DANIEL ABELLA
www.daniel-abella.com

Daniel Abella

Profissional PMP, PMI-ACP e CSM, Gerente de Projetos no Virtus & UniFacisa e Professor do Curso de Sistemas de Informação da UniFacisa. Autor dos livros Gestão A3 (Ágil) e Scrum Arretado.

Capítulo 7

Trabalhando com *Exceptions*

Livro Java do Zero (Uma Viagem ao Mundo Java)

Conteúdo:

01 Introdução

- Exceptions é uma forma de sinalizar condições excepcionais que podem ocorrer durante a operação, como por exemplo em um método de saque, a falta de fundos.
- Para usar as Exceptions, são 3 passos, relacionados a seguir.

02 Passo 1 - Criação da Exception

- A primeira coisa que precisamos fazer é, criar uma classe que represente a nossa situação excepcional, ou seja, a nossa Exception. Existem dois tipos de Exceptions: as Checked Exception (subclasse de Exception) e Unchecked Exception (subclasse de RuntimeException).
- Com base no nosso exemplo, vamos criar uma Exception intitulada SaldoInsuficienteException, que deve ser lançada quando o cliente não tem saldo disponível para o valor do saque solicitado. Abaixo apresentamos duas versões para esta Exception, uma subclasse de RuntimeException e outra de Exception.
- Como boa prática, é interessante oferecer a quem for instanciar a Exception, ter ao menos os 2 construtores abaixo definidos, um que dispõe de uma String, na qual podemos informar alguma mensagem referente à situação exception (como causa-raiz, por exemplo) e outro vazio, no caso de não termos maiores detalhes.

UNCHECKED EXCEPTION

```
public class SaldoInsuficienteException
    extends RuntimeException {

    public SaldoInsuficienteException() {
        super();
    }

    public SaldoInsuficienteException(String message) {
        super(message);
    }
}
```

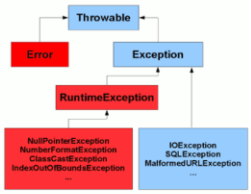
CHECKED EXCEPTION

```
public class SaldoInsuficienteException
    extends Exception {

    public SaldoInsuficienteException() {
        super();
    }

    public SaldoInsuficienteException(String message) {
        super(message);
    }
}
```

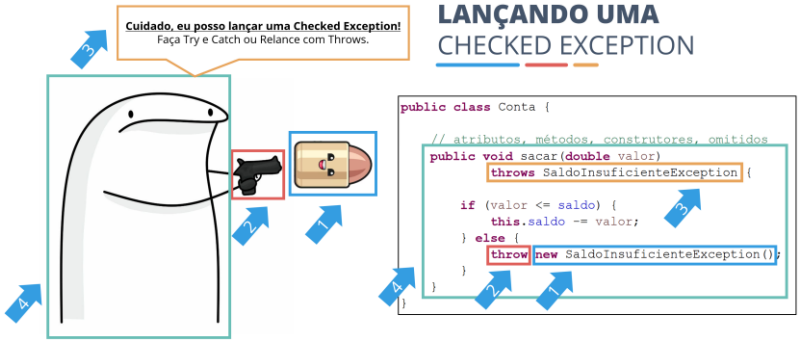
TIPOS DE EXCEPTIONS



03 Passo 2 - Lançar uma Checked Exception

- Apresentamos a seguir uma nova versão do método sacar que, lança a referida Exception quando o saldo é insuficiente para o saque.

LANÇANDO UMA CHECKED EXCEPTION



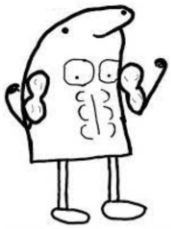
- O que gostaria que você tivesse em sua mente sempre que for criar um método como o sacar que, em dada situação pode lançar uma Checked Exception é a do Flork com uma arma na mão, dizendo que pode (não quer dizer que vai) atirar uma Checked Exception.
- A bala, propriamente dita, é a instância da Checked Exception, conforme indicado na seta 1. A instrução throw é a que atira, ou seja, a pistola, conforme indicado na seta 2. Devido ao fato de ser uma Checked Exception, precisamos avisar ao método que vai chamar da possibilidade de ele receber uma bala (instância de uma Checked Excepton). Isto é feito, conforme indicado na seta 3, através da instrução throws **NomeDaClasseDaCheckedException**. Por fim, na seta 4, tudo acontece dentro de um método, ou seja, feito pelo Flork.
- Perceba que, para lançar/atirar uma Checked Exception, usamos o throw (sem o s), enquanto que, para avisar da Checked Exception usamos o throws (com s).

04 Passo 3 - Método que Pode Lançar uma Checked Exception

- Para chamar um método que eventualmente lança uma Checked Exception, temos duas opções:
  - Tratar usando Try e Catch
  - Relançar usando Throws
- Ambas as opções são apresentadas abaixo e detalhadas na próxima página. A primeira opção, na parte superior da imagem, chamamos o método sacar. Note que, como optamos por tratar a Exception, temos que envolver o código que pode lançar uma Exception com o try e catch.

- Entre o try e o catch, indicado na seta 2, devemos informar o código passível de lançar Exception, neste caso, a chamada ao método sacar. Dentro do catch, indicado na seta 3, temos o código que lida com a Exception.

CHAMANDO UM MÉTODO COM CHECKED EXCEPTION

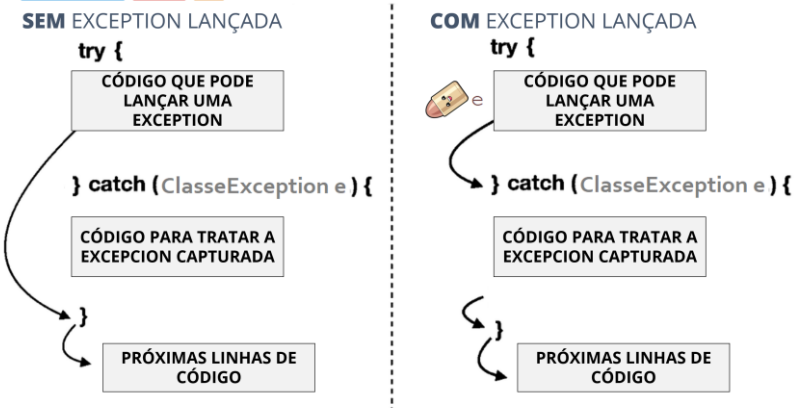


```
public class Teste {
    public static void main(String[] args) {
        Conta novaConta = new Conta(1,123,"Daniel",1000);
        try {
            novaConta.sacar(10);
        } catch (SaldoInsuficienteException e) {
            e.printStackTrace();
        }
    }
}
```

```
public class Teste {
    public static void main(String[] args)
            throws SaldoInsuficienteException {
        Conta novaConta = new Conta(1,123,"Daniel",1000);
        novaConta.sacar(10);
    }
}
```

- Neste exemplo, chamamos `e.printStackTrace();` . O método printStackTrace é um método herdado de uma superclasse, na qual exibe detalhes como o nome da classe e o número da linha onde a Exception ocorreu.
- O funcionamento do try e catch é apresentado a seguir. À esquerda, temos uma situação em que, dentro do try, o código não lança nenhuma Exception. Desta maneira, note que, o block catch é ignorado e seguimos a execução das próximas linhas. Por outro lado, à direita, situação em que uma Exception é lançada dentro do try, o block catch é executado e, logo depois seguimos a execução das próximas linhas.

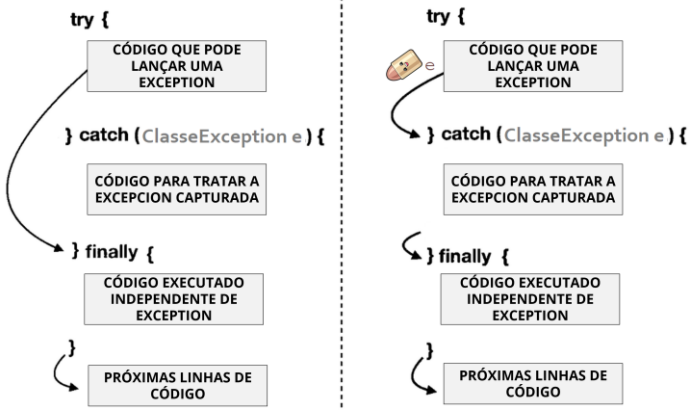
FUNCIONAMENTO DO TRY E CATCH



- Existe uma instrução opcional intitulada finally, associada ao try/catch, cujo funcionamento é relacionado a seguir. O finally é chamado independentemente de ter sido lançada uma Exception ou não, geralmente usado para liberar algum recurso usado dentro do trecho try. Um exemplo do finally é quando, dentro do try criamos e usamos uma conexão com o banco de dados e, podemos usar o finally para encerrar a conexão.

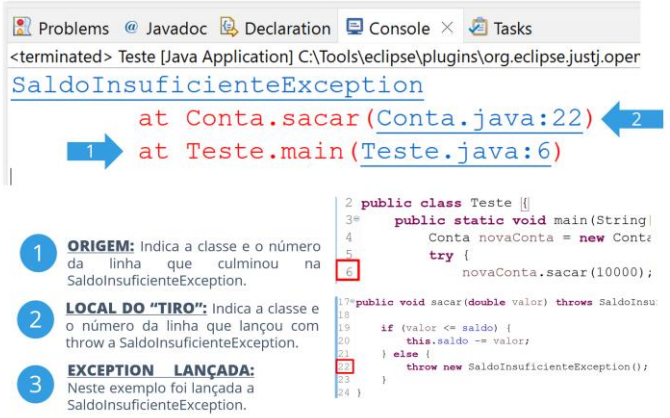
> O finally funciona tanto para Checkex Exceptions e Unchecked Exceptions. O seu objetivo é finalmente encerrar ações criadas no try/catch.

FUNCIONAMENTO DO TRY, CATCH E FINALLY



- O método printStackTrace() apresenta o que aconteceu na Exception, a qual chamamos de track trace.

ENTENDENDO A STACKTRACE

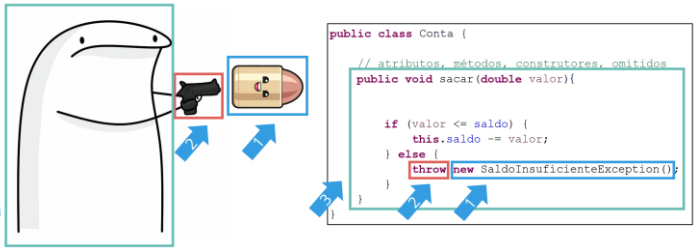


- A partir de agora, apresentaremos como lidar com Unchecked Exceptions. Note que, o passo 1 (criação da Exception) foi apresentada anteriormente.

05 Passo 2 - Lançar uma Unchecked Exception

- Apresentamos abaixo uma nova versão do método sacar que, lança a referida RuntimeException quando o saldo é insuficiente para o saque.

LANÇANDO UMA UNCHECKED EXCEPTION



- A bala é uma instância da Checked Exception, conforme indicado na seta 1. A instrução throw é a que atira, ou seja, a pistola, conforme indicado na seta 2. Devido ao fato de ser uma Unchecked Exception, NÃO precisamos avisar ao método que vai chamar da possibilidade de ele receber uma bala (instância de uma Unchecked Excepton), que era feito com throws. Por fim, na seta 3, tudo acontece dentro de um método, ou seja, feito pelo Flork.

06 Passo 3 - Chamando um Método que Pode Lançar uma Unchecked Exception

- Para chamar um método que eventualmente lança uma Unchecked Exception não somos obrigados a tratar usando Try e Catch nem como relançar usando Throws, conforme apresentado a seguir. No primeiro exemplo, que não usamos try/catch, caso seja lançada uma Unchecked Exception, esta "vai cair na cara do usuário", pois não teve tratamento, diferente do segundo exemplo que, apesar de ser Unchecked, pode ocorrer algum tratamento dentro do catch.

CHAMANDO UM MÉTODO
COM UNCHECKED EXCEPTION

```
public class Teste {
    public static void main(String[] args) {
        Conta novaConta = new Conta(1, 123, "Daniel", 1000);
        novaConta.sacar(10);
    }
}
```

```
public class Teste {
    public static void main(String[] args) {
        Conta novaConta = new Conta(1,123,"Daniel",1000);
        try {
            novaConta.sacar(10);
        } catch (SaldoInsuficienteException e) {
            e.printStackTrace();
        }
    }
}
```

TRATAMENTO COM TRY/CATCH OPCIONAL PARA UNCHECKED EXCEPTION

CHECKED VS. UNCHECKED

Agora que sabemos o funcionamento de *Checked* e *Unchecked Exceptions*, vamos entender quando uma Exception deve ser de um tipo ou de outro.

Checked Exceptions, como vimos, exige que, quem chame um método que eventualmente lance uma Exception deste tipo, trate com try/catch ou relance com throws. Isto é o caso de ValorInvalidoException, SaldoInsuficienteException, entre outras, na qual dentro do bloco catch podemos fazer algum tratamento como informar que o valor está inválido e solicitar que digite novamente (no caso de ValorInvalidoException) e informar o saldo disponível para saque (no caso de SaldoInsuficienteException).

Unchecked Exceptions, não exige nenhum tratamento e é o caso de situações inerparadas em que a situação excepcional ocorre e não pode ser tratado, como é o caso do NullPointerException inesperado. Isto é o caso de NullPointerException, ConexaoBancoDadosInvalidaException, entre outros.

07 Trabalhando com Vários Catch

- Nos exemplos nas seções anteriores, apresentamos sempre um try associado a um único catch. Entretanto, é possível termos vários blocos catch seguidos, cada um tratando um tipo específico de Exception.
- A seguir, temos um exemplo de múltiplos catch. Se o código dentro do try lançar uma ArrayIndexOutOfBounds, será executado o bloco indicado na seta 1. Caso seja lançada uma ArithmethicException, será executado o bloco indicado na seta 2. Por fim, se for lançada alguma Exception que não seja nenhuma das duas supracitadas, o bloco indicado na seta 3 será executado.
- À esquerda da imagem a seguir, note que, colocamos algumas luvas. A luva de beisebol (ao lado da seta 3), representa o catch genérico, que trata todas as Exceptions. Se você trocar os blocos das setas 1 e 3 de ordem, teremos um erro de compilação. Deve-se ao fato que, se o catch genérico é o primeiro da ordem, os catchs abaixo nunca seriam chamados.

MÚLTIPLOS
CATCH

```
public class TesteException {
    public static void main(String[] args) {
        try {
            int[] numeros = { 1, 2, 3 };
            System.out.println(numeros[5]);
            // Tentando acessar um índice inválido
            //Lança ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException ex) {
            // Tratamento específico para a exceção ArrayIndexOutOfBoundsException
            ex.printStackTrace();
        } catch (ArithmeticException ex) {
            // Tratamento específico para a exceção ArithmeticException
            ex.printStackTrace();
        } catch (Exception ex) {
            // Tratamento genérico para qualquer outra exceção que possa ocorrer
            System.out.println("Ocorreu uma exceção: " + ex.getMessage());
        }
    }
}
```

1. **ArrayIndexOutOfBoundsException:** Catch específico para tratamento desta Exception.
2. **ArithmeticException:** Catch específico para tratamento desta Exception.
3. **Catch Genérico:** Senão for lançada outra *Exception* que não seja ArrayIndexOutOfBounds (catch da seta 1) nem ArithmeticException (catch da seta 2), vai cair neste Catch genérico.

- Uma outra possibilidade é termos apenas um catch, o catch genérico, como apresentado na imagem a seguir. Esta abordagem funciona se, você quiser tratar todas as Exceptions da mesma maneira, o que nem sempre acontece.

```
public class TesteException {
    public static void main(String[] args) {
        try {
            int[] numeros = { 1, 2, 3 };
            System.out.println(numeros[5]);
            // Tentando acessar um índice inválido
            // Lança ArrayIndexOutOfBoundsException
        } catch (Exception ex) {
            // Tratamento genérico para qualquer outra exceção que possa ocorrer
            System.out.println("Ocorreu uma exceção: " + ex.getMessage());
        }
    }
}
```

08 Try with Resources

- Try with Resources é um recurso do Java 7 que, possibilita encerrar automaticamente recursos abertos no try. Abaixo temos a sua estrutura e exemplo.

TRY-WITH-RESOURCES (JAVA 7+)

```
try ( declaração dos recursos ) {
    USO DOS RECURSOS
    USO DOS RECURSOS
}
```

```
try ( declaração dos recursos ) {
    USO DOS RECURSOS
    USO DOS RECURSOS
} catch ( MinhaException e ) {
    TRATAMENTO DA EXCEPTION
} finally {
    TRATAMENTO DA EXCEPTION
}
```

- Para este funcionamento, a classe precisa implementar AutoClosable ou Closable.

```
1  import java.io.PrintWriter;
2
3  public class TesteException {
4
5      public static void main(String[] args) {
6
7          try (PrintWriter pw = new PrintWriter ("teste.txt")) {
8              pw.println ("teste");
9          } catch (Exception e) {
10             e.printStackTrace();
11         }
12     }
13 }
```