TRATAMENTO DE EXCEÇÕES EM JAVA

Curso que apresenta conceitos introdutórios sobre como podemos identificar e tratar Exceptions na linguagem Java. Curso este ministrado por mim em nome da **DIGITAL INNOVATION ONE** 🧡💛 e disponibilizado de forma gratuita para a comunidade dos desenvolvedores Java.

🛑 Pré-requistos

- [x] Java JDK 8 ou superior
- [x] IDE para desenvolvimento Java
- [x] Conhecimento BÁSICO em OOP
- [x] Estar disposto a aprender

📚 Ementa

1. Visão Geral
2. Unchecked Exception
3. Checked Exception
4. Exception Personalizada

Visão Geral

- Exceção é um evento que interrompe o fluxo normal de processamento de uma classe.
- O uso correto de exceções torna o programa mais robusto e confiável.
- Com o tratamento de exceções, um programa pode continuar executando depois de lidar com um problema.
- *Importante:* Incorpore sua estratégia de tratamento de exceções no sistema desde o princípio do processo de projeto. Pode ser difícil incluir um tratamento de exceções eficiente depois que um sistema foi implementado.

Error:

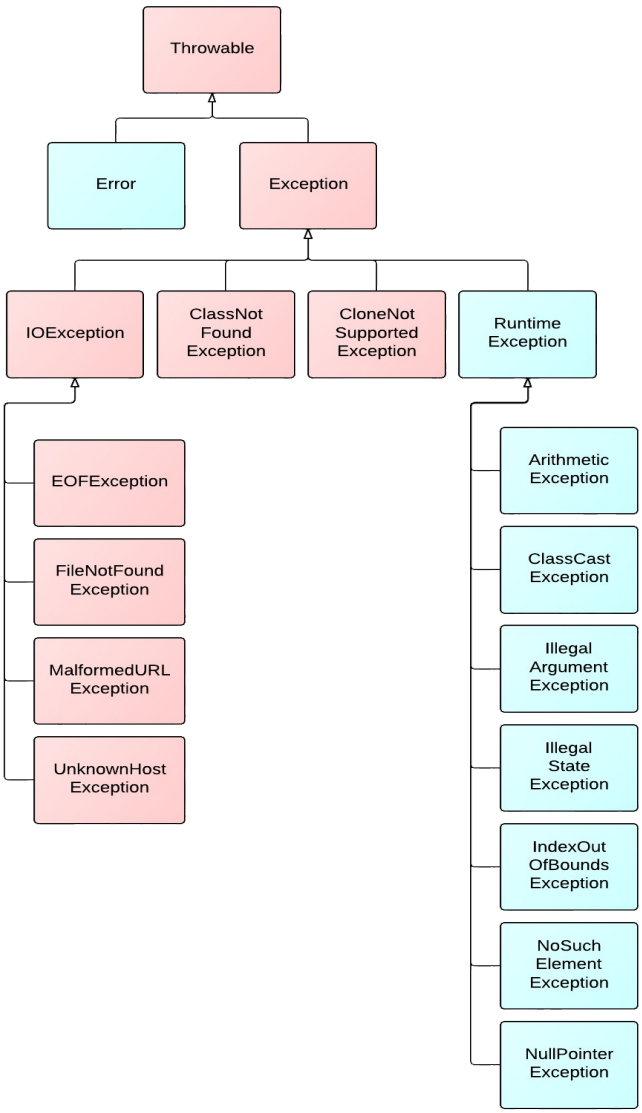
Usado pela JVM que serve para indicar se existe algum problema de recurso do programa, tornando a execução impossível de continuar.

Unchecked (Runtime):

Exceptions que PODEM ser evitados se forem tratados e analisados pelo desenvolvedor.

Checked Exception:

Exceptions que DEVEM ser evitados etratados pelo desenvolvedor para o programa funcionar.



Hierarquia Exceptions

Palavras Reservadas:

- try, catch, finally: Cada uma dessas palavras, juntas, definem blocos para o tratamento de exceções.
- throws: Declara que um método pode lançar uma ou várias exceções.
- throw: Lança explicitamente uma exception.

🔶 Error

- Usado pela JVM, serve para indicar quando existe algum problema de recurso do programa, tornando a execução impossível de continuar.
- 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 que obrigatóriamente devem ser tratadas.

Unchecked Exception

- Herdam da classe *RuntimeException* ou da classe *Error*.
- O compilador não verifica o código para ver se a exceção foi capturada ou declarada.
- Se uma exceção não-verificada ocorrer e não tiver sido capturada, o programa terminará ou executará com resultados inesperados.
- Em geral, podem ser evitadas com uma codificação adequada.

Checked Exception

- As exceções que são herdadas da classe *Exception*, mas não de *RuntimeException*.
- O compilador impõe um requisito do tipo "capturar ou declarar".
- O compilador verifica cada chamada de método e declaração de método para determinar se o método lança (*throws*) exceções verificadas.
- Se lançar, o compilador assegura que a exceção verificada é capturada ou declarada em uma cláusula *throws*.
- Caso não capturada nem declarada, ocorre um erro de compilação.

Exception Personalizada

- Programadores podem achar útil declarar suas próprias classes de exceção.
- Essas Exceptions são específicas aos problemas que podem ocorrer quando outro programador empregar suas classes reutilizáveis.
- Uma nova classe de exceção deve estender uma classe de exceção existente que assegura que a classe pode ser utilizada com o mecanismo de tratamento de exceções, logo essas Exceções customizadas são derivadas da classe Exception.
- *Importante*: Antes de criar a nossa própria exceção, é recomendado verificar se já existe alguma exceção na biblioteca Java que já nos forneça o que precisamos. Afinal, não queremos reinventar a roda!

Blocos try/catch/finally

Bloco ***try***:

- Região onde se encontra o código que queremos verificar se irá ou não lançar uma exceção.
- Caso ocorra uma exceção em algum ponto, o restante do código contido no bloco *try* não será executado.
- O bloco *try* não pode ser declarado sozinho, por tanto, precisa estar seguido de um ou vários blocos *catch* e/ou de um bloco *finally*.

Bloco ***catch***:

- Região onde se encontra o possível tratamento da exceção. Isso significa que só será executado caso o bloco *try* apresentar alguma exceção.
- Recebe como argumento a classe ou subclasse da possível exceção.
- No seu escopo ficam as instruções de como tratar essa exceção.

- Pode haver mais de um bloco *catch*, porém, será executado apenas o primeiro bloco que identificar a exceção.
- *Importante*: Caso você utilize mais de um *catch* e houver exceções de uma mesma hierarquia de classes, certifique-se que a classe mais genérica esteja como argumento do último *catch*. Caso contrário, qualquer exceção sempre cairá neste primeiro *catch*, assim fazendo com que a exception mais específica não seja verificada.

Bloco ***finally***:

- Este bloco é opcional, mas caso seja construído, quase sempre será executado. (A menos que seja forçada sua parada, por exemplo, com um System.exit(0), no *catch*)
- Dentro do bloco *finally*, poderá conter outros blocos *try, catch,* bem como outro *finally*. Geralmente utilizado quando precisamos executar algum código independente se ocorrer exception ou não.

🔶 Cláusulas throws e throw

Cláusula ***throws***

- Usada na assinatura do método.
- Necessária apenas para exceções checked.
- Informa ao chamador que este método pode lançar uma das exceções listadas no escopo do método. Isso obriga a fazer a captura dessa exception (*try-catch*) ou relançar o *throws*.

Cláusula ***throw***

- É usada para lançar explicitamente uma exceção de um método ou de qualquer bloco de código.
- Usada principalmente para lançar exceções personalizadas
Importante:
- O fluxo de execução "normal" do programa para imeditamente apos a execução da cláusula *throw*. O bloco *try* envolvente mais próximo é verificado para encontrar um bloco *catch* que corresponda ao tipo de exceção.
- Caso encontre essa correspondência, o controlado é transferido para esse bloco. Caso contrário, o próximo bloco *try* envolvente é verificado e assim por diante.
- Outro caso, é se nenhuma captura for encontrada, o manipulador da exceção padrão interromperá o programa.

🔗 Referências

- https://www.devmedia.com.br/trabalhando-com-excecoes-em-java/27601
- https://www.youtube.com/watch?v=ld2C4GcAtsg&t=296s
- https://www.programcreek.com/2009/02/diagram-for-hierarchy-of-exception-classes/
- https://www.projetojavaweb.com/certificado-aluno/plataforma-curso/aulaatual/467726283/idcurso/1/idvideoaula/161
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Error.html
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Exception.html
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Exception.html
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/RuntimeException.html