11 Erros que desenvolvedores Java cometem quando usam Exceptions

Por Rafael Del Nero
Publicado em Decembro 2019

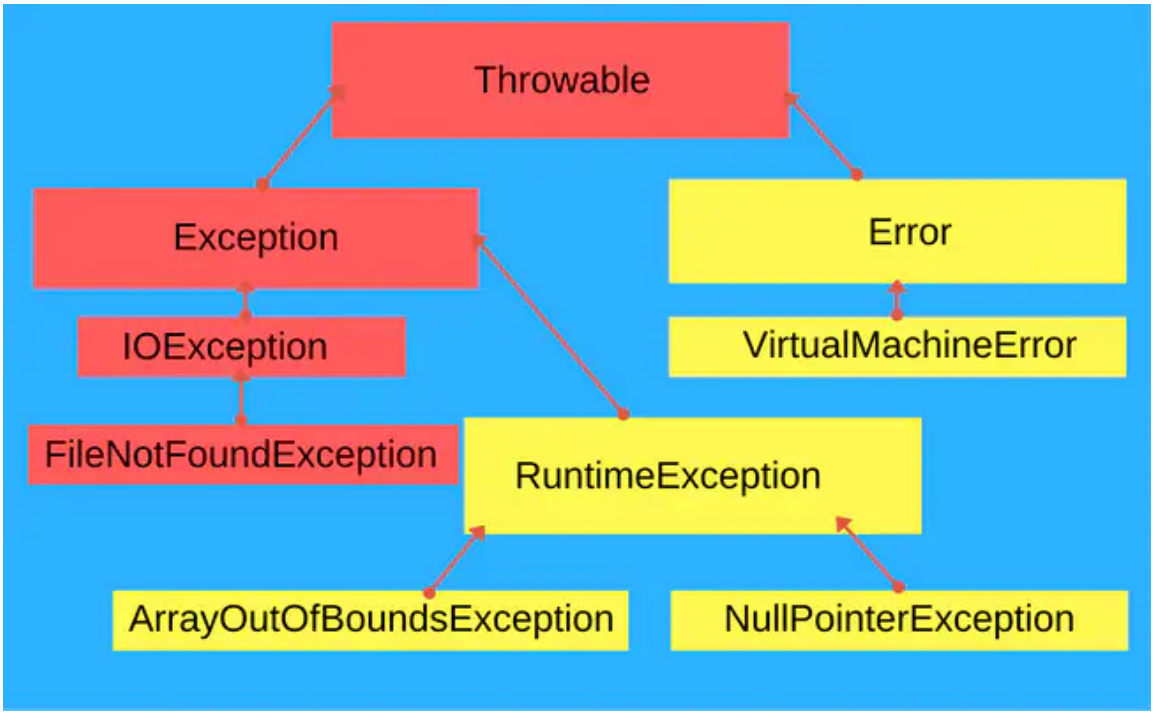
Revisado por Juan Pablo Guizado

Se você usar Exceções da maneira errada, será muito difícil encontrar erros. Se você sempre usa exceções genéricas, como outros desenvolvedores podem saber qual erro ocorreu? Você precisa entender por que usamos exceções e como usá-las efetivamente!

Veja os 11 erros que os desenvolvedores de Java cometem ao usar exceções.

Primeiro, vamos ver a hierarquia das classes Exception.

Exceções marcadas / Exceções não verificadas



1 - Usando apenas a classe Exception

Um erro comum que os desenvolvedores cometem e de usar catch com a classe Exception por qualquer erro. É muito mais difícil identificar o erro se uma Exception genérica está sendo usada. A solução para esse problema é criar exceções específicas que mostrem qual é o erro que ocorreu claramente.

2 - Criando muitas exceções específicas

Não crie exceções para tudo. Seu sistema ficará cheio de classes sem necessidade, e você terá mais trabalho para manter seu código. Em vez disso, crie exceções para requisitos de negócios realmente importantes. Por exemplo, se você estiver desenvolvendo um sistema bancário, uma possível exceção seria ao tentar sacar dinheiro e o saldo for zero: BalanceNotAvailableException. Outra exceção seria a transferência de dinheiro para outra pessoa e a conta não existe. Você pode criar: BankAccountNotFoundException e então mostre uma mensagem de exceção compreensível ao usuário.

RuntimeException pode ser usado quando o servidor do banco estiver fora de serviço. Aqui você pode usar, por exemplo: ServerNotAvailableException. O sistema deve falhar por esse tipo de erro. Não há recuperação.

3 - Criando um log para cada captura

O log de Exceptions em toda árvore de catch em sua aplicação, por exemplo, desde a camada de banco de dados, servico, controle, somente poluirá seu código. Para evitar isso, faça o log somente uma vez que a última Exception foi capturada. Você não perderá o Stacktrace se fizer o log da sua Exception somente na último catch, se você passar a Stacktrace em cada Exception claro. Se você estiver trabalhando com aplicativos da web, você pode capturar a Exceção na camada do controlador e efetue o log com uma clara mensagem de erro.

4 - Não saber a diferença entre Checked Exceptions e Unchecked Exceptions

Quando as checked Exceptions devem ser usadas? Use checked Exceptions quando houver um erro recuperável ou um requisito de negócio importante.

A exceção verificada mais comum é a classe Exception. Duas classes relacionadas da Exception são FileNotFoundException e SQLException. Você é obrigado a manipular ou declarar essas exceções. Você deve lançar ou capturar a exceção, caso contrário o código não será compilado.

Quando as unchecked Exceptions devem ser usadas? Use unchecked Exceptions quando não houver recuperação. Por exemplo, quando a memória do servidor é usada em excesso.

RuntimeException é usado para erros quando seu aplicativo não pode recuperar. Por exemplo, NullPointerException e ArrayOutOfBoundsException. Você pode evitar uma RuntimeException com um comando 'if'. Você não deve lidar com isso ou capturar a Exception.

Há também a classe Error. Também é uma exceção não verificada. Nunca tente capturar ou manipular esse tipo de exceção. Eles são erros da JVM e são o tipo mais grave de exceção em Java. Você deve analisar a causa de exceções como essa e corrigir seu código.

5 - Exceções silenciadas

Nunca pegue a Exceção e não faça nada, por exemplo:

```
try {
    System.out.println("Never do that!");
} catch (AnyException exception) {
    // Do nothing
}
```

A captura será inútil. É impossível saber o que aconteceu e a exceção será silenciada. O desenvolvedor será obrigado a debugar o código e ver o que aconteceu. Se criarmos uma boa mensagem de log neste caso, seria muito mais fácil de descobrir o que está acontecendo

6 - Não seguir o princípio “throw early, catch later”

Se você precisar lidar com a exceção, por exemplo, em seu serviço, faça duas coisas:

Lance a exceção desde o topo da árvore

Capture a Excecao mais tarde possivel e lide com ela.

7 - Não usar mensagens claras nas exceções

Sempre use mensagens claras em suas exceções. Fazer isso ajudará bastante a encontrar erros. Melhor ainda, crie um arquivo de propriedades com todas as mensagens de exceção. Você pode usar o arquivo na camada View do seu sistema e mostrar aos usuários mensagens sobre os requisitos de negócio com internacionalização.

8- Não limpar depois de manipular a exceção

Depois de usar recursos como arquivos e conexão com o banco de dados, limpe-os e feche-os para não prejudicar o desempenho do sistema. Você pode usar try with resources ou o bloco finally para fazer isso para garantir que os recursos serão limpos mesmo quando um erro ocorrer.

9 - Não documentar exceções com javadoc

Para evitar dores de cabeça, sempre documente a razão que a exceção está sendo lançada no seu método. Documente sua exceção e explique por que você a criou e também pense em nome claro que identifique a finalidade da sua Exception. No caso de sua Exception já tiver um nome claro você pode desconsiderar de comentar sua classe de Exception.

10 - Perdendo o rastreamento de pilha (causa raiz)

Ao agrupar uma exceção em outra, não basta lançar a outra exceção, mantenha o Stacktrace.

Código incorreto:

Copy

```
try {
    // Do the logic
} catch (BankAccountNotFoundException exception) {
    throw new BusinessException();
}
```

Bom código:

Copy

```
try {
    // Do the logic
} catch (BankAccountNotFoundException exception) {
    throw new BusinessException(exception);
}
```

Fazendo isso, você vai recuperar a causa raiz da exceção.

11 - Não organizar a hierarquia de exceções específicas

Se você não organizar a hierarquia de suas exceções, o relacionamento será difícil entre as partes do sistema. Organizando as suas exceções por módulo de negócio e uma boa ideia porque vai ser possível saber de qual módulo do seu sistema que está dando erro e quando o sistema tiver qualquer erro será mais fácil e mais rápido de achar e corrigir.

Exception	
BusinessException	
AccountingException	**HumanResourcesEx**
BillingCodeNotFoundException	**EmployeeNotFoundI**

Para mais artigos sobre como evitar bugs, e como ir fundo na linguagem Java acesse:

https://javachallengers.com/

Rafael Del Nero is Brazilian, creator of NoBugsProject, author of "No Bugs, No Stress - Create a Life-Changing Software Without Destroying Your Life". Rafael believes that there are many things involved to create high-quality software and most times developers are not even aware of them. His life's purpose is to help Java developers use better programming practices to code quality software for stress-free projects with fewer bugs.

Este artigo foi revisto pela equipe de produtos Oracle e está em conformidade com as normas e práticas para o uso de produtos Oracle.