Prog. Orientada a Objeto

Aula 18 – Tratamento de Erros

Prof. Nemésio Freitas Duarte Filho

Ementa

- Introdução ao JAVA
- Introdução e Fundamentos de POO
- Classes, Objetos, Atributos e Métodos
- Encapsulamento, Herança e Polimorfismo
- Interfaces gráficas
- Tratando erros
- Pacotes - Organizando suas classes e bibliotecas
- Ferramentas: jar e javadoc

Ementa

- Introdução ao JAVA
- Introdução e Fundamentos de POO
- Classes, Objetos, Atributos e Métodos
- Encapsulamento, Herança e Polimorfismo
- Interfaces gráficas
- **Tratando erros**
- Pacotes - Organizando suas classes e bibliotecas
- Ferramentas: jar e javadoc

Quais os principais tipos de erros?

1.Lógica de programação

Ex: limites do vetor ultrapassados, divisão por zero

Atitude: Correção do programa. Muitos erros são detectados pelo compilador

2.Ambiente de execução

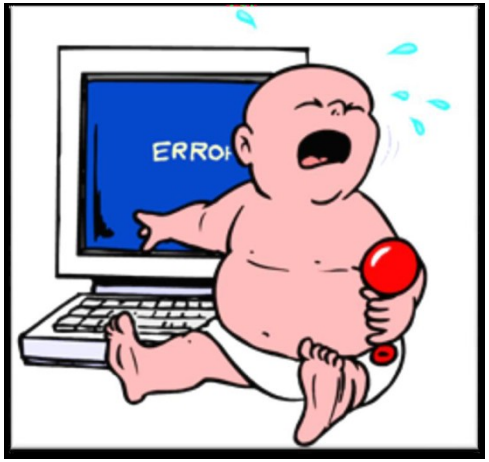
Ex: arquivo não encontrado, rede fora do ar, etc.

Atitude: Precisam ser tratados para evitar o travamento do programa

3.Erros graves

Ex: falha catastrófica, erro interno do JVM

Atitude: Não há o que fazer!



Erros

- Os erros em sistema computacionais podem ocorrer por vários motivos
  - Comportamento imprevisível de usuários
  - Falhas de hardware
  - Problemas de conexão ou comunicação
  - Falta de direito de acesso a recursos
  - Erros de memória, alocação, estouro de capacidade

- Abordagens antigas utilizavam variáveis globais para identificar o erro
  - A lógica do programa fica entrelaçada com o código do erro
  - Condições de erro eram pouco documentadas

Exceções

A palavra **exception** é usada no sentido de “eu abro uma exceção para isso“

No ponto onde o problema ocorre você pode não saber o que fazer com ele, mas você sabe que não pode continuar normalmente

Outro benefício significativo das **exceções é que elas eliminam o código de tratamento de erro**

Em vez de verificar por um erro em particular e lidá-lo em múltiplos lugares no seu programa, você só necessita tratar o problema em um único lugar, o assim chamado **tratamento de erros**

Exceções

O tratamento de exceções é o único meio formal da linguagem Java para reportar erros

É forçado pelo compilador Java, ou seja, em situações onde há probabilidade de ocorrer erros é obrigatório o uso de exceções

Exceções (Resumo)

Uma exceção é um sinal que indica que algum tipo de condição excepcional ocorreu durante a execução do programa.

Assim, exceções estão associadas a condições de erro que não tinham como ser verificadas durante a compilação do programa. As duas atividades associadas à manipulação de uma exceção são:

- **geração:** a sinalização de que uma condição excepcional (por exemplo, um erro) ocorreu, e

- **captura:** a manipulação (tratamento) da situação excepcional, onde as ações necessárias para a recuperação da situação de erro são definidas.

Exceções (Resumo)

- Para cada exceção que pode ocorrer durante a execução do código, um bloco de ações de tratamento (um *exception handler) deve ser especificado. O compilador Java verifica e força que toda* exceção "não-trivial" tenha um bloco de tratamento associado.

- A sinalização da exceção é propagada a partir do bloco de código onde ela ocorreu através de toda a pilha de invocações de métodos até que a exceção seja capturada por um bloco manipulador de exceção.

- Eventualmente, se tal bloco não existir em nenhum ponto da pilha de invocações de métodos, a sinalização da exceção atinge o método main(), fazendo com que o interpretador Java apresente uma mensagem de erro e aborte sua execução.

Instruções

- Try: bloco onde pode ocorrer um erro;
- Catch: bloco onde será tratado o erro;
- Finally: bloco que será sempre executado (útil quando se quer liberar um recurso, por exemplo);
- Throw: instrução para forçar a ocorrência de uma exceção;

Tratamento de Exceções

- O tratamento de exceções no Java é realizado através das instruções try-catch-finally

```
try {
    // Instruções a serem executadas
}
catch (TipoExceção erro) {
    // Tratamento de exceções de um tipo específico
}
catch (Exception erro) {
    // Tratamento genérico de exceções
}
finally {
    // Instruções sempre executadas, ocorram ou não erros
}
```

Tratamento de Exceções

- A captura e o tratamento de exceções em Java se dá através da especificação de blocos **try, catch e finally**, definidos através destas mesmas palavras reservadas da linguagem. A estruturação desses blocos obedece à seguinte sintaxe:

```
try {
   //  código que inclui comandos/invocações de métodos
   //  que podem gerar uma situação de exceção.
}
catch (XException ex) {
   //  bloco de tratamento associado à condição de
   //  exceção XException ou a qualquer uma de suas
   //  subclasses, identificada aqui pelo objeto
   //  com referência ex
}
catch (YException ey) {
   //  bloco de tratamento para a situação de exceção
   //  YException ou a qualquer uma de suas subclasses
}
finally {
   //  bloco de código que sempre será executado após
   //  o bloco try, independentemente de sua conclusão
   //  ter ocorrido normalmente ou ter sido interrompida
}
```

Bloco *Try*

- O bloco *try* é um bloco de execução protegida onde deve ser implementado o fluxo principal do programa

  - O bloco *try* é obrigatório
  - O Java tentará executar todas as instruções no bloco *try*
  - Se nenhuma instrução gerar uma exceção, todas serão executadas
  - Se ocorrer algum erro, a execução é pulada para os blocos *catch* ou *finally*. As instruções restantes no try são ignoradas
  - Se nenhum *catch* capturar a exceção, ela é capturada pelo JVM e a execução do aplicativo pode ser encerrada, dependendo do tipo de erro

Bloco *Catch*

- O bloco *catch* é o bloco de manipulação da exceção onde devem ser implementados os tratamentos de erro
  - O bloco *catch* é opcional
  - Um bloco *try* pode possuir vários blocos *catch*
  - Os manipuladores devem ser escritos do mais específico para o mais genérico
  - A classe mais genérica de erro é a *Exception*
  - A exceção é consumida no bloco *catch*, ou seja, *a priori* apenas um *catch* é executado

Bloco *Finally*

- No bloco *finally* são incluídas as instruções que devem necessariamente ser executadas, ocorram ou não erros
  - O bloco *finally* é opcional
  - Um bloco *try* possui apenas um bloco *finally*
  - Blocos *finally* são utilizados para liberação de recursos como manipuladores de arquivos e conexões com bancos de dados
  - Numa execução sem erros são executados os blocos *try* e *finally*
  - Numa execução com erros são executados os blocos *try* (parcialmente), *catch* (de acordo com o tipo do erro) e *finally* (sempre)

Ex: try/catch/finally

- Estruturas possíveis para o tratamento de exceções

```
try {
    // bloco
}
catch {
    // bloco
}
```

```
try {
    // bloco
}
finally {
    // bloco
}
```

```
try {
    // bloco
}
catch {
    // bloco
}
finally {
    // bloco
}
```

```
try {
    // bloco
}
catch {
    // bloco
}
catch {
    // bloco
}
// ...
finally {
    // bloco
}
```

Tratamento de Exceções

- Alguns exemplos de exceções já definidas no pacote java.lang incluem:

- **ArithmeticException:** indica situações de erros em processamento aritmético, tal como uma divisão inteira por 0.

- **NumberFormatException:** indica que tentou-se a conversão de uma string para um formato numérico, mas seu conteúdo não representava adequadamente um número para aquele formato.

- **IndexOutOfBounds:** indica a tentativa de acesso a um elemento de um agregado aquém ou além dos limites válidos.

Tratamento de Exceções

- **NullPointerException:** indica que a aplicação tentou usar uma referência a um objeto que não foi ainda definida;

- **ClassNotFoundException:** indica que a máquina virtual Java tentou carregar uma classe mas não foi possível encontrá-la durante a execução da aplicação.

Exceções geradas pelo compilador

- Tipos de exceções: 100+

Exceção	Descrição
ArrayTypeMismatchException	Lançado quando um array não pode armazenar um elemento porque os tipos do elemento e do array não são compatíveis.
DivideByZeroException	Lançado quando ocorre uma tentativa de dividir um valor inteiro por zero.
IndexOutOfRangeException	Lançado quando ocorre uma tentativa de utilizar um índice fora dos limites de um array.
NullReferenceException	Lançado quando ocorre uma tentativa de referenciar um objeto (cujo valor é) nulo.
InvalidCastException	Lançado quando uma conversão explícita falha em tempo de execução

Exceções geradas pelo compilador

Exceção	Descrição
NullReferenceException	Lançado na tentativa de referenciar um objeto cujo valor é nulo.
OutOfMemoryException	Lançado quando não há quantidade suficiente de memória disponível.
OverflowException	Lançado quando uma operação aritmética em um contexto "checked" estoura o limite do tipo.
StackOverflowException	Lançado quando a pilha do sistema está cheia (muitas chamadas a métodos; geralmente, em uma recursão infinita).
TypeInitializationException	Lançado quando um construtor estático lança uma exceção e não há um catch para tratar a exceção.

Tratamento de Exceções

- Uma exceção contém pelo menos uma *string que a descreve, que pode ser obtida pela aplicação* do método getMessage(), mas pode eventualmente conter outras informações.

- Por exemplo, InterruptedIOException inclui um atributo público do tipo inteiro, bytesTransferred, para indicar quantos bytes foram transferidos antes da interrupção da operação ocorrer.

- Outra informação que pode sempre ser obtida de uma exceção é a seqüência de métodos no momento da exceção, obtenível a partir do método printStackTrace().

Throw

Exceções não são apenas um sinal de que algo catastrófico ocorreu

Podem servir para alertar sobre alguma condição

– Um método que não deve ser utilizado

– Um sinal de que um método não executou devidamente

Exceções podem ser geradas por meio de throw

Tratamento de Exceções - Trow

- Para lançar uma exceção que seja instância das classes de verificação obrigatória a linguagem obriga o programador a declarar no cabeçalho do método quais as classes de exceção podem ter instâncias lançadas.

- Portanto, o formato completo do cabeçalho de definição de um método é:
  <modificadores> <tipo> <nome>(<parametros>) throws <classes>

```
static void metodoX() throws ErroDoTipoX, ErroDoTipoY,
  ErroDoTipoZ{
    ...
    throw new ErroDoTipoX();
    ...
    throw new ErroDoTipoY();
    ...
    throw new ErroDoTipoZ();
  }
```

Tratamento de Exceções - Trow

```
public class RelatorioFinanceiro {
    public void metodoMau() throws ExcecaoContabil {
        if (!dadosCorretos) {
            throw new ExcecaoContabil("Dados Incorretos");
        }
    }
    public void metodoBom() {
        try {
            ... instruções ...
            metodoMau();
            ... instruções ...
        } catch (ExcecaoContabil ex) {
            System.out.println("Erro: " + ex.getMessage());
        }
        ... instruções ...
    }
}
```

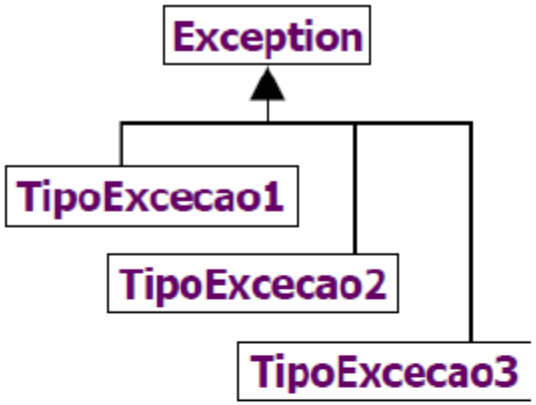
instruções que sempre serão executadas

instruções serão executadas se exceção não ocorrer

instruções serão executadas se exceção não ocorrer ou se ocorrer e for capturada

Try e Catch

- *O bloco try "tenta" executar um bloco de código que pode causar exceção*
- *Deve ser seguido por*
  - *Um ou mais blocos catch(TipoDeExcecao ref)*
  - *E/ou um bloco finally*
- *Blocos catch recebem tipo de exceção como argumento*
  - *Se ocorrer uma exceção no try, ela irá descer pelos catch até encontrar um que declare capturar exceções de uma classe ou superclasse da exceção*
  - *Apenas um dos blocos catch é executado*




```
try {
    ... instruções ...
} catch (TipoExcecao1 ex) {
    ... faz alguma coisa ...
} catch (TipoExcecao2 ex) {
    ... faz alguma coisa ...
} catch (Exception ex) {
    ... faz alguma coisa ...
}

... continuação ...
```

Finally

- *O bloco try não pode aparecer sozinho*
  - *deve ser seguido por pelo menos um catch ou por um finally*
- *O bloco finally contém instruções que devem se executadas independentemente da ocorrência ou não de exceções*

```
try {
  // instruções: executa até linha onde ocorrer exceção
} catch (TipoExcecao1 ex) {
   // executa somente se ocorrer TipoExcecao1

} catch (TipoExcecao2 ex) {
   // executa somente se ocorrer TipoExcecao2
} finally {
   // executa sempre ...
}

// executa se exceção for capturada ou se não ocorrer
```

O que não se deve fazer

NUNCA *escreva o seguinte código:*

```
try {
// .. código que pode causar exceções
} catch (Exception e) {}
```

Ele pega até NullPointerException, e não diz nada. O mundo se acaba, o programa trava ou funciona de modo estranho e ninguém saberá a causa a não ser que mande imprimir o valor de e, entre as chaves do catch.

Exemplo

- *Considere o seguinte código*

```
1.  try {
2.    URL u = new URL(s); // s is a previously defined String
3.    Object o = in.readObject(); // in is valid ObjectInputStream
4.    System.out.println("Success");
5.  }
6.  catch (MalformedURLException e) {
7.    System.out.println("Bad URL");
8.  }
9.  catch (IOException e) {
10.   System.out.println("Bad file contents");
11. }
12. catch (Exception e) {
13.   System.out.println("General exception");
14. }
15. finally {
16.   System.out.println("doing finally part");
17. }
18. System.out.println("Carrying on");
```

Exemplo

1. Que linhas são impressas se os métodos das linhas 2 e 3 completarem com sucesso sem provocar exceções?

- *A. Success*
- *B. Bad URL*
- *C. Bad File Contents*
- *D. General Exception*
- *E. Doing finally part*
- *F. Carrying on*

Exemplo

2. Que linhas são impressas se o método da linha 3 provocar um OutOfMemoryError?

- *A. Success*
- *B. Bad URL*
- *C. Bad File Contents*
- *D. General Exception*
- *E. Doing finally part*
- *F. Carrying on*

Exemplo

3. Que linhas são impressas se o método da linha 2 provocar uma MalformedURLException?

- *A. Success*
- *B. Bad URL*
- *C. Bad File Contents*
- *D. General Exception*
- *E. Doing finally part*
- *F. Carrying on*

Exemplo

4. Que linhas são impressas se o método da linha 3 provocar um StreamCorruptedException?

- *A. Success*
- *B. Bad URL*
- *C. Bad File Contents*
- *D. General Exception*
- *E. Doing finally part*
- *F. Carrying on*

Exemplo Pratico

- **ArrayIndexOutOfBoundsException**: Este tipo de exceção acontece quando há a tentativa de recuperar o conteúdo de uma posição de um array utilizando um indexador maior que a última posição (índice) do array.

```
public class TesteException1{

   public static void main (String args[]){

      int vetor[] = {3,6,9};
      int index = 3;

      try{
         System.out.println("A terceira posicao é: " + vetor[index]);
      }
      catch (ArrayIndexOutOfBoundsException e){
         System.out.println("A posicao " + index + " não é valida");
      }
   }
}
```

Exemplo Pratico

- **NullPointerException**: Este tipo de exceção acontece quando você tenta acessar um objeto que está nulo (null), ou seja, um objeto que não foi instanciado.

```
import javax.swing.JOptionPane;
public class TesteException2{
   public static void main (String args[]){
      String str;
      String mensagem = "Digite a informacao";
      int icone = JOptionPane.INFORMATION_MESSAGE;

      try{
      str = JOptionPane.showInputDialog(null, mensagem, "msg", icone);
      char c = str.charAt(0);
      }
      catch (NullPointerException e){
         System.out.println("A variavel str está nula (null) ");
      }
   }
}
```

Exemplo Pratico

- **ArithmeticException**: Este tipo de exceção acontece quando uma condição aritmética excepcional acontece. Por exemplo uma divisão por zero.

```
public class TesteException3{
   public static void main (String args[]){
      int a = 25;
      int b = 0;

      try{
            int c = a / b;
      }
      catch (ArithmeticException e){
            System.out.println("Divisao por zero");
      }
   }
}
```

Exemplo Pratico

- **NumberFormatException**: Este tipo de exceção acontece quando tentamos converter uma string para um tipo primitivo numérico, porém a string não contém apenas caracteres numéricos.

```
public class TesteException4{
   public static void main (String args[]){
      String str = "12x";
      try{
            int num = Integer.parseInt(str);
      }
      catch (NumberFormatException e){
            System.out.println("Numero " + str + " inválido");
      }
   }
}
```

Exemplo Pratico

- O bloco finally é opcional e serve para delimitar um bloco de instruções que devam ser executadas independente do que ocorra no bloco try.

```
public class TesteException5{
   public static void main (String args[]){
      int vetor[] = {3,6,9};
      int index = 3;

      try{
         System.out.println("A terceira posicao é: " + vetor[index]);
      }
      catch (ArrayIndexOutOfBoundsException e){
         System.out.println("A posicao " + index + " não é valida");
      }
      finally{
         System.out.println("Execução independente de ser gerada uma
exceção ou não");
      }
   }
}
```

Exemplo Pratico

- É possível utilizar tantos catchs quantos forem necessários

```
public class TesteTryCatch{
   public static void main (String args[]){
      try{
        String str1 = args[0];
        String str2 = args[1];

        int num1 = Integer.parseInt(str1);
        int num2 = Integer.parseInt(str2);

        double resp = num1 / num2;
        System.out.println("O resultado é: " + resp);
      }
      catch (ArrayIndexOutOfBoundsException e){
         System.out.println("Devem ser informados 2 argumentos");
      }
      catch (NumberFormatException e){
      System.out.println("Erro na conversão de string para inteiro");
      }
      catch (ArithmeticException e){
        System.out.println("Ocorreu uma divisão por zero!");
      }
   }
}
```

Exercícios

- Quais os principais tipos de erros que podem ocorrer em um sistema computacional? Defina em poucas palavras o que seria uma exceção.

- A captura e o tratamento de exceções em Java se dá através da especificação de blocos **try, catch e finally,** qual a diferença deles?

- Alguns exemplos de exceções já são definidas no pacote java.lang. É possível que o próprio desenvolvedor crie novas exceções em relação aos novos métodos criados? Explique.