


Laboratório de Engenharia de Software

# INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho  
[ivan@inf.puc-rio.br](mailto:ivan@inf.puc-rio.br)

## Programa – Capítulo 6




Laboratório de Engenharia de Software

- Arrays
- Arrays de Arrays
- Object Wrappers
- Métodos `get()` e `set()`
- Exercício

© LES/PUC-Rio 2

Programa – Capítulo 6




Laboratório de Engenharia de Software

- **Arrays**
- Arrays de Arrays
- Object Wrappers
- Métodos `get()` e `set()`
- Exercício

© LES/PUC-Rio

3

Arrays



Laboratório de Engenharia de Software


- Em Java, os **arrays** são objetos;
- Por isso, uma variável do tipo **array** é, na verdade, uma referência para um objeto;
- Assim sendo, a expressão `int qtd[]` não aloca nenhuma área de memória para o **array**; ela apenas define uma referência para um **array** de inteiros;
- Note, também, que nenhuma referência ao tamanho do array é feita na declaração anterior;
- Apenas na criação do **array** é que se aloca espaço em memória e que seu tamanho é definido.

© LES/PUC-Rio

4

Laboratório de Engenharia de Software

## Por que arrays são objetos?




- Porque os tipos **array** são referências, da mesma forma que os tipos objeto;
- Porque os **arrays** são alocados com o operador `new`, do mesmo modo que os objetos;
- Porque os **arrays** são alocados na área de memória dinâmica (heap), e não na pilha, da mesma forma que os objetos;
- Porque `Object` é a classe ancestral de todos os **arrays**. Logo, pode-se aplicar métodos como `toString()` sobre **arrays**.

© LES/PUC-Rio

5

Laboratório de Engenharia de Software

## Por outro lado...



- Não se pode estender (herança) um **array** da mesma forma que se faz com as classes regulares;
- Os **arrays** têm uma sintaxe diferente da sintaxe das classes regulares;
- Não se pode definir métodos adicionais para os **arrays**;
- Deve-se pensar nos **arrays** como objetos especiais, que possuem algumas característica em comum com os objetos regulares.

© LES/PUC-Rio

6

## Criação de arrays



- A declaração `int qtd []` informa que `qtd` é uma referência para um **array** de inteiros de qualquer tamanho;
- Dessa forma, `qtd` pode referenciar um **array** já existente ou um **array** criado por meio do operador `new`;
- Um **array** deve ser explicitamente criado por meio de uma expressão de criação;
- Após ser criado, não se pode alterar o tamanho de um **array**;
- Pode-se, entretanto, criar um **array** maior e copiar os elementos do primeiro **array** para o recém criado.

© LES/PUC-Rio

7

## Cópia de array – Método `System.arraycopy( )`



```
public class Ex {
    public static void main(String[] args) {

        int qtd[]=new int[]{1,2,3,4,5};
        int num[];

        num=new int[100];

        System.arraycopy(qtd,    //origem
                        0,      //posição na origem
                        num,    //destino
                        0,      //posição no destino
                        5);     //quantidade de elementos

        for(int i=0;i<qtd.length;i++)
            System.out.println(Integer.toString(num[i]));
    }
}
```

© LES/PUC-Rio

8

## Cópia de array – Método `Object.clone()`



```
public class Ex {
    public static void main(String[] args) {

        int qtd[]=new int[]{1,2,3,4,5};
        int num[]=(int[]) qtd.clone();

        for(int i=0;i<num.length;i++)
            System.out.println(Integer.toString(num[i]));
    }
}
```

- Enquanto `arraycopy()` copia elementos para um **array** já existente, `clone()` cria um novo **array**;
- Como `clone()` retorna um `Object`, deve-se fazer uma conversão explícita (`(int [])`).

© LES/PUC-Rio

9

## Inicialização de uma array (1)



- Um **array** pode ser criado e inicializado na sua declaração;

```
byte b[]={0,1,2,3,4};
String dia={"seg","ter","qua","qui","sex","sab","dom"};
```

- Ele é implicitamente criado quando a expressão de inicialização é avaliada;
- Não é possível, entretanto, usar esse procedimento após uma variável de **array** ter sido definida. Isto é, não se pode usá-lo, por exemplo, em um comando de atribuição.

```
// erro: constantes de array podem ser usadas apenas
// em inicializadores
dia={"seg","ter","qua","qui","sex","sab","dom"};
```

© LES/PUC-Rio

10

## Inicialização de uma array (2)



- Para inicializar um **array** na sua criação pode-se, também, usar uma expressão de criação:

```
String dia[];
dia=new String[]{"seg","ter","qua","qui","sex","sab","dom"};
```

- Pode-se criar objetos na própria expressão de inicialização:

```
UmaClasse vet[]=new UmaClasse[]{new UmaClasse(),
                                   new UmaClasse(3,4),
                                   null};
```

© LES/PUC-Rio

11

## Validação do índice



- O índice de um **array** é sempre validado em tempo de execução;
- Se um indexador tenta acessar um elemento fora dos limites do **array** uma exceção é levantada;
- Para evitar a ocorrência de exceções deve-se sempre checar o tamanho de um **array**;
- A quantidade de elementos de um **array** está disponível na variável de instância `length`, definida na classe do **array**.

```
vet.length;    // contém o tamanho do array vet
```

© LES/PUC-Rio

12

## Compatibilidade entre arrays (1)



- Uma referência para um **array** de uma certa classe pode referenciar, em tempo de execução, **arrays** de uma descendente da classe em questão:

```
public static void main(String[] args) {
    Object vet[]=new Object[3];

    String dia[]=new String[]{"seg","ter","qua",
                              "qui","sex","sab","dom"};

    vet=dia;

    for(int i=0;i<vet.length;i++)
        System.out.println((String)vet[i]);
}
```

© LES/PUC-Rio

13

## Compatibilidade entre arrays (2)



- Isso se aplica, obviamente, à passagem de parâmetros:


```
public static void main(String[] args) {
    String dia[]=new String[]{"seg","ter","qua","qui",
                              "sex","sab","dom"};

    umMetodo(dia);
}

public static void umMetodo(Object vet[]) {
    for(int i=0;i<vet.length;i++)
        System.out.println((String)vet[i]);
}
```

© LES/PUC-Rio


14

**Programa – Capítulo 6**


Laboratório de Engenharia de Software

- Arrays
- Arrays de Arrays**
- Object Wrappers
- Métodos `get()` e `set()`
- Exercício

© LES/PUC-Rio
15

**Arrays de arrays**


Laboratório de Engenharia de Software

- A linguagem Java não possui **arrays** multidimensionais;
- No lugar deles a linguagem provê a possibilidade de se criar **arrays** de **arrays**:

```
Object mat[][];
```

- A declaração acima informa que `mat` é uma referência para um **array** em que cada elemento é um **array** de `Object`;
- A criação e a inicialização de cada **array** deve ser feita individualmente:

```
Object mat[][];

mat=new UmaClasse[20][3];      // um array[20] de array[3]
mat[i]=new UmaClasse[15];     // um array[15]
mat[i][j]=new UmaClasse();    // um objeto individual
```

© LES/PUC-Rio
16



## Array triangular de arrays (1)



- Os **arrays** dos níveis inferiores não precisam ter o mesmo tamanho;
- Os exemplos a seguir exibem algumas alternativas para criar e preencher um **array** de **arrays** triangular:

```
// uso de várias expressões de criação

int mat[][]=new int[][]{new int[]{0},
                        new int[]{0,1},
                        new int[]{0,1,2},
                        new int[]{0,1,2,3}};
```

© LES/PUC-Rio

17

## Array triangular de arrays (2)



```
// uso de vários inicializadores

int mat[][]=new int[][]{{0},
                        {0,1},
                        {0,1,2},
                        {0,1,2,3}};


// inicialização individual de cada array
// por meio de várias expressões de criação

int mat[][]=new int[4][];
mat[0]=new int[]{0};
mat[1]=new int[]{0,1};
mat[2]=new int[]{0,1,2};
mat[3]=new int[]{0,1,2,3};
```

© LES/PUC-Rio

18

## Uma nova versão do comando `for` (1)




Laboratório de Engenharia de Software

- A versão JDK 1.5 introduziu uma nova forma do comando `for`, com o objetivo de iterar sobre coleções;
- A nova versão do `for` utiliza três nomes:
  - O tipo dos elementos;
  - O nome da variável que irá receber os sucessivos elementos da coleção;
  - O nome da coleção que será percorrida.

© LES/PUC-Rio

19

## Uma nova versão do comando `for` (2)



Laboratório de Engenharia de Software

- O exemplo abaixo lista todos os elementos do **array** triangular de **arrays** do exemplo anterior:

```

for(int[] v: mat) {
    for(int i: v)
        System.out.print(i+" ");
    System.out.println();
}
```

© LES/PUC-Rio

20

## Arrays e tipos enumerados



- Os **arrays** também podem receber valores de um enumerado.

```
public enum Estacoes {
    verao,outono,inverno,primavera;
}

Estacoes est[]=new Estacoes[]{Estacoes.primavera,Estacoes.outono,
    Estacoes.verao,Estacoes.inverno};

for(Estacoes e: est)
    System.out.print(e+" ");
```

© LES/PUC-Rio

21

## O tipo array



- Quando os colchetes usados na declaração de **arrays** aparecem logo após o tipo dos seus elementos, eles passam a fazer parte do tipo, e se aplicam a todas as variáveis da declaração;
- No exemplo a seguir, **j** é um **array** de **short** e **i** um **array** de **arrays** de **short**:

```
short[] j,i[];
```

- No próximo exemplo, **a** é um **array** de **int** e **b** um **int**:

```
int a[],b;
```

© LES/PUC-Rio

22

## Uma função pode retornar um array



- Ao contrário de C/C++, um método Java pode retornar um **array** – na verdade uma referência para um **array**:

```
public static void main(String[] args) {
    int[] s;
    s=umMetodo(5);
    for(int x: s)
        System.out.print(x+" ");
}

public static int[] umMetodo(int tam) {
    if(tam<1)
        return null;
    int[] v=new int[tam];
    for(int i=0;i<tam;i++)
        v[i]=i;
    return v;
}
```

© LES/PUC-Rio

23

## Programa – Capítulo 6



- Arrays
- Arrays de Arrays
- **Object Wrappers**
- Métodos get() e set()
- Exercício


© LES/PUC-Rio

24

Laboratório de Engenharia de Software

## Object wrappers

- Cada um dos oito tipos primitivos de Java possui uma classe correspondente, definida na biblioteca de classes de Java. Elas são conhecidas como object wrappers e servem a vários propósitos.
  - Elas são um meio conveniente para conter constantes, como, por exemplo, o maior e o menor valor que um determinado tipo primitivo pode armazenar;
  - Elas possuem métodos para a conversão, em ambos os sentidos, de valores de um tipo de/para **Strings**;
  - Alguns estruturas de dados existentes na biblioteca de Java operam apenas sobre objetos (subclasses de `Object`). Desse modo, guardar valores primitivos em objetos é uma boas saída para usar tais estruturas com valores de tipos primitivos.




© LES/PUC-Rio
25

Laboratório de Engenharia de Software

## Object wrappers dos tipos primitivos


Tipo Primitivo	Classe <i>Wrapper</i> Correspondente
boolean	java.lang.Boolean
char	java.lang.Character
int	java.lang.Integer
long	java.lang.Long
byte	java.lang.Byte
short	java.lang.Short
double	java.lang.Double
float	java.lang.Float



© LES/PUC-Rio
26

Laboratório de Engenharia de Software

## Exemplos de uso dos wrappers



```

int i=15;
Integer myInt=new Integer.valueOf(i);

//obtem uma versão do inteiro para impressão
String s=myInt.toString();

//obtem uma versão em hexa do inteiro para impressão
s=myInt.toHexString(255); //imprime "ff"

//converte uma string em um inteiro
i=Integer.parseInt("2047");


//gera um objeto Integer a partir de um inteiro
myInt=Integer.valueOf(18);

```

© LES/PUC-Rio
27

Laboratório de Engenharia de Software

## Autoboxing e unboxing



- Autoboxing** é uma novidade que veio com o JDK 1.5;
- Ele reconhece a relação muito próxima entre variáveis de tipos primitivos e objetos dos seus wrappers;
- Autoboxing** significa que podemos converter de uma forma de representação para a outra sem que seja necessário fazê-lo explicitamente; o compilador faz o trabalho necessário.

```

Double dObj1=20.0; //boxing
Double dObj2=10.0; //boxing
double result=dObj1+dObj2; //unboxing
Double dObj3=dObj1+23.0; //boxing e unboxing

```

© LES/PUC-Rio
28

Programa – Capítulo 6




Laboratório de Engenharia de Software

- Arrays
- Arrays de Arrays
- Object Wrappers
- **Métodos get() e set()**
- Exercício

© LES/PUC-Rio

29

Métodos get() e set() (1)



Laboratório de Engenharia de Software

- Frequentemente é necessário recuperar o valor de uma variável de instância de um objeto ou alterar o seu valor;
- Isso pode ser feito por meio de métodos get() e set():

```
public class umaClasse {  
  
    private int matricula;  
    private String nome;  
    public int getMatric(){  
        return matricula;  
    }  
    public String getNome(){  
        return nome;  
    }  
    public void setMatric(int m){  
        matricula=m;  
    }  
    public void setNome(String n){  
        nome=n;  
    }  
}
```

© LES/PUC-Rio

30

## Métodos get() e set() (2)



- Versões alternativas desses métodos são apresentadas abaixo:

```
public void set(int var, Object o) {
    switch(var) {
        case 1: matricula=(Integer)o;
                break;
        case 2: nome=(String)o;
                break;
    }
}

public Object get(int var) {
    switch(var) {
        case 1: return matricula;
        case 2: return nome;
    }
    return null;
}
```

© LES/PUC-Rio

31

## Métodos get() e set() (3)



- Eles poderiam ser usados da seguinte maneira:

```
public static void main(String[] args) {
    umaClasse p=new umaClasse();


    p.set(1,12345);
    p.set(2,"Manuel Joaquim");

    System.out.println("matricula: "+(Integer)p.get(1)+
        " nome: "+(String)p.get(2));
}
```

© LES/PUC-Rio

32



Laboratório de Engenharia de Software	<b>Programa – Capítulo 6</b>	
	<ul style="list-style-type: none"><li>• Arrays</li><li>• Arrays de Arrays</li><li>• Object Wrappers</li><li>• Métodos <code>get()</code> e <code>set()</code></li><li>• <b>Exercício – Vendas em um Supermercado</b></li></ul>	
© LES/PUC-Rio		33