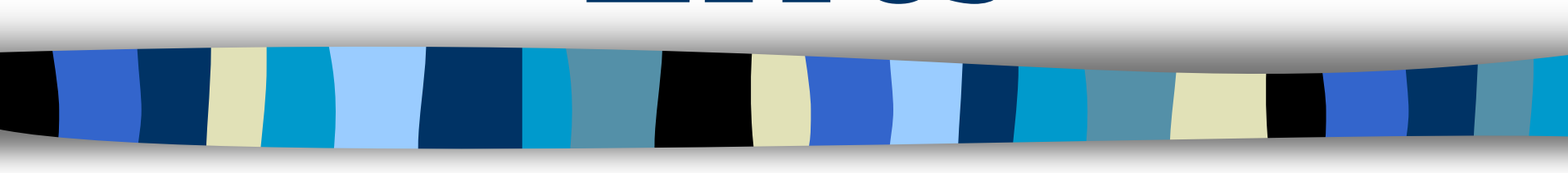
Tratamento de Erros



Gerenciamento de erros com Exceções em Java

O que é uma exceção ?

- O termo “exceção” é uma contração da frase “evento excepcional”
- Uma exceção é um evento que ocorre durante a execução de um programa que interfere no fluxo normal das instruções deste programa

O que ocasiona uma exceção ?

- Muitos tipos de erros podem causar uma exceção, como por exemplo:
  - tentar acessar um array fora de seus limites,
  - tentar abrir um arquivo inexistente,
  - uma falha geral no disco,
  - tentar abrir uma URL inexistente,
  - tentar dividir por zero,
  - tentar calcular a raiz quadrada de um número inexistente
  - etc.

Quando ocorre um erro ...

- Quando tal erro ocorre dentro de um método, este cria um objeto da classe `Exception` e passa este objeto para o sistema de *runtime*.
- Este objeto contém informações sobre a exceção, incluindo, por exemplo, seu tipo e o estado do programa quando o erro ocorreu

uma exceção é lançada !

- A partir deste momento, o sistema de *runtime* se responsabiliza por achar algum código que trate o erro ocorrido.
- Em Java, criar-se um objeto da classe `Exception` e passá-lo para o sistema de *runtime* denomina-se “lançar uma exceção” (*throwing an exception*)

Quando uma exceção é lançada...

- O sistema passa a procurar alguém capaz de tratar esta exceção
- A lista de “candidatos” para este tratamento vem da pilha de chamada de métodos que antecederam o método que lançou a exceção
- O sistema de runtime “desmonta” a pilha de chamadas, começando com o próprio método onde ocorreu o erro, buscando um método que possua um “manipulador de exceção” adequado

busca-se um *exception handler*

- Um “*exception handler*” é considerado adequado quando a exceção que ele manipula é do mesmo tipo da exceção lançada.
- Quando ele é encontrado, recebe o controle do programa para que possa tratar o erro ocorrido
- Em outras palavras, diz-se que ele “capturou” a exceção (*catch the exception*)

Se ele não é encontrado...

- Se nenhum dos métodos pesquisados pelo sistema de *runtime* provê um manipulador de exceções adequado, então o programa Java em questão é abruptamente encerrado.

Principal vantagem da manipulação de erros por exceções

- Separação do código para manipulação de erros do código “normal” do programa. Exemplo de algoritmo:

```
lerArquivo()
{
  abrir o arquivo;
  determinar seu tamanho;
  alocar memória suficiente;
  ler o arquivo para a memória
  fechar o arquivo;
}
```

Tratamento “complicado” de erros

```
tipoErro leArquivo()
{
  tipoErro códigoErro = 0;
  abrir arquivo;
  se (arquivo abriu) então {
    determinar tamanho do arquivo;
    se (conseguiu obter tamanho do arquivo) então {
      alocar memória suficiente;
      se (conseguiu memória suficiente) então {
        ler o arquivo para memória;
        se (leitura falhou) então
          códigoErro = -1;
      }
      senão
        códigoErro = -2
    }
    senão
      códigoErro = -3
    fechar o arquivo;
    se (arquivo não fechou)
      códigoErro = -4
  }
  senão
    códigoErro = -5
  retorne códigoErro;
}
```

Tratamento “fácil” de erros

```
lerArquivo() {
  try {
    abrir o arquivo;
    determinar seu tamanho;
    alocar memória suficiente;
    ler o arquivo para a memória
    fechar o arquivo;
  }
  catch (Exceção falhouAbrirArquivo) {
    fazAlgumaCoisa;
  }
  catch (Exceção falhouDeterminarTamanho) {
    fazAlgumaCoisa;
  }
  catch (Exceção falhouAlocarMemória) {
    fazAlgumaCoisa;
  }
  catch (Exceção falhouLerArquivo) {
    fazAlgumaCoisa;
  }
  catch (Exceção falhouFecharArquivo) {
    fazAlgumaCoisa;
  }
}
```

Se bem que...

- É importante lembrar que as exceções não fazem “mágica”, ou seja, não diminuem o esforço necessário para se detectar, reportar e manipular erros.
- O que elas permitem é a separação do código fonte regular do código responsável “por se fazer alguma coisa quando algo ruim acontece no programa”

Terminologia das exceções:

- ***Exception***: uma condição de erro que ocorre durante *runtime*
- ***Throwing***: lançamento de uma exceção
- ***Catching***: capturando uma exceção que acabou de ocorrer e executando instruções que tentam resolvê-la de alguma maneira
- ***Catch clause***: bloco de instruções que tentam resolver uma exceção
- ***Stack trace***: a seqüência de chamadas de métodos executados até o ponto onde ocorreu a exceção

Palavras chave Java para o tratamento de exceções:

- try
- catch
- throw
- throws
- finally

Formato geral do código para lançar e tratar exceções

Captura da Exceção

```
try
{
  //código que pode gerar uma exceção
}
catch (Exception e)//captura a exceção
{
  //código para tratar a exceção
}
finally
{
}
```

Passagem da Exceção pelo Método

```
void método1() throws IOException
{
  //código que pode gerar uma IOException
}
```

Lançamento de Exceções

```
void método2() throws IOException
{
  //testa condição de exceção
  if (excep)then throw (new IOException());
}
```

Exemplo sem tratamento de exceção

```
class Teste01
{
  public static void main (String args[])
  {
    int i = 1, j = 0, k;
    k = i/j;
  }
}
/*  causa o erro:
  Exception in thread "main"
  java.lang.ArithmeticException: / by zero
        at Teste01.main(Teste01.java:6)
  quando da execução do programa, que termina abruptamente
*/
```

Exemplo com tratamento de exceção

```
class Teste01
{
  public static void main (String args[])
  {
    int i = 1, j = 0, k;
    try
    {
      k = i/j;
    }
    catch (ArithmeticException e)
    {
      System.out.println("Deu pau");
      System.out.println(e);
    }
  }
}
```

Mais um exemplo:

```
class Conta
{
  public static void main (String args[])
  {
    int divisor = 0;
    int numero = 30;
    int indice = 100;
    int c[] = {1};
    try
    {
      int resultado = numero / divisor;
      c[indice] = resultado;
    }
    catch (ArithmeticException a)
    {
      System.out.println("Divisão por zero");
      divisor = 1;
    }
    catch (IndexOutOfBoundsException x)
    {
      System.out.println("Estourou o índice do array");
      indice = 0;
    }
  }
}
```

Variações sobre o mesmo exemplo:

```
class Conta
{
  public static void main (String args[])
  {
    int divisor = 0;
    int numero = 30;
    int indice = 100;
    int c[] = {1};
    try
    {
      int resultado = numero / divisor;
      try
      {
        c[indice] = resultado;
      }
      catch (IndexOutOfBoundsException x)
      {
        System.out.println("Estourou o índice do array");
        indice = 0;
      }
    }
    catch (ArithmeticException a)
    {
      System.out.println("Divisão por zero");
      divisor = 1;
    }
  }
}
```

Um exemplo diferente...

```
import java.io.*;
public class LeArquivo
{
  private FileReader entrada;
  public LeArquivo(String nomeArquivo)
  {
    entrada = new FileReader(nomeArquivo);
  }
  public String pegaPalavra()
  {
    int c;
    StringBuffer buf = new StringBuffer();
    do
    {
      c = entrada.read();
      if (Character.isWhitespace((char)c))
        return buf.toString();
      else
        buf.append((char)c);
    }
    while (c != -1);
    return buf.toString();
  }
}
```

...que não compila !

```
C:\>javac LeArquivo.java
LeArquivo.java:7: Exception java.io.FileNotFoundException must
be caught, or it must be declared in the throws clause of
this constructor.
    entrada = new FileReader(nomeArquivo);
              ^
LeArquivo.java:15: Exception java.io.IOException must be
caught, or it must be declared in the throws clause of
this method.
      c = entrada.read();
                  ^
2 errors
```

Pois o compilador sabe que, em ambos os métodos, estão sendo utilizadas instruções que podem ocasionar erro e, portanto, que lançam exceções. Só que estas exceções não estão sendo tratadas

Possíveis soluções:

- Tratar as exceções dentro dos próprios métodos (através de `try` e `catch`)
- Fazer com que os métodos possam lançar (`throws`) exceções que deverão ser tratadas por quem se utilizar destes métodos
- Esta última solução é mais adequada, pois o tratamento de exceções pode variar, dependendo de quem se utiliza dos métodos

Agora o exemplo compila !

```
import java.io.*;
public class LeArquivoOK
{
  private FileReader entrada;
  public LeArquivoOK(String nomeArquivo) throws FileNotFoundException
  {
    entrada = new FileReader(nomeArquivo);
  }
  public String pegaPalavra() throws IOException
  {
    int c;
    StringBuffer buf = new StringBuffer();
    do
    {
      c = entrada.read();
      if (Character.isWhitespace((char)c))
        return buf.toString();
      else
        buf.append((char)c);
    }
    while (c != -1);
     return buf.toString();
  }
}
```

Throws

- ***Throws***
  - –é usado para especificar quais os tipos de exceções que um método pode devolver !

E quem usa tem que fazer o seguinte...

```
import java.io.*;
import LeArquivoOK;
public class TesteLeArquivosOK
{
  public static void main (String args[])
  {
    LeArquivoOK l;
    try
    {
      l = new LeArquivoOK(”123DeOliveira4.txt");
    }
    catch (FileNotFoundException f)
    {
      System.out.println("Deu pau ! " + f.getMessage());
      f.printStackTrace();
    }
    System.out.println("Abriu");
  }
}
```

Ou algo um pouco mais sofisticado...

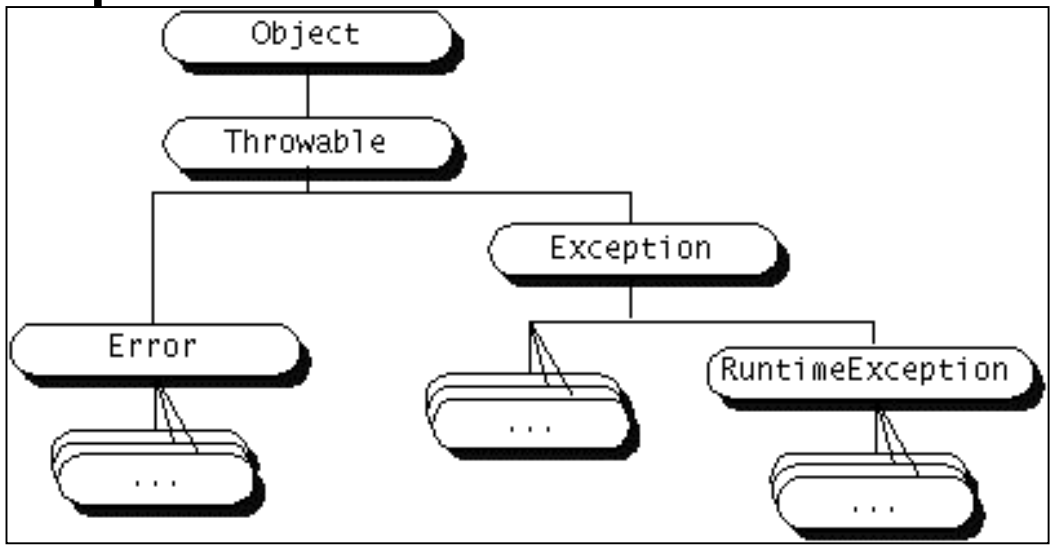
```
import LeArquivoOK;
import java.io.*;
public class TesteLeArquivosOK
{ private LeArquivoOK l;
  private String nomeArq = "12345678.txt";
  private boolean abriu = false;
  public void tentaAbrir()
  {
    while (!abriu)
    {
      try
      {
        l = new LeArquivoOK(nomeArq);
        abriu = true;
      }
      catch (FileNotFoundException f)
      {
        System.out.println("Deu pau ! " + f.getMessage()+"\n");
        f.printStackTrace();
        nomeArq = "c:\\autoexec.bat";
      }
    } // while
    System.out.println("Abriu o arquivo " + nomeArq);
  }
  public static void main (String args[])
  {
    TesteLeArquivosOK t = new TesteLeArquivosOK();
    t.tentaAbrir();
  }
}
```

Favor tentar fazer o seguinte:

- Acrescentar ao programa `TesteLeArquivosOK.java` a chamada ao método `pegaPalavra()` da classe `LeArquivoOK`

throw

- Favor não confundir com ***Throws***
- a palavra reservada ***throw*** lança uma exceção
- para lançar esta exceção, ela cria um objeto (que deve pertencer a uma classe que derive da classe `Throwable`)



Exemplo do uso de *throw*

```
public Object pop() throws EmptyStackException
{
  Object obj;
  if (size == 0)
    throw new EmptyStackException();
  obj = objectAt(size - 1);
  setObjectAt(size - 1, null);
  size--;
  return obj;
}
```

Cláusula ***finally***

- Relembrando a sintaxe:

Captura da Exceção

```
try
{
  //código que pode gerar uma exceção
}
catch (Exception e)//captura a exceção
{
  //código para tratar a exceção
}
finally
{
}
```

- A cláusula ***finally*** é opcional

Cláusula *finally*

- Ela provê um mecanismo que permite que seu método execute instruções de finalização, independente do que acontece dentro do bloco try
- geralmente estas instruções servem para fechar arquivos ou liberar eventuais recursos do sistema

Um exemplo do uso de *finally*

```
public void exemploBobo()
{
  PrintWriter saida = null;
  try
  {
    System.out.println("Entrando no bloco try");
    saida = new PrintWriter(new FileWriter("ArqSaida.txt"));
    for (int i = 0; i < tamanho; i++)
      saida.println("Valor em: " + i + " = " + vetor.elementAt(i));
  }
  catch (ArrayIndexOutOfBoundsException e)
  {
    System.err.println("Capturada ArrayIndexOutOfBoundsException: " +e.getMessage());
  }
  catch (IOException e)
  {
    System.err.println("Capturada IOException: " + e.getMessage());
  }
  finally
  {
    if (saida != null)
    {
      System.out.println("Fechando o PrintWriter");
      saida.close();
    }
    else
    {
      System.out.println("O PrintWriter não chegou a abrir");
    }
  }
}
```

Para encerrar: espiada em algumas classes de exceção (1/4)

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
           |
           +--AclNotFoundException
           +--ActivationException,
           +--AlreadyBoundException,
           +--ApplicationException,
           +--AWTException,
           +--BadLocationException,
           +--ClassNotFoundException,
           +--CloneNotSupportedException,
           +--DataFormatException,
           +--ExpandVetoException,
           +--GeneralSecurityException,
           +--IllegalAccessException,
           +--InstantiationException,
           +--InterruptedException,
```

Para encerrar: espiada em algumas classes de exceção (2/4)

```
+--IntrospectionException,
+--InvocationTargetException,
+--IOException
|  |
|  +--ChangedCharSetException,
|  +--CharConversionException,
|  +--EOFException,
|  +--FileNotFoundException,
|  +--InterruptedIOException,
|  +--MalformedURLException,
|  +--ObjectStreamException,
|  +--ProtocolException,
|  +--RemoteException,
|  +--SocketException,
|  +--SyncFailedException,
|  +--UnknownHostException,
|  +--UnknownServiceException,
|  +--UnsupportedEncodingException,
|  +--UTFDataFormatException,
|  +--ZipException
```

Para encerrar: espiada em algumas classes de exceção (3/4)

```
+--LastOwnerException,
+--NoninvertibleTransformException,
+--NoSuchFieldException,
+--NoSuchMethodException,
+--NotBoundException,
+--NotOwnerException,
+--ParseException,
+--PrinterException,
+--PrivilegedActionException,
+--PropertyVetoException,
+--RemarshalException,
+--RuntimeException,
|   |
|   +--ArithmeticException,
|   +--ArrayStoreException,
|   +--CannotRedoException,
|   +--CannotUndoException,
|   +--ClassCastException,
|   +--CMMException,
```

Para encerrar: espiada em algumas classes de exceção (4/4)

```
|  +--ConcurrentModificationException,
|  +--EmptyStackException,
|  +--IllegalArgumentException,
|  +--IllegalMonitorStateException,
|  +--IllegalPathStateException,
|  +--IllegalStateException,
|  +--ImagingOpException,
|  +--IndexOutOfBoundsException,
|  +--MissingResourceException,
|  +--NegativeArraySizeException,
|  +--NoSuchElementException,
|  +--NullPointerException,
|  +--ProfileDataException,
|  +--ProviderException,
|  +--RasterFormatException,
|  +--SecurityException,
|  +--SystemException,
|  +--UnsupportedOperationException
+--ServerNotActiveException,
+--SQLException,
+--TooManyListenersException,
+--UnsupportedFlavorException,
+--UnsupportedLookAndFeelException,
```