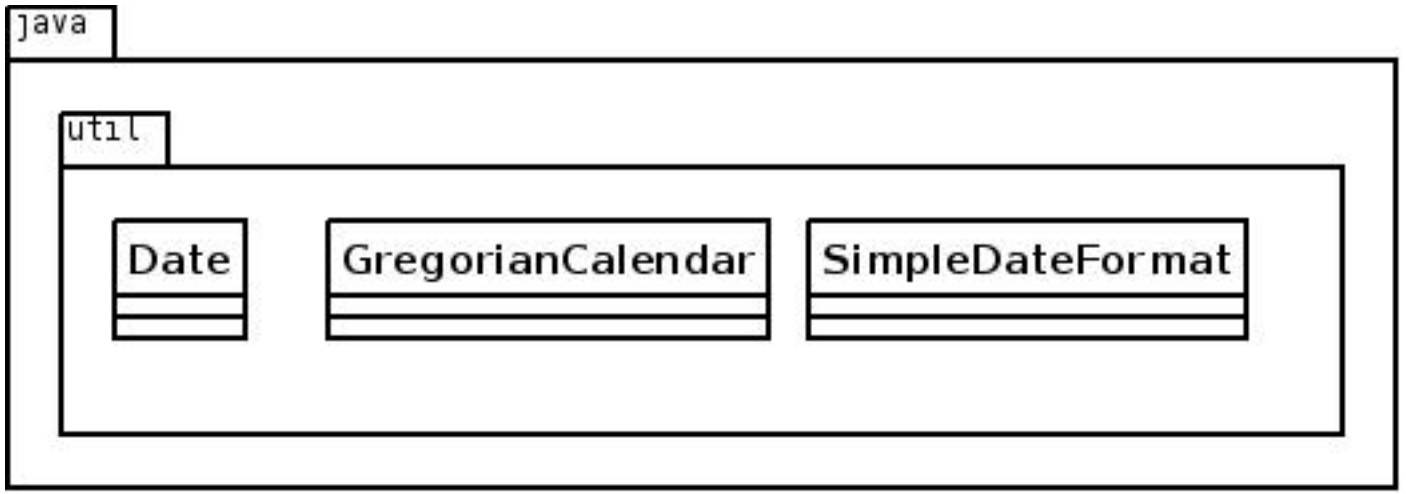
Colocar todos os arquivos em um mesmo local, sem organização, pode aumentar a possibilidade de colisão de classes (classes com o mesmo nome no mesmo escopo) e dificultar a localização de um determinado código

Pacotes

Pacote ou **package** corresponde ao conjunto de classes localizadas na mesma estrutura hierárquica de diretórios

Pacotes

Os diretórios estão diretamente relacionados aos chamados pacotes e costumam agrupar classes de funcionalidades similares ou relacionadas



Padrão da nomenclatura dos pacotes

O nome dos pacotes é relativo ao nome da empresa que desenvolveu a classe, ou seja, o nome do domínio invertido:

br.com.nomedaempresa.nomedoprojeto.subpacote

br.com.nomedaempresa.nomedoprojeto.subpacote2

br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3

Padrão da nomenclatura dos pacotes

br.com.nomedaempresa.nomedoprojeto.subpacote

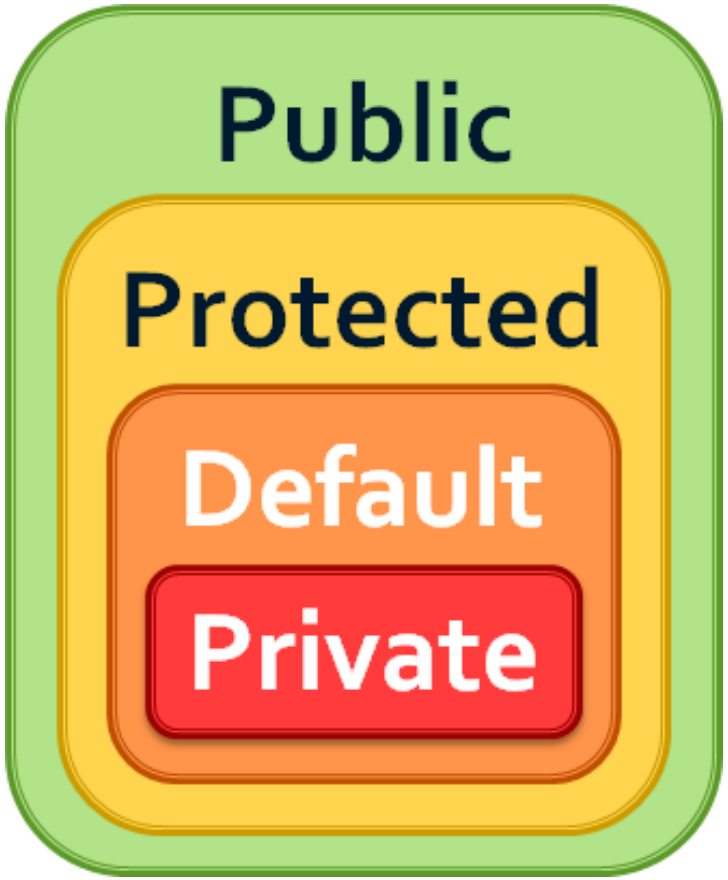
Os pacotes só têm letras minúsculas, não importa quantas palavras estejam contidas neles. Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes.

Pacotes mais comuns

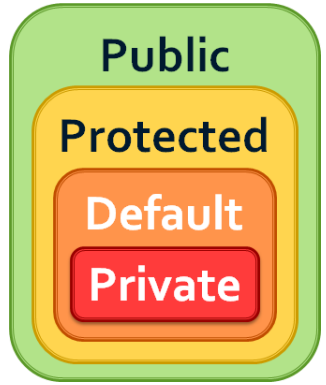
Pacote	Funcionalidade
java.awt	AWT, ou Abstract Window Toolkit. Contém todas as classes para a criação de aplicações com interfaces gráficas com o usuário (ou GUI, Graphical User Interface).
java.io	Fornece as classes para realizar operações de entrada (input) e saída (output) através do sistema de fluxos de dados (streams) e serialização de objetos.
java.lang	Fornece classes que são fundamentais para a concepção da linguagem de programação Java.
java.net	Fornece as classes para implementar aplicações de rede.
java.util	Para trabalhar com coleções, entrada de dados (Scanner) e componentes de data e hora.
javax.swing	O pacote Swing é um pacote de extensão que complementa o pacote AWT.

As classes do pacote padrão de bibliotecas não seguem a nomenclatura que foi dada para bibliotecas de terceiros.

Voltando aos Modificadores de acesso



Modificadores de acesso



Visibilidade	public	protected	default	private
A partir da mesma classe	SIM	SIM	SIM	SIM
Qualquer classe no mesmo pacote	SIM	SIM	SIM	NÃO
Qualquer classe filha no mesmo pacote	SIM	SIM	SIM	NÃO
Qualquer classe filha em pacote diferente	SIM	SIM	NÃO	NÃO
Qualquer classe em pacote diferente	SIM	NÃO	NÃO	NÃO

Modificadores de acesso

public
Uma declaração com o modificador public pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.

private
Os membros da classe definidos como não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.

protected
O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

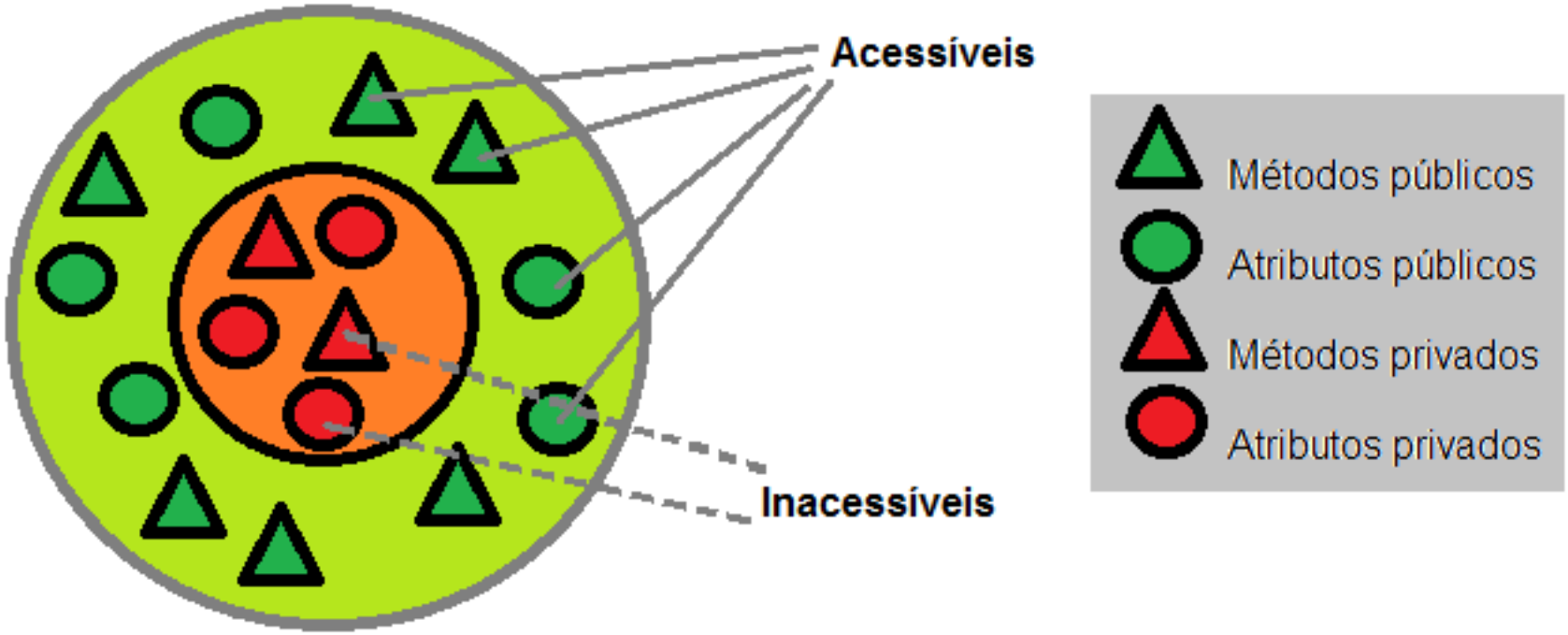
default (padrão):
A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador.

Modificadores de acesso

Cenário e solução	
Você precisa criar uma variável de instância que fique disponível somente para a classe em que ela for declarada. Que modificador de acesso usaria?	Use o modificador de acesso `private`.
Você precisa criar um método que só fique disponível para outros métodos do mesmo pacote ou para uma subclasse da classe em que ele for definido. Que modificador de acesso usaria?	Use o modificador de acesso `protected`
Você precisa criar um método que esteja disponível para todos os outros métodos do aplicativo. Que modificador de acesso usaria?	Use o modificador de acesso `public`.
Você precisa criar uma variável de instância que só fique disponível para outros objetos do mesmo pacote. Que modificador de acesso usaria?	Use o modificador *package-private* (padrão).

Modificadores de acesso

Se alguns atributos ou métodos forem facilmente visíveis e modificáveis, isso pode dar liberdade para que alterações sejam feitas, resultando em efeitos colaterais imprevisíveis.



Encapsulamento

Utilizando modificadores de acesso em atributos e métodos, impede-se o chamado vazamento de escopo, onde um atributo ou método é visível por alguém que não deveria vê-lo, como outro objeto ou classe. Isso evita a confusão do uso de variáveis globais no programa, deixando mais fácil de identificar em qual estado cada variável vai estar a cada momento do programa, já que a restrição de acesso nos permite identificar quem consegue modificá-la

Modificador *static*

Usado para criar variáveis e métodos que irão existir independentemente de quaisquer instâncias criadas para a classe. Os membros static existem antes mesmo de você criar uma nova instância de uma classe, e haverá apenas uma cópia do membro static, independentemente do número de instâncias dessa classe.

Pode ser usado em:

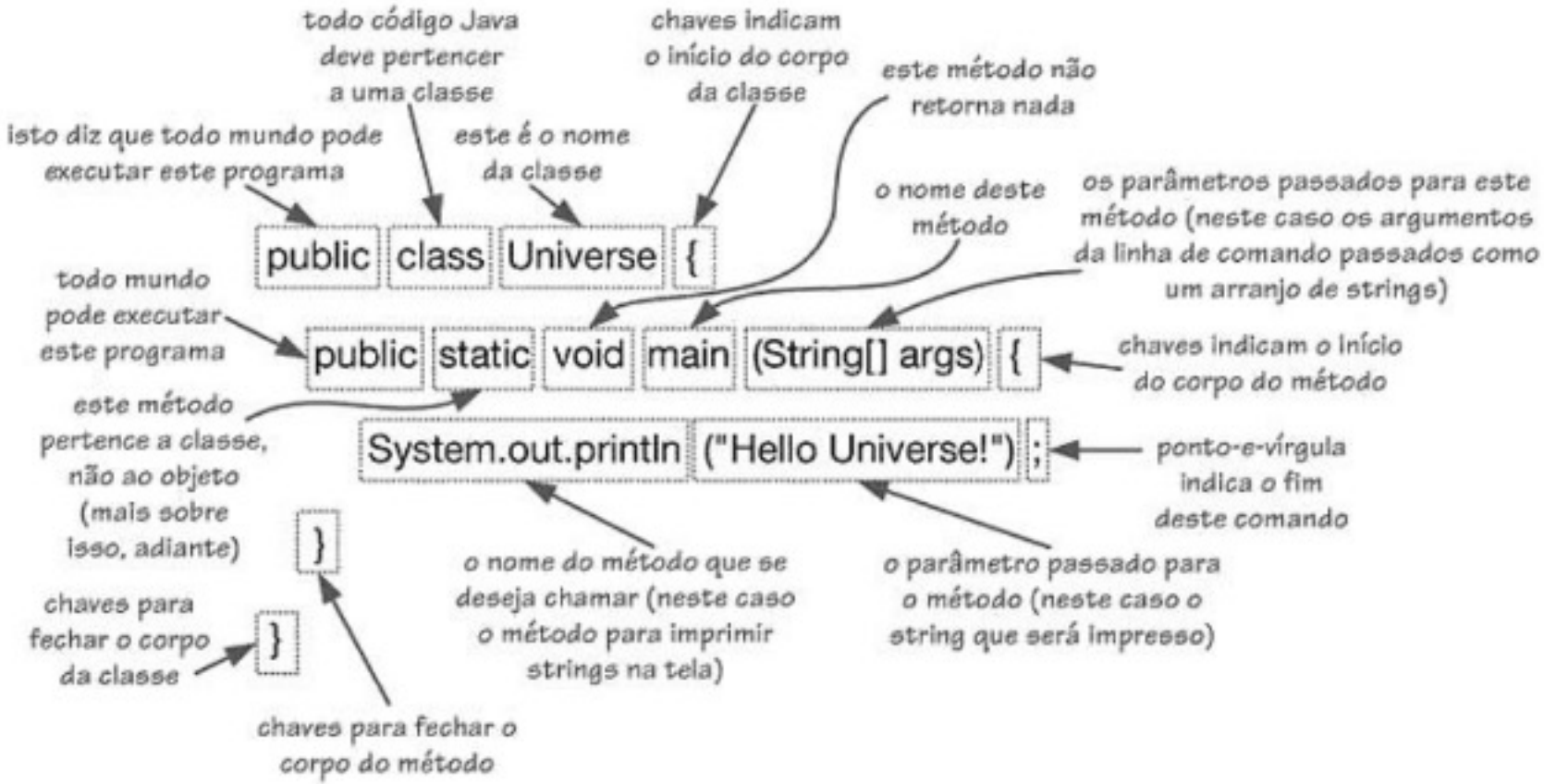
- Blocos de inicialização;
- Variáveis;
- Métodos;
- Uma classe aninhada dentro de outra classe.

O método main()

Um bloco de inicialização estático é executado quando a classe é carregada pela JVM (Java Virtual Machine) e serve para inicializar alguma informação necessária antes mesmo de se ter acesso a uma instância da classe.

O método main() é diferente de outros métodos estáticos, ele é por padrão o método que diz ao Java que essa é a classe principal do projeto. Assim, o JRE saberá qual método será responsável pela inicialização da aplicação. Em uma aplicação Java SE, precisamos de pelo menos um método main() para executá-la.

Resumindo


todo código Java deve pertencer a uma classe
chaves indicam o início do corpo da classe
este método não retorna nada
isto diz que todo mundo pode executar este programa
este é o nome da classe
o nome deste método
os parâmetros passados para este método (neste caso os argumentos da linha de comando passados como um arranjo de strings)
public class Universe {
todo mundo pode executar este programa
public static void main (String[] args) {
chaves indicam o início do corpo do método
este método pertence a classe, não ao objeto (mais sobre isso, adiante)
System.out.println ("Hello Universe!") ;
ponto-e-vírgula indica o fim deste comando
}
o nome do método que se deseja chamar (neste caso o método para imprimir strings na tela)
o parâmetro passado para o método (neste caso o string que será impresso)
chaves para fechar o corpo da classe
}
chaves para fechar o corpo do método

Modificador *final*

Quando é aplicado na classe, não permite estende-la, nos métodos impede que o mesmo seja sobrescrito (overriding) na subclasse, e nos valores de variáveis não pode ser alterado depois que já tenha sido atribuído um valor.

String é ***final*** para não permitir que outros a estendam e destruam sua imutabilidade.

Modificador *final* e String

- Parâmetros de segurança são normalmente representados como String em conexões de rede, URLs de conexão de banco de dados, nomes de usuário/senhas, etc. Se fosse mutável, esses parâmetros poderiam ser alterados facilmente.

- Sincronização e simultaneidade tornando String imutável automaticamente os torna seguros para threads, resolvendo assim os problemas de sincronização.

Modificador *final* e String

- Cache quando o compilador otimiza nossos objetos String, parece que se dois objetos tiverem o mesmo valor (a =" test" e b =" test") e, portanto, precisamos apenas de um objeto de String (para a e b, esses dois apontar para o mesmo objeto).

- Carregamento de classe String é usado como argumentos para carregamento de classe. Se mutável, pode resultar no carregamento da classe errada (porque objetos mutáveis mudam seu estado).

Constantes

Variáveis estáticas também podem ser usadas na criação de constantes no código. Uma constante é simplesmente uma variável que tem um valor definido em tempo de compilação e que não pode ser alterado. Uma constante Java é criada pela inclusão da palavra-chave final a uma variável de classe. A palavra-chave final indica apenas que essa varável não pode ter seu valor alterado:

```
public static final double PI = 3.14;
```

Sobrecarga

PERMITE QUE VOCÊ CRIE **MÉTODOS E CONSTRUTORES** COM O

MESMO NOME, DESDE QUE ELES **TENHAM PARÂMETROS**

DIFERENTES

Sobrecarga

Sobrecarregue os métodos da classe calculadora para receber varags

```
public class Calculadora {

    public double soma(double valor1, double valor2){
     ...
    }

    public double soma(double... valores){
     ...
    }


    ...
}
```

Package, import, class

Ordem

1. Uma (ou nenhuma) vez o package,
2. Pode aparecer um ou mais imports e,
3. Por último, as declarações de classes.

É possível importar um pacote inteiro (todas as classes do pacote, exceto os subpacotes) por meio do coringa *****, ou **classes específicas**:

- **import** java.util.*;
- **import** java.util.**Scanner**;

Package, import, class

Importar todas as classes de um pacote não implica na perda de performance em tempo de execução, mas pode trazer problemas com classes de mesmo nome.

Além disso, importar de um em um é considerado boa prática, pois facilita a leitura a outros programadores. IDEs e editores avançados auxiliam nesta tarefa, e também na organização em diretórios.

Package, import, class

Crie um projeto com dois pacotes:

1. br.edu.ifsp.teste.sem.varargs

2. br.edu.ifsp.teste.com.varargs

Em seguida cria duas classes Calculadora, uma com os métodos de calculo com varargs e outra sem. Utilize-as numa classe Main dentro do pacote **br.edu.ifsp.teste**.

Convenções de nomenclatura

Elemento	Uso de letras	Característica	Exemplo
Nome de classe	Começa com maiúscula e segue com a notação CamelCase	Substantivo	`SpaceShip`
Nome de interface	Começa com maiúscula e segue com a notação CamelCase	Adjetivo terminando com “able” ou “ible” quando se trata de uma qualidade. Caso contrário é um substantivo	`Dockable`
Nome de método	Começa com minúscula e segue com a notação CamelCase	Verbo. Pode incluir adjetivo ou substantivo	`orbit`
Nomes de variáveis de instância e estáticas	Começam com minúscula e seguem com a notação CamelCase	Substantivo	`moon`

Convenções de nomenclatura

Elemento	Uso de letras	Característica	Exemplo
Parâmetros e variáveis locais	Começam com minúscula e seguem com a notação CamelCase se várias palavras forem necessárias	Palavras individuais, acrônimos ou abreviações	`lop` (line of position)
Parâmetros de tipos genéricos	Uma única letra maiúscula	A letra *T* é recomendada	`T`
Constante	Todas as letras são maiúsculas	Quando são várias palavras elas são separadas por sublinhados	`LEAGUE`
Enumeração	Começa com maiúscula e segue com a notação CamelCase; o conjunto de objetos deve ficar todo em maiúsculas	Substantivo	`enum Occupation {MANNED, SEMI_MANNED, UNMANNED}`
Pacote	Todas as letras são minúsculas	Pacotes públicos devem ter o nome de domínio invertido da organização	com.ocajexam.sim

Símbolos e separadores

Símbolo	Descrição	Finalidade
( )	Parênteses	Delimitam o conjunto de argumentos do método, delimitam tipos para coerção (*cast*), determinam a precedência em expressões aritméticas
{ }	Chaves	Delimitam blocos de código, inicializam arrays
[ ]	Colchetes retos	Declaram arrays, inicializam arrays
< >	Colchetes angulares	Delimitam tipos genéricos
;	Ponto e vírgula	Termina a instrução no fim de uma linha
,	Vírgula	Separa identificadores em declarações de variáveis, separa valores, separa expressões em um laço for
.	Ponto	Delineia nomes de pacote, seleciona o campo ou o método de um objeto, suporta o encadeamento de métodos
:	Dois pontos	Vem após rótulos de laços
' '	Aspas simples	Definem um único caractere
->	Operador de seta	Faz a separação entre os parâmetros do lado esquerdo e a expressão do lado direito
" "	Aspas duplas	Definem uma string de caracteres
//	Barras inclinadas	Indicam um comentário de linha única
/* */	Barras inclinadas com asteriscos	Indicam um comentário de bloco com várias linhas
/** */	Barras inclinadas com um asterisco duplo e um simples	Indicam comentários Javadoc

Bibliografia

BARNES, David J.; KOLLING, Michael. **Programação orientada a objetos com Java**. 4. ed. São Paulo: Prentice Hall – Br, 2009.

BOENTE, Alfredo. **Aprendendo a programar em C++: usando classes e objetos**. Rio de Janeiro: Brasport, 2004. ISBN 9788574521749.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. **Análise orientada a objetos**. 2. ed. Florianópolis: Visual Books, 2006.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. 10. ed. São Paulo: Pearson, 2017. ISBN 9788543004792. Disponível em: https://ifsp.bv3.digitalpages.com.br/users/publications/9788543004792. Acesso em: 14 jun. 2019.

Sierra, Kathy. **Use A Cabeça Java**. S.l: Alta Books, 2009. ISBN: 8576081733

INSTITUTO FEDERAL
São Paulo

Câmpus
São Paulo

Obrigado!

Gustavo Fortunato Puga

gustavo.puga@ifsp.edu.br