SCC0504 - Programação Orientada a Objetos

Exceções

Prof.: Leonardo Tórtoro Pereira

leonardop@usp.br

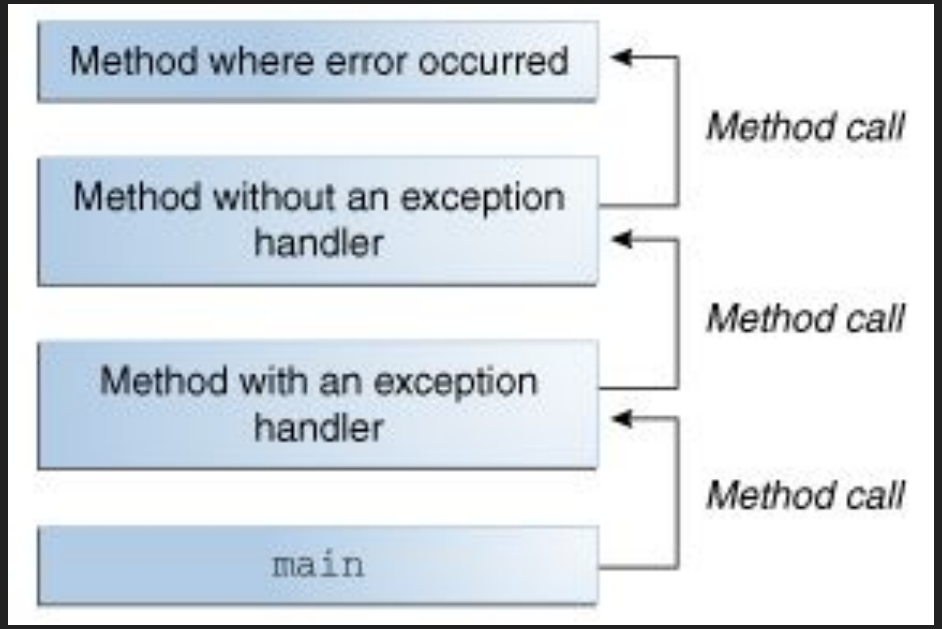
Exceções

Exceções [1]

- Origem no termo “evento excepcional”
- Quando ocorre um erro dentro de um método é criado um objeto que é entregue ao sistema de tempo de execução
- O objeto é chamado de objeto de exceção (*exception*)
- Ele contém informações sobre o erro, incluindo seu tipo e estado do programa quando o erro ocorreu
- Entregar esse objeto ao sistema é chamado de “jogar” uma exceção (*throwing*)

Exceções [1]

- Depois que um método lança uma exceção, o sistema tenta encontrar algo para lidar com ela
- O conjunto de “algos” é uma lista ordenada de métodos que foram chamados para chegar até o método quando o evento ocorreu
  - Conhecida como pilha de chamada (*call stack*)


Method where error occurred
Method without an exception handler
Method with an exception handler
main
Method call
Method call
Method call

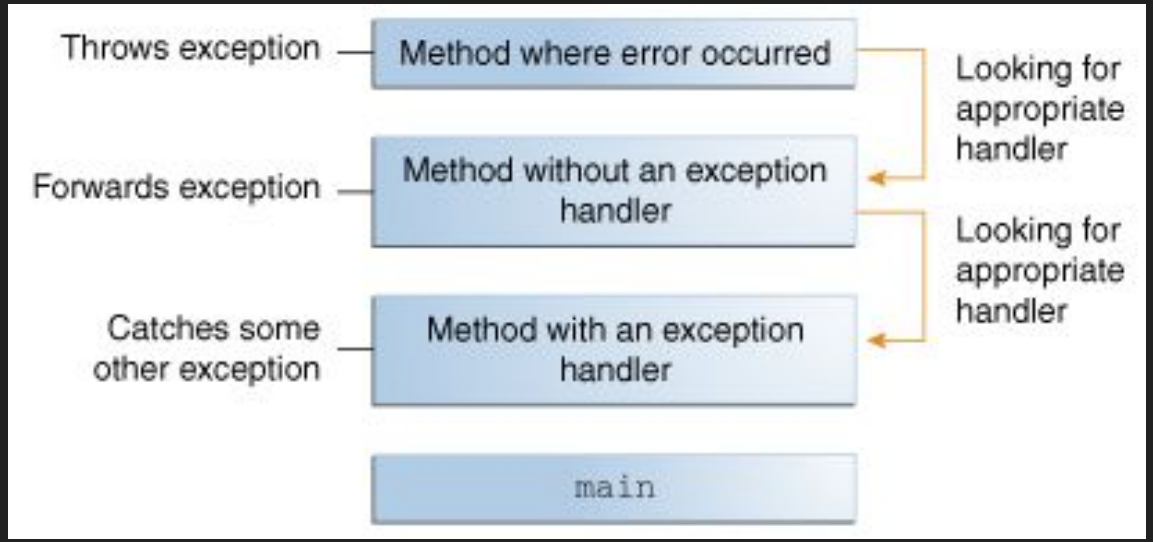
Fonte: [1]

Exceções [1]

- O sistema procura na pilha por um método que tenha um bloco de código que saiba lidar com uma exceção
  - Esse bloco é chamado de manipulador de exceção
    - *Exception Handler*
- Por ser uma pilha, a busca começa pelo método em que ocorreu a exceção e percorre inversamente a ordem em que os métodos foram chamados

Exceções [1]

- Quando um manipulador de exceção apropriado é encontrado, o sistema passa a exceção para ele
  - Ele é considerado apropriado caso o tipo do objeto de exceção lançado seja correspondente ao tipo que pode ser manipulado por ele
- O manipulador escolhido “pega” a exceção (*catch*)
- Se nenhum for encontrado, o sistema termina
  - Assim como o programa


Throws exception
Method where error occurred
Looking for appropriate handler
Forwards exception
Method without an exception handler
Looking for appropriate handler
Catches some other exception
Method with an exception handler
main

Fonte: [1]

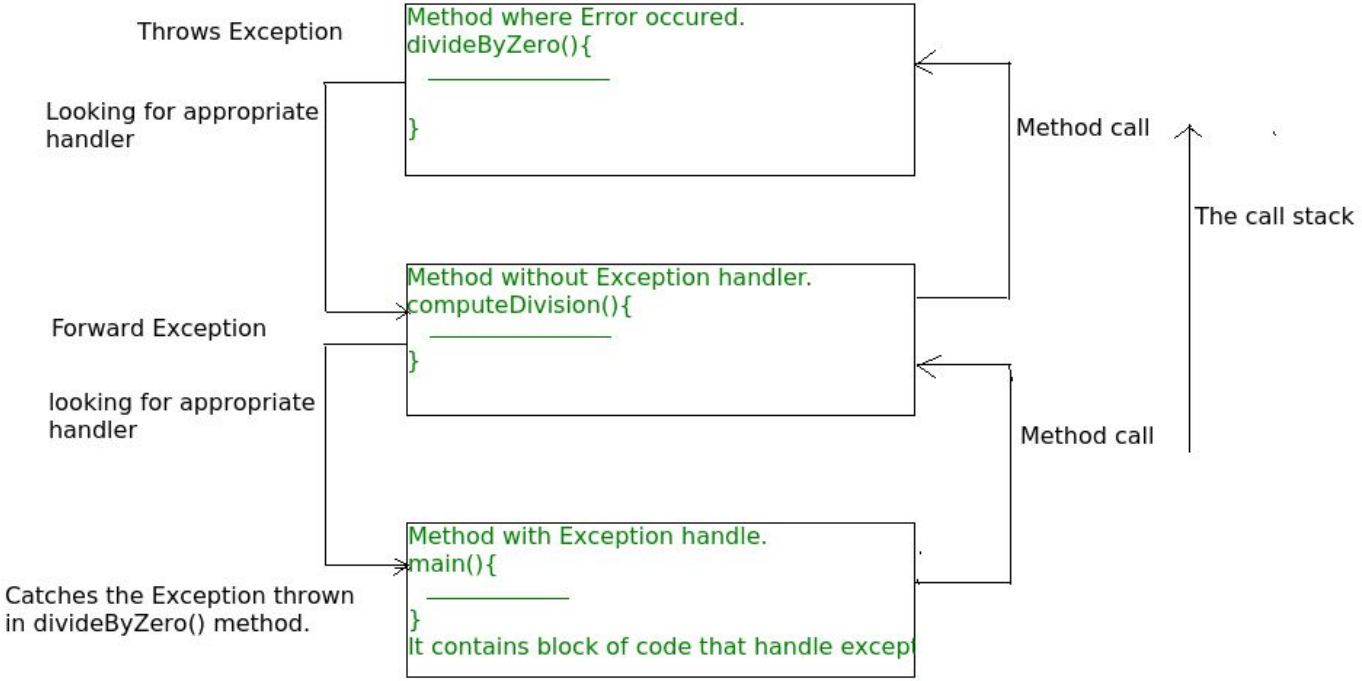
Exceções [2]

```
class ExceptionThrown {
    static int divideByZero(int a, int b){
        int i = a/b;
        return i;
    }
    static int computeDivision(int a, int b) {
        int res =0;
        try {
          res = divideByZero(a,b);
        }
        catch(NumberFormatException ex) {
           System.out.println("NumberFormatException has occurred");
        }
        return res;
    }
```

Exceções [2]

```
    public static void main(String args[]){
        int a = 1;
        int b = 0;
        try {
            int i = computeDivision(a,b);

        }
        catch(ArithmeticException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```



The call stack and searching the call stack for exception handler.

Fonte: [2]

Exceções [1]

- ➔ Um código que joga uma exceção precisa estar cercado ou por um comando *try* que pega a exceção ou um método que especifica que pode jogar uma exceção
- ➔ O bloco *try* precisa providenciar um manipulador de exceções
- ➔ O método precisa providenciar uma cláusula *throw* que lista a exceção

Exceções [1]

- Um código que pode gerar uma exceção deve ser cercado por uma expressão *try*, seguida de um bloco *catch* ou *finally*
- É possível colocar cada linha de código que pode gerar uma exceção em seu bloco *try* individual, com suas exceções separadas
- Ou colocar todos em um único bloco com vários manipuladores

Exceções [1]

➔ Exemplo de bloco *try-catch* com mais de uma exceção

```
try {
    //Código que gera exceção
} catch (ExceptionType name) {
    //Código de tratamento de exceção do tipo 1
} catch (ExceptionType2 name2) {
    //Código de tratamento de exceção do tipo 2
}
```

Exceções [1]

- O bloco *catch* contém código que é executado SE e QUANDO o manipulador da exceção é invocado
- O sistema invoca o manipulador quando ele é o primeiro da pilha de chamadas com o tipo correspondente ao da exceção jogada
  - Corresponde se o objeto jogado puder ser atribuído ao argumento do manipulador

Exceções [1]

➔ Exemplo de bloco *try-catch* com mais de uma exceção

```
PrintWriter out = null;
try {
    System.out.println("Entered try statement");
    out = new PrintWriter(new FileWriter("OutFile.txt"));
    for (int i = 0; i < SIZE; i++)
         out.println("Value at: " + i + " = " + list.get(i));
} catch (IndexOutOfBoundsException e) {
    System.err.println("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```

Exceções [1]

- O bloco *finally* SEMPRE executa ao final de um bloco *try*
- Ele garante que o bloco *finally* é executado mesmo quando uma exceção inesperada ocorre
- Ele é útil também para que o programador evite que códigos de limpeza sejam ignorados graças a um *return continue* ou *break*
- É sempre uma boa prática, mesmo quando não espera-se exceções

Exceções [1]

- O bloco *finally* SEMPRE executa ao final de um bloco *try*
- Ele garante que o bloco *finally* é executado mesmo quando uma exceção inesperada ocorre
- Ele é útil também para que o programador evite que códigos de limpeza sejam ignorados graças a um *return continue* ou *break*
- É sempre uma boa prática, mesmo quando não espera-se exceções

Exceções [1]

➔ Exemplo de bloco *finally* com o código anterior

```
finally {
    if (out != null) {
        System.out.println("Closing PrintWriter");
        out.close();
    } else {
        System.out.println("PrintWriter not open");
    }
}
```

Exceções [1]

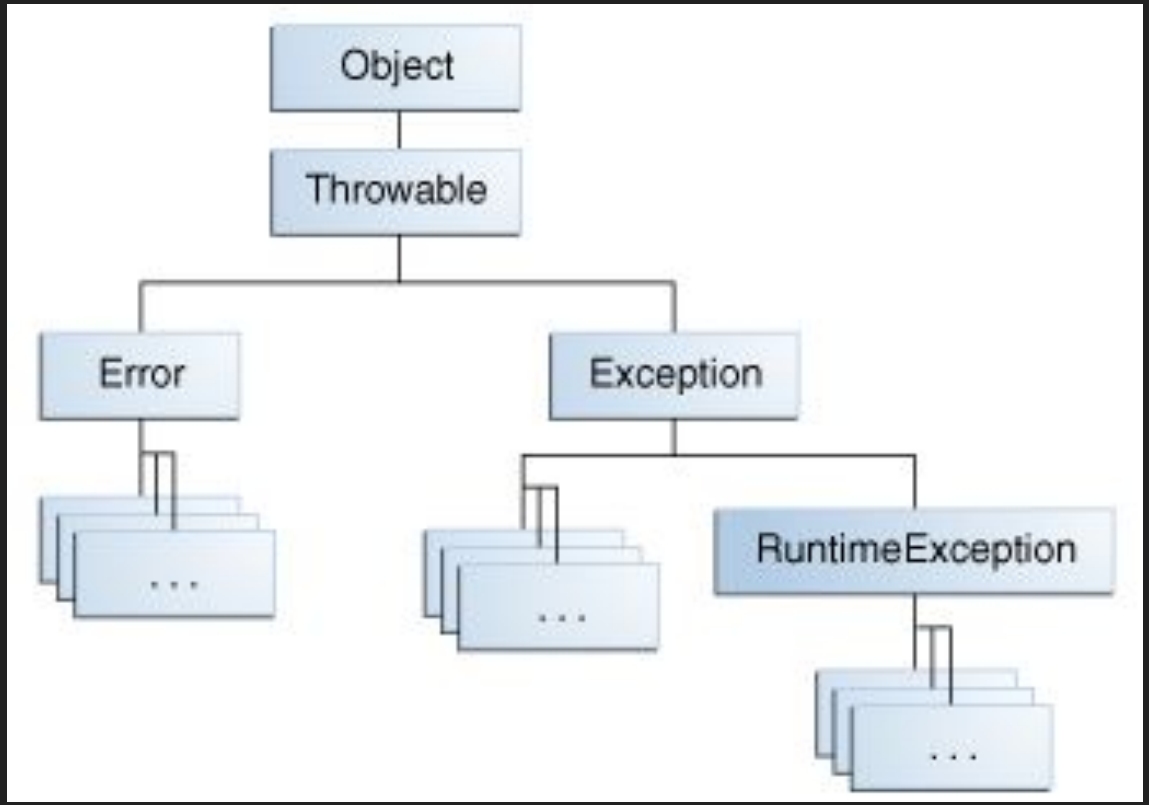
- É possível deixar que exceções sejam tratadas por métodos acima do atual na pilha de chamadas
  - Ele precisa *jogar* a exceção
- Adicione a cláusula *throw* à declaração do método
  - *throw Excepction1, Exception2 { method }*
- `public void writeList() throws IOException, IndexOutOfBoundsException { }`

Exceções [1]

- Antes de *pegar* uma exceção, algum código em outro lugar precisa *lançar* uma
- Seu código, o de um pacote escrito por outro programador, um pacote java...
- É preciso usar a palavra *throw*
  - Seu único argumento é um objeto lançável
- Todas as classes de exceção são herdeiras de *Throwable*
  - É possível criar a sua própria exceção!

Exceções [1]

```
public Object pop() {
    Object obj;
    if (size == 0) {
        throw new EmptyStackException();
    }
    obj = objectAt(size - 1);
    setObjectAt(size - 1, null);
    size--;
    return obj;
}
```



Fonte: [1]

Exceções [3]

```
class ThrowExcep {
    static void fun()     {
        try         {
            throw new NullPointerException("demo");
        }
        catch(NullPointerException e)         {
            System.out.println("Caught inside fun().");
            throw e; // rethrowing the exception
        }
    }
    public static void main(String args[])    {
        try         {
            fun();
        }
        catch(NullPointerException e)        {
            System.out.println("Caught in main.");
        }
    }
}
```

Exceções [4]

➔ Para criar sua própria exceção é só criar uma classe que herda de Exception

```
public class IncorrectFileNameException extends Exception {
    public IncorrectFileNameException(String errorMessage) {
        super(errorMessage);
    }
}
```

Exceções [5]

- ➔ Vamos finalizar vendo esses 2 exemplos [5]

Referências

1. https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html
2. https://www.geeksforgeeks.org/exceptions-in-java/
3. https://www.geeksforgeeks.org/throw-throws-java/
4. https://www.baeldung.com/java-new-custom-exception
5. https://www.devmedia.com.br/trabalhando-com-excecoes-em-java/27601