Exceções

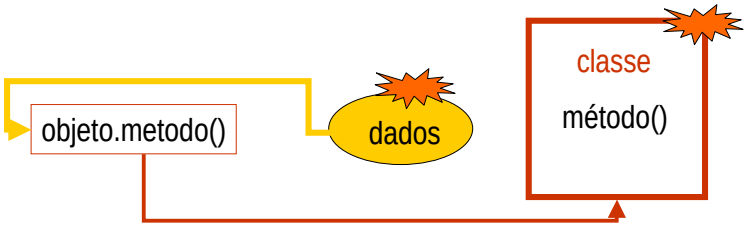
- Componentes robustos
- Problemas comuns
- Sistemas de tratamento de exceções
- Tratando exceções
- Classes de exceções
- Disparando exceções
- Criando novas classes de exceções

Componentes robustos

- Componentes podem ter problemas durante a execução e reagir como:
  - terminar o programa;
  - retornar um valor de erro indicando falha;
  - retornar e ignorar o problema;
  - chamar uma função para tratar o erro, etc...
- Problemas: erros e exceções
  - representam situações anormais (exceções) ou inválidas (erros) durante o processo de execução
- Componentes robustos: tentam sanar os problemas

Problemas comuns

- Os problemas mais comumente encontrados:
  - falha na aquisição de um recurso (new, open..)
  - tentativa de fazer algo impossível (divisão por zero, índice inválido..)
  - outras condições inválidas (lista vazia, overflow...)



Hipótese: ignorar exceções

Exemplo:

```
public class Zero {
    public static void main ( String[] args){
            int numerador = 10;
            int denominador = 0;
            System.out.println(numerador/denominador);
            } // main
    } // Zero
```

- execução: java.lang.ArithmeticException: / by zero
  at Zero.main(Zero.java:xx)
- xx: número da linha do arquivo onde ocorreu o erro

Sistemas de tratamento de exceções

- Antecipação de problemas pelo programador
- Situações de erro podem ser revertidas
- Solução ideal: tratamento de problemas separado do código normal
- Mecanismo: sistemas de tratamento de exceções
  - um sistema de tratamento de exceções deve ser capaz de: detectar e sinalizar (disparar), capturar e tratar uma exceção (ativar tratador)

Tratando exceções

- Dado o seguinte enunciado: Escreva um programa em JAVA que recebe dois strings, S1 e S2 e um número N pela linha de comando e imprima:
  - Os N primeiros caracteres do String S1 separados por um "-";
  - Os N primeiros caracteres do String S1 de traz para diante;
  - Os N primeiros caracteres do String S2 separados por um "-";
  - Os N primeiros caracteres do String S2 de traz para diante.

Tratando exceções

- Verifique se o programa a seguir corresponde a uma "boa solução":

```
class ImpString {
  public static void impSep(String str,int n){
   String aux = "";
   for(int i=0; i<n; i++){
       aux = aux + str.charAt(i);
       if (i < n-1) aux = aux + '-';
    }
   System.out.println(aux);
  }

public static void impInv(String str,int n){
   String aux = "";
    for(int i=n-1; i>=0; i--)
       aux = aux + str.charAt(i);
    System.out.println(aux);
  }
}
```

```
public class TestaExceptions{
  static public void main(String args[]){
    String s1,s2;
    int n;

    s1 = args[0];
    s2 = args[1];
    n  = Integer.parseInt(args[2]);

    ImpString.impSep(s1,n);
    ImpString.impInv(s1,n);
    ImpString.impSep(s2,n);
    ImpString.impInv(s2,n);
  }
}
```

Tratando exceções

- A solução anterior está correta se:
  - O usuário informar os 3 parâmetros na linha de comando;
  - O terceiro parâmetro for um número;
  - O valor de N for menor ou igual ao comprimento dos strings informados;

O TRATAMENTO DE SITUAÇÕES DE ERRO É FUNDAMENTAL EM APLICAÇÕES REAIS !!

Tratando exceções

- Exemplo:

Entrada: Teste exceções 4 **Saída:** T-e-s-t tseT e-x-c-e ecxe	**Entrada:** Teste exceções 10 **Saída:** java.lang.StringIndexOutOfBoundsException: String index out of range: 5
Entrada: Teste Teste Teste **Saída:** java.lang.NumberFormatException: Teste	**Entrada:** Teste **Saída:** java.lang.ArrayIndexOutOfBoundsException: 1

Solução convencional para o tratamento das situações de erro: acrescentou 14 linhas executáveis

```
public class TestaExceptions{
  static public boolean intOk(String s){
    for(int i=0; i<s.length(); i++)
       if (Character.isDigit(s.charAt(i)) == false) return(false);
    return(true);
  }
public static void main(String args[]){
    String s1,s2;
    int n;
    if (args.length != 3){
      System.out.println("Sintaxe: <string> <string> <int>");
      System.exit(0);
    }
   if (intOk(args[2]) == false){
      System.out.println("O terceiro parâmetro deve ser um inteiro");
      System.exit(0);
    }
```

(continua...)

Solução convencional para o tratamento das situações de erro: acrescentou 14 linhas executáveis

```
(...continua)

 s1 = args[0];  s2 = args[1];
 n  = Integer.parseInt(args[2]);
 if (n < s1.length()){
    ImpString.impSep(s1,n);
    ImpString.impInv(s1,n);
  }
 else System.out.println("O valor de n é maior que o tamanho de S1");

 if (n < s2.length()){
    ImpString.impSep(s2,n);
    ImpString.impInv(s2,n);
  }
 else System.out.println("O valor de n é maior que o tamanho de S2");
}
}
```


Um bloco try deve agrupar sentenças que tenham afinidade

- Exemplo 2: Acrescentou 8 linhas executáveis em relação a solução original

```
public class TestaExceptions{
  static public void main(String args[]){
  String s1="";
  String s2="";
  int n=0;
  try{
    s1 = args[0];
    s2 = args[1];
    n  = Integer.parseInt(args[2]);
  }
  catch(Exception e){
     System.out.println("Sintaxe: <string> <string> <int>");
     System.exit(0);
  }
```

```
try{
    ImpString.impSep(s1,n);
    ImpString.impInv(s1,n);
    ImpString.impSep(s2,n);
    ImpString.impInv(s2,n);
  }
catch(Exception e){
     System.out.println("O valor de N é maior que S1 ou S2");
     System.exit(0);
  }
 }
}
```

Tratando exceções

- Os métodos Java “geram uma exceção” quando, por uma razão qualquer, falham.
- O controle, então, passa imediatamente ao “gestor de exceções” apropriado.

```
try
{
  // código que pode gerar exceção
}
catch (Exception e)
{
  // código que trata exceção
}
finally
{
  // tratamento geral
}
```


Tratando exceções

- O comando try/catch/finally suporta o tratamento de exceções:
  - no bloco try estão colocados os comandos que podem provocar o lançamento de uma exceção
  - essas exceções são capturadas em um ou mais comandos catch, colocados após o bloco try
  - o comando finally contém código a ser executado, independente de outros comandos. É opcional, mas quando presente, é sempre executado

Tratando exceções

- Um programa JAVA deve tratar, na medida do possível, todas as exceções que puderem ser geradas de maneira que o programa não seja interrompido de forma anormal.
- Todas as exceções são derivadas da classe “Exception”.
- Uma vez lançada, uma exceção procura por uma cláusula catch capaz de referenciá-la e tratá-la. Se não houver, o programa irá parar com erro.

Tratando exceções

- Uma cláusula catch pode referenciar qualquer exceção do tipo que declara ou derivadas. No caso do exemplo anterior, as cláusulas catch poderiam interceptar qualquer exceção.
- Deve-se evitar tratar apenas as exceções mais genéricas porque as mensagens de erro tornam-se genéricas demais.
- Um único bloco try pode ter várias cláusulas catch.
- A ordem em que as cláusulas catch aparecem, importa. Por esta razão, as exceções mais genéricas devem ser tratadas após as mais específicas.

Observe o trecho inicial do exemplo anterior re-escrito para tratar as exceções específicas:

```
public class TestaExceptions{
  static public void main(String args[]){
    String s1="";
    String s2="";
    int n=0;
    try{
      s1 = args[0];
      s2 = args[1];
      n  = Integer.parseInt(args[2]);
      ImpString.impSep(s1,n);
      ImpString.impInv(s1,n);
      ImpString.impSep(s2,n);
      ImpString.impInv(s2,n);
    }
    catch(NumberFormatException e){
      System.out.println("O terceiro argumento deve ser um int");
    }
    catch(IndexOutOfBoundsException e){
      System.out.println("Sintaxe: <string> <string> <int>");
    }
    catch(Exception e){
      System.out.println("O valor de N é maior que S1 ou S2");
    }
  }
}
```

Observe que a ordem das cláusulas catch, importa !!!

Classes de exceções

- Exceções são objetos, instâncias de alguma sub-classe de java.lang.Throwable
  - podem conter dados
  - podem definir métodos
- Um dos dados é um String inicializado em tempo de criação do objeto, consultado por
  - Throwable.getMessage()
- Throwable tem duas sub-classes: Error e Exception.
  - Exceptions podem ser capturadas e tratadas
  - Errors correspondem a problemas mais graves, que não devem ser capturadas

O finally

- A cláusula finally é utilizada para forçar a execução de um bloco de código
- Pode ser utilizada com ou sem o bloco catch
- A cláusula finally é executada nas seguintes condições:
  - fim normal do método
  - devido a uma instrução return ou break
  - caso uma exceção tenha sido gerada

Uso do bloco finally

```
public class TestaExceptions{
 static public void main(String args[]){
   String s1="";
   String s2="";
   int n=0;
   try{
     s1 = args[0];
     s2 = args[1];
     n  = Integer.parseInt(args[2]);
     ImpString.impSep(s1,n);
     ImpString.impInv(s1,n);
     ImpString.impSep(s2,n);
     ImpString.impInv(s2,n);
   }  catch(NumberFormatException e){
     System.out.println("O terceiro argumento deve ser um int");
   }  catch(IndexOutOfBoundsException e){
     System.out.println("Sintaxe: <string> <string> <int>");
   }  catch (Exception e){
     System.out.println("O valor de N é maior que S1 ou S2");
   }  finally{
     System.out.println("Sempre passa por aqui !!");
   }
 }
}
```

O bloco finally sempre é executado (havendo ou não exceção) !!!

Hierarquia de exceções pré-definidas: (lançadas automaticamente pelo JAVA)

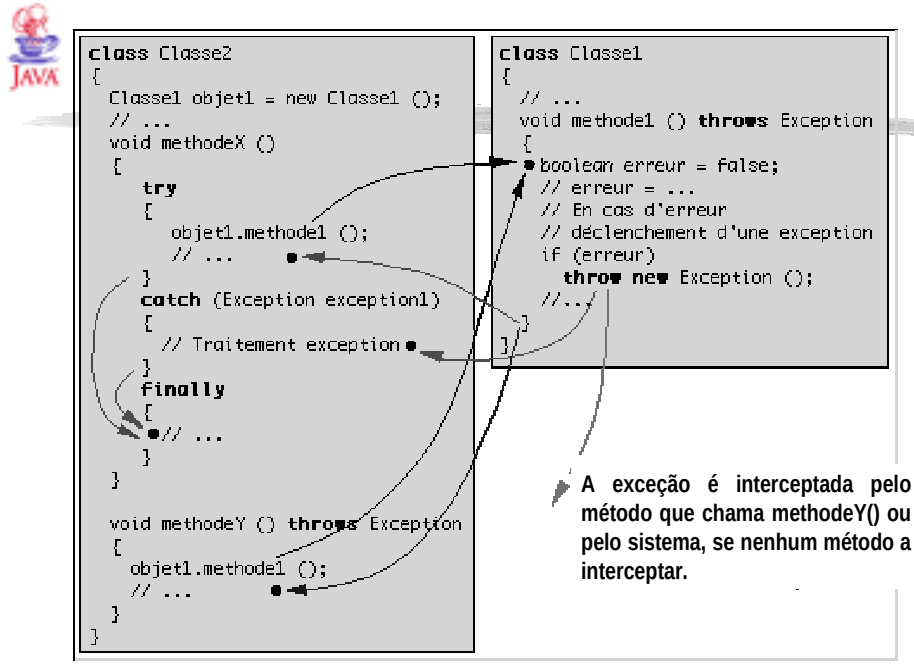
java.lang.Exception
- java.lang.ClassNotFoundException
- java.lang.CloneNotSupportedException
- java.lang.IllegalAccessException
- java.lang.InstantiationException
- java.lang.InterruptedException
- java.lang.NoSuchFieldException
- java.lang.NoSuchMethodException
- java.lang.RuntimeException
  - java.lang.ArithmeticException
  - java.lang.ArrayStoreExceptionclass
  - java.lang.ClassCastException
  - java.lang.IllegalArgumentException
    - java.lang.IllegalThreadStateException
    - java.lang.NumberFormatException
  - java.lang.IllegalMonitorStateException
  - java.lang.IllegalStateException
  - java.lang.IndexOutOfBoundsException
    - java.lang.ArrayIndexOutOfBoundsException
    - java.lang.StringIndexOutOfBoundsException
  - java.lang.NegativeArraySizeException
  - java.lang.NullPointerException
  - java.lang.SecurityException
  - java.lang.UnsupportedOperationException

Gerando exceções

- Para forçar a ocorrência de uma exceção, utiliza-se a palavra reservada throw (no singular)

```
public void metodo1( ) {
  try {
    metodo2( );
  }  catch (IOException e) {
     System.err.println(e);
  }
}
```

```
public void metodo2( ) throws IOException {
  if (...problema...)
     throw new IOException( );
}
```



Gerando exceções

```
public static void impSep(String str,int n) throws
  InterruptedException {
   String aux = "";
   if (n == 0)  throw(new InterruptedException());

   for (int i=0; i<n; i++){
        aux = aux + str.charAt(i);
        if (i < n-1) aux = aux + '-';
       }
   System.out.println(aux);
}
```

- throw funciona como um desvio para o local que trata a exceção
- Se throw não estiver localizado dentro de um bloco try-catch, então a rotina deverá declarar que lança determinada exceção.

Gerando exceções

```
public static void impSep(String str,int n) throws
   InterruptedException{
  String aux = "";
  if (n == 0)  throw(new InterruptedException());

  for (int i=0; i<n; i++){
    aux = aux + str.charAt(i);
    if (i < n-1) aux = aux + '-';
  }
  System.out.println(aux);
}
```

- Desta forma a exceção é passada “para cima”, ou seja, para a rotina chamadora e assim sucessivamente até encontrar um bloco catch adequado ou a rotina “main”.

Criando suas próprias exceções

```
class MinhaExcecao extends Exception{
  private int val;
  public MinhaExcecao(int n){
    super("Minha excecao: valor de val="+n);
    val = n;
  }
  public int getVal(){ return(val); }
}

class ImpString {
  public static void impSep(String str,int n) throws MinhaExcecao{
    String aux = "";
    if (n == 0)  throw(new MinhaExcecao(n));
    for(int i=0; i<n; i++){
      aux = aux + str.charAt(i);
      if (i < n-1) aux = aux + '-';
    }
    System.out.println(aux);
  }
```

(continua...)

Criando suas próprias exceções

```
(...continua)
 public static void impInv(String str,int n){
   String aux = "";
   for(int i=n-1; i>=0; i--)
     aux = aux + str.charAt(i);
   System.out.println(aux);
 }
}
public class TestaExceptions{
 static public void main(String args[]){
   String s1="";
   String s2="";
   int n=0;
   try{
     s1 = args[0];
     s2 = args[1];
     n  = Integer.parseInt(args[2]);
     ImpString.impSep(s1,n);
     ImpString.impInv(s1,n);
     ImpString.impSep(s2,n);
     ImpString.impInv(s2,n);
   }
```


(continua...)

Criando suas próprias exceções

```
   (...continua)

   catch(NumberFormatException e){
        System.out.println("O terceiro argumento deve ser um int");
      }
   catch(IndexOutOfBoundsException e){
        System.out.println("Sintaxe: <string> <string> <int>");
      }
   catch(MinhaExcecao e){
        System.out.println(e.getMessage());
        int x = e.getVal();
      }
   catch(Exception e){
        System.out.println("O valor de N é maior que S1 ou S2");
      }
    finally{
        System.out.println("É obrigado a passar por aqui");
      }
    }
   }
```

Exceções (resumo)

- Uma exceção é um sinal que indica a ocorrência de uma condição excepcional, como um erro.
- Gerar (throw) uma exceção é sinalizar a ocorrência de uma condição excepcional.
- Capturar (catch) uma exceção permite tratá-la, tomando ações necessárias para recuperar a condição correta de execução.
- Exceções propagam-se na estrutura léxica de blocos de um método e, se não capturadas, propagam-se para o método que a invocou e, sucessivamente, no stack de métodos.
- Caso não sejam capturadas, originam erro no main().

Exercícios

1) Faça um programa Java que solicite dois números ao usuário e, em seguida, imprime o resultado da divisão do primeiro pelo segundo. Trate a exceção "ArithmeticException". Teste o método "getMessage" e "printStackTrace" de "Exception" e observe o tipo de mensagem.
2) Re-escreva o exercício 1 de maneira que os números que compõem a razão sejam recebidos pela linha de comando. Trate todas as exceções necessárias.

Exercícios

3) Faça uma rotina que recebe um inteiro “n” por parâmetro e retorne uma referência para um vetor de “n” posições de inteiros. Trate a exceção “NegativeArraySizeException”.

4) Faça um programa que leia 2 strings e compara se o primeiro é lexicograficamente maior, menor ou igual ao segundo. Teste “NullPointerException” para o caso de um dos strings ser nulo.

5) Pesquise o significado das demais exceções.

Exercícios

6) Faça um programa que recebe 4 argumentos e os imprime. O programa deve gerar uma exceção específica caso o número de argumentos esteja incorreto. A mensagem exibida, neste caso, deve indicar a quantidade de argumentos passados e a quantidade correta.

7) Desenvolva um método em JAVA que recebe um string como parâmetro e verifica se o mesmo é composto apenas por caracteres maiúsculos. O método deve gerar 2 tipos de exceções específicas: uma para indicar se existe algum caracter que não é uma letra e outra para indicar se alguma das letras não é uma maiúscula. Para verificar o tipo dos caracteres use os métodos “isLetter” e “isUpperCase” da classe “Character” (ambos static).