Java

Prof. Renato Pimentel

2023/2

Sumário

1 Java

A linguagem Java I

- Aplicativos ou softwares Java são criados baseados na **sintaxe** do Java.
- Os programas em Java são escritos baseados na criação de classes.
- Cada classe segue as estruturas de sintaxe do Java, definida pelo agrupamento de **palavras-chave**, ou **palavras reservadas**, e nomes de classes, atributos e métodos.

A linguagem Java II

Palavras reservadas do Java:

abstract	boolean	break	byte
case	catch	char	class
continue	default	do	double
else	extends	false	final
finally	float	for	if
implements	import	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	void	volatile	while

Palavras reservadas, mas não usadas pelo Java: `const`, `goto`

A linguagem Java III

```
1  public class Media {
2     public static void main(String[] args) {
3        //variaveis do tipo real
4        float nota1, nota2, nota3, nota4, mediaAritmetica;
5        //entrada de dados
6        Scanner entrada = new Scanner (System.in);
7        System.out.println("Entre com a nota 1: ");
8        nota1= entrada.nextFloat ();
9        System.out.println("Entre com a nota 2: ");
10       nota2= entrada.nextFloat();
11       System.out.println("Entre com a nota 3: ");
12       nota3= entrada.nextFloat();
13       System.out.println("Entre com a nota 4: ");
14       nota4= entrada.nextFloat();
```


A linguagem Java IV

```
15       //processamento
16       mediaAritmetica = (nota1+nota2+nota3+nota4)/4;
17       //resultados
18       System.out.printf ("A média aritmética: %.2f",
     mediaAritmetica);
19       if (mediaAritmetica >= 7.0){
20          System.out.printf("Aluno aprovado!");
21       }//fim do if
22    } //fim do método main
23 } //fim da classe Média
```

Declarando classes I

Uma classe é declarada com a palavra reservada `class`, seguida do nome da classe e de seu corpo *entre chaves* (como em C).

```
<modificador de acesso> class NomeDaClasse
{
    // declaracao dos atributos
    // declaracao dos métodos
}
```


Declarando classes II

O nome da classe é também chamado de **identificador**.

Regras para nome de uma classe:

- Não pode ser uma palavra reservada do Java
- Deve ser iniciado por uma letra, ou _ ou $
- Não pode conter espaços

```
public class Pessoa
{
    //declaracao dos atributos
    //declaracao dos métodos
}
```

Declarando classes III

Sugestão de estilo para nomes de classes (boa prática de programação):

- A palavra que forma o nome inicia com letra maiúscula, assim como palavras subsequentes:
  - Exemplos: `Lampada`, `ContaCorrente`, `RegistroAcademico`, `NotaFiscalDeSupermercado`, `Figura`, ...
- Devem preferencialmente ser **substantivos**
- Sugere-se que cada classe em Java seja gravada em um **arquivo separado**, cujo nome é o nome da classe seguido da extensão .java

Declarando classes IV

Ex.: classe `Pessoa` no arquivo **Pessoa.java**

```
1 public class Pessoa {
2    //declaracao dos atributos
3    //declaracao dos métodos
4 }
```

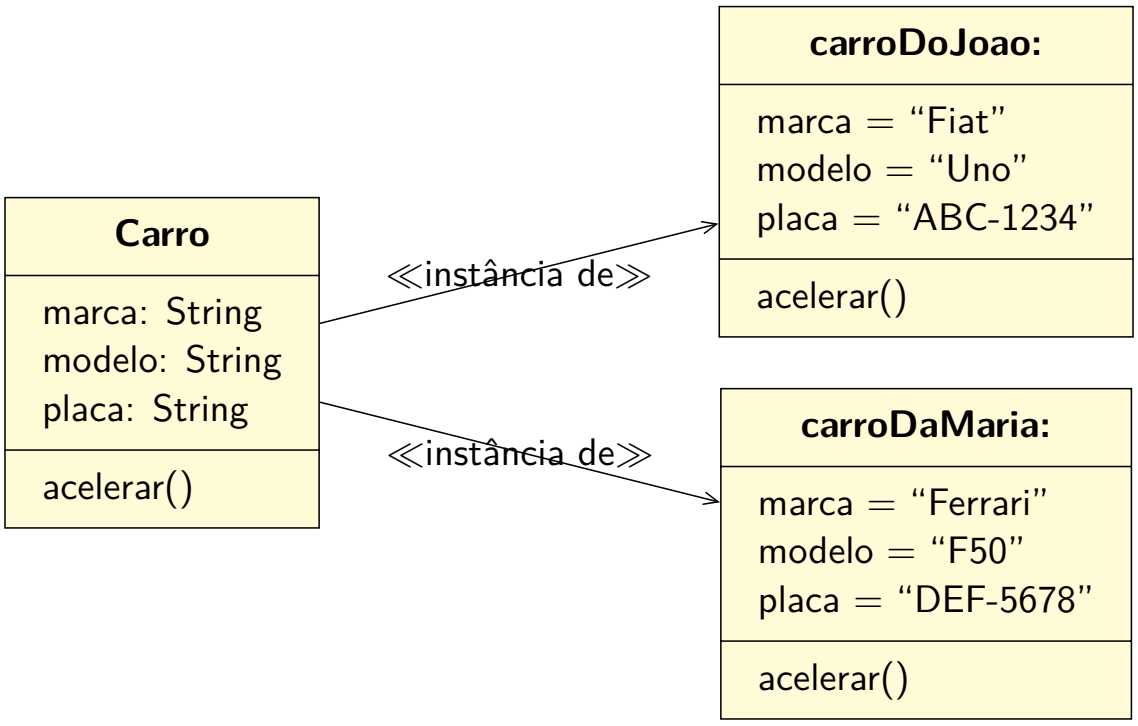
Atributos / variáveis de instância I

- **Atributos / variáveis de instância**: espaço de memória reservado para armazenar dados, tendo um nome para referenciar seu conteúdo;
- Um **atributo** ou **variável de instância** (ou ainda **campo**) é uma variável cujo valor é específico a cada objeto;
- Ou seja, cada objeto possui uma cópia particular de atributos/variáveis de instância com seus próprios valores.
- **Estilo de nome (Java)**: "Camel Case" com primeira letra da primeira palavra minúscula, primeira das demais maiúscula. **Ex.:** `nome`, `dataDeNascimento`, `caixaPostal`.

Atributos / variáveis de instância II

Exemplo: classe `Carro`

Atributos / variáveis de instância III

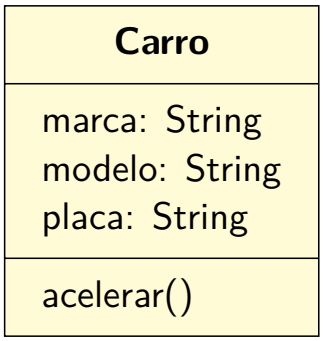


Atributos / variáveis de instância IV

```
public class NomeDaClasse
{
    //variáveis de instâ
    ncia
    //métodos
}
```

- Variáveis de instância são definidas dentro da classe, fora dos métodos;
- Inicializadas automaticamente.
  - Ou por meio de **construtores** dos objetos da classe.

```
public class Carro
{
    String marca;
    String modelo, placa;
    // métodos ...
}
```



Tipos de dados I

- **Tipos**: representam um valor ou uma coleção de valores, ou mesmo uma coleção de outros tipos.
- Tipos **primitivos**:
  - Valores são armazenados nas variáveis diretamente;
  - Quando atribuídos a outra variável, valores são copiados.
- Tipos **por referência**:
  - São usados para armazenar referências a objetos (localização dos objetos);
  - Quando atribuídos a outra variável, somente a referência é copiada (não o objeto).

Tipos de dados II

Tipos primitivos:

- Números inteiros:

Tipo	Descrição	Faixa de valores
`byte`	inteiro de 8 bits	-128 a 127
`short`	inteiro curto (16 bits)	-32768 a 32767
`int`	inteiro (32 bits)	-2147483648 a 21474863647
`long`	inteiro longo (64 bits)	-2^{63} a $2^{63}-1$

- Números reais (IEEE-754):

Tipo	Descrição	Faixa de valores
`float`	decimal (32 bits)	$-3,4e{+}038$ a $-1,4e{-}045$; 1,4e$-$045 a 3,4e$+$038
`double`	decimal (64 bits)	$-1,8e{+}308$ a $-4,9e{-}324$; 4,9e$-$324 a 1,8e$+$308

Tipos de dados III

- Outros tipos

Tipo	Descrição	Faixa de valores
`char`	um único caractere (16 bits)	`'\u0000'` a `'uFFFF'` (0 a 65535 Unicode ISO)
`boolean`	valor booleano (lógico)	`false`, `true`

Tipos de dados IV

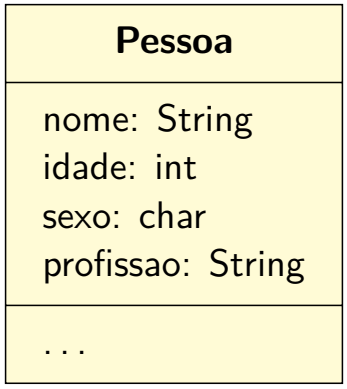
Tipos **por referência** (ou não-primitivos)

- Todos os tipos não primitivos são tipos por referência (ex.: as próprias classes)
- *Arrays* e *strings*:
  - Ex.: `String teste = "UFU Sta. Monica";`
  - Ex.: `int []v = {3,5,8};`

Exercício I

Criar um modelo em Java para uma pessoa. Considere os atributos `nome`, `idade`, `sexo` e `profissão` (não se preocupe ainda com o comportamento/métodos).

Modelagem:



Exercício II

```
public class Pessoa
{
   String nome;
   int idade;
   char sexo;
   String profissao;
// métodos ...
}
```

Métodos em Java I

- Os **métodos** implementam o *comportamento* dos objetos;
- Um método agrupa, em um bloco de execução, uma sequência de comandos que realizam uma determinada função;
- Os métodos em Java são declarados dentro das classes que lhes dizem respeito;
- Eles são capazes de **retornar informações** quando completam suas tarefas.
- **Estilo de nome (Java)**: "Camel Case" igual à forma usada para atributos. Ex.: `salvar`, `salvarComo`, `abrirArquivo`.

Métodos em Java II

Sintaxe da declaração de um método (semelhante a C):

```
1  public class NomeDaClasse
2  {
3      // declaração dos atributos
4      // ...
5
6      <mod. acesso> <tipo de retorno> nomeDoMétodo (
       argumento[s])
7      {
8          // corpo do método
9      }
10 }
```

Métodos em Java III

Exemplo:

```
1  public class Carro
2  {
3      // declaração dos atributos
4      // ...
5
6      public void acelerar()
7      {
8          // corpo do método
9      }
10 }
```


Métodos em Java IV

- Em geral, um método recebe argumentos / parâmetros, efetua um conjunto de operações e retorna algum resultado.
- A definição de método tem cinco partes básicas:
  - modificador de acesso (`public`, `private`, ...);
  - nome do método;
  - tipo do dado retornado;
  - lista de **parâmetros** (argumentos);
  - corpo do método.

Métodos em Java V

- Os métodos aceitam a passagem de um número determinado de parâmetros, que são colocados nos parênteses seguindo ao nome;
- Os argumentos são **variáveis locais** do método – assim como em C.

Exemplo:

```
float hipotenusa (float catetoA, float catetoB)
{
   // corpo do método
   // ...
}
```


Métodos em Java VI

Se o método não utiliza nenhum argumento, parênteses vazios devem ser incluídos na declaração.

Exemplo:

```
public class Carro
{
   String marca;
   String modelo;
   String placa;
   public void acelerar()
   { // corpo do método
   }
}
```

Métodos em Java VII

Um método pode devolver um valor de um determinado tipo de dado.

- Nesse caso, existirá no código do método uma linha com uma instrução `return`, seguida do valor ou variável a devolver.

```
1 class Administrativo {
2     float salario;
3     public float retorneSalario( )
4     {
5         //calcular salario
6         return salario;
7     }
8 }
```


Métodos em Java VIII

Se o método não retorna nenhum valor, isto deve ser declarado usando-se a palavra-chave `void` – como em C.
Exemplo:

```
public class Carro {
    String marca;
    String modelo;
    String placa;
    public void acelerar() // void indica que nao ha
    retorno de valor
    {
    }
}
```

Exercício – métodos I

Criar um modelo em Java para uma lâmpada. As operações que podemos efetuar nesta lâmpada também são simples: podemos ligá-la ou desligá-la. Quando uma operação for executada, imprima essa ação.

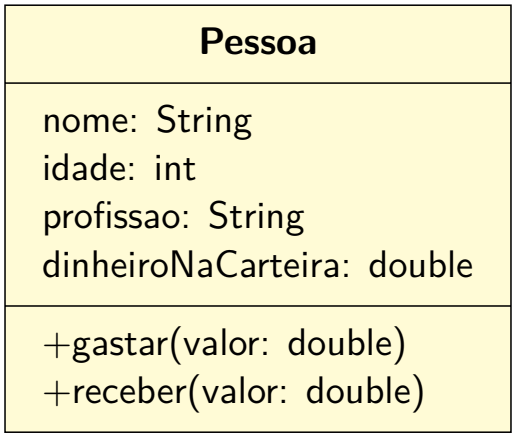
Lampada
estadoDaLampada: boolean
+acender() +apagar()

Exercício – métodos II

```
1  public class Lampada {
2     boolean estadoDaLampada;
3
4     public void acender( )
5     {
6        estadoDaLampada = true;
7        System.out.println("A acao acender foi executada");
8     }
9     public void apagar( )
10    {
11       estadoDaLampada = false;
12       System.out.println("A acao apagar foi executada");
13    }
14 }
```

Exercício – métodos III

Criar um modelo em Java para uma pessoa. Considere os atributos nome, idade e profissão. A pessoa pode gastar ou receber uma certa quantia de dinheiro (assuma que o dinheiro se encontra na carteira).



Exercício – métodos IV

```java
public class Pessoa
{
   String nome, profissao;
   int idade;
   double dinheiroNaCarteira;
   public void gastar( double valor )
   {
      dinheiroNaCarteira = dinheiroNaCarteira - valor;
   }
   public void receber( double valor )
   {
      dinheiroNaCarteira = dinheiroNaCarteira + valor;
   }
}
```

Instanciação de objetos I

- **Declaração**: a seguinte instrução declara que a variável `nomeDoObjeto` refere-se a um objeto / instância da classe `NomeDaClasse`:

```
NomeDaClasse nomeDoObjeto;
```

- **Criação**: a seguinte instrução cria (em memória) um novo objeto / instância da classe `NomeDaClasse`, que será referenciado pela variável `nomeDoObjeto` previamente declarada:

```
nomeDoObjeto = new NomeDaClasse();
```

As duas instruções acima podem ser combinadas em uma só:

```
NomeDaClasse nomeDoObjeto = new NomeDaClasse();
```


Instanciação de objetos II

```
public class Carro {
   String marca;
   String modelo;
   String placa;
   public void acelerar()
   {
   }
}
```

Instanciação de objetos III

```
...
Carro carro1 = new Carro();
...
```


Instanciação de objetos IV

O comando new cria uma instância de uma classe:

- Aloca espaço de memória para armazenar os atributos;
- Chama o **construtor** da classe;
- O construtor inicia os atributos da classe, criando novos objetos, iniciando variáveis primitivas, etc;
- Retorna uma *referência* (**ponteiro**) para o objeto criado.

Comando ponto I

Para acessar um atributo de um objeto, usa-se a notação ponto:
<nome do objeto>.<nome da variavel>

```
public class Carro {
   String marca;
   String modelo;
   String placa;
   public void acelerar()
   {
   }
}
```

```
...
Carro car1 = new Carro();
Carro car2 = new Carro();
//inicializando car1
car1.marca = "Fiat";
car1.modelo = "2000";
car1.placa = "FRE-6454";
//inicializando car2
car2.marca = "Ford";
car2.modelo = "1995";
car2.placa = "RTY-5675";
...
```


Comando ponto II

Para acessar um método de um objeto, usa-se a notação ponto:
<nome do objeto>.<nome do metodo>

```
public class Carro {
   String marca;
   String modelo;
   String placa;
   public void acelerar()
   { // corpo do metodo
   }
   public void frear()
   { // corpo do metodo
   }
}
```

```
...
Carro car1 = new Carro();
//inicializando car1
car1.marca="Fiat";
car1.modelo="2000";
car1.placa="FRE-6454";
//usando os métodos
car1.acelerar();
car1.frear();
...
```

Passagem de mensagens I

- Para *mandar mensagens* aos objetos utilizamos o operador ponto, seguido do método que desejamos utilizar;
- Uma **mensagem** em um objeto é a ação de efetuar uma chamada a um método.

```
Pessoa p1;
p1 = new Pessoa();
p1.nome = "Vitor Josue Pereira";
p1.nascimento = "10/07/1966";
p1.gastar( 3200.00 ); // Mensagem sendo passada ao
   objeto p1
```


Passagem de mensagens II

Um programa orientado a objetos nada mais é do que vários objetos dizendo uns aos outros o que fazer.

- Quando você quer que um objeto faça alguma coisa, você envia a ele uma "mensagem" informando o que quer fazer, e o objeto faz;
- Se o método for **público**, o objeto terá que executá-lo;
- Se ele precisar de outro objeto para o auxiliar a realizar o "trabalho", ele mesmo vai cuidar de enviar mensagem para esse outro objeto.

O método `main()`

- O método main() é o ponto de partida para todo aplicativo em Java.
- É nele que são instanciados os primeiros objetos que iniciarão o aplicativo.
- A forma mais usual de se declarar o método `main()` é mostrada abaixo:

```
public class ClassePrincipal
{
   public static void main (String args [])
   {
      //corpo do método
   }
}
```


Exercício I

Criar um modelo em Java para uma lâmpada. Implemente o modelo, criando dois objetos `Lamp1` e `Lamp2`. Simule a operação `acender` para `Lamp1` e `apagar` para `Lamp2`.

Lampada
estadoDaLampada: boolean
+acender() +apagar()

Exercício II

```java
public class Lampada {
   boolean estadoDaLampada;

   public void acender( )
   {
      estadoDaLampada = true;
      System.out.println("A acao acender foi executada");
   }
   public void apagar( )
   {
      estadoDaLampada = false;
      System.out.println("A acao apagar foi executada");
   }
}
```


Exercício III

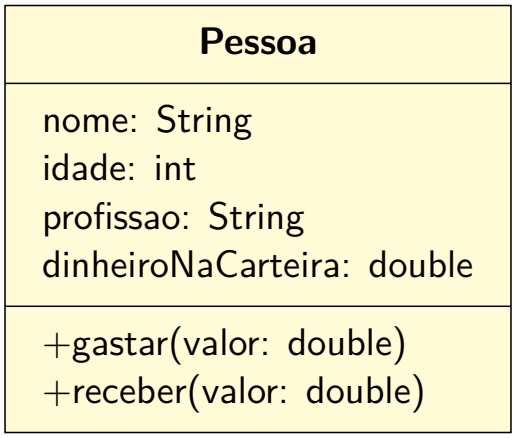
```java
public class GerenciadorDeLampadas {
   public static void main(String args[])  {
      // Declara um objeto Lamp1 da classe Lampada
      Lampada Lamp1;
      // Cria um objeto da classe Lampada
      Lamp1 = new Lampada();
      //Simulando operação sobre objeto Lamp1
      Lamp1.acender();

      // Declara um objeto Lamp2 da classe Lampada
      Lampada Lamp2;
      // Cria um objeto da classe Lampada
      Lamp2 = new Lampada();
      //Simulando operação sobre objeto Lamp2
      Lamp2.apagar();
   }
}
```

Exercício IV

Criar um modelo em Java para uma pessoa. Considere os atributos `nome`, `idade` e `profissão`. A pessoa pode gastar ou receber uma certa quantia de dinheiro (assuma que o dinheiro se encontra na carteira). Implemente o modelo, criando dois objetos `p1` e `p2`. Assuma que o objeto `p1` tem 3.200 na carteira, e o objeto `p2` tem 1.200. Simule operações de gasto e recebimento.

Exercício V



Exercício VI

```java
public class Pessoa
{
   String nome, profissao;
   int idade;
   double dinheiroNaCarteira;
   public void gastar( double valor )
   {
      dinheiroNaCarteira = dinheiroNaCarteira - valor;
   }
   public void receber( double valor )
   {
      dinheiroNaCarteira = dinheiroNaCarteira + valor;
   }
}
```


Exercício VII

```java
public class GerenciadorDePessoas
{
   public static void main(String args[])
   {
      // Declara um objeto da classe Pessoa
      Pessoa p1;
      // Cria um objeto da classe Pessoa
      p1 = new Pessoa();
      //Atribuindo valor aos atributos do objeto p1
      p1.nome = "Vitor Pereira";
      p1.idade = 25;
      p1.profissao = "Professor";
      p1.dinheiroNaCarteira = 3200.00;
      System.out.println( "Salário de " + p1.nome + " = " +
      p1.dinheiroNaCarteira );
      // Vitor recebeu 1000 reais
      p1.receber( 1000.00 );
```

Exercício VIII

```
18          System.out.println( p1.nome + "tem " +
19          p1.dinheiroNaCarteira + " reais");
20          // Vitor gastou 200 reais
21          p1.gastar( 200.00 );
22          System.out.println( p1.nome + "tem agora " +
23          p1.dinheiroNaCarteira + " reais");
24
25          // Declara e cria um outro objeto da classe Pessoa
26          Pessoa p2 = new Pessoa();
27          //Atribuindo valor aos atributos do objeto p2
28          p2.nome = "João Silveira";
29          p2.idade = 30;
30          p2.dinheiroNaCarteira = 1200.00;
31          System.out.println( "Salário de " + p2.nome + " = " +
32          p2.dinheiroNaCarteira );
```


Exercício IX

```
33          // João recebeu 400 reais
34          p2.receber( 400.00 );
35          System.out.println( p2.nome + "tem " +
36          p1.dinheiroNaCarteira + " reais");
37          // João gastou 100 reais
38          p2.gastar( 100.00 );
39          System.out.println( p2.nome + "tem agora " +
40          p1.dinheiroNaCarteira + " reais");
41      }
42  }
```

Comandos básicos do Java I

Declaração de variáveis:

```
<tipo> <nomeDaVariável> [=
 <valorInicial>];
```

```
int x1;
double a, b = 0.4;
float x = -22.7f;
char letra;
String nome = "UFU"
boolean tem;
```

Atribuição de valores:

```
<nomeDaVariável> = <valor
>;
```

```
double soma;
soma = 0.6;
obj.nome = "UFU";
```


Comandos básicos do Java II

Desvio condicional simples:

```
if (<condição>)
   <instrução>;


// {}: mais de uma instruç
   ão
if (<condição>)
{
   <instruções>;
}
```

Desvio condicional composto:

```
if (<condição>)
{
   <instruções>;
}
else
{
   <instruções>;
}
```

Comandos básicos do Java III

Escolha-caso:

```
switch (expressão)
{
    case <valor1>:
        <comando(s);>
        break;
    case <valor2>:
        <comando(s);>
        break;
        ...
    case <valorN>:
        <comando(s);>
        break;
    default:
        <comando(s);>
}
```


Comandos básicos do Java IV

Repetição
Com `while`

```
while (<condição>)
    <comando>;
```

```
while (<condição>)
{
    <comandos>;
}
```

Com `do-while` – teste ao final

```
do
{
    <comandos>;
} while (<condição>);
```

Comandos básicos do Java V

Repetição com o laço `for`

```
for ([tipo] <var = valInicial> ; <condição> ; <
    incremento>)
{
    <comandos>;
}
```

Obs.: passar o tipo quando estiver declarando a variável no comando `for`

Comandos básicos do Java VI

Exemplo – variável `i` não declarada antes do `for`:

```
for(int i = 1; i <= 10; i++) {
    <comando 1>;
    <comando 2>;
}
```

Vetores e matrizes I

Vetor (*array*): agrupamento de valores de um mesmo tipo primitivo ou de *objetos de uma mesma classe*.
Em Java, primeiro elemento sempre tem índice 0

- Em vetor de `n` elementos, índices variam de `0` a `n-1`

Exemplo:

0	1	2	3	4
v[0]	v[1]	v[2]	v[3]	v[4]

- Todos os *arrays* são objetos da classe `java.lang.Object` – importada automaticamente, ver Java API adiante;
- Em Java, vetores têm *tamanho fixo* (dado pelo atributo `length`) e **não podem ser redimensionados**;
- Para redimensionar, deve-se criar um novo e fazer cópia.

Vetores e matrizes II

A utilização de vetores na linguagem Java envolve três etapas:

1. Declarar o vetor;
2. Reservar espaço na memória e definir o tamanho do vetor;
3. Armazenar elementos no vetor.

Vetores e matrizes III

Para declarar um vetor em Java é preciso acrescentar um par de colchetes antes, ou depois, do nome da variável
Exemplo:

```
int idade[];          // Colchetes depois do
double salario[];     // nome da
String diasDaSemana[];  // variável

double []nota;        // Colchetes antes do
String []nome;        // nome da variável:
char []vet1, vet2, vet3;// Esta notação é útil quando
    queremos declarar mais de um vetor, pois não há
    necessidade de repeti-los.
```


Vetores e matrizes IV

É preciso definir o tamanho do vetor – isto é, a quantidade total de elementos que poderá armazenar;
Em seguida é necessário reservar espaço na memória para armazenar os elementos. Isto é feito pelo operador new.

```
// Sintaxe 1
int idade[];
idade = new int[10];
double salario[];
salario = new double[6];

// Sintaxe 2
double []nota = new double [125];
String nome[] = new String [70];
```

Vetores e matrizes V

A cópia de vetores é possível através de um método da classe `System`, denominado `arraycopy()`:

```
int v1[] = {1,2,3,4,5};
v1[5] = 10; // erro: índice fora dos limites

int v2[] = new int[10];
System.arraycopy(v1, 0, v2, 0, v1.length);
v2[5] = 10; // ok, os índices de v2 vão de 0 a 9
```

Sintaxe: `arraycopy(v1,i1,v2,i2,n)`: Copia n elementos do vetor `v1` – a partir do elemento de índice $i1$ – para o vetor `v2`, a partir do índice $i2$.

Vetores e matrizes VI

Conforme já adiantado no exemplo anterior, há um atalho que resume os três passos vistos anteriormente (declaração, reserva de espaço, atribuição de valor).
Outro exemplo deste atalho:

```
// 10 primeiros elementos da sequência de Fibonacci
long fibonacci[] = {1, 1, 2, 3, 5, 8, 13, 21, 34, 55};
```

Vetores e matrizes VII

Também é possível armazenar **objetos** em vetores:

```
Pessoa [] clientes = new Pessoa[3];
clientes[0] = new Pessoa(); // Necessário neste caso
clientes[0].nome = "João";
clientes[0].idade = 33;
clientes[0].profissao = "gerente";
clientes[1] = new Pessoa();
clientes[1].nome = "Pedro";
clientes[1].idade = 25;
clientes[1].profissao = "caixa";
clientes[2] = new Pessoa();
clientes[2].nome = "Maria";
clientes[2].idade = 20;
clientes[2].profissao = "estudante";
```


Vetores e matrizes VIII

Percorrendo *arrays*: podemos fazer `for` especificando os índices – como fazemos em C.
Porém, se o **vetor todo será percorrido**, podemos usar o ***enhanced for***:

```
public void somaVetor()
{
   int[] vetor = {87, 68, 94, 100, 68, 39, 10};
   int total = 0;
   for (int valor: vetor) // enhanced for
      total = total + valor;
      System.out.println("Total = " + total);
}
```

Vetores e matrizes IX

Vetores multidimensionais:

- Vetor bidimensional: **matriz**.
  - Ex.: `double[][] matriz = new double[5][10];`
- É possível construir vetores multidimensionais **não retangulares**:

```
// matriz de inteiros, 2 x 3 (2 linhas e 3 colunas)
int v[][] = new int[2][];
v[0] = new int[3];
v[1] = new int[3];
// vetor bidimensional não-retangular.
int vet[][] = new int[3][];
vet[0] = new int[2];
vet[1] = new int[4];
vet[2] = new int[3];
```


Vetores e matrizes X

Em Java, é possível criar métodos que recebem um número não especificado de parâmetros – **usa-se um tipo seguido por reticências**:

- Método recebe número variável de parâmetros desse tipo;
- Reticências podem ocorrer apenas uma vez;
- No corpo do método, a lista variável é tratada como um vetor.

```
public class Media {
   public double media(double... valor) {
      double soma = 0.0;
      for (int i=0; i<valor.length; i++)
         soma = soma + valor[i];
      return soma/valor.length;
   }
}
```

Java API I

Java API (*Applications Programming Interface*): conjunto de classes pré-definidas do Java;
API está organizadas em pastas (pacotes)

Java API II

Exemplos:

Pacote	Descrição
`java.lang`	Classes muito comuns (este pacote é **importado automaticamente** pelo compilador `javac` em todos os programas Java).
`java.io`	Classes que permitem entrada e saída em arquivos.
`java.math`	Classes para executar aritmética de precisão arbitrária
`javax.swing`	Classes de componentes GUI Swing para criação e manipulação de interfaces gráficas do usuário
`java.util`	Utilitários como: manipulações de data e hora, processamento de números aleatórios, armazenamento e processamento de grandes volumes de dados, quebras de *strings* em unidades léxicas etc.

Java API III

A classe `java.lang.Math`, por exemplo, contém valores de constantes, como `Math.PI` e `Math.E`, e várias funções matemáticas. Alguns métodos:

Método	**Descrição**
`Math.abs(x)`	Valor absoluto de `x`.
`Math.ceil(x)`	Teto de `x` (menor inteiro maior ou igual a `x`).
`Math.cos(x)`	Cosseno de `x` (`x` dado em radianos).
`Math.exp(x)`	Exponencial de `x` (e^x).
`Math.floor(x)`	Piso de `x` (maior inteiro menor ou igual a `x`).

Métodos estáticos

Chamados **a partir da classe** (e não de um objeto específico)

Java API IV

A classe `String` opera sobre *cadeias de caracteres*.

Método	**Descrição**
`ob.concat(s)`	Concatena objeto `ob` ao `String s`.
`ob.replace(x,y)`	Substitui, no objeto `ob`, todas as ocorrências do caractere `x` pelo caractere `y`.
`ob.substring(i,j)`	Constrói novo String para `ob`, com caracteres do índice i ao índice $j-1$.

Métodos não são estáticos

Chamados a partir de objetos da classe `String`.

Armazenamento em memória I

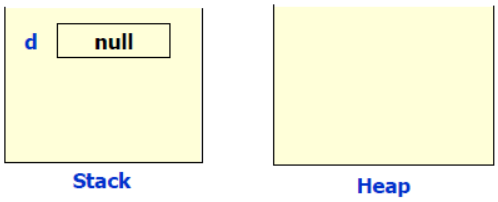
Variáveis de tipos primitivos ou referências são criados em uma área de memória conhecida como **Stack** (pilha);
Por sua vez, objetos são criados em área de memória conhecida como **Heap** (monte). Uma instrução `new Xxxx()`:

- Aloca memória para a variável de referência ao objeto na pilha e inicia-a com valor **null**;
- Executa construtor, aloca memória na *heap* para o objeto e inicia seus campos (atributos);
- Atribui endereço do objeto na *heap* à variável de referência do objeto na pilha.

Armazenamento em memória II

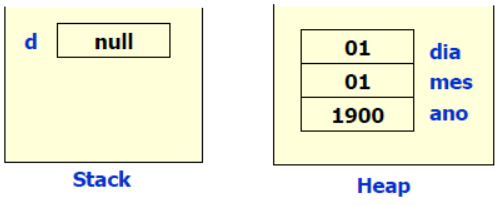
1

```
MinhaData d = new MinhaData (17, 3, 2009);
```



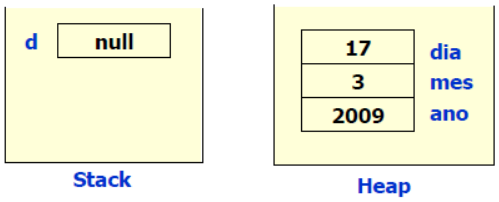
2

```
MinhaData d = new MinhaData (17, 3, 2009);
```



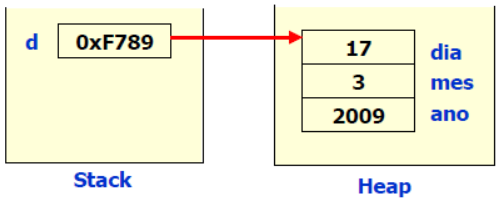
3

```
MinhaData d = new MinhaData (17, 3, 2009);
```



4

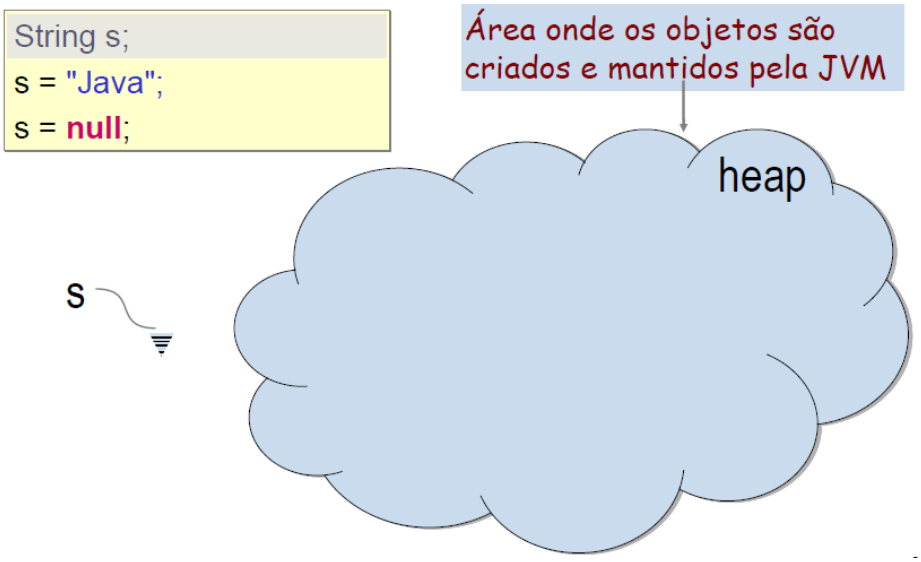
```
MinhaData d = new MinhaData (17, 3, 2009);
```



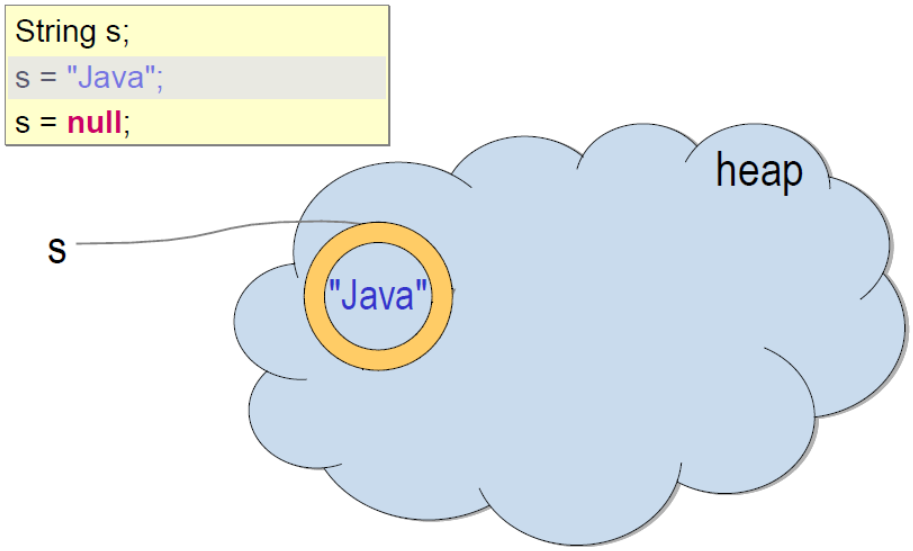
Coleta de lixo I

- Periodicamente, a JVM realiza uma **coleta de lixo**:
  - Retorna partes de memória não usadas;
  - Objetos não referenciados;
  - Não existem então primitivas para alocação e desalocação de memória (como `calloc`, `free`, etc).

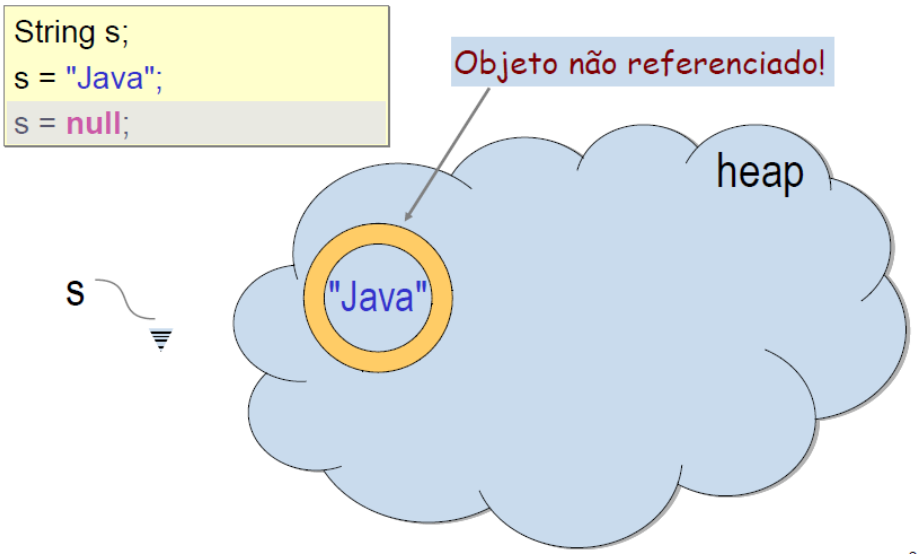
Coleta de lixo II

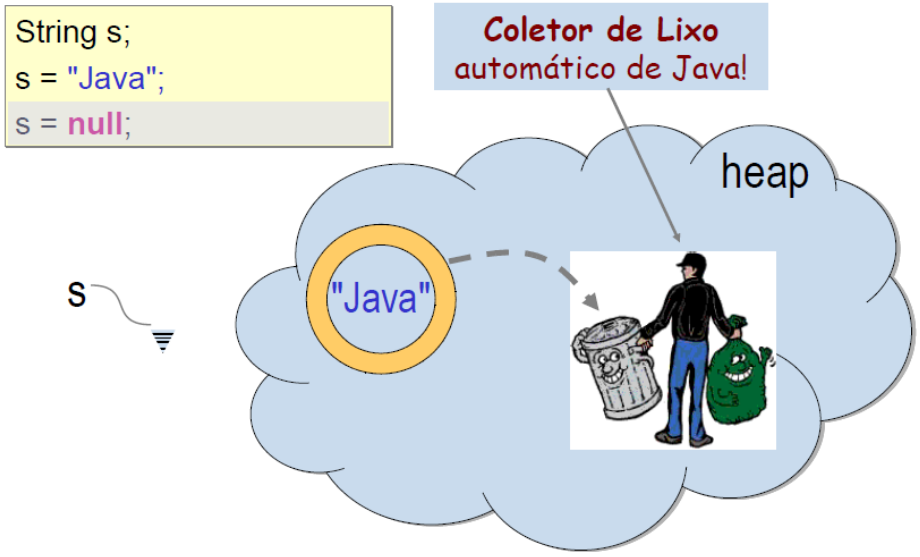


Coleta de lixo III



Coleta de lixo IV





Encapsulamento e modificadores de acesso I

A linguagem Java oferece mecanismos de controle de acessibilidade (visibilidade):

Encapsulamento

Encapsulamento é a capacidade de controlar o acesso a classes, atributos e métodos

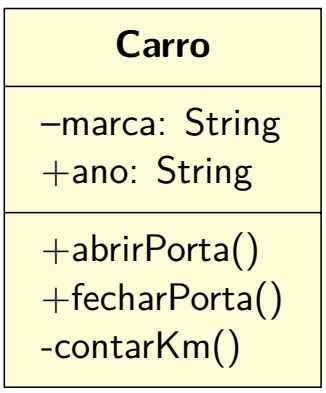
- O encapsulamento é implementado através dos **modificadores de acesso**;
  - São palavras reservadas que permitem definir o encapsulamento de classes, atributos e métodos;
- **Modificadores de acesso**: `public`, `private`, `protected`.
- Quando se omite, o acesso é do tipo *package-only*.
- Classes: apenas `public` – ou omitido (*package-only*) – é permitido.

Encapsulamento e modificadores de acesso II

- *Package-only*
  - Caso padrão (quando modificador de acesso é omitido);
  - Permite acesso a partir das classes do mesmo pacote – ou classe de outro pacote, desde que seja *subclasse* (herdeira) da classe em questão (conceito de herança).
- `protected`
  - Permite acesso a partir de uma classe que é herdeira de outra.
- `public`
  - Permite acesso irrestrito a partir de qualquer classe (mesmo que estejam em outros arquivos);
  - Único que pode ser usado em classes.
- `private`
  - Permite acesso apenas por objetos da própria classe. O elemento é visível apenas dentro da classe onde está definido.

Encapsulamento e diagrama de classes I

- +: `public` – visível em qualquer classe;
- –: `private` – visível somente dentro da classe.
- #: `protected` – visibilidade associada à herança




```java
public class Carro
{
    private String marca;
    public String ano;
    public void abrirPorta()
    {
        //corpo do método
    }
    public void fecharPorta()
    {
        //corpo do método
    }
    private void contarKm()
    { /*corpo do método*/ }
}
```

```java
public class Carro
{
   private String marca;
   public String ano;
   public void abrirPorta()
   {
      //corpo do método
   }
   public void fecharPorta()
   {
      //corpo do método
   }
   private void contarKm()
   { /*corpo do método*/ }
}
```

```java
public class UsaCarro
{
   public static void main(String
    args[])
   {
      Carro car1 = new Carro();
      car1.ano = "2000";
      car1.marca = "Fiat";

      car1.fecharPorta();
      car1.contarKm();
   }
}
```

Erros de semântica: linhas 7 e 10.

- Um método `public` pode ser invocado (chamado) dentro da própria classe, ou a partir de qualquer outra classe;
- Um método `private` é acessível apenas dentro da classe a que pertence.

Encapsulamento e atributos I

- Atributos públicos podem ser acessados e modificados a partir de qualquer classe;
- A menos que haja razões plausíveis, os atributos de uma classe devem ser definidos como `private`;
- Tentar acessar um componente privado de fora da classe resulta em **erro de compilação**.

Encapsulamento e atributos II

Mas então como acessar atributos, ao menos para consulta (leitura)?

- Para permitir o acesso aos atributos de uma maneira controlada, a prática mais comum é de criar *dois métodos*;
- Os dois métodos são definidos na própria classe onde o atributo se encontra;
- Um dos métodos retorna o valor da variável
- Outro método muda o seu valor;
- Padronizou-se nomear esses métodos colocando a palavra *get* ou *set* antes do nome do atributo.

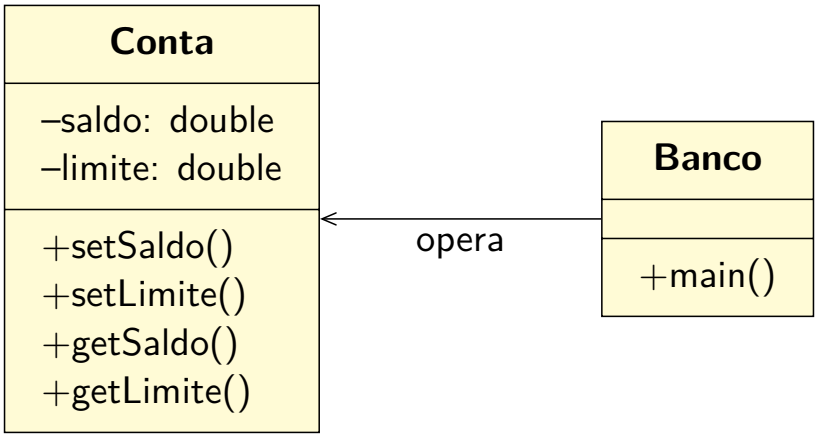
Métodos *getters* e *setters*

- Com atributos sendo `private`, é frequente usar **métodos acessores/modificadores** (*getters*/*setters*) para manipular atributos;
- Porém, devemos ter cuidado para não quebrar o encapsulamento:

Se uma classe chama `objeto.getAtrib()`, manipula o valor do atributo e depois chama `objeto.setAtrib()`, o atributo é essencialmente público.
De certa forma, estamos quebrando o encapsulamento!

Exercício – simulação de um banco (v0)

Construa um programa para simulação de um banco que possui apenas uma `conta`, com os atributos privados `saldo` e `limite`. Utilize métodos *getters* e *setters* para manipular os valores dos atributos e visualizá-los. Uma entidade `banco` é responsável pela criação da conta e sua operação.



Classe Conta.java (versão 0)

```java
public class Conta {
   private double limite;
   private double saldo;
   public double getSaldo() {
      return saldo;
   }
   public void setSaldo(double x) {
      saldo = x;
   }
   public double getLimite() {
      return limite;
   }
   public void setLimite(double y) {
      limite = y;
   }
}
```

Questão

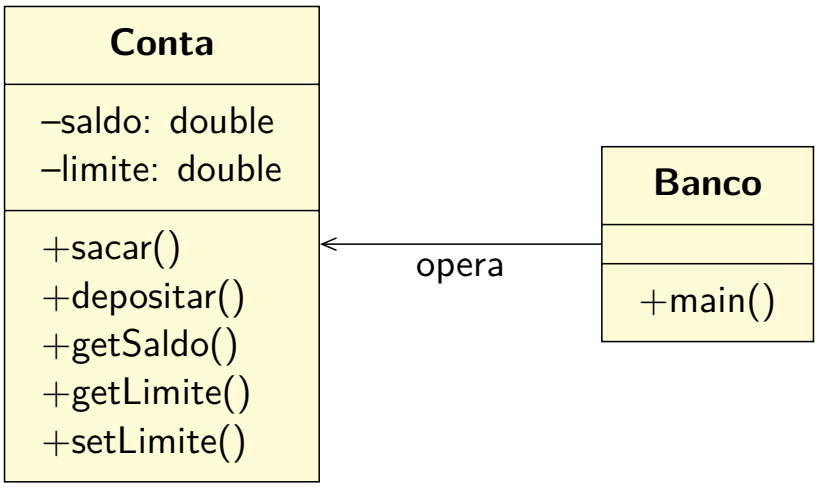
Esta classe permite alterar seus atributos como se fossem públicos!

Classe Banco.java (versão 0)

```java
public class Banco
{
   public static void main ( String args[] )
   {
      Conta c1 = new Conta();
      c1.setSaldo(  1000 );
      c1.setLimite( 1000 );
      double saldoAtual = c1.getSaldo();
      System.out.println( "Saldo atual é " + saldoAtual );
      double limiteConta = c1.getLimite();
      System.out.println( "Limite é " + limiteConta );
   }
}
```

Exercício – simulação de um banco (v1)

Construa um programa para simulação de um banco que possui apenas uma `conta`, com os atributos privados `saldo` e `limite`. Uma entidade `banco` é responsável pela criação da conta e sua operação. Ela pode sacar e depositar dinheiro da conta, visualizar seu saldo atual, assim como verificar e atualizar o limite.



Classe Conta.java (versão 1)

```
1  public class Conta {
2     private double saldo;
3     private double limite;
4     public void depositar( double x
       )
5     {
6        saldo = saldo + x;
7     }
8     public void sacar( double x )
9     {
10       saldo = saldo - x;
11    }
12    public double getSaldo()
13    {
14       return saldo;
15    }
16    public void setLimite( double x
       )
17    {
18       limite = x;
19    }
20
21    public double getLimite()
22    {
23       return limite;
24    }
25 }
```

Já colocamos nos métodos **regras de negócio**, que representam a lógica de negócio da sua aplicação.

Classe Banco.java (versão 1) I

```
1  public class Banco
2  {
3     public static void main ( String args[] )
4     {
5        Conta c1 = new Conta();
6        c1.setLimite( 300 );
7        c1.depositar( 500 );
8        c1.sacar( 200 );
9        System.out.println( "O saldo é " + c1.getSaldo() );
10    }
11 }
```


Classe Banco.java (versão 1) II

E se sacarmos mais que o disponível?

Classe Conta.java (versão 1.1)

Reescrevemos o método `sacar()`:

```
8     public void sacar(double x) {
9        if ( saldo + limite >= x )
10          saldo = saldo - x;
11    }
```

Métodos permitem controlar os valores / atributos, evitando que qualquer objeto altere seu conteúdo sem observar regras.

Construtores I

Métodos construtores são utilizados para realizar *toda a inicialização necessária* a uma nova instância da classe;

- Diferente de outros métodos, um método construtor não pode ser chamado diretamente:
  Um construtor é invocado pelo operador new quando um novo objeto é criado;
- Determina como um objeto é inicializado quando ele é criado;
- **Vantagens**: não precisa criar métodos *get*/*set* para cada um dos atributos privados da classe (reforçando o encapsulamento), tampouco enviar mensagens de atribuição de valor após a criação do objeto.

A declaração de um método construtor é semelhante a qualquer outro método, com as seguintes particularidades:

- O nome do construtor **deve ser o mesmo da classe**;
- Um construtor não possui um tipo de retorno – sempre `void`.


```
1  class Veiculo
2  {
3     private String marca;
4     private String placa;
5     private int kilometragem;
6     public Veiculo( String m, String p, int k )
7     {
8        marca = m;
9        placa = p;
10       kilometragem = k;
11    }
12    public String getPlaca()
13    {
14       return placa;
15    }
16    public String getMarca()
17    {
18       return marca;
19    }
```

Classe Veiculo.java II

```
20     public int getKilometragem()
21     {
22        return kilometragem;
23     }
24     public void setKilometragem(int k)
25     {
26        kilometragem = k;
27     }
28 }
```


Classe AcessaCarro.java

```
1  class AcessaCarro
2  {
3     public static void main(String args[])
4     {
5        Veiculo meuCarro = new Veiculo("Escort","XYZ-3456",60000);
6        String marca;
7        int kilometragem;
8        marca = meuCarro.getMarca();
9        System.out.println( marca );
10       kilometragem = meuCarro.getKilometragem();
11       System.out.println( kilometragem );
12       meuCarro.setKilometragem( 100000 );
13       System.out.println( kilometragem );
14    }
15 }
```

Referências

- HORSTMANN, Cay S.; CORNELL, Gary. *Core Java 2: Vol.1 – Fundamentos*, Alta Books, SUN Mircosystems Press, 7a. Edição, 2005.
- DEITEL, H. M.; DEITEL, P. J. *JAVA – Como Programar*, Pearson Prentice-Hall, 6a. Edição, 2005.
- https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html