

Tratamento de Exceção

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC

Exceções

- Definição

- Uma exceção é uma indicação de um problema que ocorre durante a execução de um programa. Representa uma situação que normalmente não ocorre e representa algo de estranho ou inesperado no sistema.

Exceções

- Tratamento de Exceções

- Forma elegante e simples para tratamento de condições excepcionais

- Entrada de dados inválida;
 - Falhas no tratamento de arquivos;
 - Falhas na comunicação entre processos;
 - Reativação de threads;
 - Erros aritméticos;
 - Estouro de limites de arrays;
 - etc.

Exceções

■ Definição

- O uso de exceções permite separar a **detecção da ocorrência** de uma situação excepcional do seu **tratamento**, ao se programar um método em Java.
 - Separa “detectar” e “tratar”
 - Códigos separados na implementação

Tratamento de Exceções

```
1 Realize uma tarefa
2 Se a tarefa anterior não tiver sido executada corretamente
3     Realize processamento de erro
4
5 Realize a próxima tarefa
6 Se a tarefa anterior não tiver sido executada corretamente
7     Realize processamento de erro
8
9 ...
10
```

Neste pseudocódigo, começamos realizando uma tarefa; depois testamos se essa tarefa foi executada corretamente. Se não tiver sido, realizamos processamento de erro. Caso contrário, continuamos com a próxima tarefa.

Essa forma de tratamento de erro funciona. Contudo, quais os problemas inerentes?

Tratamento de Exceções

```
...
codigoErro = 0;
operacao1;
if (!erro1) {           //teste a ocorrencia de um tipo de erro
    operacao2;          //não ocorreu o erro 1
    if (!erro2) {       //teste a ocorrencia de outro tipo de erro
        operacao3;      //não ocorreu o erro 2
        if (!erro3) {   //outro tipo de erro: mais um teste
            ...          //não ocorreu o erro 3
        }
        else {
            codigoErro = -3;
        }
    } else {
        codigoErro = -2;
    }
} else {
    codigoErro = -1;
}
if (codigoErro < 0)
    tratarErro(codigoErro); //tomar as medidas necessarias
...
```

Tratamento de Exceções

■ Problemas

- Legibilidade do código em si é ruim
- Problemas em relação aos métodos
 - O tratamento dado a uma exceção em uma aplicação pode não ser o mesmo necessário considerando-se uma aplicação diferente. Neste caso, seria necessário alterar o código de tratamento. Isso limita a flexibilidade de se lidar com situações de exceção

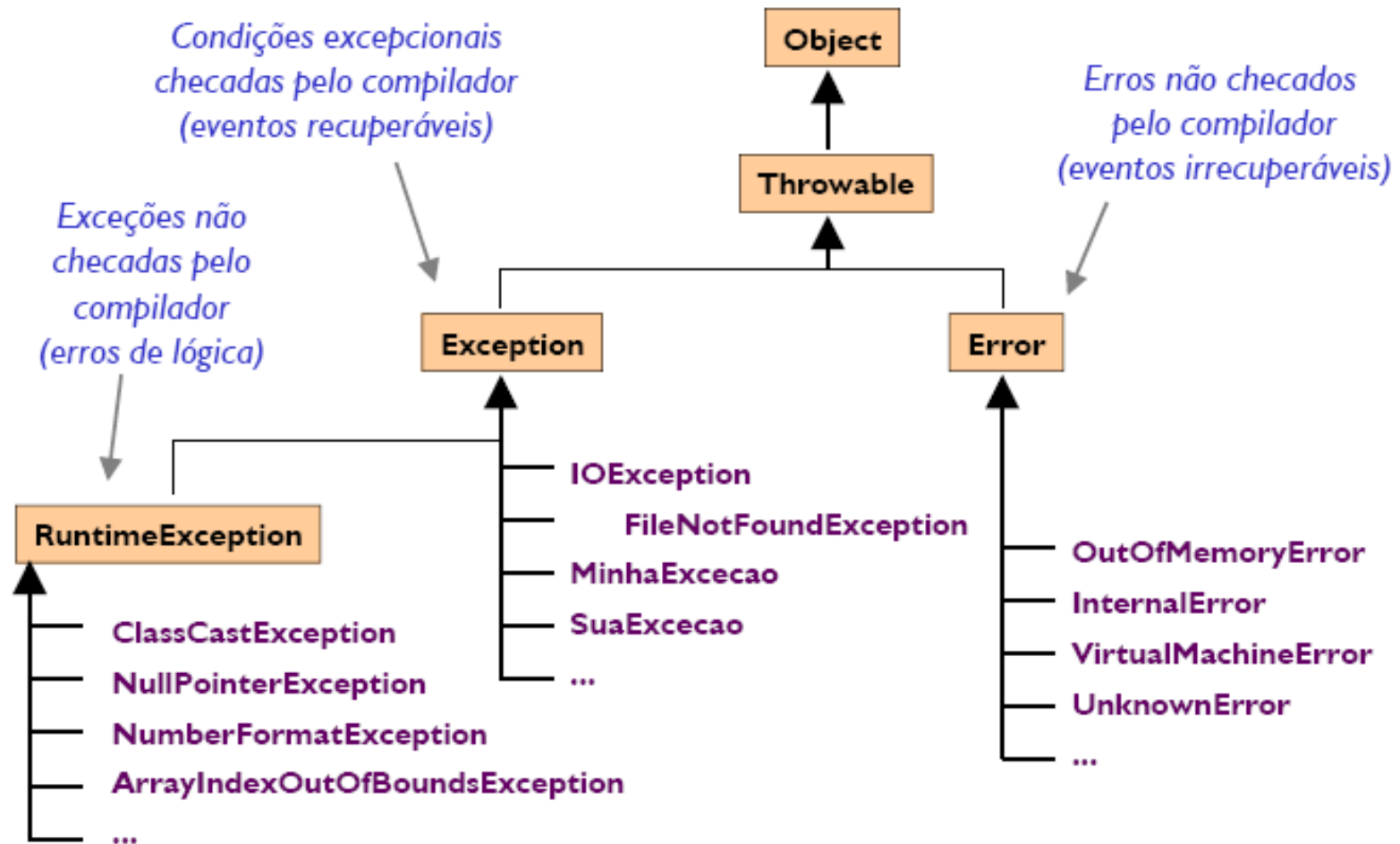
Tratamento de Exceções

```
...  
try {  
    operacao1;  
    operacao2;  
    operacao3;  
} catch (erro1) {  
    //tomar as medidas necessarias para a tratar o erro1  
} catch (erro2) {  
    //tomar as medidas necessarias para a tratar o erro2  
} catch (erro3) {  
    //tomar as medidas necessarias para a tratar o erro3  
}  
...
```

O código apresentado aqui é mais legível,
manipulável e menos propenso a erros de
programação que o anteriormente
ilustrado

Tratamento em Java

Hierarquia



Herarquia

■ RuntimeException e Error

- Exceções não verificadas em tempo de compilação
- Subclasses de Error não devem ser capturadas
 - São situações graves em que a recuperação é impossível ou indesejável
- Subclasses de RuntimeException\
 - Representam erros de lógica de programação
 - Devem ser corrigidos (podem, mas não devem ser capturadas)

■ Exception

- Exceções verificadas em tempo de compilação
 - Exceção à regra são as subclasses de RuntimeException
- Compilador exige que sejam ou capturadas ou declaradas
 - Pelo método que potencialmente as provoca

TE em Java

■ Exceções

- Erros de tempo de execução
- Objetos criados a partir de classes especiais
- Esses objetos são "lançados" quando ocorrem condições excepcionais

■ Métodos

- Podem capturar ou deixar passar exceções que ocorrerem em seu corpo. É obrigatório, para a maior parte das exceções, que o método declare quaisquer exceções que ele não capturar. Mecanismo **try-catch** é usado para tentar capturar exceções enquanto elas passam por métodos

TE em Java

■ Tipos de Erros

- 1. Erros de lógica de programação
 - Ex: limites do vetor ultrapassados, divisão por zero
 - Devem ser corrigidos pelo programador
- 2. Erros devido a condições do ambiente de execução
 - Ex: arquivo não encontrado, rede fora do ar, etc.
 - Fogem do controle do programador
 - Podem ser contornados em tempo de execução
- 3. Erros graves onde não adianta tentar recuperação
 - Ex: falta de memória, erro interno do JVM
 - Fogem do controle do programador e não podem ser contornados

TE em Java

■ Exceção

- É um tipo de objeto que sinaliza que uma condição excepcional ocorreu
- A identificação (nome da classe) é sua parte mais importante
- Precisa ser criada com **new** e depois lançada com **throw**

```
IllegalArgumentException e = new  
    IllegalArgumentException("Erro!");  
throw e; // exceção foi lançada!
```



Mais usual

```
throw new IllegalArgumentException("Erro!");
```

TE em Java

■ Declaração throws

- Uma declaração **throws** (observe o "s") é obrigatória em métodos e construtores que deixam de capturar uma ou mais exceções que ocorrem em seu interior
 - `public void m() throws Excecao1, Excecao2 {...}`
 - `public Circulo() throws ExcecaoDeLimite {...}`
- `throws` declara que o método pode provocar exceções do tipo declarado
 - Ou de qualquer subtipo
- A declaração abaixo declara que o método pode provocar qualquer exceção
 - `Public void m() throws Exception {...}`
 - Nunca faça isto

Exceção

- O que acontece
 - Uma exceção lançada interrompe o fluxo normal do programa
 - O fluxo do programa segue a exceção
 - Se o método onde ela ocorrer não a capturar...
 - ... ela será propagada para o método que chamar esse método
 - ... e assim por diante
 - Se ninguém capturar a exceção, ela irá causar o término da aplicação
 - Se em algum lugar ela for capturada, o controle pode ser recuperado

TE em Java

```
public class RelatorioFinanceiro {  
    public void metodoMau() throws ExcecaoContabil {  
        if (!dadosCorretos) {  
            throw new ExcecaoContabil("Dados Incorretos");  
        }  
    }  
    public void metodoBom() {  
        try {  
            ... instruções ...  
            metodoMau();  
            ... instruções ...  
        } catch (ExcecaoContabil ex) {  
            System.out.println("Erro: " + ex.getMessage());  
        }  
        ... instruções ...  
    }  
}
```

*instruções que sempre
serão executadas*

*instruções serão executadas
se exceção não ocorrer*

*instruções serão executadas
se exceção não ocorrer ou
se ocorrer e for capturada*

Mecanismo Geral

■ Funcionamento

- No momento de uma exceção, o método onde ocorreu a exceção aborta
 - O controle da execução retorna ao método que o chamou
 - O objeto exceção que foi construído é enviado ao método chamador
-
- O método onde ocorreu o erro "lança" a exceção para o método que o chamou. Quem deverá capturar o erro e tratá-lo é uma função que usa esta função para funcionar. Em outras palavras, a função onde o erro aconteceu não trata o erro, simplesmente avisa que o erro aconteceu e interrompe seu processamento

Mecanismo Geral

Suponha um método qualquer (por exemplo, `main()`) que chama um método `g()`:

```
public static void main (String[] args) {  
    .....  
    .....  
    g() ;  
    .....  
    .....  
}
```

Método `g()` é chamado dentro
do método `main()`

Suponha também que, de dentro de `g()`, o método `f()` seja chamado:

```
public void g() {  
    .....  
    .....  
    f() ;  
    .....  
    .....  
}
```

Método `f()` é chamado dentro
do método `g()`

Mecanismo Geral

```
public void f() {  
    if (<teste da condição de erro A>) {  
        //constrói uma exceção da classe ExcecaoA e lança para quem chamou f()  
        throw new ExcecaoA(...lista de argumentos...);  
  
    else if (<teste da condição de erro B>) {  
        //constrói uma exceção da classe ExcecaoB e lança para quem chamou f()  
        throw new ExcecaoB(...lista de argumentos...);  
    }  
  
    else ....<testando outros possíveis erros e procedendo de forma similar>  
  
    else {  
        <comandos para processar f() em condições normais sem erro>  
    }  
}
```

Mecanismo Geral

```
public void f() {  
    if (<teste da condição de erro A>) {  
        //constrói uma exceção da classe ExcecaoA e lança para quem chamou f()  
        throw new ExcecaoA(...lista de argumentos...);  
  
    else if (<teste da condição de erro B>) {  
        //constrói uma exceção da classe ExcecaoB e lança para quem chamou f()  
        throw new ExcecaoB(...lista de argumentos...);  
    }  
  
    else ....<testando outros possíveis erros e procedendo de forma similar>  
  
    else {  
        <comandos para processar f() em condições normais sem erro>  
    }  
}
```

Agora o método f() não precisa mais **determinar o que fazer** quando cada caso de erro ocorrer.

Ele precisa apenas detectar que o caso de erro ocorreu. A partir daí, ele constrói, e "**lança**" para o método g() que o chamou, um **objeto especial da classe Exception** (ou de alguma subclasse).

De outro modo, teria que ser explicitamente definido no método f() o tratamento para o erro, que poderia ser imprimir uma mensagem

Captura de Exceções

```
public static void main (String[] args) {  
    .....  
    g();  
    .....  
    .....  
}
```

Método g() é chamado dentro
do método main()

```
public void g() {  
    .....  
    f();  
    .....  
    .....  
}
```

Método f() é chamado dentro
do método g()

Captura de Exceções

```
public void g() throws ExcecaoA {  
    .....  
    .....  
    try{  
        f();  
    }  
    catch(ExcecaoB exb) {  
        ....comandos para examinar a exceção referenciada por exb...  
        ....comandos para tratar o erro B...  
        .....  
    }  
    .....  
    .....  
}
```

Gerado por

Capturada/Tratada por

Exceções do tipo B são geradas pelo método f() e capturadas e tratadas pelo método g()

Captura de Exceções

```
public static void main (String[] args) {  
    .....  
    .....  
    try{  
        g();  
    }  
    catch (ExcecaoA exa) {  
        ....comandos para examinar a exceção referenciada por exa...  
        ....comandos para tratar o erro A...  
        .....  
    }  
    .....  
    .....  
}
```

Gerado por

Capturada/Tratada por

The diagram illustrates the flow of an exception. A red circle highlights the `g();` statement within the `try` block, with an arrow pointing to it from the text "Gerado por". Another red circle highlights the `main` method signature, with an arrow pointing to it from the text "Capturada/Tratada por". A red rectangle highlights the `catch (ExcecaoA exa) {` block, which is connected to the `main` method by a red line, indicating that the exception is caught and handled within the `main` method.

Exceções do tipo A são geradas pelo método `g()` e capturadas e tratadas pelo método `main()`

Captura de Exceções

```
public static void main (String[] args) {  
    .....  
    .....  
    try{  
        g();  
    }  
    catch (ExcecaoA exa) {  
        ....comandos para examinar a exceção referenciada por exa...  
        ....comandos para tratar o erro A...  
        .....  
    }  
    .....  
    .....  
}
```

Traduzindo:

Tente executar o método `g()`. Se algo der errado na execução do mesmo, então é gerado a exceção **EXA** do tipo **ExcecaoA**. O objeto/excecao **EXA** é capturado pelo método `main()`

Try e Catch

■ Try

- O bloco try "tenta" executar um bloco de código que pode causar exceção
- Deve ser seguido por
 - Um ou mais blocos catch(TipoDeExcecao ref)
 - E/ou um bloco finally

■ Catch

- Blocos catch recebem tipo de exceção como argumento
- Se ocorrer uma exceção no try
 - Ela irá descer pelos catch até encontrar um que declare capturar exceções de uma classe ou superclasse da exceção. Apenas um dos blocos catch é executado

Try e Catch

é usada para indicar um bloco de código que possa ocorrer uma exceção.

```
public static void main (String[] args) {  
    .....  
    .....  
    try{  
        g();  
    }  
    catch(ExcecaoA exa) {  
        ....comandos para examinar a exceção referenciada por exa...  
        ....comandos para tratar o erro A...  
        .....  
    }  
    .....  
    .....  
}
```

Serve para manipular as exceções
Tratar o erro

Finally

- Finally

- É opcional
- Pode ser usado para liberar recursos adquiridos no bloco try
 - Ex: fechar um arquivo que foi aberto
 - Ex: encerrar uma conexão com um banco de dados
- OBS: quando ocorre um erro, o método aborta sua execução e não volta mais de onde parou. Se um arquivo foi aberto, ele não será fechado. Se uma conexão for aberta, ela não será encerrada.

Try/Catch/Finally

```
try {  
    Bloco de instruções ..  
}  
catch (instância de uma subclasse de Exception) {  
    Bloco de instruções ..  
}  
finally {  
    Bloco de instruções ..  
}
```

Try/Catch/Finally

```
try {  
    // Código que pode gerar uma exceção  
}
```

```
catch (UmTipoDeExcecao excecao1) {  
    // Código para processar a exceção  
}
```

```
catch (OutroTipoDeExcecao excecao2) {  
    // Código para processar a exceção  
}
```

```
finally {  
    // Código que sempre será executado  
}
```

Try/Catch/Finally

A execução começa no início do bloco

Não ocorrendo uma exceção, o bloco **Finally** é executado após a execução do bloco **Try**. Havendo um **return** no bloco **Try**, o Bloco **Finally** sempre será executado antes da saída do método **Try**


Se não houver **return** no bloco **Try** nem no bloco **Finally**, a execução continua com o código após o bloco **Finally**

```
try {  
    // Código que pode gerar uma exceção  
}
```

```
catch (UmTipoDeExcecao excecao1) {  
    // Código para processar a exceção  
}
```

```
catch (OutroTipoDeExcecao excecao2) {  
    // Código para processar a exceção  
}
```

```
finally {  
    // Código que sempre será executado  
}
```



Try/Catch/Finally

A execução começa no início do bloco

```
try {  
    // Código que pode gerar uma exceção  
}
```

A execução é interrompida no ponto onde foi gerado a exceção. O controle passa para o bloco **catch** que irá capturá-la

```
catch (UmTipoDeExcecao excecao1) {  
    // Código para processar a exceção  
}
```

O bloco **finally** será executado logo após o bloco **catch**

```
catch (OutroTipoDeExcecao excecao2) {  
    // Código para processar a exceção  
}
```

A execução continua com o código após o bloco **Finally**, caso não exista **return**

```
finally {  
    // Código que sempre será executado  
}
```


Exemplo

```
class Vetor13elf {  
  
    public static void main(String[] a) {  
        int[] vet = new int[3];  
        try {  
            for (int c=0; c<4; c++) vet[c] = 0;  
        } catch (Exception e) {  
            System.out.println("Houve um erro geral! Tratando ..");  
        } finally {  
            System.out.println("Finalmente! Liberando recursos ..");  
        }  
        System.out.println("Fim do programa.");  
    }  
}
```



```
Houve um erro geral! Tratando ..  
Finalmente! Liberando recursos ..  
Fim do programa.
```

Try e Catch

```
try {  
    ... instruções ...  
} catch (TipoExcecao1 ex) {  
    ... faz alguma coisa ...  
} catch (TipoExcecao2 ex) {  
    ... faz alguma coisa ...  
} catch (Exception ex) {  
    ... faz alguma coisa ...  
}  
  
... continuação ...
```

Try e Catch

- Observações

- O bloco try não pode aparecer sozinho
 - Deve ser seguido por pelo menos um catch ou por um finally
- O bloco finally contém instruções que devem se executadas
 - Independentemente da ocorrência ou não de exceções

Finally

```
try {  
    // instruções: executa até linha onde ocorrer exceção  
} catch (TipoExcecao1 ex) {  
    // executa somente se ocorrer TipoExcecao1  
}  
catch (TipoExcecao2 ex) {  
    // executa somente se ocorrer TipoExcecao2  
}  
finally {  
    // executa sempre ...  
}  
  
// executa se exceção for capturada ou se não ocorrer
```

Capturando erros

```
class Teste {  
    public static void main(String[] args) {  
        Conta c = new Conta();  
  
        try {  
            c.deposita();  
        } catch (IllegalArgumentException e) {  
            System.out.println("Houve um erro ao depositar");  
        }  
    }  
}
```

```
class Teste {  
    public static void main(String[] args) {  
        Conta c = new Conta();  
  
        try {  
            c.deposita();  
        } catch (IllegalArgumentException e) {  
            System.out.println("Houve uma IllegalArgumentException ao depositar");  
        } catch (SQLException e) {  
            System.out.println("Houve uma SQLException ao depositar");  
        }  
    }  
}
```

Cavando a Cova

■ Problema

- Não tratar exceções e simplesmente declará-las em todos os métodos evita trabalho, mas torna o código menos robusto. Mas o pior que um programador pode fazer é capturar exceções e não fazer nada, permitindo que erros graves passem despercebidos e causem problemas difíceis de localizar no futuro.

NUNCA escreva o seguinte código:

```
try {  
    // .. código que pode causar exceções  
} catch (Exception e) {}
```

Cavando a Cova

- Problema grave
 - Ele pega até `NullPointerException`, e não diz nada.
 - O programa trava ou funciona de modo estranho
 - Ninguém saberá a causa a não ser que mande imprimir o valor de `e`,
 - O que poderia ser pior?

Criando Exceções

Modelando Exceções

```
1 public class Classe1{  
2  
3     public int div(int a, int b){  
4         return a/b;  
5     }  
6 }
```

O que acontecerá se o método "div" for chamado passando-se o parâmetro "b" como "0" ?

Modelando Exceções

- Criar Exceção
 - Basta estender `java.lang.Exception`:
 - `class NovaExcecao extends Exception {}`
 - Não precisa de mais nada. O mais importante é:
 - Herdar de `Exception`
 - Fornecer uma identificação diferente
 - Bloco `catch` usa nome da classe para identificar exceções.


Modelando Exceções

```
1 public class DivByZero extends Exception {  
2  
3     public String toString() {  
4         return "Division By Zero";  
5     }  
6  
7 }
```

Modelando Exceções

```
1 public class Classe1{  
2  
3     public int div(int a, int b){  
4         return a/b;  
5     }  
6 }
```

Alterando a classe



```
1 public class Classe1{  
2  
3     public int div(int a, int b) throws DivByZero{  
4  
5         if(b==0){  
6             throw new DivByZero();  
7         }  
8  
9         return a/b;  
10    }  
11 }
```

Modelando Exceções

```
1 public class DivByZero extends Exception {  
2  
3     public String toString(){  
4         return "Division By Zero";  
5     }  
6  
7 }
```



```
1 public class Classe1{  
2  
3     public int div(int a, int b) throws DivByZero{  
4  
5         if(b==0){  
6             throw new DivByZero();  
7         }  
8  
9         return a/b;  
10    }  
11 }
```

Modelando Exceções

Throws: usado na definição de um método para dizer que o mesmo dispara exceções de algum tipo

```
1 public class Classe1{  
2  
3     public int div(int a, int b) throws DivByZero{  
4  
5         if(b==0){  
6             throw new DivByZero();  
7         }  
8  
9         return a/b;  
10    }  
11 }
```

Throws deve indicar os tipos de exceções ele pode retornar

Throw: comando usado para, literalmente, lançar a exceção

Modelando Exceções

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         Classe1 cl1 = new Classe1();  
6  
7         try{  
8  
9             int resultado = cl1.div(10, 0);  
10            System.out.println(resultado);  
11  
12        } catch (DivByZero e) {  
13            System.out.println(e);  
14        }  
15    }  
16 }
```

Bloco TRY/CATCH

Tenta executar o que está definido no bloco TRY.
Se não executar (por algum erro), então execute o
bloco CATCH

Modelando Exceções

```
1 public class Teste {  
2  
3     public static void main(String args[]) {  
4  
5         Classe1 c11 = new Classe1();  
6  
7         try {  
8  
9             int resultado = c11.div(10, 0);  
10            System.out.println(resultado);  
11  
12        } catch (DivByZero e) {  
13            System.out.println(e);  
14        }  
15    }  
16 }
```

Problems Javadoc Declaration Console Progress
<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\
Error: Division By Zero

Modelando Exceções

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         Classe1 c11 = new Classe1();  
6  
7         try{  
8  
9             int resultado = c11.div(10, 2);  
10            System.out.println(resultado);  
11  
12        } catch (DivByZero e){  
13            System.out.println(e);  
14        }  
15    }  
16 }
```

Problems Javadoc Declaration Console Progress
<terminated> Teste [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\java
5

Tratamento de Exceção

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC