Exceções

- **Exceções**: Erro durante a execução para o qual pode existir "*tratamento*".
- **Exemplo**: divisão por zero, falta de espaço suficiente para alocação de memória em dispositivo de armazenamento de dados, chamada de método na qual valor da expressão que denota objeto alvo é `null`, etc.


Exceções: Objetivos

- Permitir que execução de programa continue mesmo depois da ocorrência de erro.
- Permitir estrutura mais adequada para programas, separando código que define comportamento "normal" (esperado) do código que realiza tratamento de exceções.


Exceções em Java

Podem ser causadas por:

- Erro detectado pelo sistema de suporte a execução.
- Comando `throw` incluído pelo programador.

Em qualquer caso: exceção em Java é objeto que representa a situação de erro ocorrida, criado automaticamente no instante em que ocorre esse erro.


Tratador de Exceção

- Comando ou método em que uma exceção pode ocorrer pode (ou tem que) especificar um **tratador**.
- **Tratador**: bloco de comandos (possivelmente parametrizado) a ser executado se e quando a exceção ocorrer.


Tratamento de Exceções

Para especificar tratador, comando que pode provocar ocorrência da exceção deve ser colocado internamente a um comando `try`:

```
int x = 0;
try { System.out. print(x/0); }
catch (ArithmeticException e)
    { System.out.println
        ("Exemplo de tratamento de exceção"); }
```


Classes *Exception* e *Throwable*

- Exceções são objetos de subclasses da classe *Throwable*
- *Exception*, subclasse de *Throwable*, contém:
  - ⋆ Construtor com parâmetro de tipo *String*, para receber informações relativas à exceção ocorrida.
  - ⋆ Método *getMessage*, que recupera essa cadeia de caracteres.


Qual Tratador?

Exceção E ocorre durante execução de comando c em método (ou construtor) m; então:

- Execução é transferida para tratador mais interno, no corpo de m, que engloba c, caso exista esse tratador;
- Caso contrário, E é **propagada** para (isto é, considerada como causada no) ponto de chamada a m.


Exceções da classe *RuntimeException*

Considere método ou construtor p:

Exceções podem precisar ou não ser especificadas no cabeçalho de p sempre que puderem ocorrer em algum comando e não existir tratador correspondente em p.

- No 1º caso, cabeçalho de p deve conter cláusula `throws` E (que indica que chamada a p pode provocar ocorrência de E).
- No 2º caso, devem ser exceções de tipo-classe $RunTimeException$.
- Objetivos: conveniência do programador e clareza dos programas.


Exceções da classe *RuntimeException*

Nenhum tratador para *NumberFormatException* — subclasse de *RunTimeException* — que pode ser causada por *parseInt*:

```
class NenhumTratador_RunTimeException
{ public static void main (String[] a)
  { System.out.print (Integer.parseInt(a[0])); }
}
```


Exemplo de Tratamento de Exceções

Programa a seguir lê dois algarismos e imprime maior inteiro que se pode escrever com esses dois algarismos.

Método *charPraNum* recebe como argumento um caractere e retorna valor inteiro correspondente, caso esse caractere seja algarismo (0 a 9). Caso contrário, exceção *AlgarismoEsperado* é gerada — usando comando `throw` — e propagada por esse método, por meio do uso da cláusula `throws` no cabeçalho do método.


Exemplo: Gerando e Propagando Exceções

```
static int charPraNum (char c)
            throws AlgarismoEsperado
{ if (c>='0' && c<='9') return (c-'0');
  else throw new AlgarismoEsperado(c); }
```


Exemplo: subclasse de *Exception*

```
import javax.swing.*;

class AlgarismoEsperado extends Exception
{ AlgarismoEsperado(char c)
  { super ("Caractere digitado: " + c + "\n" +
           "Era esperado um algarismo."); }
}
```


Exemplo: código para exceções separado

Tratamento de exceções permite que trecho do programa que trata da execução normal (no caso, a entrada de dados corretos) fique separado do trecho de tratamento de exceções (contido na cláusula `catch`).

```
static void dialog ()
{ try
    { char c1 = (JOptionPane.showInputDialog ("Digite ...: ")).charAt(0);
      int  d1 = charPraNum(c1);
      char c2 = (JOptionPane.showInputDialog ("Digite ...: ")).charAt(0);
      int  d2 = charPraNum(c2);
      int  v  = maiorInt2Alg(d1,d2);
      JOptionPane.showMessageDialog (null, "Maior inteiro formado com " + c1 +
        " e " + c2 + " = " + v, "Resultado", JOptionPane.INFORMATION_MESSAGE);}
    catch (AlgarismoEsperado exc)
      { JOptionPane.showMessageDialog(null,exc.getMessage(),
                    "Erro",JOptionPane.ERROR_MESSAGE); dialog(); } }
```


Cláusula `finally`

- Comando em cláusula `finally` (de comando `try`) é sempre executado, ocorrendo ou não exceção.
- Se exceção E for causada e execução entrar em corpo da cláusula, pode ocorrer que essa execução termine:
  - ⋆ normalmente (após atingir fim da execução do último comando): nesse caso exceção E é propagada;
  - ⋆ anormalmente: nesse caso E é descartada. Pode ocorrer que nova exceção ocorra na cláusula `finally`: nesse caso ela é sinalizada (sobrepondo-se a E).

```
class Exemplo_finally
{ public static void m1 () throws Ex2
  { try { throw new Ex1(); } finally {throw new Ex2();} }
  public static void m2 () throws Ex1
  { try { throw new Ex1(); } finally {return;} }

  public static void m3 () throws Ex3
  { try { throw new Ex1();} catch (Ex1 e) {throw new Ex2();}
    finally {throw new Ex3();} }
  public static void m4 () throws Ex2
  { try { throw new Ex1(); } catch (Ex1 e) {throw new Ex2();}
    finally { ; /* nada */ } }

  public static void main (String[] a)
  { try {m1();} catch(Ex2 e) { System.out.print('a');}
    try {m2();} catch(Ex1 e) { System.out.print('b');}
    try {m3();} catch(Ex3 e) { System.out.print('c');}
    try {m4();} catch(Ex2 e) { System.out.print('d');}}}
```


Utilidade da cláusula `finally`

```
try { ... /* recursos alocados */
      ... /* usados */ ...
catch (...)  { ... }
catch (...)  { ... }
finally { ...
          /* e liberados (com ou sem exceções) */
          ...}
```