transforme-se

serasa experian

WISEHANDS
Tecnologia e Acessibilidade

JAVA
AULA 5 - Exceções, arrays e métodos

serasa experian

WISEHANDS
Tecnologia e Acessibilidade

Exceções em Java

Exceções em Java

As exceções em Java referem-se aos erros que podem ser gerados durante a execução de um pro grama. Como o nome sugere, trata-se de algo que interrompe a execução normal do programa. Em Java, as exceções são divididas em duas categorias: Unchecked (não verificadas) e Checked (verificadas).

Unchecked Exception

Nesse tipo de exceção o compilador Java não verifica o código-fonte para determinar se a exceção está sendo capturada. Assim, o tratamento desse tipo de exceção é opcional. Apesar de termos tratado o erro de conversão pela estrutura try-catch, isso é opcional.

Mesmo se a estrutura try-catch for retirada, a classe continua compilando normalmente. Outros exemplos de exceção desse tipo são o acesso a um índice inexistente num array e o uso do método de um objeto que ainda não foi instanciado (nulo).

Checked Exception

Ao contrário de Unchecked Exception, nesse tipo de exceção o compilador Java verifica o código fonte para determinar se a exceção está sendo capturada. Se uma exceção verificada não for capturada (não estiver sendo tratada no código-fonte), o compilador acusa a possível exceção e obriga o programador a tratá-la.

Essa exceção pode ser tratada de duas maneiras: por meio da estrutura try-catch-finally ou por meio da cláusula throws.

Uso da estrutura try-catch-finally

Vamos fazer uma analogia com o mundo real. Considere que você vai fazer uma viagem de automóvel de uma cidade para outra. Durante o trajeto, podem ocorrer eventos que o obriguem a fazer uma pausa ou mudar seu trajeto. Por exemplo, ao furar o pneu do veículo você terá que parar a viagem para realizar sua troca. Se houver algum acidente na estrada, talvez você resolva pegar outro caminho. Veja que podem ocorrer vários incidentes que exigirão um tratamento diferente ou a mudança de rota.

Algo bastante semelhante pode ocorrer durante a execução de um programa de computador e cada tipo de erro necessita que seja realizado um tratamento diferente.

Portanto, a estrutura try-catch-finally tem como função desviar a execução de um programa caso ocorram certos tipos de erro, predefinidos durante o processamento das linhas, e evitar que o programador precise fazer testes de verificação e avaliação antes de realizar certas operações.

Quando um erro ocorre, ele gera uma exceção que pode ser tratada pelo programa. A estrutura try-catch-finally pode ser usada tanto com Unchecked Exceptions como com Checked Exceptions.

Uso da estrutura try-catch-finally

A estrutura try-catch-finally possui a seguinte sintaxe:

```
try{
    <conjunto de instruções>
   } catch (Nome da exceção){
    <tratamento do erro 1>
   } catch (Nome da exceção){
    <tratamento do erro 2>
   } catch (Nome da exceção){
    <tratamento do erro n>
   } finally{
    <conjunto de instruções>
   }
```

Uso da estrutura try-catch-finally

Toda vez que a estrutura try é utilizada, obrigatoriamente em seu encerramento (na chave final) deve existir pelo menos um catch, a não ser que ela utilize a instrução finally.

A sintaxe apresentada pode ser interpretada como:

1. tente executar o conjunto de instruções do try que estão entre as chaves;
2. Se houver algum erro, execute seu tratamento no catch.
3. Depois de tratado o erro, a execução do programa continua a partir do final do último catch.
4. O finally é opcional e fornece um conjunto de códigos que é sempre executado, independentemente de uma exceção ocorrer ou não.

Uso da estrutura try-catch-finally

Em exemplo de try-catch

```
1  try {
2      int valor = Integer.parseInt(JOptionPane.showInputDialog("Insira um valor numérico").toString());
3  }
4  catch (NullPointerException erro){
5      JOptionPane.showMessageDialog(null, "Tecla cancel pressionada\n" + erro.toString(),
6                                    "Cancelado pelo usuário", JOptionPane.ERROR_MESSAGE);
7  }
```

No exemplo acima, caso o usuário clique em cancelar na janela de input, não teríamos uma String para passar e sim um valor nulo, o que estoura uma NullPointerException que nós capturamos e damos o tratamento desejado nas linhas 4, 5 e 6.

Uso da cláusula throws

Em alguns momentos, pode ocorrer de o programador não querer realizar controle sobre uma exceção, isto é, não desejar tratar um erro. A linguagem Java permite ao programador que um erro seja descartado, mesmo que ele ocorra. Entretanto é preciso que esse fato seja informado na declaração do método. Esse processo pode ser realizado pela cláusula throws. Para ilustrar o uso dessa cláusula, vamos elaborar um exemplo que demonstra a criação de um arquivo no disco rígido.

```java
public static void main(String[] args) throws IOException {

    String frase = JOptionPane.showInputDialog("Entre com uma frase");

    try {
        FileWriter file = new FileWriter(CAMINHO, true);
        PrintWriter out = new PrintWriter(file);
        out.println(frase);
        JOptionPane.showMessageDialog(null, "Frase armazenada no arquivo");
    }catch (FileNotFoundException erro) {
        JOptionPane.showMessageDialog(null, "Erro, verifique se o caminho do arqvuivo está correto!");
    }
}
```

Uso da instrução throw

Conforme comentado anteriormente, a linguagem Java possui muitas exceções, incluindo sua geração e tratamento. Assim, seu estudo detalhado precisaria ser definido em um livro à parte. O objetivo deste item é apenas demonstrar que um desenvolvedor Java pode criar suas próprias exceções e dispará-las no momento em que necessitar.

A instrução throw é utilizada para disparar uma exceção, isto é, ela pode forçar que uma determinada exceção ocorra. O disparo dessa exceção pode ser realizado sempre que for fornecido um valor inválido.

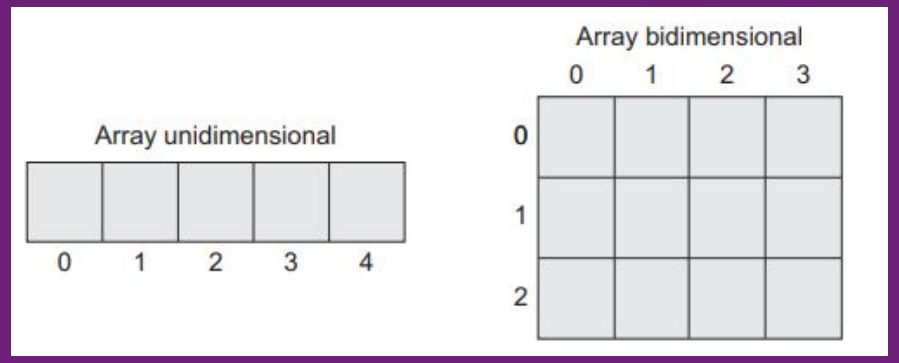
Utilização de Arrays

Definição

Suponha que seja necessário armazenar e manipular dezenas de nomes de pessoas num programa de computador. De acordo com o que estudamos até aqui, seriam necessárias dezenas de variáveis, cada uma armazenando um nome diferente, como, por exemplo, nome1=”Lucas”, nome2=”Daniel” e assim por diante.

Em vez disso, é possível a declaração de apenas uma variável indexada, chamada array. Em outras palavras, podemos definir uma variável cujos elementos são referenciados por um índice no seguinte formato: nome[0]=”Lucas”, nome[1]=”Daniel” etc.

Os arrays podem ser unidimensionais (com uma única dimensão, conhecido como vetor) ou bidimensionais (com duas dimensões, conhecido por matriz).



Arrays unidimensionais

Arrays unidimensionais

Os arrays unidimensionais são os que possuem apenas um índice para acessar seu conteúdo.
Eles são declarados da seguinte maneira:

- Tipo-de-dado[] nome-do-array = new Tipo-de-dado[quantidade], em que:
- **Tipo-de-dado** → pode ser qualquer tipo de variável.
- **Nome-do-array** → um nome válido; as mesmas regras para nomes das variáveis.
- **quantidade** → a quantidade de elementos que o array pode manipular.

Arrays unidimensionais

Exemplos:

- int[] numeros=new int[10]; → cria um array com o nome numeros que contém 10 elementos do tipo int e seu índice varia de 0 a 9.

- double[] precos=new double[5]; → cria um array com o nome precos que contém 5 elementos do tipo double e seu índice varia entre 0 e 4.

Arrays unidimensionais

Para atribuir o valor a um elemento do array, deve-se indicar o índice desejado dentro dos colchetes, como nos exemplos a seguir:

```
numeros[0] = 100;
numeros[5] = 38;
numeros[8] = 17;
meses[0] = “Janeiro”;
meses[3] = “Abril”;
meses[11] = “Dezembro”;
```

Arrays bidimensionais

Arrays unidimensionais

Um array bidimensional possui dois índices e possibilita que os valores sejam armazenados na forma de matrizes. A linguagem Java não suporta arrays bidimensionais como as outras linguagens (no formato linha, coluna, por exemplo), entretanto é possível obter a mesma funcionalidade criando um array de arrays. Os de uso mais comum são os que envolvem dois arrays, mas é possível criar arrays com quantas dimensões forem necessárias.

Esses arrays devem ser declarados da seguinte maneira:

Tipo-do-dado nome-do-array[][] = new tipo-do-dado [] []

Busca em arrays

Arrays unidimensionais

Outro aspecto pertinente ao uso de um array se refere à busca de um valor entre seus elementos. Existem diversas maneiras de realizar esse processo, umas mais simples, porém lentas, e outras mais complexas, porém mais velozes. Vamos iniciar pela forma mais simples.

```
1  String[] cores = { "amarelo", "vermelhjo", "azul"};
2  String cor = JOptionPane.showInputDialog("Digite uma cor");
3  String msg = "Cor não encontrada";
4  for(String elemento: cores) {
5      if(elemento.equals(cor)){
6          msg = "Con encontrada";
7          break;
8      }
9  }
10 JOptionPane.showMessageDialog(null, msg);
```

transforme-se

O conhecimento é o poder
de transformar o seu futuro.