Tratamento de Exceções

FURB
UNIVERSIDADE DE BLUMENAU

Tipos de erros

- Durante a execução de um programa, podemos nos deparar com dois tipos de erros:
  - Erros de lógica
  - Erros de execução

Erros de lógica

- São erros de concepção de algoritmo

```
double[] nota = new double[3];
nota[0] = 10;
nota[1] = 7;
nota[2] = 8;

double somaNotas = 0;
for (int i=0; i<=nota.length; i++) {
    somaNotas += nota[i];
}

System.out.println("A média é: " + (somaNotas/nota.length));
```

- Os erros de lógica **devem** ser evitados

Erros de execução

- Erros causados por operações que não possuem suporte pelo ambiente. Exemplos:
  - Entrada de dados inadequada
  - Manipulação indevida de arquivos
  - Operação aritmética ilegal
  - Comando não atendido pelo periférico
  - Falta de memória
- Os erros de execução correspondem a condições excepcionais ou anormais, frequentemente chamadas de **exceções**
- Os erros de execução, muitas vezes, não podem ser evitados.
  - Na ocorrência de tais erros, como o programa deve se comportar?

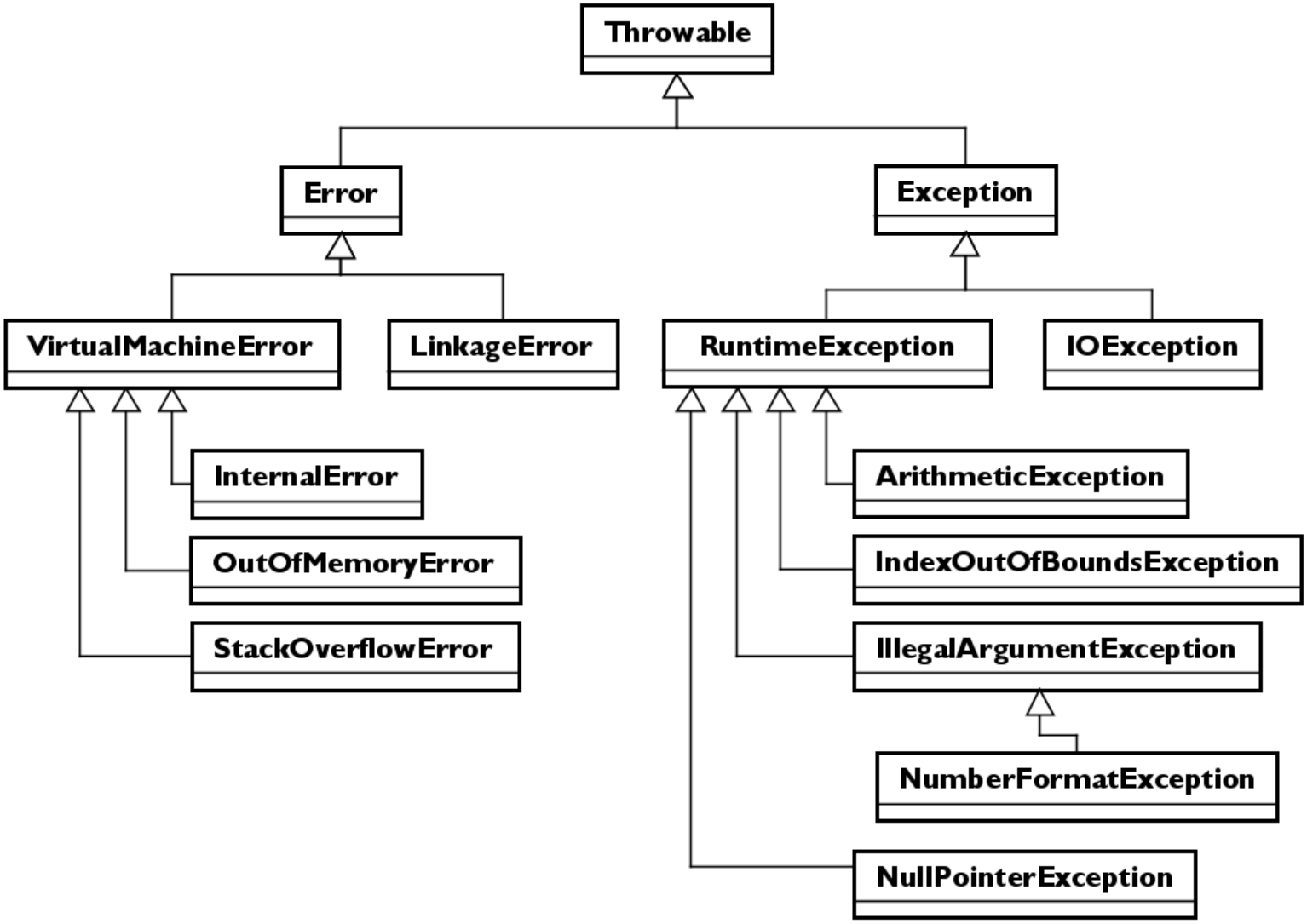
Erros

- Se o usuário digitar uma URL incorreta no navegador de internet, o que deveria acontecer?
  - Nada – O navegador finaliza
  - O navegador solicita outra URL
- Dependendo da natureza da exceção, o programa pode tratar adequadamente a situação. Chamamos este procedimento de **tratamento de exceção**.
- Quando uma exceção não é tratada, o programa é abortado – finaliza de forma anormal

Erros em Java

- Os erros quando ocorrem são detectados pela JVM
- A JVM cria um objeto que caracteriza o erro
- O programa que causou o erro é notificado pela JVM
- O programa pode tratar o erro
  - Podemos acessar o objeto criado pela JVM ao tratar a exceção
- Java possui vários tipos de classe que representam erros. As principais são:
  - Classe `Error`
  - Classe `Exception`

Hierarquia de classes de erros



Hierarquia de classes de erros

- A classe `Error` e todas suas subclasses são reservadas para indicar erros graves.
  - Não se espera que sejam tratados pelos programas
- A classe `Exception` e todas suas subclasses indicam erros que poderiam ser tratados pelos programas.

Tratamento de exceções

- Embora a JVM detecte erros, o programador também pode criar explicitamente objetos para sinalizar situações de erro;
- Também é possível criar novas classes de erro, desde que sejam suclasses de `Throwable` ou de alguma de suas subclasses.

Como tratar exceções

- O tratamento de exceções é feito através do comando `try`, cuja sintaxe básica consta abaixo:

```
try {
    comando
    comando
    comando
    ...
} catch (ClasseExcecao objeto) {
    comando caso ocorra erro
    comando caso ocorra erro
    ...
}
```

O comando `catch` exige um parâmetro.

Podemos definir qualquer classe que estenda de `Throwable`

- Qualquer erro ocorrido num dos comandos do bloco `try` causará o desvio do fluxo de execução para o bloco `catch`

O comando try..catch

No início do comando `try`, o ambiente passa a monitorar a execução de todos os comandos declarados em seu corpo. Se nenhum erro ocorrer durante a execução dos comandos monitorados, então o comando `try` termina normalmente e os comandos de bloco `catch` não são executados

Exemplo

```
1 Scanner teclado = new Scanner(System.in);
2 int idade;
3 try {
4     idade = Integer.parseInt(teclado.nextLine());
5 } catch (NumberFormatException objErro) {
6     System.out.println("Valor incorreto. ");
7 }
8 System.out.println("Fim")
```

Dizemos que este fragmento é capaz de "capturar" o erro `NumberFormatException`

Exemplo

```
 1 Scanner teclado = new Scanner(System.in);
 2 int num;
 3
 4 System.out.println("Digite um número: ");
 5 while (true) {
 6     try {
 7         num = Integer.parseInt(teclado.nextLine());
 8         System.out.println("O número informado é: " + num);
 9         break;
10     } catch (NumberFormatException objErro) {
11         System.out.println("Valor incorreto. Por favor, digite novamente");
12     }
13 }
```

Múltiplos catch

- O comando `try...catch` aceita várias cláusulas `catch`.

```
 1 Scanner teclado = new Scanner(System.in);
 2
 3 try {
 4     int a[] = new int[2];
 5     a[4] = 30 / Integer.parseInt(teclado.nextLine());
 6     System.out.println("Operação concluída com êxito");
 7 } catch (NumberFormatException e) {
 8     System.out.println("Valor digitado é inválido");
 9 } catch (ArithmeticException e) {
10     System.out.println("Falha na divisão");
11 } catch (ArrayIndexOutOfBoundsException e) {
12     System.out.println("Não conseguiu atribuir ao vetor");
13 } catch (Exception e) {
14     System.out.println("Qualquer outra exceção");
15 }
16 System.out.println("Fora do bloco");
```

Múltiplos catch

- Ao ocorrer um erro causado por um dos comandos do bloco `try`, um objeto de uma classe que caracteriza o erro é criado e o programa é notificado.
- O comando que causou o erro é interrompido.
- Em seguida, as cláusulas `catch` são verificadas, na ordem em que foram implementadas, para verificar se há alguma que pode tratar a exceção
  - Quando uma cláusula catch que pode tratar o erro é encontrada, o fluxo é desviado para o primeiro comando do bloco desta cláusula. Ao fim da execução desta cláusula, o comando try é finalizado normalmente e o fluxo prossegue após o comando try.

Tratando erros através de polimorfismo

- Um erro de uma dada classe pode ser capturado por uma cláusula catch que tenha como parâmetro uma variável da mesma classe ou de uma superclasse do erro ocorrido, utilizando uma variável polimórfica

```
1 try {
2     criarArquivo();
3     gravarArquivo();
4 } catch (IOException e) {
5     System.out.println("Falha ao manipular arquivo");
6 }
```

Tratando erros através de superclasse

- Como o fluxo da execução é desviado para a primeira cláusula-catch que pode tratar a exceção, não é válido declarar cláusulas que especifiquem classes de erros que sejam subclasses de classes especificadas em cláusulas catch anteriores.

```
 1 try {
 2     int a[] = new int[2];
 3     a[4] = 30 / Integer.parseInt(teclado.nextLine());
 4     System.out.println("Operação concluída com êxito");
 5 } catch (Exception e) {
 6     System.out.println("Aconteceu algum erro...");
 7 } catch (NumberFormatException e) {
 8     System.out.println("Valor digitado é inválido");
 9 } catch (ArithmeticException e) {
10     System.out.println("Falha na divisão");
11 } catch (ArrayIndexOutOfBoundsException e) {
12     System.out.println("Não conseguiu atribuir ao vetor");
13 }
```

Mesmo tratamento para várias classes de erro

- A partir do Java 7, é possível definir uma cláusula catch capaz de tratar várias classes de erro.
- Exemplo:

```
1 try {
2     ...
3 } catch ( IOException | SQLException ex ) {
4     ...
5 }
```

Propagação de erros

Quando um método é interrompido por erro, o controle retorna ao método chamador, que poderá tratar o erro.

```
 1 public static int obterValor() {
 2     Scanner teclado = new Scanner(System.in);
 3     int valor = Integer.parseInt(teclado.nextLine());
 4     return valor;
 5 }
 6
 7 public static void teste1() {
 8     while (true) {
 9         try {
10             int num = obterValor();
11             System.out.println("Numero informado = " + num);
12             break;
13         } catch (NumberFormatException e) {
14             System.out.println("Valor inválido. Informe novamente");
15         }
16     }
17
18     System.out.println("Fim");
19 }
```

Propagação de erros

- Se o método chamador não puder tratar o erro que causou a interrupção, também será interrompido.
- A propagação do erro ocorre até que ele possa ser tratado por algum método da cadeia de chamada de métodos.
  - O último método que tem “oportunidade” de tratar o erro é o método `main`().
  - Se o método `main`() não tratar o erro, o programa é interrompido

Exceções verificadas pelo compilador

- Alguns métodos podem solicitar ao compilador exigir que o programador trate a exceção no local em que ocorreram ou devem ter o tratamento explicitamente postergado.
- São os erros das **classes verificáveis**.
- As classes Error e RuntimeException, bem como todas as suas subclasses, não são verificáveis. As demais classes são verificáveis.

Exceções verificadas pelo compilador

```
public void abrirArquivo() {
    arquivo = new FileReader("arquivo.txt");
}
```

Este método causa erro de compilação porque `FileReader` pode lançar uma exceção da classe FileNotFoundException e deve ser tratado

```
public void abrirArquivo() {
    try {
        arquivo = new FileReader("arquivo.txt");
    } catch (FileNotFoundException e) {
        System.out.println("Arquivo não encontrado");
    }
}
```

Cláusula throws

- A claúsula `throws` é utilizada num método para informar que o método chamador deverá tratar a possibilidade daquelas exceções ocorrerem
- Utilizado com classes de erros verificáveis
- Quando um método utiliza a classe `throws`, ele delega o tratamento daquelas exceções para os métodos chamadores

```
public void abrirArquivo() throws FileNotFoundException {
    arquivo = new FileReader("arquivo.txt");
}
```

- É possível informar várias exceções na cláusula `throws`, basta separá-las por vírgula.

Cláusula finally

- A cláusula `finally` está associada ao comando `try` e é utilizado para definir um conjunto de comandos que sempre serão executados, quer o comando `try` execute com ou sem erros.
- Sintaxe:

```
try {
    comando
    comando
...
} catch (ClasseExcecao objeto) {
    comando caso ocorra erro
    comando caso ocorra erro
    ...
} finally {
    comando...
    comando...
}
```

Cláusula finally

- O uso da cláusula `finally` geralmente está associado à liberação de recursos alocados durante o bloco `try`.

Observações

- Todo comando try define dois ou mais blocos referentes ao seu corpo e às suas cláusulas-catch. Estes blocos definem escopo de variáveis.
- Se um método de uma superclasse declara exceções verificáveis, para sobrescrever o método, é obrigatório redeclarar as mesmas exceções