Introdução a Exceções

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

Quando usar exceções

Quando ocorre um erro em nosso código devido a uma situação excepcional, a forma de evitá-lo é usando exceções. Para usar exceções, temos blocos try / catch. O que podemos fazer é literalmente o que eles nos dizem:

- Experimente (o que pode nos causar problemas).
- Pegue (o problema).

Quando usar exceções

```
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   int num1,num2, divisao;

   System.out.println("Primeiro número, deve ser um valor inteiro ");
   num1=scanner.nextInt();
   System.out.println(" Divisor, um valor inteiro ");
   num2=scanner.nextInt();
   divisao= num1/num2;
   System.out.println(division);
}
```

Se executarmos este código, ele parece estar correto, mas se for inserido o número 0 no segundo número, uma **exceção** será lançada.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at com.company.Main.main(Main.java:16)
```

Solução com Exceções

Para exibir o erro, usamos **System.err.println**, isso fará com que a mensagem apareça em outra cor:

```
public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
   int num1,num2, divisao;

   System.out.println("Primeiro número, deve ser um valor inteiro ");
   num1=scanner.nextInt();
   System.out.println(" Divisor, um valor inteiro ");
   num2=scanner.nextInt();
   try{
           divisao= num1/num2;
           System.out.println(divisao);}
   catch(ArithmeticException excepcion){
           System.err.println(“Tentei dividir por zero”);
}
```

{código}

No bloco **try** estão as instruções que podem gerar um problema, neste caso, a divisão (se o divisor for 0).

O bloco **catch** "captura" a exceção, se você tentar dividir por zero, então essa exceção é capturada e, neste caso, a mensagem é exibida. Se for feita uma divisão que não seja inconveniente, a captura não funcionará.

ArithmeticException é o tipo de exceção que ocorreu. Quando ocorre, é criado um objeto, neste caso **exception**.

```
try{
        divisao= num1/num2;
        System.out.println(divisao);}
catch(ArithmeticException excepcion){
        System.err.println("Tentei dividir
   por zero");
}
```

Exceções em detalhes

Estamos protegendo o código da divisão por zero, mas ainda pode haver erros inesperados. Estamos solicitando a entrada de números inteiros, mas você pode inserir valores com decimais. Para proteger o código, vamos modificá-lo.

```
    System.out.println(""Primeiro número, deve ser um valor inteiro  "");
    num1=scanner.nextInt();
    System.out.println(" Divisor, um valor inteiro ");
    num2=scanner.nextInt();
    try{
            division= num1/num2;
            System.out.println(division);}
    catch(ArithmeticException excepcion){
            System.err.println(“Tentei dividir por zero”)”);
}
```

Exceções em detalhes

```
try{
        System.out.println("Primeiro número, deve ser um valor inteiro ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, um valor inteiro ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

 catch(ArithmeticException excepcion){
        System.err.println("Tentei dividir por zero");}
```

Agora estamos protegendo o código, mas não analisamos o erro. Se um número com decimais for inserido, teremos outra **exceção**.

```
Exception in thread "main" java.util.InputMismatchException
```

Diferentes Exceções

Os blocos try / catch nos permitem usar mais de uma captura. Dessa forma, podemos lidar com exceções específicas primeiro e, em seguida, as mais gerais. No exemplo anterior, você pode gerar:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

```
Exception in thread "main" java.util.InputMismatchException
```

Então, vamos adaptar o código para diferenciar a ocorrência.

Diferenciando erros

A primeira captura acontece na exceção que ocorreria devido a uma entrada incorreta, e a segunda está vinculada ao que ocorreria ao tentar dividir por zero.

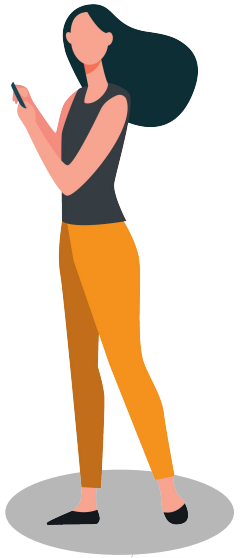
```
try{
        System.out.println("Primeiro número, deve ser um valor inteiro ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, um valor inteiro ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

catch(InputMismatchExcption excepcion){
        System.err.println("Você inseriu um tipo de dado incorreto");
}

catch(ArithmeticException excepcion){
        System.err.println("Tentei dividir por zero");}
```

O bloco finally

Você pode adicionar o bloco **finally** aos blocos **try / catch**, o que é opcional, ou seja, não é obrigatório usá-lo. O **finally** é sempre executado, se não ocorrer uma exceção e não insere o **catch**, se o finalmente for executado. Se ocorrer uma exceção e o **catch** a "pegar", o **finally** também será executado. Vamos ver como ficaria no exemplo da próxima página.



O **finally** é sempre executado e é opcional.

```
try{
        System.out.println("Primeiro número, deve ser um valor inteiro ");
        num1=scanner.nextInt();
        System.out.println(" Divisor, um valor inteiro ");
        num2=scanner.nextInt();
        division= num1/num2;
        System.out.println(division);}

catch(InputMismatchExcption excepcion){
        System.err.println("Você inseriu um tipo de dado incorreto");}

catch(ArithmeticException excepcion){
        System.err.println("Tentei dividir por zero");}
finally{
        System.out.println("O exemplo terminou");}
```


DigitalHouse>
Coding School