Linguagem de Programação

EXCEÇÕES

Prof. Me. Humberto Zanetti

Nessa aula...

- Tratamento de exceções com:
  - try, catch e finally.
  - throw/throws.

Exceções

- Exceções são ocorrência imprevistas em nosso programa, podendo ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis.
- Alguns exemplos de eventos que levam a exceção
  - Operações inválidas ou erros de lógicas:
    - Divisão por zero
    - Tentar manipular um objeto que está com o valor nulo.
    - Tentar manipular um tipo de dado como se fosse outro.
    - Tentar utilizar um método ou classe não existentes.
  - Tentar abrir um arquivo que não existe ou tentar escrever sem permissão;
  - Tentar conectar em servidor inexistente ou off-line;
  - Consultar um banco de dados indisponível;
  - entre outras...

Tratamento de exceções

- Devido à esses cenários, o programador deve “tratar” essas exceções, partindo do princípio que elas podem ocorrer.
- Para isso devemos:
  - Demarcar o trecho do código que é possível ocorrer uma exceção.
  - “Lançar” essa exceção, para que possamos tratar.

Try - catch

- Para esse tratamento usamos 2 blocos/comandos:
  - **try**{ ... } – Bloco onde são introduzidas todas as linhas de código que podem vir a lançar uma exceção.
  - **catch**(tipo_excecao e) { ... } – Bloco onde é descrita a ação que ocorrerá quando a exceção for capturada.

```
try {
        //código que poderá lançar uma exceção
}
catch(tipo_excecao e) {
        //ação a ser tomada caso ocorra a exceção
 }
```

Exemplo – divisão por zero

```
public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int a, b, resultado;
        System.out.println("Digite um número: ");
        a = entrada.nextInt();
        System.out.println("Digite um número: ");
        b = entrada.nextInt();

        resultado = a/b;
        System.out.println("Resultado:"+resultado);

    }
```

Esse código causa uma exceção, corrompendo a execução do programa se **b** for igual a zero

Digite um número:
2
Digite um número:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero

Exemplo – divisão por zero

```
public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        int a, b, resultado;
        System.out.println("Digite um número: ");
        a = entrada.nextInt();
        System.out.println("Digite um número: ");
        b = entrada.nextInt();

        try{
            resultado = a/b;
            System.out.println("Resultado:"+resultado);
        }
        catch(Exception e){
            System.out.println("Divisor não pode ser zero!");
        }
    }
```

Aqui nós colocamos o trecho no qual poderá gerar uma exceção, fazendo uma "tentativa".

Caso a exceção ocorrerá, esse trecho será executado, sem corromper a execução do resto do programa.

Digite um número:
2
Digite um número:
0
Divisor não pode ser zero!

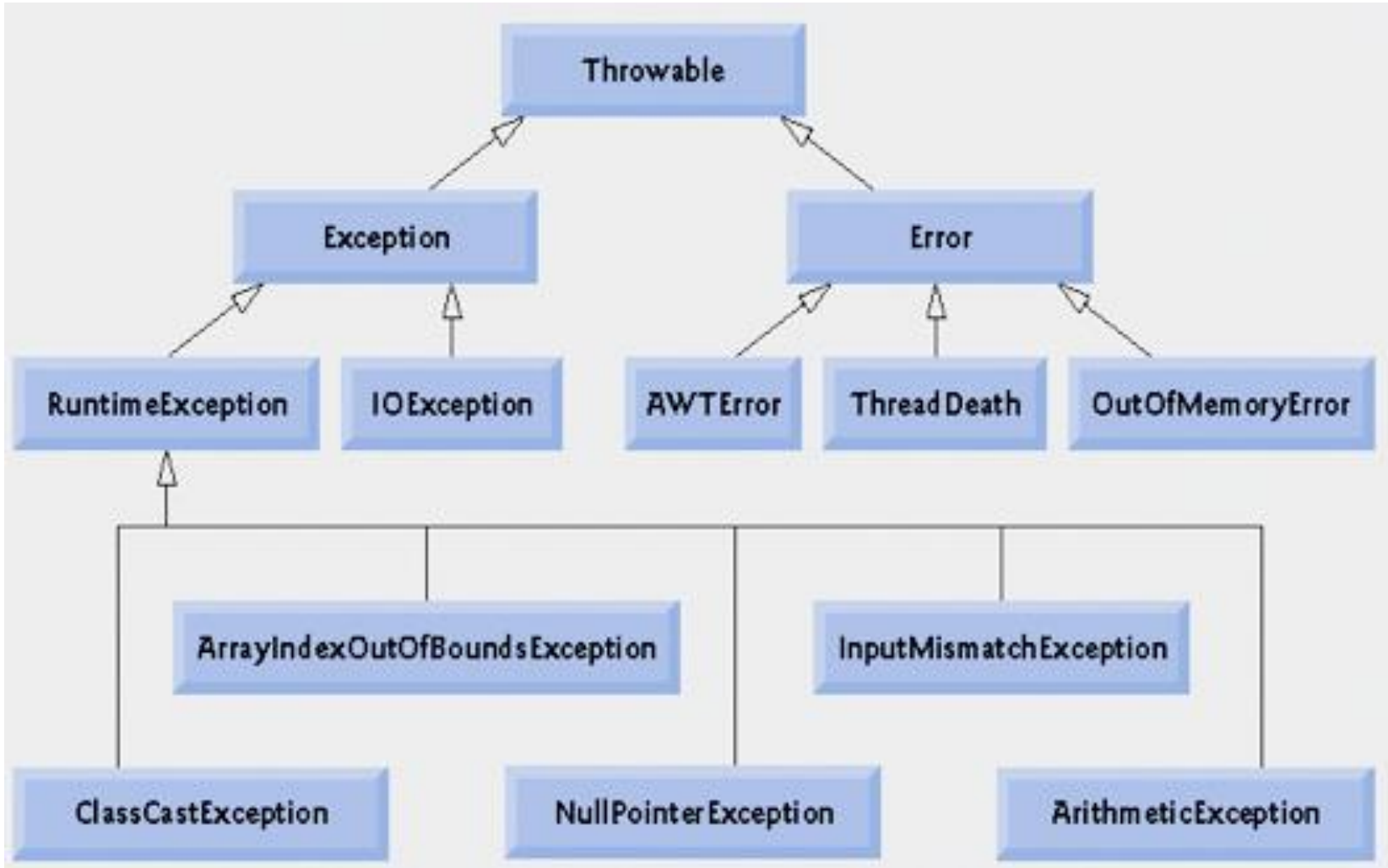
Classificações de exceções

- Em Java existem dois tipos de exceções, que são:
  - **Implícitas**: Exceções que não precisam de tratamento e demonstram serem contornáveis. Esse tipo origina-se da subclasse **Error** ou **RunTimeException**.
  - **Explícitas**: Exceções que precisam ser tratadas e que apresentam condições incontornáveis. Esse tipo origina do modelo **throw** e necessita ser declarado pelos métodos. É originado da subclasse **Exception** ou **IOException**.

Classificação de exceções

- **Checked Exception**: Erros que acontecem fora do controle do programa, mas que devem ser tratados pelo desenvolvedor para o programa funcionar.
- **Unchecked (Runtime):** Erros que podem ser evitados se forem tratados e analisados pelo desenvolvedor. Caso haja um tratamento para esse tipo de erro, o programa acaba parando em tempo de execução (Runtime).
- **Error**: Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continuar.

Hierarquia


Throwable
Exception
Error
RuntimeException
IOException
AWTError
ThreadDeath
OutOfMemoryError
ArrayIndexOutOfBoundsException
InputMismatchException
ClassCastException
NullPointerException
ArithmeticException

Exceções X Erros

- A classe **Throwable** tem duas subclasses:
  - Exception (java.lang.Exception) – É a raiz das classes originárias da classe Throwable, onde mostra as situações em que a aplicação pode querer capturar e realizar um tratamento para conseguir realizar o processamento.
  - Error (java.lang.Error) – Também é raiz das classes originárias da classe Throwable, indicando as situações em que a aplicação não deve tentar tratar, como ocorrências que não deveriam acontecer.
- Existe uma diferença entre “Erro (Error)” e “Exceção (Exception)”:
  - O “Erro” é algo que não pode mais ser tratado, ao contrário da “Exceção” que trata seus erros, pois todas as subclasses de Exception (menos as subclasses RuntimeException) são exceções e devem ser tratadas.
- Os erros da classe Error ou RuntimeException são erros e não precisam de tratamento, por esse motivo é usado o **try/catch** e/ou propagação com **throw/throws**.

Exercício

1. Faça um programa que peça para o usuário preencher um vetor com 10 valores inteiros e depois digitar um índice. O programa deverá verificar se o índice inserido pelo usuário é válido, ou seja, se ele existe dentro do vetor. Caso seja válido, mostrar o valor correspondente ao índice. Senão, mostrar uma mensagem de erro. Use try-catch.

Throw/throws

- Cláusulas **throw** e **throws** são ações que propagam exceções, ou seja, criam exceções que não possuem tratamento pela classe **Exception**.
- De maneira direta: **você pode criar suas exceções!**
- O programador pode definir que algumas ações são exceções à lógica do seu sistema, fazendo tratamentos quando ela ocorrer.
- Usualmente lançamos através de métodos.
  - **throws**: você define na assinatura do método quais são as exceções que serão lançadas.
  - **throw**: “lança” a exceção.

Exemplo

- **Exemplo**: ao inserir as notas do alunos, não permitimos que elas sejam menores que 0 e maiores que 10.

```
public static void verifica(double nota) throws Exception{
        if(nota < 0 || nota > 10){
            throw new Exception("Nota inválida");
        }
    }

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        double nota;
        System.out.println("Digite a nota: ");
        nota = entrada.nextDouble();
        try{
            verifica(nota);
        }catch(Exception e){
            System.out.println(e);
        }
    }
```

Usamos o *throw* para efetivamente lançar as exceções.

Aqui definimos que esse métodos lança um exceção.

Podemos imprimir nossa exceção!!!

```
Digite a nota:
11
java.lang.Exception: Nota inválida
```

Imprimindo exceções

- Podemos usar e/ou imprimir as mensagens de exceções:
  - **toString():** Converte os dados da exceção para String para visualização.
  - **printStackTrace():** Imprime na saída de erro padrão (geralmente console) todos os frames de onde foram detectados erros. Útil para depuração no desenvolvimento, pois mostra todo o histórico do erro, além das linhas onde foram ocasionados.
  - **getCause():** Retorna a causa da Exceção, ou null se a causa for desconhecida ou não existir.
  - **getMessage():** Retorna uma string com o erro. É uma forma simples e elegante de mostrar a exceção causada, geralmente, utilizada como forma de apresentação ao usuário.
- Podemos usar o método System.err.println para dar ênfase à mensagem de exceção/erro!

Exemplo

```
public static void verifica(double b) throws ArithmeticException{
        if(b == 0){
            throw new ArithmeticException("Divisor == 0!");
        }
    }

public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    int a, b, resultado;
    System.out.println("Digite um número: ");
    a = entrada.nextInt();
    System.out.println("Digite um número: ");
    b = entrada.nextInt();
    try{
       verifica(b);
    }catch(ArithmeticException e){
       System.err.println(e);
     }
}
```

```
Digite um número:
2
Digite um número:
0
java.lang.ArithmeticException: Divisor == 0!
```

Exemplo

```
public static void verifica(int b) throws ArithmeticException{
        if(b == 0){
            throw new ArithmeticException("Divisor == 0!");
        }
    }

public static void main(String[] args) {
    Scanner entrada = new Scanner(System.in);
    int a, b, resultado;
    System.out.println("Digite um número: ");
    a = entrada.nextInt();
    System.out.println("Digite um número: ");
    b = entrada.nextInt();
    try{
       verifica(b);
    }catch(ArithmeticException e){
       System.out.println(e.getMessage());
     }
}
```

```
Digite um número:
2
Digite um número:
0
Divisor == 0!
```

Finally

- O bloco **finally é opcional** e deve ser colocado ao final do último **catch**.
  - Podemos ter várias exceções, portanto vários ***catchs***
- Este bloco (se houver) será sempre executado, independente se houver ou não a ocorrência de uma exceção.
  - Até mesmo se o bloco try parar devido a um **break** ou **return**, o bloco finally será executado.
- Geralmente usado para dar uma mensagem, informando que o trecho do código foi executado, ou para desalocar recursos, como memória ou conexões (como em Banco de Dados, por exemplo).

Exemplo

```
Scanner entrada = new Scanner(System.in);
int a, b, resultado;
System.out.print("Digite um número: ");
a = entrada.nextInt();
System.out.print("Digite um número: ");
b = entrada.nextInt();

try{
    resultado = a/b;
    System.out.println("Resultado:"+resultado);
}catch(Exception e){
    System.out.println("Divisor não pode ser zero!");
}finally{
    System.out.println("Término da divisão!");
}
```

```
Digite um número: 2
Digite um número: 2
Resultado:1
Término da divisão!
```

```
Digite um número: 2
Digite um número: 0
Divisor não pode ser zero!
Término da divisão!
```

Exercício

1. Refaça o exercício anterior (que verifica o índice) agora usando método para lançar a exceção.
2. O cálculo de raízes de números negativos são tratados na matemática como números imaginários, saindo fora do contexto da matemática fundamental. Com isso, faça um programa que tenha um método que calcula a raiz quadrada de um número inteiro mas que trata a exceção caso seja inserido como parâmetro um valor negativo. Use a classe de exceção **IllegalArgumentException**.

Dúvidas!?

humberto.zanetti@fatec.sp.gov.br