AfroDev

Java Básico - Aula 5 - Parte 1

Exceções - Geral

- Evento que pode ocorrer durante a execução de um algoritmo que altera o fluxo normal do programa
- Quando uma exceção ocorrer, um objeto exceção é criado (contendo o tipo e o estado do programa naquele momento)
- Criar um objeto de exceção e o oferecendo ao “sistema runtime” é o mesmo que lançar uma exceção ou arremessar uma exceção (throw)
- A VM caminha na ordem inversa da pilha de execução para buscar algo que trate aquela exceção (catch)
- try-catch-finally, throw, throws e Throwable são elementos que compõe o tema “exceções em JAVA”

Exceções - Try

- É o primeiro passo para tratar uma exceção
- Se uma exceção ocorrer dentro desse bloco, ela deverá ser tratada através de um bloco catch associado a esse try

Exceções - Catch

- Associado a um único bloco try
- Pode conter um ou mais blocos catch. Isso significa que podemos tratar mais de um tipo de exceção
- Bloco de código dentro do catch só é executado quando aquela determinada exceção é lançada
- Esse bloco tem a finalidade de tentar recuperar o sistema e/ou informar que o evento é irrecuperável

Exceções - Finally

- É associado a um bloco try
- É “SEMPRE” executado (System.exit e falta de energia)
- Mesmo com return, break ou continue
- Observe fluxos mais adiante
- Esse bloco tem a função de limpar o que foi feito no bloco try (fechar o que foi aberto, efetivar ou desfazer uma transação, avisar/notificar que algo acabou, etc)

Exceções

- Um bloco de código precisa conter a palavra chave “try”. Isso indica que a execução contida nesse escopo PODE apresentar uma ou mais exceções de um único ou vários tipos. Exemplo:

```
try {
    FileReader fr = new FileReader("algumArquivo.txt");
} catch (FileNotFoundException e) {
    System.out.println("Arquivo nao existe!");
}
```

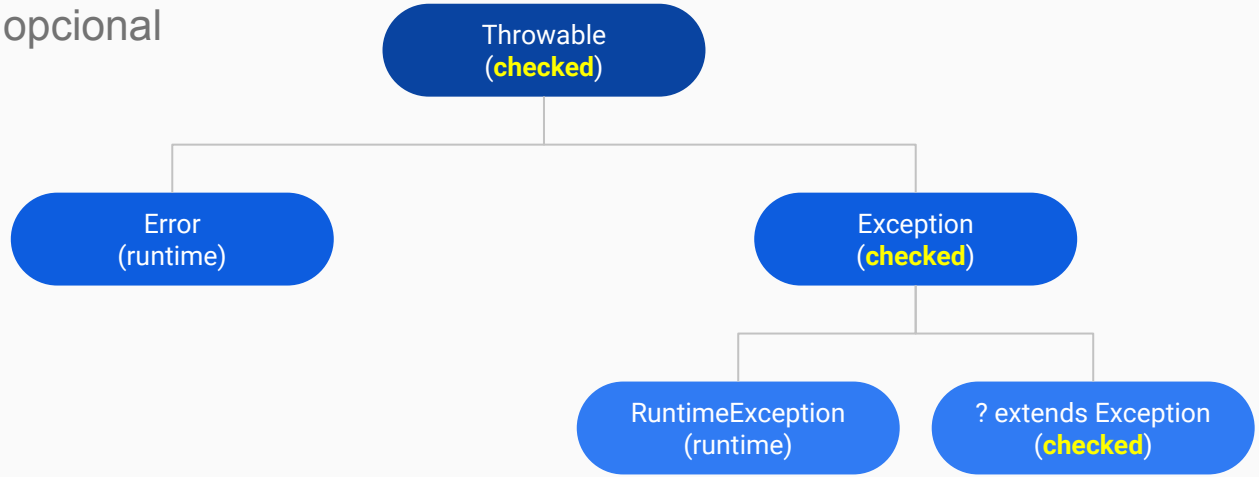
Exceções

- Pode contar a instrução *finally*
- Ideal para limpeza do que foi feito pelo *try*

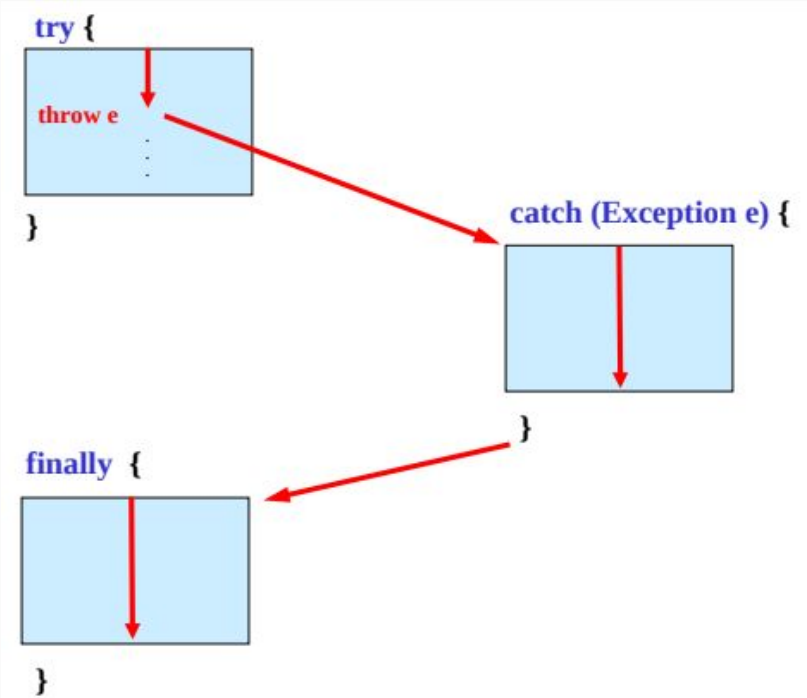
```
try {
    FileReader fr = new FileReader("algumArquivo.txt");
} catch (FileNotFoundException e) {
    System.out.println("Arquivo nao existe!");
} finally {
    System.out.println("Sempre vou passar aqui!");
}
```

Exceções - Hierarquia

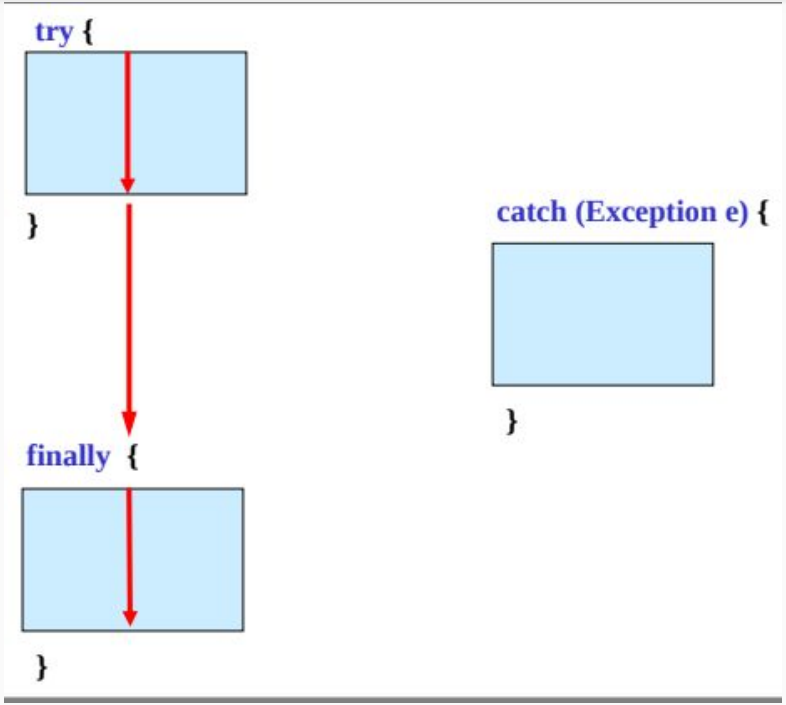
- Três tipos de exceções: *checked*, *error* e *runtime*.
- Checked é obrigatório tratar
- Runtime é opcional



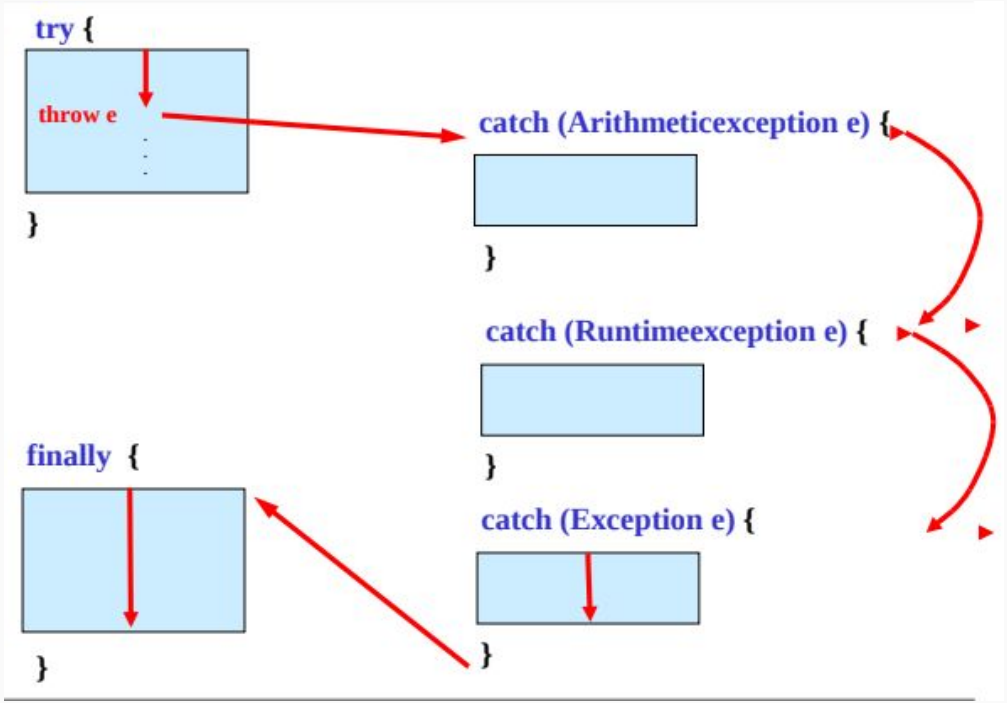
Exceções - Fluxos



Exceções - Fluxos



Exceções - Fluxos



Exceções - Métodos

- Na definição do método dizemos quais exceções ele poderá lançar
- A instrução throws indica que o método poderá lançar uma ou mais exceções
- A lista de exceções é separada por vírgulas
- Se a exceção for do tipo runtime, poderá ser omitida da lista de exceções
- Exemplo runtime e checked: public void writeList() throws IOException, IllegalArgumentException
- Exemplo checked: public void writeList() throws IOException

Exceções - Lançamento

- Para lançar (ou arremessar) uma exceção utilizamos o comando *throw*
- O comando requer um objeto. Exemplo: throw new FileNotFoundException()
- Você poderá lançar exceções pré-existente
- Você também poderá lançar exceções que você mesmo criou
- Reaproveitar um tipo de exceções genericamente pode ser uma ideia ruim…
- Exemplo: MinhaExcecaoException para todos os eventos adversos da sua aplicação. O que vai diferenciar um registro duplicado de um dado inválido?

Exceções - Criando sua própria Exception

- Estender de Error, RuntimeException ou Exception
- Usualmente você irá estender de Exception. Por que?
- Não é obrigatório, mas sim uma boa prática apendar o nome Exception ao final do nome da sua exceção. Exemplo: FaltandoDinheiroException
- Isso é suficiente para criar uma exceção própria: public class FaltandoDinheiroException extends Exception {}
- Vamos olhar com calma: https://docs.oracle.com/javase/8/docs/api/java/lang/Exception.html

Exceções - Boas práticas

- Artigo muito interessante: https://dzone.com/articles/exception-handling-a-best-practice-guide
- Vamos pensar juntos:
  - não achar algo é uma exceção?
  - achar mais de alguma coisa é uma exceção?
  - problemas de infra são considerados exceções? Como tratar de fato? Caiu no catch… e daí, o que eu faço?
- Design do código fica coeso com relação ao tratamento do erro da negra de negócio: um bloco faz o que de fato precisa ser feito e outro bloco distinto trata somente o “erro”
- Exceções custam caro a VM em runtime

Exceções - Exercícios

- Artigo muito interessante: https://dzone.com/articles/exception-handling-a-best-practice-guide
- Vamos pensar juntos:
  - não achar algo é uma exceção?
  - achar mais de alguma coisa é uma exceção?
  - problemas de infra são considerados exceções? Como tratar de fato? Caiu no catch… e daí, o que eu faço?
- Design do código fica coeso com relação ao tratamento do erro da negra de negócio: um bloco faz o que de fato precisa ser feito e outro bloco distinto trata somente o “erro”
- Exceções custam caro a VM em runtime

Exceções - Boas práticas

- https://docs.oracle.com/javase/tutorial/essential/exceptions/QandE/questions.html
- Não vale colar...

Revisão

- Exceções são primeiramente design elegante e coeso para tratamento de “erros”
- Exceções são eventos que fogem do fluxo normal esperado
- O bloco catch serve para tentar recuperar o erro ocorrido no bloco try
- O bloco finally serve para limpeza geral do bloco try
- Exceções custam caro a VM e devem ser usadas com sabedoria