Domain Driven Design

PROF. ELIANE RODRIGUES MARION SANTA ROSA
Profeliane.rosa@fiap.com.br

1

EXCEÇÕES

TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

FIAP

Quando se constrói aplicações é comum a ocorrência de erros imprevistos durante a sua execução, em Java, esses erros são conhecidos como exceções e podem ser provenientes de erros de lógica ou acesso a dispositivos ou arquivos externos.

Alguns possíveis **motivos externos** para ocorrer uma exceção são:

- Tentar abrir um arquivo que não existe;
- Tentar fazer consulta a um banco de dados que não está disponível;
- Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita;
- Tentar conectar em servidor inexistente.

TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

FIAP

Entendendo as exceções decorrentes de possíveis **erros de lógica:**

- Tentar manipular um objeto que está com o valor nulo;
- Dividir um número por zero.
- Tentar manipular um tipo de dado como se fosse outro;
- Tentar utilizar um método ou classe não existentes;

TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

Você fez um programa mas em tempo de execução ocorre um problema, como já vimos, chamamos este problema de exceção. “A exceção é um problema que pode ocorrer mas não esperamos que ocorra com frequência”.

Precisamos tratar a exceção para que nosso programa não trave ou encerre diante de um problema

TRATANDO AS EXCEÇÕES

Uma maneira de tentar contornar esses imprevistos é realizar o tratamento dos locais no código que podem vir a lançar possíveis exceções, como por exemplo, campo de consulta a banco de dados, locais em que há **divisões**, **consulta a arquivos** de propriedades ou arquivos dentro do próprio computador.

Para **tratar as exceções em Java** são utilizados os comandos try e catch.

Sintaxe:

TRATANDO AS EXCEÇÕES

```
try
{
  //trecho de código que pode vir a lançar uma exceção
}
catch(tipo_exceçao_1 e)
{
  //ação a ser tomada
}
catch(tipo_exceçao_2 e)
{
  //ação a ser tomada
}
catch(tipo_exceçao_n e)
{
  //ação a ser tomada
}
```

TRATANDO AS EXCEÇÕES

Onde:

try{ ... } - Neste bloco são introduzidas todas as linhas de código que podem vir a lançar uma exceção.

catch(tipo_excessao e) { ... } - Neste bloco é descrita a ação que ocorrerá quando a exceção for capturada.

EXEMPLO DE EXCEÇÃO

Exemplo:

1. Criar a classe Conversor com o atributo privado valor do tipo String.
2. Criar os getters e setter.
3. Criar o método converter() abaixo:

```
public void converter() {
    int resultado = Integer.parseInt(valor);
    System.out.println("Valor convertido: " + resultado);
}
```

EXEMPLO DE EXCEÇÃO

1. Criar a classe TestaConversor com o método main.

2. Instanciar a classe Conversor e atribuir o valor “12”.

3. Executar e verificar o resultado

4. Altere o valor para “12aa” e execute novamente.

5. Aparecerá a seguinte mensagem:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "12s"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)
        at java.base/java.lang.Integer.parseInt(Integer.java:658)
        at java.base/java.lang.Integer.parseInt(Integer.java:776)
        at Conversor.converter(Conversor.java:14)
        at TesteConversor.main(TesteConversor.java:8)
```

CONTINUAÇÃO DO EXEMPLO

- Para corrigir este erro podemos lançar o try...catch, verificando o erro podemos identificar o tipo de exceção: NumberFormatException.

```
try {
    Conversor c1 = new Conversor();
    c1.setValor("12s");
    c1.converter();
} catch (NumberFormatException e) {
    System.out.println("Valor informado não é um número!");
}
```

6. Ao executar novamente aparecerá a seguinte mensagem:

Valor informado não é um número!

2

EXERCÍCIO

EXERCÍCIO SOBRE EXCEÇÃO

Criar a classe Divisor com os atributos inteiros privados (numero1 e numero2).

2 - Criar os getters e setter.

3 - Criar o método calcular() abaixo:

```
public void calcular() {
        double resultado = numero1 / numero2;
        System.out.println("Resultado: " + resultado);
}
```

EXERCÍCIO

4 - Criar a classe TestaDivisor com o método main.

5 - Instanciar a classe Divisor e atribuir os valores numero1 = 10 e numero2 = 2.

6 - Executar para ver o resultado feliz :)

```
public class Teste {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Divisor d = new Divisor();
        d.setNumero1(10);
        d.setNumero2(2);
        d.calcular();

    }

}
```

Resultado

```
Resultado: 5.0
```

EXERCÍCIO

7 - Alterar numero2 = 0 e executar novamente:

```java
public class Teste {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Divisor d = new Divisor();
        d.setNumero1(10);
        d.setNumero2(0);
        d.calcular();

    }

}
```

Resultado

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Divisor.calcular(Divisor.java:20)
        at Teste.main(Teste.java:9)
```

EXERCÍCIO

8 – Tratar a exceção informada:

```
public class Teste {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Divisor d = new Divisor();

        try {
            d.setNumero1(10);
            d.setNumero2(0);
            d.calcular();
        }
        catch (ArithmeticException e) {
            System.out.println("Operação Invalida! | Não existe divisão por 0");
        }
    }
}
```

EXCEÇÃO

As exceções são objetos que descrevem um erro, indicando a causa do erro, a linha de código onde o erro aconteceu, e possivelmente outras informações.

- Existem várias classes de exceção, como ArithmeticException, NullPointerException, Error e muitas outras.
- Todas as classes que representam exceções são subtipos, direta ou indiretamente, da classe Throwable (lançável)

3

throw e throws

TRATAMENTO DE EXCEÇÕES

Quando uma exceção é lançada (throw), a JVM entra em estado de alerta e verificará se o método atual toma alguma precaução ao tentar executar o código.

Exceções mais comuns:

Exception: é a super-classe de todas as classes que tratam exceções.

NullPointerException: quando você aponta para algo que está nulo, como por exemplo, um atributo de um objeto nulo.

ArrayIndexOutOfBoundsException: dispara quando você aponta para uma posição que não existe dentro de uma matriz. Exemplo, apontar para o elemento 11 sendo que a matriz possui 10 elementos.

TRATAMENTO DE EXCEÇÕES

FIAP

NumberFormatException: lança a exceção quando o formato do dado passado não condiz com o tipo de dado esperado, exemplo, quando digita-se uma letra para um tipo de dado inteiro.

FileNotFoundException: quando não encontra o arquivo especificado no código.

IllegalArgumentException: dispara quando é fornecido um parâmetro fora dos padrões estabelecidos, por exemplo, uma referência nula.

4

Checked Exceptions

CHECKED EXCEPTIONS

- Obriga quem chama o método ou construtor a tratar essa exceção.
- O compilador checará se ela está sendo devidamente tratada.

Exemplo:

```
public class TesteCheckedException {

    public static void main(String[] args) {

        new java.io.FileInputStream("arquivo.txt");

    }

}
```

CHECKED EXCEPTIONS

- O código não compila e avisa que é necessário tratar o FileNotFoundException.
- Existe duas formas de resolver:
- 1ª Usando o try...catch

```
try {
    new java.io.FileInputStream("arquivo.txt");
}
catch (java.io.FileNotFoundException e) {
    System.out.println("Nao foi possível abrir o arquivo para leitura");

}
```

CHECKED EXCEPTIONS

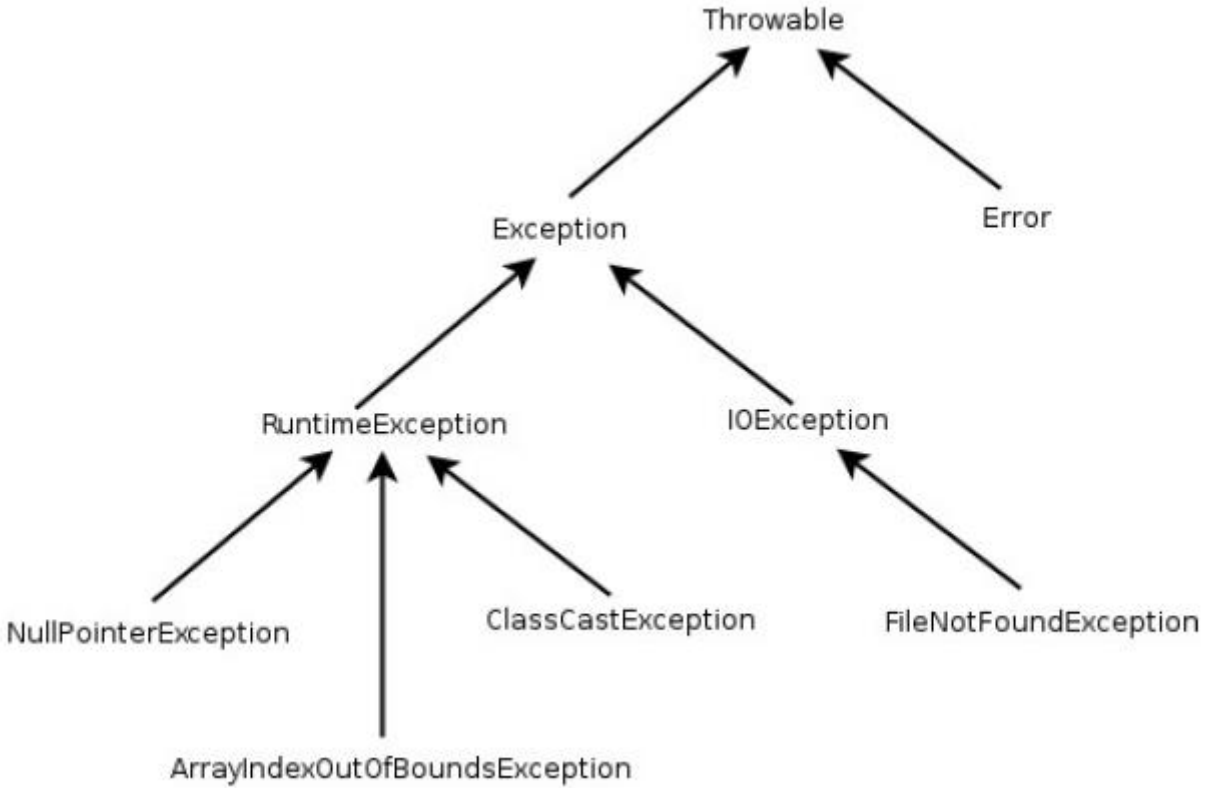
FIAP

2ª Delegá-lo a quem chamou o nosso método, isto é, passar para a frente.

```
public static void main(String[] args) throws java.io.FileNotFoundException {

    new java.io.FileInputStream("arquivo.txt");
```

FAMÍLIA THROWABLE



TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

FIAP

Podemos tratar mais de um erro tanto usando o try..catch quanto lançando a exceção throw.

Com o try e catch:

```
try {
        objeto.metodoQuePodeLancarIOeSQLException();
} catch (IOException e) {
// ..
} catch (SQLException e) {
// ..
}
```

Com o throws :

```
public void abre(String arquivo) IOException,
SQLException {throws
// ..
}
```

TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

FIAP

Dependendo da situação é possível tratar uma exceção e lançar outras

```
public void abre(String arquivo) throws IOException {
      try {
            objeto.metodoQuePodeLancarIOeSQLException();
      } catch (SQLException e) {
      // ..
      }
}
```

THROW

Podemos também lançar uma exceção. A palavra-chave **throw**, que está no imperativo, lança uma Exception. Isso é bem diferente de throws , que está no presente do indicativo e só avisa da possibilidade daquele método lançá-la, obrigando o outro método que vá utilizar-se daquele a se preocupar com essa exceção em questão.

Exemplo:

```
public void saca(double valor) {
    if (this.saldo < valor) {
        throw new RuntimeException();
    } else {
        this.saldo-=valor;
    }
}
```

THROW

- Nesse exemplo foi lançada uma exceção do tipo ***unchecked.***
- RuntimeException é a exception mãe de todas as exceptions unchecked.
- A desvantagem aqui é que ela é muito genérica; quem receber esse erro não saberá dizer exatamente qual foi o problema. Podemos então usar uma exception mais específica:

```
public void saca(double valor) {
    if (this.saldo < valor) {
        throw new IllegalArgumentException();
    } else {
        this.saldo-=valor;
    }
}
```

THROW

- Nesse exemplo foi lançada uma exceção do tipo ***unchecked.***
- IllegalArgumentException: algo foi passado como argumento, e seu método não gostou. Ela é uma Exception unchecked, pois estende de RuntimeException , e já faz parte da biblioteca do Java.
- É a melhor escolha quando um argumento sempre é inválido, por exemplo, números negativos, referências nulas, etc.).
- Quando lançamos uma exceção, precisamos trata-la ao invocar o método.

Bloco Finally

- Finally é um bloco opcional, que pode ser incluído após os blocos catch e reúne todas as instruções que serão executadas obrigatoriamente no programa, por exemplo o fechamento do objeto.

```
try {
        // bloco try
} catch (IOException ex) {
        // bloco catch 1
} catch (SQLException sqlex) {
        // bloco catch 2
} finally {
        // bloco que será sempre executado, independente
        // se houve ou não exception e se ela foi tratada ou não
}
```

5

EXERCÍCIO

EXERCÍCIO SOBRE EXCEÇÃO

1. Importe o projeto pw2-exceção para sua workspace.

2 - No Eclipse, localize (use a combinação Ctrl + Shift + R) o arquivo Exemplo2Application e execute o arquivo.

3 - Coloque uma data de nascimento no formato 01/01/2022 e clique em Cadastrar.

4 - Imagine que, no sistema, um cliente deva ter idade maior ou igual a 18 anos. Para isso, abra a classe ValidatorUtil com a combinação CTRL + SHIFT + R.

5 - O método de validação de data de nascimento possui a indicação //TODO Implementar o lançamento da exceção (aproximadamente linha 44).

EXERCÍCIO SOBRE EXCEÇÃO

FIAP

5 - Troque a instrução pelo lançamento de exceção de RuntimeException com a mensagem descritiva referente a data de nascimento.

6 - Localize (use a combinação Ctrl + Shift + R) o arquivo ClienteService.

7 - Inclua a chamada para o método validarNascimento() de ValidatorUtil passando a data de nascimento do cliente.

8 - Refaça os testes com os mesmos valores.

9 - Observe que o console do Eclipse exibirá o erro java.lang.RuntimeException.

10 - Na classe ValidatorUtil, complete o método validarNome() para verificar se foi digitado o nome completo do nome passado como parâmetro, lançando a exceção RuntimeException.

FIAP

6

CRIANDO SUAS PRÓPRIAS EXCEÇÕES

LANÇANDO NOSSAS PRÓPRIAS EXCEÇÕES

O JAVA possui uma série de exceções nativas, que podemos utilizar para identificar e personalizar as mensagens e ações, entretanto haverá situações em que precisaremos lançar exceções específicas da aplicação.

- Para lançar uma exceção utilizamos o throw

LANÇANDO NOSSAS PRÓPRIAS EXCEÇÕES

FIAP

```
public class Funcionario {

   private String nome;
   private String email;
   private Double salario;

//Getters and Setters
  public void setSalario(Double salario) {
    if(salario < 0.0) {
       throw new IllegalArgumentException("O salário não pode
ser negativo");
    }
  this.salario = salario;
  }
```

Utilizamos neste exemplo a exceção do tipo **IllegalArgumentException**

É ela que deve ser disparada quando um argumento ilegal for passado para dentro de um método ou função.

Mas podemos criar nossas próprias exceções personalizadas, exemplo:

```
3  public class MinhaExcecao extends Exception{
4
5
6      private static final long serialVersionUID = 1L;
7      public MinhaExcecao(String texto) {
8          super(texto);
9      }
10
11 }
```

TRATAMENTO DE EXCEÇÕES E CONTROLE DE ERROS

FIAP

Para lançar uma exceção do tipo criado o programador deve indicar a subclasse que foi criada.

```
1  public class ClassExemplo {
2      public void MetodoExemplo() throws MinhaExcecao{
3          throw new MinhaExcecao("Exceção personalizada");
4      }
5  }
```

TRATANDO AS EXCEÇÕES

Usando o try..catch a exceção pode ser identificada da seguinte maneira:

```
1  try {
2      //acao
3  }catch(MinhaExcecao excecao){
4      System.out.println(excecao.getMessage());
5  }
```