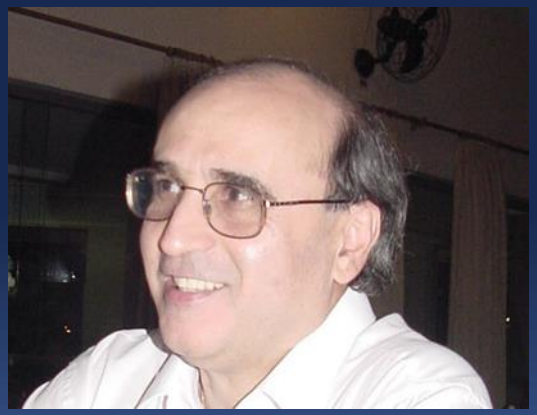
Programação Orientada a Objetos

Unidade 5 – Tratamento de Exceções



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia

- Beginning Java 2 – Ivor Horton – 1999 WROX
- Java2 – The Complete Reference – 7th Edition – Herbert Schildt – Oracle Press
- Core Java Fundamentals – Horstmann / Cornell – PTR- Volumes 1 e 2 – 8th Edition
- Inside the Java 2 – Virtual Machine Venners – McGrawHill
- Understanding Object-Oriented Programming with JAVA – Timothy Budd – Addison Wesley
- Head First Java, 2nd Edition by Kathy Sierra and Bert Bates
- Effective Java, 2nd Edition by Joshua Bloch
- Thinking in Java (4th Edition) by Bruce Eckel
- Java How to Program - 9th Edition by Paul Deitel and Harvey Deitel

Introdução

- Numa aplicação em um mundo ideal, o usuário jamais entraria com dados em formato inválido, arquivos sempre existiriam, e o código **nunca** teria **bugs**.

Erros ...

- **Infelizmente**, durante a execução de uma aplicação podem ocorrer **erros**.

Tente executar este código !

```
public class Erro1 {
  public static void main(String[] args){

      int x = 4, y=0;
      System.out.println(x/y);

  }
}
```

Tente executar este código !

```
public class Erro2 {

  public static void main(String[] args){

      int[] vet = new int[3];
      for (int i=0; i<4; i++)
          vet[i] = i+1;
   }
}
```

Quais as causas de erros ?

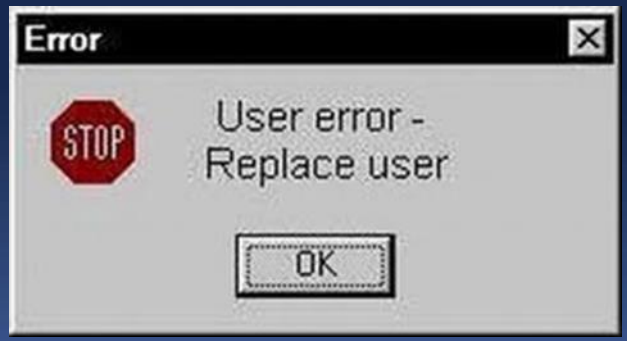
Causas de erros...

- O erro pode ser causado por um arquivo com informação inválida, um problema de conexão na rede, ou índice de array inválido, ou ainda a tentativa de referenciar um objeto que aponta para null, ou erros de input do usuário, erros de devices, limitações **físicas**, erros de codificação, etc.

Como prevenir erros ?

Como prevenir erros ?

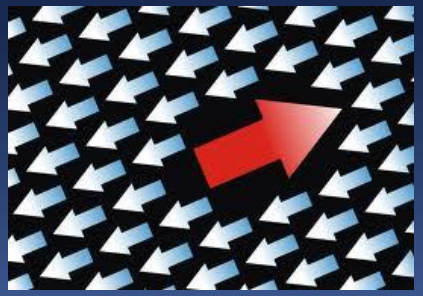
- Java usa uma forma de tratamento de erros, chamada **exception-handling**. (manuseio de exceções...)
- Este mecanismo é semelhante a **C++** , **C#.**



O que é uma exception ?

Exception

- Uma exception é um **objeto**, que ocorre durante a execução de um programa, e encapsula todas as informações do erro ocorrido.
- Uma exception é sempre uma instância da classe **Throwable**.



Terminologia

- Uma exceção é dita ser lançada (**thrown**) a partir do ponto onde ocorreu e é dita apanhada (**caught**) no ponto para o qual o controle é transferido.

thrown

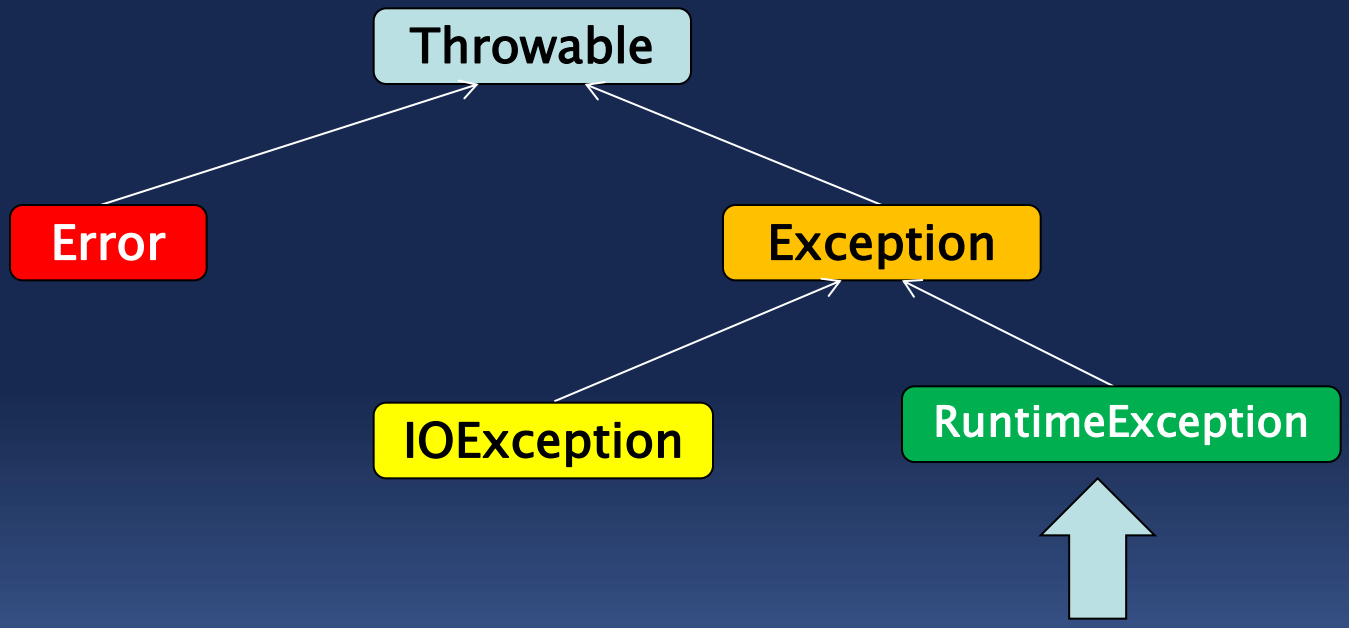
caught

Comando throw

- Programas podem lançar **exceções** explicitamente, por meio do comando **throw**.
- O uso **explícito** do comando **throw** torna os programas mais robustos e menos propensos a comportamento indesejado.

A classe Throwable

- Todas as exceções descendem da classe **Throwable**.
- Logo abaixo dela, descendem duas hierarquias: **Error** e **Exception**.



Exceptions herdadas de RuntimeException

- Uma operação de **casting** incorreta;
- Erro de **endereçamento** de acesso à arrays;
- Acesso a objetos com pointer null;
- Erros em operações aritméticas.

Subclasses da RunTimeException

- ArithmeticException
- IndexOutOfBoundException
- Negative ArraySizeException
- NullPointerException
- ArrayStoreException
- ClassCastException
- IllegalArgumentException
- SecurityException
- IllegalMonitorStateException
- IllegalStateException
- UnSupportedOperationException

Manuseando exceções

- Para tratarmos **exceções** em um programa quando elas ocorrerem, há três tipos de blocos de código que podemos incluir em um método para manuseá-las:
  - ✓ **bloco try**
  - ✓ **bloco catch**
  - ✓ **bloco finally**

Bloco try

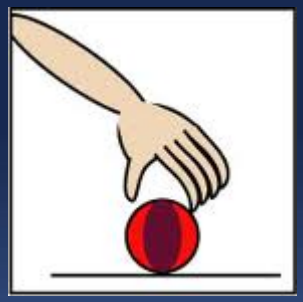
- Define um **bloco de código** que pode causar uma ou mais exceções.
- Ou seja, quando você quizer interceptar exceções (catch) o código que pode causá-las deve estar delimitado pelo bloco **try**.
- Código que cause exceção e que não esteja delimitado pelo bloco **try** não será capaz de capturá-la (por meio de catch correspondente).

Bloco try

```
try   {
          // código que pode disparar
          // uma ou mais exceções. . .
}
```

Bloco catch

- Define o código que irá manusear a **exceção** de um determinado tipo;
- Deve seguir imediatamente o bloco **try**.



Bloco catch

```
try {
            // codigo que pode disparar
            // uma ou mais exceções. . .
}
catch (ArithmeticException e) {
            // codigo para tratar a exceção
}
```

Bloco catch

- No exemplo anterior, o bloco **catch** somente manuseia (trata) exceções do tipo **ArithmeticException**;
- Isto implica que este é o único tipo de exceção que pode ser disparada (thrown) no bloco **try**.

Bloco catch

- Em geral, o parâmetro para um bloco **catch** deve ser do tipo **Throwable** ou uma de suas subclasses;
- Se a **classe** especificada como parâmetro tem subclasses,então o bloco **catch** irá tratar exceções desta classe mais todas as suas subclasses.

Exemplo

```
public class ExemploTryCatch  {
  public static void main(String[] args)   {
    int i = 1;
    int j = 0;
    try {
      System.out.println(" Entrada bloco try  " + "i = "+ i + " j = "+j);
      System.out.println(i/j);      // Divisao por 0 - exception thrown
      System.out.println(" Fim bloco try ");
    }
    // Catch the exception
    catch(ArithmeticException e) {
      System.out.println(" Arithmetic exception caught ");
    }
    System.out.println(" Apos bloco try ");
    return;
  }
}
```

O par try/catch

- O par **try catch** forma um casal;
- Você não deve separá-los incluindo comandos entre os mesmos.

Múltiplos blocos Catch

- Se um bloco **try** pode disparar (throw) diversos tipos de exceções, então você poderá definir diversos blocos **try** para manusear estas diferentes exceções.

Múltiplos blocos Catch

```
try {
        // codigo que pode disparar
        // (throw) exceções
}
catch (ArithmeticException e) {
        // codigo para tratar as excecões do tipo
        // ArithmeticException
}
catch (IndexOutOfBoundsException e) {
        // codigo para tratar as excecões do tipo
        // IndexOutOfBoundsException
}
// execução continua aqui…
```

Múltiplos blocos Catch

- Quando você necessitar manusear (catch) exceções de diferentes tipos em um bloco **try**, a ordem dos blocos **catch** é importante;
- Quando uma exceção é disparada (thrown), ela será tratada pelo primeiro bloco **catch** com o parâmetro de mesmo tipo da exceção ou um tipo que é uma superclasse do tipo da exceção.

Múltiplos blocos Catch

- Isto tem uma importante implicação;
- Os blocos **catch** devem estar na sequência do tipo mais específico primeiro, e o tipo mais básico por último;
- Caso contrário, o código não compilará;
- A razão é simples: se o primeiro for mais geral e o segundo derivado do primeiro, então o segundo jamais será executado. . .

Múltiplos blocos Catch

```
try  {
        // bloco de codigo try
}
catch (Exception  e) {
        // manuseio generico de exceção
}
catch (AritmeticException  e) {
        // tratamento especializado
        // este catch jamais será executado. . .
        // compilador irá marcar como erro . . .
}
```

O bloco finally

- A exceção pode introduzir a possibilidade de deixar o programa em estado insatisfatório;
- Por exemplo, você abriu um arquivo, e por causa de uma exceção, o código para fechar o arquivo não foi executado;
- O bloco **finally** provê o meio para sincronizar a execução do bloco **try**.

O bloco finally

- Um bloco **finally** é sempre executado, independentemente do que ocorre durante a execução do método;
- Se um arquivo precisa ser fechado, ou se algum recurso crítico precisa ser liberado, você pode garantir esta providência por meio do bloco **finally**.

O bloco finally

```
finally {
        // codigo de clean-up para ser
        // executado por último
}
```

- Deve sempre ser seguido dos blocos **try** e **catch**;
- Se não houver bloco **catch**, então deve seguir imediatamente após o bloco **try**.

Estruturando um método

```
try {
        // código que pode disparar exceções...
}
catch   (ExceptionType1 e ){
        // código para tratar exceção do tipo
        // ExceptionType1 ou subclasses
}
catch   (ExceptionType2 e ){
        // código para tratar exceção do tipo
        // ExceptionType2 ou subclasses
}
// mais blocos catch se necessarios. . .
finally {
        // código para ser executado após
        // o bloco try
}
```

Estruturando um método

- Você não pode ter apenas um bloco **try**. Cada bloco **try** deve sempre ser seguido por ao menos um bloco catch ou **finally**.
- Você não pode incluir código entre um bloco **try** e seus blocos **catch**, ou entre o bloco **try** e o bloco **finally**.
- Você pode ter outros blocos **try** em um método.