



Universidade Federal do ABC  
Centro de Matemática, Computação e Cognição

# Tratamento de Exceções

Monael Pinheiro Ribeiro, D.Sc.

# Exceções

- É uma indicação de um problema que ocorre durante a execução de um programa;
- O tratamento permite ao programador criar programas que resolve ou trata exceções às regras;
- Essa prática permite que os programas ao encontrarem um erro não terminem de forma abrupta e continuem executando com objetivo de recuperar-se da exceção ou simplesmente terminar de forma mais elegante;

# Exceções

- Geralmente um programa testa condições que determinam como a execução de um programa deve prosseguir;

Instrução 1;

Se Erro 1 ocorrer então

Realize um processamento do Erro 1;

Instrução 2;

Se Erro 2 ocorrer então

Realize um processamento do Erro 2;

...

- Embora essa forma de tratamento de erro funcione corretamente, mesclar a lógica do programa com a lógica do tratamento de erro dificulta sua legibilidade, manutenabilidade e depuração

# Exemplo

```
double divisao(int a, int b)
{
    return double(a)/b;
}

int main()
{
    int a, b;
    while(std::cin >> a >> b)
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    return 0;
}
```

# Exemplo

```
double divisao(int a, int b)
{
    return double(a)/b;
}

int main()
{
    int a, b;
    while(std::cin >> a >> b)
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    return 0;
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

# Exemplo

```
double divisao(int a, int b)
{
    return double(a)/b;
}
```

```
int main()
{
    int a, b;
    while(std::cin >> a >> b)
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    return 0;
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

```
Resultado: 45
Resultado: 30
Resultado: 0
Resultado: inf
Resultado: 3
Resultado: -3
Resultado: -inf
```

# Criando uma Classe de Exceção

- Em C++

```
#include <stdexcept>
```

```
class DivideByZeroException : public runtime_error
```

```
{
```

```
    public:
```

```
        DivideByZeroException():runtime_error("Erro divisao por zero")
```

```
        { }
```

```
};
```

# Criando uma Classe de Exceção

- Em C++

```
#include <stdexcept>
```

```
class DivideByZeroException : public runtime_error
{
    public:
        DivideByZeroException():runtime_error("Erro divisao por zero")
        { }
};
```

- `DivideByZeroException` deriva de `runtime_error` da biblioteca padrão.
- `runtime_error` deriva de `exception` também da biblioteca padrão.
- Não é obrigatória essa derivação, mas é recomendada, pois padroniza uma forma de tratamento de exceções.
- O construtor de `runtime_error` recebe uma string com a descrição do erro.
- `exception` fornece um método virtual `what()` que retorna uma string com a descrição do erro.



# Lançando uma Exceção

- Em C++

```
double divisao(int a, int b)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```

# Lançando uma Exceção

- Em C++

```
double divisao(int a, int b)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```

- O comando `throw` lança uma exceção.
- O comando `throw` espera um objeto que será lançado.
- Ao encontrar um comando `throw`, o fluxo é interrompido e o objeto é lançado para eventual captura e tratamento.

# Capturando uma Exceção

- Em C++

```
...
while(std::cin >> a >> b)
{
    try
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    catch(DivideByZeroException e)
    {
        std::cout << e.what() << std::endl;
    }
}
```

# Capturando uma Exceção

- Em C++

```
...
while(std::cin >> a >> b)
{
    try
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    catch(DivideByZeroException e)
    {
        std::cout << e.what() << std::endl;
    }
}
```

- O comando `try` define um bloco de código onde exceções podem ocorrer.
- Um comando `try` tem um ou muitos comandos `catch`.
- Cada comando `catch` tem como argumento o tipo da exceção que ele trata.
- Geralmente o bloco de código de um comando `catch` informa o erro ao usuário, registra o erro em um log, termina a execução de forma elegante ou busca alternativas para contornar o erro ocorrido.

# Capturando uma Exceção

- Em C++

```
try
{
    doSomething();
}
catch(ExceptionType1 e)
{ /* bloco que trata a excecao 1*/ }
catch(ExceptionType2 e)
{ /* bloco que trata a excecao 2*/ }
catch(ExceptionType3 e)
{ /* bloco que trata a excecao 3*/ }
catch(ExceptionType4 e)
{ /* bloco que trata a excecao 4*/ }
catch(ExceptionType5 e)
{ /* bloco que trata a excecao 5*/ }
catch(ExceptionType6 e)
{ /* bloco que trata a excecao 6*/ }
catch(ExceptionType7 e)
{ /* bloco que trata a excecao 7*/ }
```

# Capturando uma Exceção

- Em C++

```
...
while(std::cin >> a >> b)
{
    try
    {
        std::cout << "Resultado: " << divisao(a, b) << std::endl;
    }
    catch(DivideByZeroException e)
    {
        std::cout << e.what() << std::endl;
    }
}
```

- O comando `try` define um bloco de código onde exceções podem ocorrer.
- Um comando `try` tem um ou muitos comandos `catch`.
- Cada comando `catch` tem como argumento o tipo da exceção que ele trata.
- Geralmente o bloco de código de um comando `catch` informa o erro ao usuário, registra o erro em um log, termina a execução de forma elegante ou busca alternativas para contornar o erro ocorrido.

# Especificando uma Lista de Exceções

- Em C++
  - Também chamada de lista `throw` é opcional e enumera uma lista de exceções que o método pode lançar.
  - Caso a lista `throw` seja omitida, significa que o método pode lançar qualquer exceção.
  - Caso a lista `throw` seja declarada como vazia, significa que o método não lança nenhuma exceção.
  - Caso o método lance uma exceção fora da sua lista `throw` ou quando tem lista `throw` vazia, então a função `unexpected()` é invocada.
  - Já se a exceção não tiver um `catch` correspondente, então a função `terminate()` é invocada.
  - `unexpected()` por sua vez invoca `terminate()` que invoca `abort()` sem chamar nenhum destrutor e pode levar a vazamento de recursos.

# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```



# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```

 **Pode lançar qualquer exceção.**

# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b) throw(DivideByZeroException)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```

# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b) throw(DivideByZeroException)
{
    if(b==0)
    {
        throw DivideByZeroException();
    }
    return double(a)/b;
}
```



**Lança exceções da Lista.**

# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b) throw()  
{  
    if(b==0)  
    {  
        throw DivideByZeroException();  
    }  
    return double(a)/b;  
}
```

# Especificando uma Lista de Exceções

- Em C++

```
double divisao(int a, int b) throw()  
{  
    if(b==0)  
    {  
        throw DivideByZeroException();  
    }  
    return double(a)/b;  
}
```

 **Não lança nenhuma exceção**

# Exemplo com Tratamento de Exceção

- Em C++

```
class DivideByZeroException : public runtime_error
{
    public:
        DivideByZeroException():runtime_error("Erro divisao por zero")
        { }
};
```

```
double divisao(int a, int b) throw (DivideByZeroException)
{
    if(b==0)
        throw DivideByZeroException();
    return double(a)/b;
}
```

```
int main()
{
    int a, b;
    while(std::cin >> a >> b)
    {
        try
        {
            std::cout << "Resultado: " << divisao(a, b) << std::endl;
        }
        catch(DivideByZeroException e)
        {
            std::cout << e.what() << std::endl;
        }
    }
    return 0;
}
```

# Exemplo com Tratamento de Exceção

- Em C++

```
class DivideByZeroException : public runtime_error
{
    public:
        DivideByZeroException():runtime_error("Erro divisao por zero")
        { }
};
```

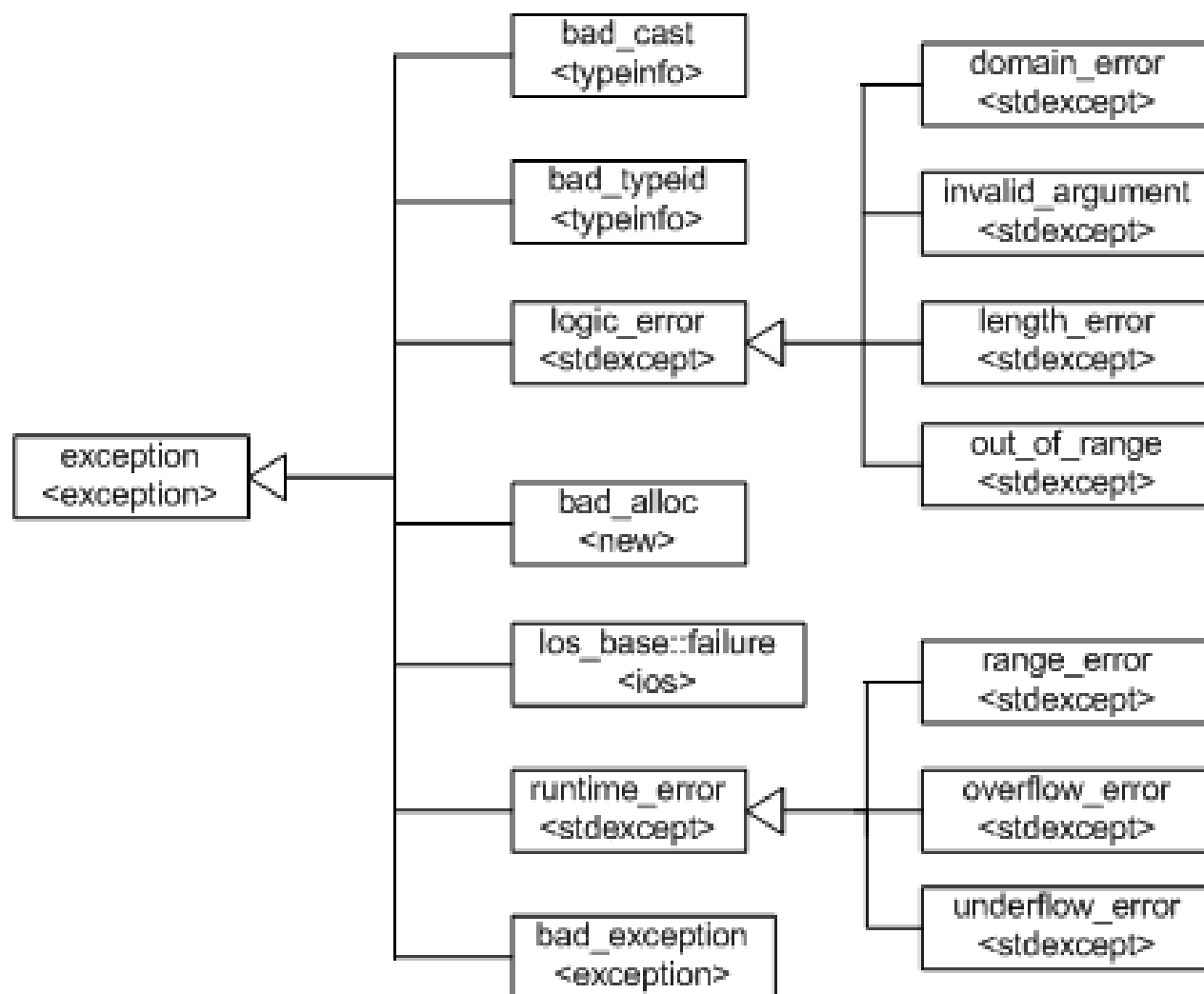
```
double divisao(int a, int b) throw (DivideByZeroException)
{
    if(b==0)
        throw DivideByZeroException();
    return double(a)/b;
}
```

```
int main()
{
    int a, b;
    while(std::cin >> a >> b)
    {
        try
        {
            std::cout << "Resultado: " << divisao(a, b) << std::endl;
        }
        catch(DivideByZeroException e)
        {
            std::cout << e.what() << std::endl;
        }
    }
    return 0;
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

```
Resultado: 45
Resultado: 30
Resultado: 0
Erro divisao por zero
Resultado: 3
Resultado: -3
Erro divisao por zero
```

# Hierarquia de Exceções C++





# Exemplo em JAVA

```
public class Main
{
    public static double divisao(int a, int b)
    {
        return a/(double)b;
    }

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            System.out.println("Resultado: " + divisao(a,b));
        }
    }
}
```

# Exemplo em JAVA

```
public class Main
{
    public static double divisao(int a, int b)
    {
        return a/(double)b;
    }

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            System.out.println("Resultado: " + divisao(a,b));
        }
    }
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

# Exemplo em JAVA

```
public class Main
{
    public static double divisao(int a, int b)
    {
        return a/(double)b;
    }

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            System.out.println("Resultado: " + divisao(a,b));
        }
    }
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

```
Resultado: 45
Resultado: 30
Resultado: 0
Resultado: Infinity
Resultado: 3
Resultado: -3
Resultado: -Infinity
```

# Criando uma Classe de Exceção

- Em JAVA

```
class DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Erro divisao por zero");
    }
};
```

# Lançando uma Exceção

- Em JAVA

```
public static double divisao(int a, int b) throws DivideByZeroException
{
    if(b==0)
    {
        throw new DivideByZeroException();
    }
    return a/(double)b;
}
```

# Capturando uma Exceção

- Em JAVA

```
...
try
{
    System.out.println("Resultado: " + divisao(a,b));
}
catch(DivideByZeroException e)
{
    System.err.println(e.getMessage());
}
```

# Exemplo com Tratamento de Exceção

- Em JAVA

```
public class Main
{
    public static double divisao(int a, int b) throws DivideByZeroException
    {
        if(b==0)
            throw new DivideByZeroException();
        return a/(double)b;
    }
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            try {
                System.out.println("Resultado: " + divisao(a,b));
            }
            catch(DivideByZeroException e) {
                System.err.println(e.getMessage());
            }
        }
    }
}
```

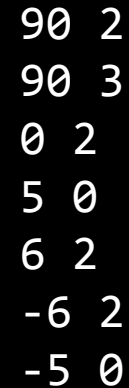
```
class DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Erro: Divisao por zero");
    }
}
```

# Exemplo com Tratamento de Exceção

- Em JAVA

```
class DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Erro: Divisao por zero");
    }
}
```

```
public class Main
{
    public static double divisao(int a, int b) throws DivideByZeroException
    {
        if(b==0)
            throw new DivideByZeroException();
        return a/(double)b;
    }
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            try {
                System.out.println("Resultado: " + divisao(a,b));
            }
            catch(DivideByZeroException e) {
                System.err.println(e.getMessage());
            }
        }
    }
}
```



```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```



# Exemplo com Tratamento de Exceção

- Em JAVA

```
class DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Erro: Divisao por zero");
    }
}
```

```
public class Main
{
    public static double divisao(int a, int b) throws DivideByZeroException
    {
        if(b==0)
            throw new DivideByZeroException();
        return a/(double)b;
    }
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        int a, b;
        while(scan.hasNext())
        {
            a = scan.nextInt();
            b = scan.nextInt();
            try {
                System.out.println("Resultado: " + divisao(a,b));
            }
            catch(DivideByZeroException e) {
                System.err.println(e.getMessage());
            }
        }
    }
}
```

```
90 2
90 3
0 2
5 0
6 2
-6 2
-5 0
```

```
Resultado: 45
Resultado: 30
Resultado: 0
Erro divisao por zero
Resultado: 3
Resultado: -3
Erro divisao por zero
```

# Parte da hierarquia de Exceções em JAVA

