java.lang.IOException: Um breve resumo

Camila Cavalcante abril/21 - Digital Innovation One

Sumário:

Motivação: Algumas classes e métodos do pacote *java.io* nos obriga a fazer a utilizar blocos *try-catch* para capturar e tratar exceções, ou que declare a possibilidade de lançar essas exceções, colocando uma cláusula *throws* no cabeçalho do método.
É necessário fazer este tratamento para que o programa não pare abruptamente.

As exceções de um modo geral podem ser do tipo checked ou unchecked.

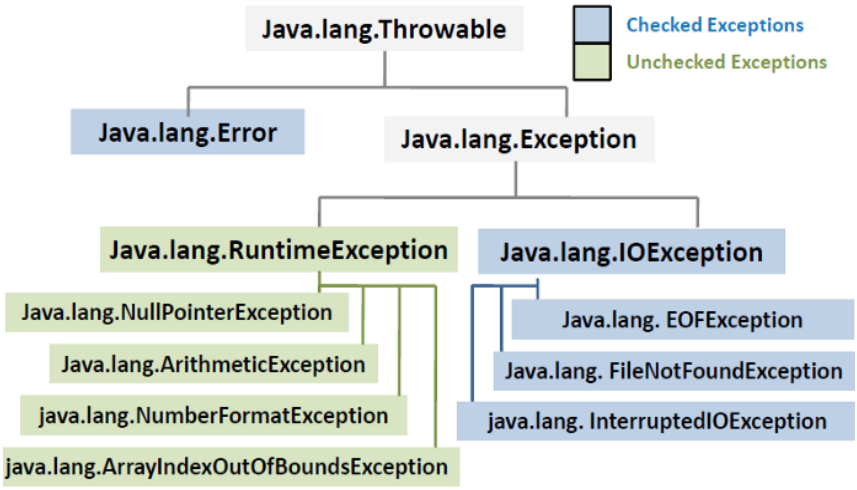

Figura 1: Hierarquia Exception - fonte

Importante: Não estudaremos neste artigo sobre exceções *unchecked*, nem sobre *java.lang.Error* pois sairia do escopo do artigo. Nosso foco será as exceções checked do pacote *java.lang.IOException*.

Class IOException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.io.IOException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AttachOperationFailedException, ChangedCharSetException, CharacterCodingException, CharConversionException, ClosedChannelException, ClosedConnectionException, EOFException, FileLockInterruptionException, FileNotFoundException, FilerException, FileSystemException, HttpRetryException, HttpTimeoutException, IIOException, InterruptedByTimeoutException, InterruptedIOException, InvalidPropertiesFormatException, JMXProviderException, JMXServerErrorException, MalformedURLException, ObjectStreamException, ProtocolException, RemoteException, SaslException, SocketException, SSLException, SyncFailedException, TransportTimeoutException, UnknownHostException, UnknownServiceException, UnsupportedEncodingException, UserPrincipalNotFoundException, UTFDataFormatException, WebSocketHandshakeException, ZipException

Figura 2: Classe IOException e suas subclasses diretamente conhecidas - fonte

Exceções Checked:

- O compilador impõe a restrição de capturar ou sinalizar para exceções verificadas (toda exceção que estende Exception e não estende RuntimeException)
- Exceções verificadas devem ser capturadas (*catch*) ou sinalizadas (*throws*)

Palavras Reservadas:

- *try, catch, finally*: definem um bloco para tratamento de exceção.
- *throws*: declara que um método pode lançar uma ou várias exceções.
- *throw*: lança uma exceção.

A seguir, veremos formas de receber um **Stream de Character** através da classe **FileReader**.

Tratamento de Exceções: try-catch-finally

try:

- Região onde fica o código que queremos verificar se vai ou não lançar exceção.
- Se ocorrer uma exceção em algum ponto, o restante do código contido no bloco *try* não será executado.
- O bloco try não pode ficar sozinho, por tanto, precisa ser seguido de um ou vários blocos *catch* ou de um *finally*.

catch:

- Região onde fica o possível tratamento da exceção. Isso significa que só será executado quando o bloco *try* apresenta alguma exceção.
- Recebe como argumento a classe (no nosso caso, IOException) ou subclasse (mostradas na figura 2) da possível exceção e no seu escopo ficam as instruções do que devemos fazer com essa exceção.
- Pode haver mais de um bloco *catch*, porém, será executado apenas o primeiro bloco que identificar a exceção. Por exemplo, podemos escrever no console uma mensagem que ajudará a identificar a exceção e como podemos resolvê-la.

finally:

- Esse bloco é opcional, mas caso seja construído, sempre será executado (a menos que seja forçado um System.in(0) no catch).
- Por exemplo, podemos utilizar esse bloco para fechar um arquivo.

Importante: Caso você utilize mais de um catch e houver exceções de uma mesma hierarquia de classes, nunca coloque a classe mais genérica como argumento do primeiro *catch*, pois qualquer exceção sempre cairá neste primeiro *catch* e a classe mais específica nunca será verificada.

```java
public class ExemploTryCatchFinally {
    public static void main(String[] args) {
        File f = new File( pathname: "dica-para-ler-e-escrever-arquivo-java-IO.txt"); //arquivo existe
        File f1 = new File( pathname: "test.txt"); //arquivo não existe
        FileReader fr = null; //abre uma stream de caracter
        try { //tente receber o stream f e f1
            fr = new FileReader(f.getName()); //recebe o stream. Sendo positivo, o try segue sendo executado.
            System.out.println("Stream recebido com sucesso! " + f.getName());
            //rotina aqui no meio...
            fr.close(); //fechamos a stream.

            fr = new FileReader(f1.getName()); //esse arquivo não existe, logo o try é interrompido e vai para o catch.
            System.out.println("Stream recebido com sucesso! " + f.getName());
            //rotina aqui no meio...
            fr.close(); //fechamos a stream
        } catch (FileNotFoundException e) { //se o arquivo não existe
            System.out.println("Arquivo não encontrado, motivo: " + e.getCause()); //mostra a causa
            System.out.println("Por gentileza, confira se o arquivo existe " + e.getMessage()); //mostra o nome
            //e.printStackTrace();
        } catch (IOException | NullPointerException e) { //caso o stream estivesse aberto, mas impedido de ser fechado
            System.out.println("Arquivo não pode ser fechado!" + e.getCause()); //mostra a causa
        } finally { //(opcional)
            System.out.println("A execução do finally independe se apresentou exception ou não!");
        }
        System.out.println("** Programa continua normalmente! **"); //apesar de ter apresentado uma exception.
        //caso tivesse outro try por aqui ou qualquer outra rotina, seria executado!
    }
}
```

Figura 3 : Exemplo da construção do *try-catch-finally*

```
Run:  ExemploTryCatchFinally ×
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij-idea-community/289/lib/idea_rt.jar=39837:/sna
Stream recebido com sucesso! dica-para-ler-e-escrever-arquivo-java-IO.txt
Arquivo não encontrado, motivo: null
Por gentileza, confira se o arquivo existe test.txt (No such file or directory)
A execução do finally independe se apresentou exception ou não!
** Programa continua normalmente! **

Process finished with exit code 0
```

Figura 4: Resposta ao executar a classe da figura 3

Link código fonte exemplo *try-catch-finally*

Lançando Exceções: throws

- Usada na assinatura do método
- Necessária apenas para exceções *checked (nosso caso de estudo)*
- Informa ao chamador se este método pode lançar uma das exceções listadas no escopo do método, obrigando o fazer try-catch ou lançar outro throws

Importante: o *throws* serve para convencer o compilador que estamos ciente que pode ocorrer uma exceção, porém não evita o encerramento abrupto do programa caso a exceção não seja tratada corretamente.

```java
public class ExemploThrows {
    public static void recebeStream(String nomeArquivo) throws IOException { //método não trata exceptions, ele lança.
        FileReader fr = new FileReader(nomeArquivo); //FileReader pede um try-catch ou throws. Usamos throws
        System.out.println("Stream recebido com sucesso! " + nomeArquivo);
        //rotina aqui no meio...
        fr.close(); //fechamos o stream.
    }

    public static void main(String[] args) { //chamador do método recebeStream. Note que o método utiliza throws
        File f = new File( pathname: "dica-para-ler-e-escrever-arquivo-java-IO.txt"); //arquivo existe
        File f1 = new File( pathname: "test.txt"); //arquivo não existe

        try { //trataremos as exceptions que não foram tratadas no próprio método recebeStream
            recebeStream(f1.getName()); //exige um try-catch ou throws. Faremos um try-catch
            recebeStream(f.getName()); //exige um try-catch ou throws. Faremos um try-catch
        } catch (IOException e) { //se acontece algum erro correspondente com IOException
            System.out.println("Arquivo não encontrado, motivo: " + e.getCause()); //mostra a causa
            System.out.println("Por gentileza, confira se o arquivo existe " + e.getMessage()); //mostra o nome
            //e.printStackTrace();
        } finally { //(opcional)
            System.out.println("A execução do finally independe se apresentou exception ou não!");
        }

        System.out.println("** Programa continua normalmente! **"); //apesar de ter apresentado uma exception.
        /*Neste caso o try foi interrompido porque na primeira execução do método recebeStream já apresenta
         uma exception. O arquivo que existe (dica-para-ler-e-escrever-arquivo-java-IO.txt) não foi lido pois está
         dentro do try que apresentou a exception, porém a rotina fora do try continua.*/
    }
}
```

Figura 5: Exemplo utilizando o throws no método

```
Run:  ExemploThrows ×
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij-idea-community/289/lib/idea_rt.jar=42829:/snap
Arquivo não encontrado, motivo: null
Por gentileza, confira se o arquivo existe test.txt (No such file or directory)
A execução do finally independe se apresentou exception ou não!
** Programa continua normalmente! **

Process finished with exit code 0
```

Figura 6: Resposta ao executar a classe da figura 5

Link do código fonte exemplo *throws*

Criando nossa própria exceção: ***throw***

- Usada principalmente para lançar exceções personalizadas (nosso caso de estudo)
- Antes de criar a nossa própria exceção, precisamos saber se já existe alguma na biblioteca da linguagem já que nos forneça o que precisamos (não queremos reinventar a roda)
- É usada para lançar explicitamente uma exceção de um método ou qualquer bloco de código
- Pode ser utilizada tanto para exceções *checked* ou *unchecked*

Importante: O fluxo de execução do programa para imediatamente após a execução da instrução throw e o bloco try envolvente mais próximo é verificado para ver se possui uma instrução catch que corresponda ao tipo de exceção.
Se encontrar uma correspondência, o controlado é transferido para essa instrução, caso contrário, o próximo bloco try envolvente é verificado e assim por diante. Se nenhuma captura correspondente for encontrada, o manipulador de exceção padrão interromperá o programa.

```java
class NaoAbreStreamException extends IOException { //exeption customizada
    private String nomeDoArquivo; //criamos um atributo para conter o nome do arquivo

    public NaoAbreStreamException(String nomeDoArquivo) { //criamos um contrutor para iniciar o atributo
        this.nomeDoArquivo = nomeDoArquivo;
    }

    private String diretorioArquivo() {//método para exibir o diretório do arquivo que estamos tentando abrir
        File file = new File(this.nomeDoArquivo); //criamos um objeto File para utilizamos o método getAbsolutePath()
        return file.getAbsolutePath();  //método que retorna o caminho completo do arquivo
    }

    @Override
    public String toString() { //sobrescrevemnos o toString
        return "Arquivo '" + this.nomeDoArquivo + "' não encontrado. Verique o diretório: " + this.diretorioArquivo();
    }
}
```

Figura 7: Exemplo exception customizada

```java
public class ExemploThrow {
    public static void recebeStream(String nomeArquivo) throws NaoAbreStreamException { //trataremos no chamador
        FileReader fr = null; //FileReader pede um try-catch ou throws. Usamos try-catch
        try {
            fr = new FileReader(nomeArquivo); //tente abrir o stream
            System.out.println("Stream recebido com sucesso! " + nomeArquivo);
            //rotina aqui no meio...
            fr.close(); //fechamos o stream.
        } catch (FileNotFoundException e) { //caso o arquivo não seja encontrado
            throw new NaoAbreStreamException(nomeArquivo); //será exibida a exception que nós criamos
        } catch (IOException e) { //caso a stream seja aberta, porém impedida de ser fechada.
            System.out.println("O arquivo não pode ser fechado, motivo: " + e.getCause());
        }
    }

    public static void main(String[] args) { //chamador do método recebeStream. Note que o método utiliza throws
        File f = new File( pathname: "dica-para-ler-e-escrever-arquivo-java-IO.txt"); //arquivo existe
        File f1 = new File( pathname: "test.txt"); //arquivo não existe

        try { //trataremos a exception customizada que foi lançada no próprio método recebeStream
            recebeStream(f.getName()); //exige um try-catch ou throws. Faremos um try-catch
            recebeStream(f1.getName()); //exige um try-catch ou throws. Faremos um try-catch
        } catch (NaoAbreStreamException e) { //utilizando nossa exception customizada
            System.out.println(e); //exibe mensagem que passamos no toString da nossa exception NaoAbreStreamException
            //e.printStackTrace();
        } finally { //(opcional)
            System.out.println("A execução do finally independe se apresentou exception ou não!");
        }
        System.out.println("** Programa continua normalmente! **"); //apesar de ter apresentado uma exception.
        //caso tivesse outro try por aqui ou qualquer outra rotina, seria executado!
    }
}
```

Figura 8: Lógica criada para utilizar a exception customizada (throw + throws) da figura 7

```
Run:  ExemploThrow ×
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij-idea-community/289/lib/idea_rt.jar=38381:/sna
Stream recebido com sucesso! dica-para-ler-e-escrever-arquivo-java-IO.txt
Arquivo 'test.txt' não encontrado. Verique o diretório: /home/cami/git/curso-dio-intro-java-io/test.txt
A execução do finally independe se apresentou exception ou não!
** Programa continua normalmente! **

Process finished with exit code 0
```

Figura 9: Resposta ao executar a classe da figura 8

Link código fonte exemplo *throw*

Exceptions para o curso introdutório java.io: throws

Como já sabemos, a maioria das classes e métodos do pacote *java.io* utilizam exceções *checked*, logo precisam ser tratadas (*try-catch*) ou lançadas (*throws*).
Como nosso curso será focado em File Input/Output, não vamos fazer o *try-catch*, pois como vimos nos exemplos acima, o código fica mais seguro, em contrapartida, dificulta a visualização do mesmo. Resumindo: Iremos sempre declarar o *thows* no método main. Isso implica:

- Levaremos em consideração que em nossos exemplos não apresentarão *Exceptions*
- Caso apresente, essa *Exception* irá aparecer no console sem nenhum tratamento e o nosso programa para abruptamente
- O *throws* nesse caso servirá apenas para informar ao compilador que estamos cientes da possibilidade de apresentar alguma *Exception* em nosso método. Entretanto, permitiremos que esta *Exception* caia no console e resolveremos de outra forma.

```java
public class ExemploThrowsParaUtilizarNoCurso {
    //throws: aviso para o compilador que sabemos da possibilidade de ocorrer um erro, mas não queremos tratar agora.
    public static void main(String[] args) throws IOException {
        FileReader fr = null; //abre um stream de caracter
        File f = new File( pathname: "dica-para-ler-e-escrever-arquivo-java-IO.txt"); //arquivo existe
        File f1 = new File( pathname: "test.txt"); //arquivo não existe

        fr = new FileReader(f.getName()); //recebe o stream. Sendo positivo, o try segue sendo executado.
        System.out.println("Stream recebido com sucesso! " + f.getName());
        //rotina aqui no meio...
        fr.close(); //fechamos a stream.

        fr = new FileReader(f1.getName()); //esse arquivo não existe, o programa encerra aqui e mostra no console.
        //daqui pra frente nada mais acontece
        System.out.println("Stream recebido com sucesso! " + f.getName());
        //rotina aqui no meio...
        fr.close(); //fechamos a stream

        System.out.println("Apresentando qualquer erro antes dessa linha, a mesma nunca será lida.");
    }
}
```

Figura 10: Exemplo utilizando o throws no método main

```
Run:  ExemploThrowsParaUtilizarNoCurso ×
/usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -javaagent:/snap/intellij-idea-community/289/lib/idea_rt.jar=39919:/sna
Stream recebido com sucesso! dica-para-ler-e-escrever-arquivo-java-IO.txt
Exception in thread "main" java.io.FileNotFoundException Create breakpoint : test.txt (No such file or directory)
    at java.base/java.io.FileInputStream.open0(Native Method)
    at java.base/java.io.FileInputStream.open(FileInputStream.java:219)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:157)
    at java.base/java.io.FileInputStream.<init>(FileInputStream.java:112)
    at java.base/java.io.FileReader.<init>(FileReader.java:60)
    at br.com.dio.java.io.IOException.ExemploThrowsParaUtilizarNoCurso.main(ExemploThrowsParaUtilizarNoCurso.java:19)

Process finished with exit code 1
```

Figura 11: Resposta ao executar a classe da figura 10

Link código fonte exemplo *throws* que usaremos no curso file Input/Output

Bibliografia

1. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Exception.html
2. https://marciobueno.com/arquivos/ensino/poo/POO_14_Excecoes.pdf
3. https://pt.slideshare.net/regispires/java-13-excecoes-presentation
4. https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/tratamento-excecao_v01.pdf
5. http://www.inf.ufsc.br/~frank.siqueira/INE5605/13.Excecoes.pdf
6. https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/IOException.html
7. https://www.geeksforgeeks.org/throw-throws-java/
8. http://www.mauda.com.br/?p=2315
9. https://www.devmedia.com.br/como-tratar-excecoes-na-linguagem-java/39163#11