

# Array em Java

---

# Array em Java

## ■ Definição

- É uma estrutura de dados que permite o armazenamento de um conjunto de variáveis (elementos) de um mesmo tipo (ou referências para instâncias de uma mesma classe) e que são acessadas individualmente por um índice

```
char[] letrasAlfabeto = {'a', 'b', 'c', 'd', 'e', 'f', 'g',  
                          'h', 'i', 'j', 'k', 'l',          'm', 'n', 'o', 'p', 'q',  
                          'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'};
```

```
letrasAlfabeto[0] → a
```

```
letrasAlfabeto[1] → b
```

```
...
```

```
letrasAlfabeto[25] → z
```

# Array em Java

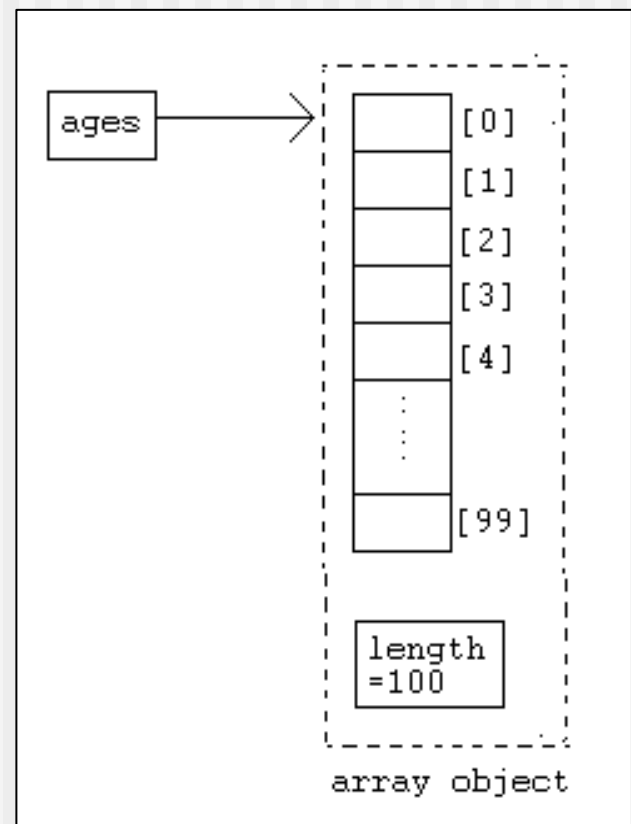
## ■ Usos mais comuns

- Armazenar grandes quantidades de dados de um mesmo tipo ou classe
  - 1440 amostras do tipo `double` da temperatura de um paciente internado, medida a cada minuto, durante um dia
  - As 38 instâncias da classe `Aluno`, componentes de uma turma
- Utilizar variáveis individuais e com nomes distintos em casos como estes é extremamente trabalhoso e sujeito a erros
  - O uso de arrays permite usar um único nome para denotar um conjunto homogêneo de variáveis, que são acessadas individualmente através de índices

# Array em Java

- Armazena múltiplos itens de um mesmo tipo de dado em um bloco contínuo de memória, dividindo-o em certa quantidade de posições

```
// declaração  
int [] ages;  
  
int ages[];  
  
// construção  
ages = new int[100];  
  
// declarar e construir  
int ages[] = new int[100];
```



# Array em Java

```
boolean results[] = { true, false, true, false };
```

```
double []grades = {100, 90, 80, 75};
```

```
String days[] =  
    { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };
```

```
// atribuir 10 ao primeiro elemento do array  
ages[0] = 10;
```

```
// imprimir o último elemento do array  
System.out.print(ages[99]);
```

# Tipos de Array

---

## ■ Unidimensionais

- Elementos são acessados por um único índice

- `alunos[0]`
- `alunos[1]`

## ■ Bidimensional

- Elementos são acessados por um par de índices, especificando linha e coluna de uma matriz

- `turmaAluno[0][1]`
- `turmaAluno[1][10]`

## ■ Multidimensional

- Elementos são acessados por um número arbitrário de índices

# Declaração e alocação de arrays unidimensionais

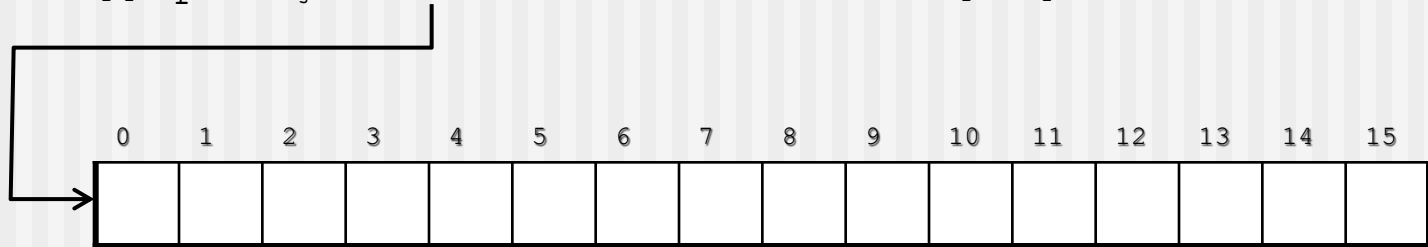
- Java adota a notação de colchetes para declarar arrays
  - Uma variável de um tipo (classe) específico seguido de um par de colchetes declara uma referência a um array de elementos desse tipo
    - `int[] posiçõesDeMemória;`
    - `char letrasAlfabeto[];`
    - `double[] medidaTemperatura;`
  - A declaração acima define apenas referências para arrays, mas não cria nenhuma variável do tipo array na memória
    - É preciso criar (alocar) o array na memória com um número predeterminado de posições
    - Opcionalmente, pode-se inicializá-lo na cláusula de declaração

# Declaração e alocação de arrays unidimensionais

## ■ Alocação de arrays

- Cria a variável correspondente ao array na memória com para um número de elementos predeterminado
  - Comando realizado pela palavra-chave **new**, seguida pelo tipo do dado do array e do número de elementos a alocar, entre colchetes

- `int[] posiçõesDeMemória = new int[16];`



- `int tamanho = 32768;`
- `posiçõesDeMemória = new int[tamanho];`
- `double[] medidasTemperatura = new int[24 * 60 * 60];`



# Declaração e alocação de arrays unidimensionais

## ■ Inicialização

- Pode ser realizada na cláusula de declaração, para pequeno número de elementos sabidos *a priori*
  - `int[] ultimoDiaMes = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};`
  - `String[] universidades = {"UEPB", "UFCG", "UEPB", "CEFET", "UNIFE"};`
- Pode ser realizada após a alocação/criação do array
  - `String[] universidades = new String[5];`  
`universidades[0] = "UEPB";`  
`universidades[1] = "UFCG";`  
`universidades[2] = "UEPB";`  
`universidades[3] = "CEFET";`  
`universidades[4] = "UNIFE";`

0	1	2	3	4
UEPB	UFCG	UEPB	CEFET	UNIFE

# Acesso a elementos

- O índice para acesso a um elemento deverá ser um valor do tipo inteiro entre 0 e o tamanho do array subtraído de 1
  - `nome_do_array[indice]`
- Qualquer tentativa de acesso com valor de índice fora dessa faixa resulta em erro em tempo de execução, com a interrupção da execução do programa, indicado pela exceção
  - `ArrayIndexOutOfBoundsException`
- O tamanho –quantidade de elementos– de um array pode ser lido através da variável `length`, existente em todo array
  - `universidades.length` → 5
  - `ultimoDiaMes.length` → 12

# Arrays de instâncias de classes

- Array no qual os elementos são instâncias de classes
  - Cada elemento deverá ser criado para inicializar o array

```
CartaoDeCredito[] void defineCartoes(int nCartoes) {  
    System.out.println("Configuracao do sistema de cartoes 'SysCard'.");  
    System.out.println("Entre com a bandeira de "+nCartoes+" cartoes ...");  
    CartaoDeCredito[] cartoes = new CartaoDeCredito[nCartoes];  
    Scanner entrada = new Scanner (System.in);  
    for (int n = 0; n < cartoes.length; n++) {  
        cartoes[n] = new CartaoDeCredito();  
        System.out.print("Entre com a bandeira do cartão '"+n+"': ");  
        String bandeira = entrada.nextLine();  
        cartoes[n].defineBandeira(bandeira);  
        System.out.println();  
    }  
    return cartoes;  
}
```

# Arrays multidimensionais

- Elementos são acessados por um número arbitrário de índices
  - Matrizes matemáticas

```
double[][] matriz = new double[5][10];
```

```
...
```

```
public double maiorValor() {  
    double maiorAtéAgora = matriz[0][0];  
    for (int lin= 0; lin < matriz.length; lin++)  
        for (int col = 0; col < matriz[lin].length; col++)  
            if (matriz[lin][col] > maiorAtéAgora)  
                maiorAtéAgora = matriz[lin][col];  
    return maiorAtéAgora;  
}
```

# Arrays multidimensionais irregulares

- Não têm o mesmo número de elementos em cada dimensão
- Um array em Java nada mais é do que um array cujos elementos são arrays, com exceção do elemento mais interno

```
double[] [] matriz = new double[5][];  
matriz[0] = new double[2];  
matriz[1] = new double[5];  
matriz[2] = new double[3];  
...  
public double maiorValor() {  
    double maiorAtéAgora = matriz[0][0];  
    for (int lin= 0; lin < matriz.length; lin++)  
        for (int col = 0; col < matriz[lin].length; col++)  
            if (matriz[lin][col] > maiorAtéAgora)  
                maiorAtéAgora = matriz[lin][col];  
    return maiorAtéAgora;  
}
```

# Trabalhando com Strings

---

# A classe String

- Usada para representação e manipulação seqüências de caracteres
  - Inclui métodos úteis para processamento de textos
    - Tamanho do string, i. e., a quantidade de caracteres armazenados
    - Caractere existente numa dada posição do string
      - `"Java".length()` → 4
      - `"Java".charAt(0)` → `'J'`
      - `"Java".charAt(2)` → `'v'`
      - `"Java".charAt(4)` → `java.lang.StringIndexOutOfBoundsException`
      - `"Java".toArray()` → `{'J','a','v','a'}`
  - Construção de instâncias da classe `String`
    - `String curso = "Ciência da Computação";`
    - `String curso = new String("Ciência da Computação");`
    - `char[] caracteres = {'C','i','ê','n','c','i','a',' ','d','a',' ','C','o','m','p','u','t','a','ç','ã','o'};`
    - `String curso = new String(caracteres);`

# A classe String

- Métodos para comparação de string
  - Strings não devem ser comparados com ==  
Por que?
    - String curso = "Computação";
    - "Computação".equals(curso) ➡ true
    - "computação".equals(curso) ➡ false
    - curso.equals("Computação") ➡ true
    - curso.equals("Comuptação") ➡ false
    - curso.equals(curso) ➡ true
    - "Computação".equalsIgnoreCase(curso) ➡ true
    - "cOmPutação".equalsIgnoreCase(curso) ➡ true
    - curso.equalsIgnoreCase("cOmPutação") ➡ true
    - curso.equalsIgnoreCase("cOmPutação") ➡ false
    - curso.equalsIgnoreCase(curso) ➡ true



# A classe String

- Métodos para comparação de string
  - Considerando apenas o início ou o fim do string
    - `String curso = "Computação";`
    - `curso.startsWith("Comp") ➡ true`
    - `curso.startsWith("comp") ➡ false`
    - `curso.startsWith("Computação") ➡ true`
    - `curso.startsWith("Computaçãoo") ➡ false`
    - `curso.startsWith("") ➡ true`
    - `curso.endsWith("ação") ➡ true`
    - `curso.endsWith("Ação") ➡ false`
    - `curso.endsWith("Computação") ➡ true`
    - `curso.endsWith("Computaçãoo") ➡ false`
    - `curso.endsWith("") ➡ true`

# A classe String

- Métodos para comparação de string
  - Métodos para procura de substrings (Verificar se um string contém outro string)
    - `String curso = "Computação";`
    - `curso.indexOf("ação") ➡ 6`
    - `curso.indexOf("o") ➡ 1`
    - `curso.indexOf("uta") ➡ 4`
    - `curso.indexOf("cação") ➡ -1`
    - `curso.indexOf("") ➡ 0`
    - `curso.indexOf("Comp") ➡ 0`

# A classe String

## ■ Métodos para transformação de strings

### ■ String em Java são imutáveis

- A “modificação” de um string é, de fato, a criação de um

Os métodos de processamento de string podem ser combinados

`curso.toUpperCase().trim().substring(12)` ➔ “OMPUTAÇÃO”

- `curso.substring(12)` ➔ Computação
- `curso.substring(12, 16)` ➔ “Comp”
- `curso.substring(16, 12)` ➔  
`java.lang.StringIndexOutOfBoundsException`
- `curso.substring(16, curso.length())` ➔  
`java.lang.StringIndexOutOfBoundsException`

# A classe String

- Métodos para conversão de tipos com strings
  - Converte valores de tipos nativos para strings e vice-versa
    - Tipos nativos para String
      - `String.valueOf(10)` ➡ `"10"`
      - `String.valueOf(15.4)` ➡ `"15.4"`
      - `String.valueOf('v')` ➡ `"v"`
  - `char[] carac = {'J', 'a', 'v', 'a'};`  
`String.valueOf(carac)` ➡ `"Java"`
    - String para tipos Nativos
      - `Integer.parseInt("10")` ➡ `10`
      - `Float.parseFloat("3.2383")` ➡ `3.2383`
      - `Integer.parseInt("10.33")` ➡ `java.lang.NumberFormatException`
      - `Float.parseFloat("3.2a383")` ➡ `java.lang.NumberFormatException`

# A classe StringBuffer

- Permite criação e manipulação de strings modificáveis
  - Lembre que instância da classe `String` são imutáveis
- A criação de instâncias de `StringBuffer` é mais custosa que a criação de instâncias da classe `String`, porém a manipulação é bem mais eficiente
- Instâncias da classe `StringBuffer` têm duas variáveis de instância do tipo inteiro associadas
  - Uma para indicar o *comprimento* do string, isto é, a quantidade de caracteres armazenados num dado momento
  - Outra para indicar a capacidade, que corresponde ao número máximo de caracteres que podem ser armazenados pela instância num dado momento
    - A capacidade pode ser expandida automática ou manualmente

# A classe StringBuffer

- `StringBuffer str = new StringBuffer(50);`
- `StringBuffer str = new StringBuffer("Java");`
- `str.append(' '); str.append("Language ");`
- `str.append(10);`
- `System.out.println(str);` → Java Language 10
- `str.insert(14, "é ");`
- `System.out.println(str);` → Java Language é 10
- `str.delete(5, 13);`
- `System.out.println(str);` → Java é 10
- `str.reverse();`
- `System.out.println(str);` → 01 é avaJ

# Pacotes

---

# Pacotes (*package*)

---

- Um **package** é um agrupamento de classes e interfaces que estão logicamente relacionadas.
- Em Java é um conjunto de classes e interfaces que estão dentro de um mesmo diretório de arquivos.
- O Java utiliza os packages para:
  - garantir a unicidade do nome da classe, ou seja, não permitir que existam nomes de classes iguais em um mesmo pacote;
  - e para realizar controles de acesso;
- Membros de uma classe sem a especificação do controle de acesso são considerados membros que podem ser acessados pelas classes do mesmo package.



- Para colocar uma classe em um determinado pacote, deve-se colocar a palavra reservada `package` na primeira linha de código do arquivo fonte (\*.java).

```
package java.lang;
```

- Para utilizar classes definidas em outros pacotes deve-se importar as classes através da palavra chave `import`.

```
import java.lang.String;
```

ou ainda

```
import java.lang.*;
```

Importa apenas a classe String

Importa todas as classes do package lang

# Principais Pacotes do Java

---

- `java.applet` - Classes para criação de Applets.
- `java.awt` - Classes para criação de interfaces gráficas.
- `java.io` - Classes para gerenciamento de Entradas e Saídas.
- `java.lang` - Classes Básicas da linguagem (incluídas automaticamente, não necessita o uso do `import`).
- `java.net` - Classes para trabalho em rede.
- `java.util` - Classes de utilitários (Data, Dicionário, Pilha, Vetor, Random, etc.)
- `java.sql` - Classes para manipulação de Banco de Dados

# Exceções

---

# Tratamento de Exceções

- Uma exceção é um evento que ocorre durante a execução do programa, que foge à normalidade do fluxo de instruções.
- Linguagens que tratam adequadamente as exceções têm a vantagem de separar a lógica da aplicação do tratamento de erros.
- Exceções são ocorrências tratáveis pelo desenvolvedor da aplicação.
- Quando uma exceção acontece pode-se:
  - emitir exceção para o método chamador, através do comando `throws`;
  - capturar e tratar através dos comandos `try`, `catch` e `finally`.

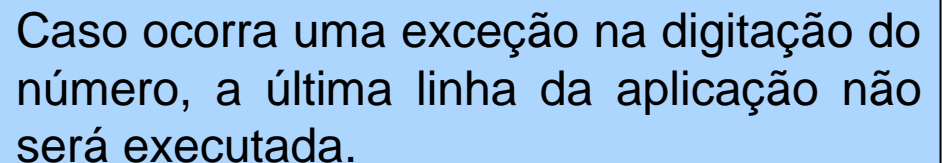
# Emitindo Exceções

- Os métodos podem emitir exceções de forma explícita. Um método pode declarar que é capaz de emitir uma exceção, usando para isso a cláusula `throws`.
- Ao emitir uma exceção está se considerando que o método chamador irá tratar a exceção.
- Caso o método chamador não trate a exceção será apresentado um erro no console do computador e a aplicação pode parar de rodar.
- No próximo exemplo, caso seja digitado uma letra em lugar de um número, a máquina virtual Java emite um erro e para a execução da aplicação.

# Exemplo de Emissão de Exceções

```
import java.io.*;
class RepassandoExcecoes
{   public static void main(String args[])
    throws java.io.IOException
    {   float n=0;
        String line;
        BufferedReader in = new BufferedReader(new I
                                                nputStreamReader(System.in));

        System.out.println("Digite um número: ");
        line = in.readLine();
        n = Float.valueOf(line).floatValue();
        System.out.println("O número digitado foi: " + n);
    }
}
```



Caso ocorra uma exceção na digitação do número, a última linha da aplicação não será executada.

# Capturando Exceções

- Para capturar exceções coloque o trecho que pode originar a exceção num bloco **try**.
- Manipule cada exceção num bloco **catch**.
  - Podem existir vários catch para um único try, já que num único bloco try podem ocorrer tipos de exceções diferentes.
- Realize procedimentos obrigatórios e comuns num bloco **finally**.

```
try{  
    //origina a  
    //exceção  
}  
catch(exception1) {  
    //manipula exc1  
}  
catch(exception2) {  
    //manipula exc2  
}  
finally {  
    //proc.  
    //obrigatórios  
}
```

---

# TRABALHO AVALIATIVO

Para ser feito individualmente.

Deverá ser entregue (apresentada a execução) na aula em 07/11/2014



# Exercício Avaliativo: Jogo da Forca

1. Inicializa-se um string com palavra a ser adivinhada
  2. Cria-se um array de String, cada posição correspondendo a um caractere da palavra, inicializada com espaço, para indicar que nenhum caractere foi adivinhado.
  3. Cria-se um array de 26 posições booleanas ou String para indicar as letras que já foram utilizadas
  4. Esses arrays mostram para o usuário quais letras já foram adivinhadas e quais letras já foram utilizadas (
  5. O usuário entrar com uma letra, que é incluída no rol das letras usadas e marca-se a letra nas posições correspondentes da palavra a ser adivinhada, se pertinente
- Caso todas as letras da palavra tenha sido adivinhada, então o jogo é encerrado, caso contrário, volta-se para o passo 2