ORIENTAÇÃO A OBJETOS

AULA 4

Estruturas Homogêneas (Array e String) e Exceções

Vandor Roberto Vilardi Rissoli

APRESENTAÇÃO

- Estrutura de Dados Homogênea
- Classe Array
- Classes String
- Correção de leitura (*buffer*)
- Referências

Estrutura de Dados Homogênea

Uma estrutura de dados homogênea consiste em um conjunto de dados que são armazenados na memória do computador de forma organizada e mais eficiente.

Geralmente, estas estruturas são chamadas de **variáveis compostas homogêneas**, quando podem variar os dados que armazenam (como variáveis):

- composta - porque um único identificador é criado com capacidade de armazenar vários valores
- homogênea - porque todos os valores armazenados nesta estrutura serão de um mesmo tipo de dado ou objeto

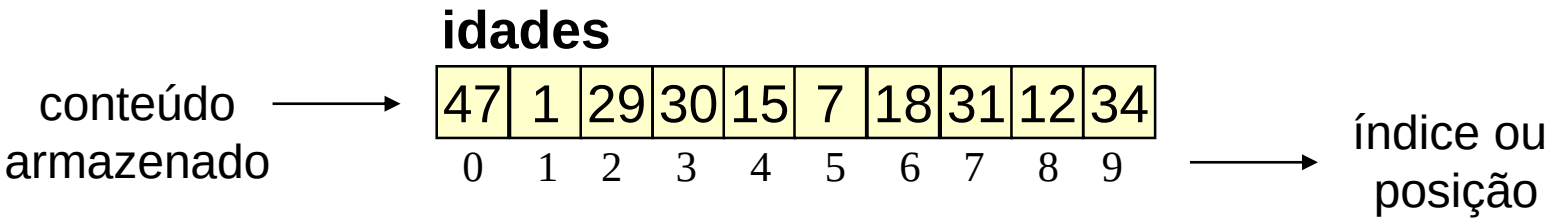
→ Esta estrutura pode variar em algumas dimensões para organizar seu armazenamento de dados, por exemplo:
•bidimensional – possui variação em linha e coluna;
•tridimensional – variação em linha, coluna e página...

Estrutura de Dados Homogênea

Principais Características

- Capacidade de armazenamento de vários valores de um mesmo tipo
- Um único identificador para toda estrutura
- Independência de cada valor armazenado de acordo com seu índice ou posição na estrutura
- O primeiro índice ou posição na estrutura sempre é zero (0)

Exemplo de estrutura de inteiros:



Estrutura de Dados Homogênea - Array

Em Java, o armazenamento de um conjunto de valores, que não são somente do tipo caracter (String), pode ser realizado através da classe Array.

Esta classe é utilizada para guardar vários valores de um mesmo tipo de dado, ou objeto, possuindo um único identificador e tornando mais eficiente a manipulação da memória alocada a todos seus elementos (alocação contínua).

Declaração de Array

```
int [ ] idades;
int idades[ ];
```

As duas formas acima declaram um Array de inteiros, onde os **colchetes** podem estar em qualquer uma das posições indicadas, sendo o identificador definido como **idades**.

Estrutura de Dados Homogênea - Array

Principais Características

- Consiste em uma estrutura de dados composta homogênea de tipos primitivos ou de outros objetos
- Seu primeiro elemento é armazenado na posição ou índice zero (0)
- Pode possuir uma (unidimensional) ou várias dimensões (bidimensional, tridimensional, etc.), como uma matriz, definidas por meio dos colchetes

```
int [][] idades;    // possíveis declarações bidimensionais
int idades[][];
int [] idades[];
```

- O tamanho do Array deve ser definido em sua criação, **não podendo ser alterado** (fixo) em tempo de execução (alterado dinamicamente)

Estrutura de Dados Homogênea - Array

Criação do Array em Memória

Após sua declaração, é necessário criar o Array na memória do computador, ou seja, alocar seu espaço.

```
int [] idades;   // declaração do vetor (Array) inteiro
idades = new int[10];   // cria o Array na memória (aloca)
```

identificador

Operador new
(aloca memória)

Tamanho do Array

Este processo de declaração e criação pode ser realizado de maneira mais concisa, como:

```
int [ ] idades = new int[10];    // mesma declaração anterior
```

Estrutura de Dados Homogênea - Array

Esta estrutura de dados pode ser criada e inicializada simultaneamente, sem o uso do operador **new**, por exemplo:

```
int [ ] idades = {9, 5, 20, 14, 37, 61, 18, 21, 91, 43, 51};
```

- Declara o Array idades e aloca seu espaço em memória (cria o objeto)
- O tamanho deste Array é exatamente a quantidade de elementos que foram inicializados, ou seja, dez (10)
- São necessárias as chaves (**{ }**) para inicialização dos elementos no Array
- A criação do Array sem atribuição de valores a cada uma de suas posições inseri os valores padrões Java para cada **tipo primitivo** e para objetos o valor inicial é **null**

Estrutura de Dados Homogênea - Array

O acesso ou a manipulação do conteúdo armazenado em uma posição do Array exige a identificação de seu índice (ou posição) para cada uma das dimensões definidas para esta estrutura de dados (Array).

```
idades[5] = 61;   // armazena elemento no Array
System.out.print("A sexta idade = " + idades[5]); // mostra
idades[5] = 10; // altera o valor do elemento no Array
```

Para declaração e criação de um Array de 10 elementos, o acesso e a manipulação de seus valores só podem acontecer nas posições de zero até o tamanho da declaração, menos um elemento:

```
int [ ] idades= new int[10];  // posição válida de 0 até (10 - 1)
```

Estrutura de Dados Homogênea - Array

Tamanho do Array

A obtenção do tamanho (número de elementos) do Array pode usar o atributo **length**, por exemplo:

```
quantidade = idades.length;   // idades teria tamanho 10
```

Um uso bem comum para este atributo é:

```
for (int aux=0; aux < idades.length; aux++)
```

onde a definição do tamanho das dimensões do Array pode ser realizada por meio de uma constante, facilitando a manutenção sobre seu programa.

```
final int QUANTIDADE = 10;  // constante para Array
int [ ] idades = new int[QUANTIDADE];    // cria Array
```

Estrutura de Dados Homogênea - Array

```
/** Síntese
 *   Objetivo: manipulação de Array de tipo
 *             primitivo caracter e objeto String
 *    Entrada: sem entrada
 *    Saída:   análise sobre os arrays manipulados
 *             (tamanho, elementos, concatenação)
 */
public class CaracteresString {
  public static void main(String[] args) {
   // Cria um Array de Caracteres
   char CaracterArray[]={'a','b','c','d','e','f'};
   System.out.println("Mostra Array de Caracteres = " +
                          String.valueOf(CaracterArray));
   System.out.println("\tQuantidade de Elementos = " +
                                 CaracterArray.length);
   System.out.println("\tApresenta suas 3 primeiras
      posições = "+ String.valueOf(CaracterArray,0,3));
   String caracteres = ""; // inicializa String
   for (int aux=0; aux < CaracterArray.length; aux++)
     caracteres=caracteres + CaracterArray[aux] + " ";
   System.out.println("Mostra o Array de Caracteres
                          concatenado = " + caracteres);
```

Estrutura de Dados Homogênea - Array

```
// continuação do exemplo anterior
  // Cria um Array de Strings
  String StringArray[] = {"Estudando","a","Linguagem",
                                                "Java"};
  String frase = ""; // inicializa o objeto String
  // Armazena Strings no Array frase (concatenações)
  System.out.print("\nMostra Array de String = ");
  for (int aux=0; aux < StringArray.length; aux++) {
     System.out.print(StringArray[aux]);
     frase = frase + StringArray[aux] + " ";
  }
  System.out.println("\n\tQuantidade de elementos = "+
                                      StringArray.length);
  System.out.println("\tMostra primeiro elemento = " +
                                        StringArray[0]);
  System.out.println("\tMostra último elemento   = " +
                     StringArray[StringArray.length-1]);
  System.out.println("Mostra Array concatenado em
                                       Frase = " + frase);
 }
}
```

Exercício de Fixação

1) Faça um programa que armazene a temperatura média de cada dia do mês de novembro em um vetor e apresente qual foi a maior e a menor temperatura deste mês. Indique também qual foi o dia, ou dias no caso de temperaturas iguais, em que estas temperaturas aconteceram.

Elabore os métodos adequados a programação orientada a objeto para solução completa deste problema, que apresentará os valores finais em um método único que mostrará todos os dados finais acionado pelo método principal.

Volte à Aula 3 e continue o estudo sobre **Classes**

Estrutura de Dados Homogênea - Array

Array de Referência

O Array de uma classe guarda a referência ao endereço de memória onde o objeto está armazenado.

```
Conta contasPessoais [ ];
contasPessoais = new Conta[100];
```

Em cada posição do Array contasPessoais existe o valor **null**, quando sua criação acontecer sem inicialização.

Cada uma destas posições armazenará uma referência ao objeto Conta, sendo possíveis 100. A instrução abaixo resultará em erro de execução, pois a primeira posição deste Array não está referenciando nenhuma Conta (não foi alocado nenhum espaço na memória para armezená-la).

```
System.out.print(contasPessoais[0].saldo);
```

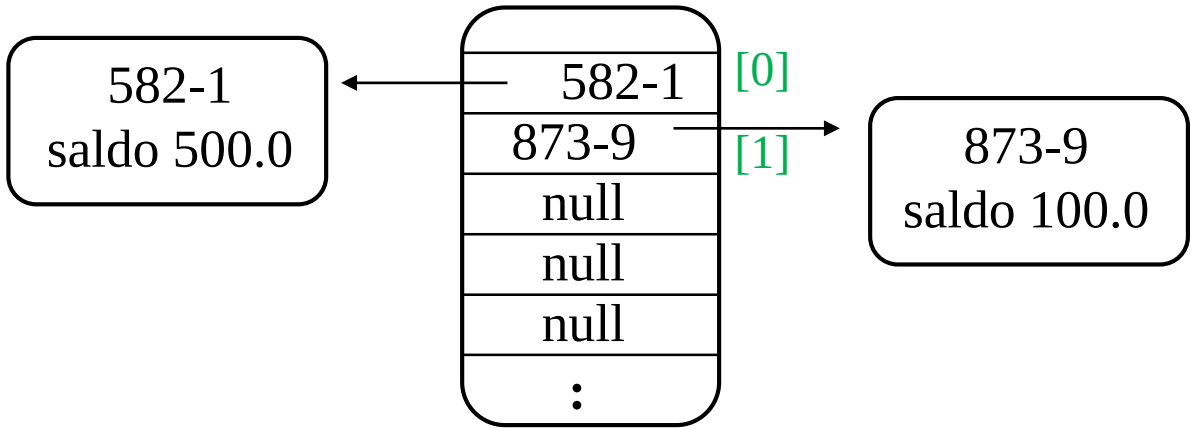
Estrutura de Dados Homogênea - Array

Após a criação deste tipo de estrutura elas também precisam ser populadas (atribuídos valores).

Exemplo:

```
Conta novaConta = new Conta();
novaConta.saldo = 500.0;
contasPessoais[0] = novaConta;
```

O Array de tipos de dados primitivos armazena os valores atribuídos a cada posição do Array, porém o Array de objetos guarda **somente a referência** ao estes objetos.



Estrutura de Dados Homogênea - Array

A utilização de Array de objetos contribui com a codificação mais organizada e eficiente na programação.

Exemplo:

```
for(int aux=0; aux < 100; aux++)
   contasPessoais.saldo = 0;  // inicializa todas as contas
```

No trecho de código acima todos os objetos terão seu saldo zerados. Caso não existisse este Array, mas 100 contas independentes, cada uma com seu identificador, seriam necessárias 100 linhas de código para zerar todas estas contas, pois todas elas estariam sendo armazenadas na memória, mas sem uma organização eficiente para o computador as localizar.

Estrutura de Dados Homogênea - Array

Array Bidimensional

A linguagem Java não possui estrutura de dados com várias dimensões, similar a outras linguagens que implementam matrizes.

No entanto, a mesma funcionalidade é alcançada por sua implementação de Array de arrays, sendo possível trabalhar com várias dimensões.

```
float notas [ ][ ] = new float [5][2];    // Array bidimensional
   :                          :
System.out.print("Nota1 = " + notas[0][0] + "\t");
System.out.print("Nota2 = " + notas[0][1] + "\n");
System.out.println("Média = " + (notas[0][0]+notas[0][1])/2);
```

⇒ Um Array bidimensional corresponde a um Array dentro de outro Array

Estrutura de Dados Homogênea - Array

Passagem de Array como Parâmetro

Os métodos podem receber valores como parâmetros e até retornarem um único valor como resultado. Isso também é possível usando Array, onde esta estrutura pode ser enviada como parâmetro ou mesmo uma única estrutura pode ser retornada por um método.

- Quando um método é acionado um Array é enviado como parâmetro, similar a qualquer outro parâmetro
- Este Array é manipulado normalmente com o identificador de seu parâmetro dentro do método
- Caso exista a necessidade de retornar o Array totalmente ele deve ser especificado no **return**
- Ás vezes um Array é necessário dentro de um método para apuração de um único valor que será retornado, ou mesmo para um operação que não necessite de returno (método sem retorno – procedimento)

Estrutura de Dados Homogênea - Array

```
/** Síntese
 *    Objetivo:  ordena valores inteiros positivos
 *    Entrada: valores inteiros positivos
 *    Saída:   valores em ordem crescente
 */
import java.util.Scanner;
public class PassagemArray {
   public static int[] ordenaArray(int nros[]) {
     int cont1,cont2,auxiliar;
     for(cont1=0;cont1<nros.length;cont1++)
       for(cont2=0;cont2<nros.length;cont2++)
         if(nros[cont1]<nros[cont2]) {
           auxiliar=nros[cont2]; // processo de trocas
           nros[cont2]=nros[cont1];
           nros[cont1]=auxiliar;
         }
     return nros;
   }

   public static void mostraArray(int nros[]) {
      for(int aux=0; aux < nros.length;aux++)
         System.out.print(nros[aux] + " ");
   }
```

Estrutura de Dados Homogênea - Array

```
// continuação do exemplo anterior
 public static void main(String[] args) {
   final int MAX = 5;
   Scanner ler = new Scanner(System.in);
   // Criando um Array de MAX números
    int numeros[] = new int[MAX];
    System.out.println("Informe números positivos");
    for(int aux=0; aux < MAX; aux++) {
      System.out.print("Número[" + (aux+1) + "] = ");
      numeros[aux] = ler.nextInt();
      if (numeros[aux]<0) {
         System.out.println("Número inválido. Digite
                                              novamente: ");
         aux--;
      }
    }
    numeros = ordenaArray(numeros);   // método ordenar
    System.out.println("\nNúmeros em Ordem Crescente");
    mostraArray(numeros); // método sem retorno
 }
}
```

Exercícios de Fixação

2) Crie uma classe que tenha a capacidade de armazenar o nome e a quantidade de Campeonatos Brasileiros de futebol que este time já ganhou. Em outra classe executável deve ser solicitado até 10 times ao usuário para cadastro em um Array de times, onde quando este Array de times estiver preenchido totalmente ou o usuário quiser encerrar seu programa, deverá ser apresentado o nome de todos os times cadastrados, sendo um em cada linha, e somente ao final dos nomes será mostrado o total de títulos deste campeonato que estes times somados já conseguiram ganhar. Sua solução deverá evitar a criação de métodos estáticos, tendo seu programa pelo menos 3 classes e no máximo 2 métodos estáticos.

Exercícios de Fixação

3) Uma espetacular loja de biquíni tem um único vendedor e comercializa 15 tipos fantásticos de biquínis. Seu salário mensal é R$250,00, acrescido de 10% de toda sua venda no mês. O valor unitário de cada um destes biquínis deve ser solicitado ao usuário (***precos***), enquanto que a quantidade vendida de cada peça ficará em outro vetor (***vendas***). Elabore um programa que cadastre os valores nestes vetores para um mês e apresente um relatório contendo: quantidade e valor total de cada biquíni vendido, além do total geral de biquínis e do valor das vendas. Mostre também o valor da comissão do vendedor e o total de seu salário neste mês analisado.

Tratamento de Exceções

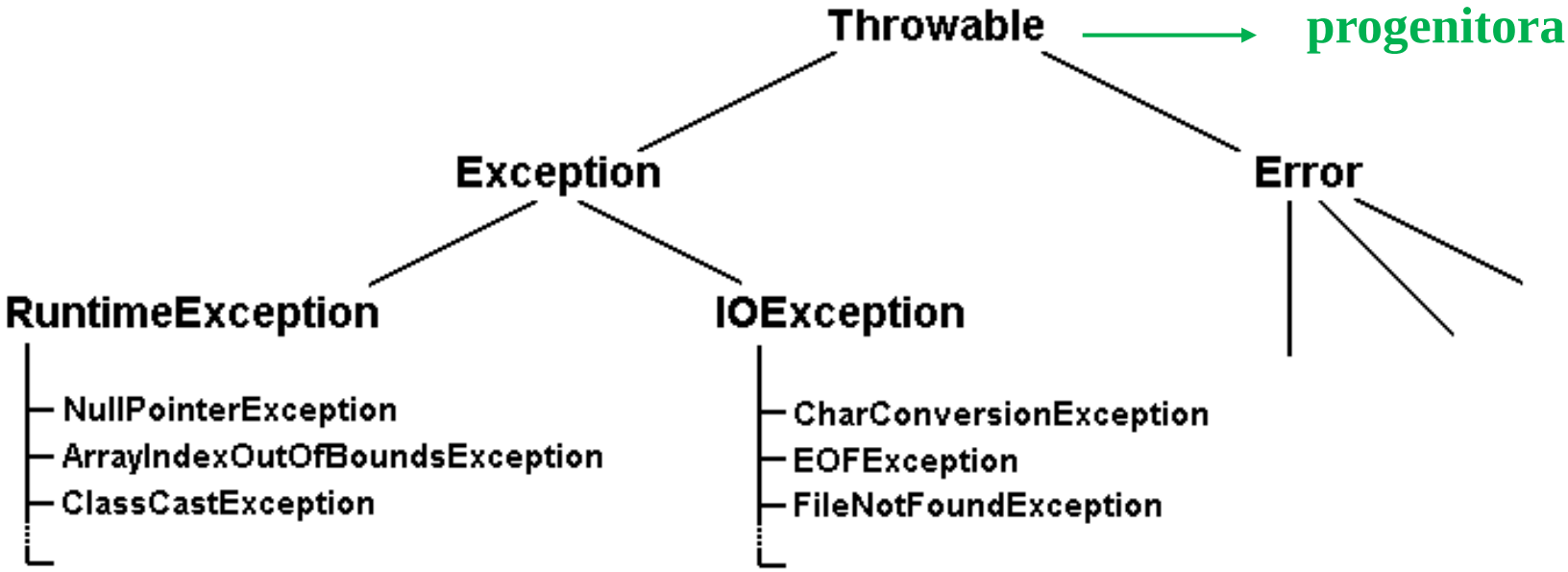
A ocorrência de uma situação inesperada no fluxo normal de processamento pode acontecer durante a execução de um programa, geralmente deixando seus usuários descontentes e até prejudicando as atividades realizadas até aquele momento inoportuno.

Estas ocorrências são tratadas de maneira especial em Java, sendo denominadas de *tratamento de exceção*.

As exceções se referem aos erros que podem ser gerados durante a execução de um programa, sendo aplicada a esta situação um conjunto de recursos Java que permitirão a manipulação e o tratamento da maioria destas possíveis situações.

Tratamento de Exceções

Estas possíveis exceções em Java possuem uma sintaxe própria para serem identificadas e tratadas, além de fazerem parte de uma hierarquia especial de heranças.



O tratamento das exceções se concentra, quase que totalmente, na Exception, pois praticamente a totalidade da Error esta fora de seu controle, sendo interessante na ocorrência desta última o encerramento de seu processo.

Tratamento de Exceções

Dois tipos de tratamento de exceção podem ser implementados em um programa, sendo estes classificados em:

- **Verificados**: exigem um tratamento de exceção no desenvolvimento do programa

  ⇒ provoca erro de compilação quando não é implementado em seu código (método)

  Exemplo: abertura de um arquivo de dados

- **Não verificados**: ou estão fora do controle do programa que está sendo elaborado, ou seu programa não deveria permitir que os mesmos acontecessem

  ⇒ não provoca erro de compilação se não for implementado, pois é considerado de responsabilidade de seu programador

  Exemplo: divisão por zero

Tratamento de Exceções

Instrução try... catch

Uma das formas de tratar as exceções é o uso da instrução try...catch, que possibilita a averiguação de uma ou várias exceções em um conjunto de instruções.

Suas principais características são:

- Abertura e fechamento de bloco de instrução é obrigatória para cada sub-bloco que compõem esta instrução (**{ }**)
- A instrução finally corresponde a um outro bloco de **instrução opcional** na instrução try...catch, similar ao else na instrução if
- Para cada try pode haver um ou mais catch, mas somente um finally ou ambos (catch e finally) nesta instrução try
- Cada bloco catch define o tratamento de **uma única exceção**, recebendo como parâmetro sua identificação
- Esta exceção deve pertencer a classe **Throwable** ou uma de suas subclasses

Tratamento de Exceções

```
/** Síntese
 *    Objetivo: analisar a paridade de um número
 *    Entrada:  número inteiro
 *    Saída:    paridade do número
*/
import java.util.Scanner;
public class Paridade {
  public static void main(String[] args) {
    int numero;
    Scanner ler = new Scanner(System.in);
    System.out.println("Digite um número inteiro.");
    numero = ler.nextInt();
    System.out.print("\n\n\nO número "+ numero +" é\t");
    if((numero % 2) == 0)
       System.out.print("PAR");
    else
       System.out.println("IMPAR");
  }
}
```

Tratamento de Exceções

Após observar o funcionamento correto do programa anterior, verifique o que a desatenção de um usuário pode ocasionar no uso do mesmo.

Imagine que o valor informado pelo usuário foi um valor real, ou ainda ele digitou por engano uma letra ou um outro caracter especial. Este programa não previa esta possibilidade e por isso será encerrado sem uma orientação do motivo disso acontecer para seu usuário.

Para se evitar esta situação tente elaborar no mesmo um tratamento de exceção que impeça este término de programa mais indelicado e oriente seu usuário sobre o que aconteceu usando o tratamento de exceção.

→ Muitas são as exceções disponíveis em Java, sendo por isso interessante testar a ocorrência das mesmas e implementar seu tratamento, fazendo uma **cópia de sua identificação**.

Tratamento de Exceções

```
/** Síntese
 *    Objetivo: analisar a paridade de um número
 *    Entrada:  número inteiro
 *    Saída:    paridade do número
*/
import java.util.Scanner;
import java.util.InputMismatchException;
public class Paridade {   // novo código com exceção
  public static void main(String[] args) {
    int numero;
    Scanner ler = new Scanner(System.in);
    System.out.println("Digite um número inteiro.");
    try {
      numero = ler.nextInt();
      System.out.print("\n\nO número "+ numero +" é\t");
      if((numero % 2) == 0)
          System.out.print("PAR");
      else
          System.out.println("IMPAR");
    } catch (InputMismatchException excecao) {
        System.out.print("Valor incorreto. Paridade ");
        System.out.println("não pode ser verificada.");
    }
  }
}   // identificação da exceção ocorrida anteriormente
```

Tratamento de Exceções

Usando finally

O bloco opcional do try...catch pode se tornar obrigatório caso o bloco try não possua nenhum catch definido, sendo possível então as seguintes variações no uso desta instrução.

```
 :
try {
   // bloco de instruções
} catch (exceção) {
   // bloco de tratamento
}
 :
```

```
 :
try {
   // bloco de instruções
} finally {
   // bloco de continuidade
}
 :
```

Além destas variações mais simples, também é possível o tratamento de exceções múltiplas em um mesmo try, bem como o uso do finally com estes.

Tratamento de Exceções

```
  :
try {
    // bloco de instruções
} catch (exceção1) {
    // bloco de tratamento
} catch (exceção2) {
    // bloco de tratamento
}
  :
```

```
  :
try {
    // bloco de instruções
} catch (exceção1) {
    // bloco de tratamento
} catch (exceção2) {
    // bloco de tratamento
} finally {
    // bloco de continuidade
}
```

- finally é sempre executado, independente de ter ocorrido ou não uma exceção
- A ocorrência de alguma exceção executará sempre sua catch e em seguida o bloco finally, se houver sua definição
- O finally permite a liberação de recursos para continuidade do processamento, após a ocorrência de alguma exceção
- Seu uso ainda permite o tratamento de algumas exceções, sendo a ocorrência de outras tratadas exatamente no finally

Tratamento de Exceções

Características Importantes

- Evite usar exceções que não sejam para manipulação de erros, pois sua utilização interfere diretamente no desempenho do programa, tornando-o bem **mais lento**
- A execução do bloco try é **interrompida** na instrução que gerar uma exceção, não sendo suas outras instruções executadas neste bloco, pois esta execução é desviada ao bloco catch que corresponda a tal tipo de exceção
- Sobre a catch será procurada uma classe específica ou uma de suas subclasses
  - O catch específico tem maior precedência que um geral
  - Só o primeiro catch identificado será executado num try
- O bloco finally pode também não ser executado totalmente, se dentro dele ocorrer uma exceção, que deverá ser tratada como uma outra exceção qualquer

Exercício de Fixação

4) Elabore um programa que analise dados de uma amostra de pessoas a serem pesquisadas sobre a contaminação de dengue. Cada pessoa participante desta pesquisa deverá informar seu nome completo, idade e sexo, coletados do usuário através de uma janela de interação adequada. Quando o pesquisador solicitar o encerramento da coleta deverá ser apresentado o relatório final com as porcentagens de:
   - pessoas que já tiveram dengue entre todas;
   - homens que já tiveram dengue em relação a quantidade de homens participantes da pesquisa;
   - crianças (idade até 12 anos) que já tiveram dengue em relação a todos os participantes.

   Apresente na **console** as porcentagens acima, além dos dados complementares: total de pessoas participantes, quantidade de homens e de mulheres pesquisados. Esta solução deverá possuir a classe Pessoa somente com o controle de seus dados cadastrais, sendo estático somente o método *main* e os demais métodos de serviços na solução.

Estrutura de Dados Homogênea - String

O conteúdo de uma String é **imutável**, ou seja, não pode ser alterado. Porém, a referência do tipo String não precisa apontar sempre para o mesmo objeto, podendo variar seu apontamento, onde o coletor de lixo automático Java liberará a área de memória que não está mais em uso.

```
String saudar = new String("Olá");
saudar = "Bom dia!";
```

- A primeira instrução declara um objeto String saudar e aciona seu método construtor
- Na segunda instrução o objeto saudar **modifica sua referência** para um outro local onde estará armazenado outro conteúdo String desejado (Bom dia!)

Estrutura de Dados Homogênea - String

Classe StringBuffer e StringBuilder

Estas 2 classes strings são **mutáveis**, ao contrário da classe String, o que permiti a alteração desejada sobre o próprio objeto String criado na memória.

A diferença entre estas classes mutáveis consiste na primeira ser sincronizada e a outra não (StringBuilder), sendo uma equivalente a outra.

Exemplo: suponha a criação destes objetos String.

```
String saudar = "Bom ";
saudar = saudar + "dia!";
```

→ O objeto saudar (String), após a segunda instrução acima, referenciará um novo objeto String contendo "Bom dia!“

→ Um programa *multi-thread* exigi o uso da StringBuffer.

Estrutura de Dados Homogênea - String

As classes StringBuffer e StringBuilder operam sobre o próprio objeto já criado em memória, evitando a criação de diversos objetos indesejáveis na memória.

Geralmente, a conversão (*casting*) dos objetos instanciados por estas classes é realizada de maneira automática (ou implícita). Porém, existem situações onde o método toString() deverá ser utilizado para realizar esta conversão explicitamente.

Exemplo: suponha a criação deste objeto StringBuilder.

```
StringBuilder saudar = "Bom ";
saudar = saudar + "dia!";
System.out.print(saudar);   // conversão implícita
```

Estrutura de Dados Homogênea - String

Alguns dos métodos construtores destas classes são apresentados a seguir e demonstram a sobrecarga (conceito já estudado) que acontecem com os mesmos.

CONSTRUTOR	**OPERAÇÃO**
StringBuffer()	Aloca um novo objeto StringBuffer sem nenhum caracter armazenado (nulo), com capacidade inicial de **16**
StringBuffer(int length)	Aloca um novo objeto StringBuffer sem nenhum caracter, mas com capacidade definida em seu parâmetro
StringBuffer(String)	Aloca um novo objeto String com o mesmo conteúdo do objeto String definido no parâmetro

Estrutura de Dados Homogênea - String

Os objetos destas classes mutáveis de String não podem ser carregados pela entrada via teclado com os métodos da classe Scanner. Para isso, os dados lidos do teclado deverão ser carregados, primeiramente, por um objeto String (não mutável) com seu método nextLine() e posteriormente ser atribuído a um objeto da classe StringBuffer ou StringBuilder (mutáveis).

Exemplo:

```
   :
String cadeia = new String();
StringBuilder frase = new StringBuilder();
Scanner ler = new Scanner(System.in);
System.out.println("Informe a frase desejada:");
cadeia = ler.nextLine();
frase.append(cadeia);  // conversão de String para
  :                    // StringBuilder
```

Estrutura de Dados Homogênea - String

Os principais métodos disponíveis na classe String também estão disponíveis para estas classes mutáveis, tais como: charAt(int), length(), equals(Object), indexOf(String), substring(int início, int fim), replace(int, int), entre outros métodos equivalentes.

MÉTODO	**FUNCIONALIDADE**
setCharAt(int, char)	modifica o valor char na posição indicada
toString()	cria um objeto String a parti da uma das classes mutáveis
setLength(int)	trunca ou expande a classe mutável preenchendo com valor nulo as expansões
append(Object)	objeto é convertido para String e concatenado ao final da classe mutável, aumentando seu tamanho quando necessário
insert(int, String)	insere o String desejado na posição indicada
delete(int inicial, int final)	apaga caracteres da posição inicial até a posição **final -1**
:	:

Exercícios de Fixação

5) Faça um programa em OO que armazene o primeiro e o último nome de uma pessoa em variáveis diferentes, garantindo que nenhuma destas strings estará vazia. Em seguida, concatene estas duas strings no formato usado no cadastro de passageiros aéreos, utilizando os recursos mais coerentes para manipulação de strings sem criação inadequada de vários objetos temporários na memória do computador. Após a concatenação deverá ser apresentado ao usuário o seu nome padronizado e duas linhas abaixo será solicitado se este deseja informar outro nome.

   Usando da POO elabore métodos não estáticos para solução da lógica que será necessária neste exercício, onde o padrão aéreo consiste no último nome concatenado com uma barra (/) e seguida do primeiro nome, por exemplo:

   SILVA / LUIZ

Estrutura de Dados Homogênea - String

Método split()

Este método divide uma String em um Array de Strings, respeitando determinado critério especificado no seu argumento.

```
public static void main(String[] args) {
 // Declarações
  String frase = new String("Eu te amo.");
  String palavras[] = frase.split(" ");    // critério

 // Instruções
  for(int aux = 0; aux < 3; aux++)
    System.out.println("Palavras = " + palavras[aux]);
}
```

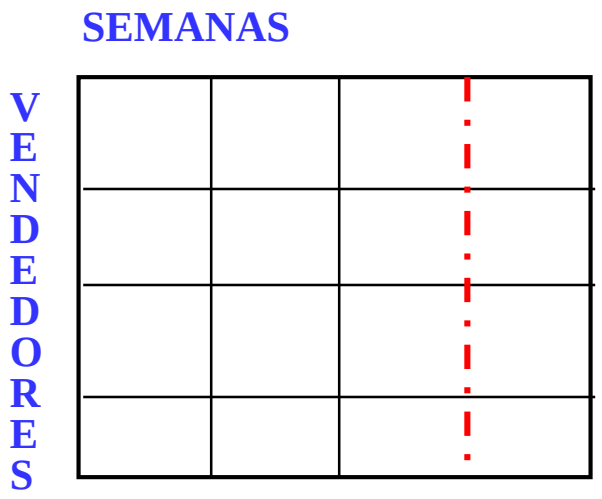
Exercícios de Fixação

6) Elabore um programa que registre o nome completo de uma pessoa e separe cada um de seus sobrenomes em uma outra estrutura de dados homogênea (array), tendo o cuidado de colocar todos os caracteres informados pelo usuário em maiúsculo. Apresente em cada linha de seu programa os nomes que formam o nome completo de seu usuário. Realize esta operação enquanto o usuário desejar informar nomes e não aceite a entrada de dados vazia (sem nome algum).

7) Desenvolva um programa que receba uma frase do usuário e mostre a quantidade de vezes que cada letra se repete na mesma, além da própria letra, sem considerar a igualdade entre as letras maiúsculas e minúsculas, ou seja, sua solução tratará de forma diferente cada uma das letras. Por exemplo: **B** é diferente de **b**.

Exercícios de Fixação

8) Desenvolva um programa que armazene, em uma matriz, as vendas semanais de 4 vendedores durante um mês, que poderá variar entre 3 ou 4 semanas contábeis. Após estes cadastros apresente o total de vendas de cada semana, o total de vendas de cada vendedor neste mês e o total geral de vendas no mês. A identificação de quantas semanas contábeis tem o mês e quantidade de vendas de cada vendedor deverão ser validadas de maneira coerentes. Nesta solução ainda deverão existir um método que valide a quantidade de vendas, outro que valide a quantidade de semanas e um último, sem retorno, que apresentará os valores totais solicitados neste problema. Outros métodos deverão ser elaborados mantendo a coerência com a implementação adequada em POO.



Referência de Criação e Apoio ao Estudo

Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G. Core Java2 , volume 1, Makron Books, 2001.
  - Capítulos 3, 5 e 11
- FURGERI, S. Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 2002.
  - Capítulos 3, 4 e 8
- ASCENCIO, A. F. G.; CAMPOS, E. A. V. Fundamentos da programação de computadores, 2 ed., São Paulo: Pearson Prentice Hall,2007.
  - Capítulos 6, 7 e 9
- Universidade de Brasília (UnB Gama)
  - http://cae.ucb.br/conteudo/unbfga

  (escolha a disciplina **Orientação a Objetos** no menu superior)