

Tratamento de Exceções em Java

Exercício

Vinicius Takeo Friedrich Kuwaki

Universidade do Estado de Santa Catarina

Seções

Exemplo

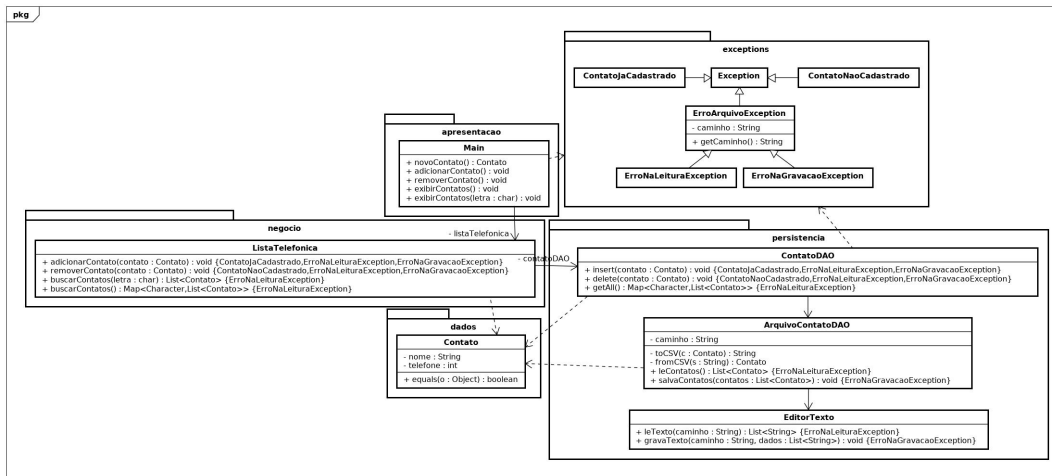
Resolução

Exercício

Exemplo

- Partindo do exercício da Aula Prática 7: Persistência de Dados em Arquivos, crie as exceções necessárias para tratar os possíveis problemas relacionados à manipulação do arquivo de dados e às inserções e remoções que não ocorrerem com sucesso na classe ContatoDAO. Ainda, trate-as corretamente no pacote de apresentação;
- O diagrama a seguir apresenta a modelagem das exceções lançadas pelos métodos do projeto do exercício da Aula Prática 7.

Exemplo - Diagrama



Exemplo - Descrição

- É necessário criar uma classe chamada `ErroArquivoException` que estende a classe `Exception`;
- Essa classe deve conter um atributo do tipo `String` chamado `caminho`;
- Ela deve ser especializada em duas outras classes chamadas: `ErroNaLeituraException` e `ErroNaGravaçãoException`;
- Os métodos da classe `EditorTexto` devem lançar essas exceções, enviando dentro delas o caminho do arquivo;
- Crie mais duas classes que estendem a classe `Exception` chamadas: `ContatoJaCadastrado` e `ContatoNaoCadastrado`;
- Os métodos **`insert()`** e **`delete()`** da classe `ContatoDAO` devem lançar essas exceções quando o contato já estiver cadastrado ou quando o contato não for encontrado, respectivamente;
- Para verificar a existência ou não do contato é necessário implementar o método **`equals()`** na classe `Contato`;

Seções

Exemplo

Resolução

Exercício

Resolução - Classe Contato

- Vamos utilizar como base a classe Contato implementada na Aula Prática 7;
- Nela, vamos adicionar a implementação do método **equals()**;
- Por padrão, ele recebe um objeto do tipo Object;
- Vamos fazer um Casting para Contato;
- A condição de comparação dos contatos adotada é a igualdade dos seus telefones;
- Não iremos nos preocupar com o nome:

```
public boolean equals(Object object) {  
    Contato c = (Contato) object;  
    return c.getTelefone() == this.telefone;  
}
```

Resolução - Classe ErroArquivoException

- Vamos criar agora o pacote exceptions;
- A primeira classe desse pacote vai ser a ErroArquivoException;
- Ela vai ter um atributo do tipo String que conterà o caminho do arquivo de dados;
- Dentro do seu construtor será invocado o construtor da super classe Exception, passando uma mensagem a ser recebida como parâmetro:

```
public class ErroArquivoException extends Exception {  
    private String caminho;  
    public ErroArquivoException(String mensagem) {  
        super(mensagem);  
    }  
}
```

- A implementação do get e set do caminho não será mostrada;
- Essa classe é especializada em duas outras que serão mostrada a seguir.

Resolução - Classe ErroNaLeituraException

- A classe ErroNaLeituraException estende a classe ErroArquivoException, chamando o construtor da super classe com uma mensagem alertando sobre o erro:

```
public class ErroNaLeituraException extends ErroArquivoException {  
    public ErroNaLeituraException() {  
        super("Erro durante a leitura do arquivo");  
    }  
}
```

Resolução - Classe ErroNaGravacaoException

- O mesmo para a classe ErroNaGravacaoException:

```
public class ErroNaGravacaoException extends ErroArquivoException {  
    public ErroNaGravacaoException() {  
        super("Erro durante a gravação do arquivo");  
    }  
}
```

Resolução - Classe ContatoJaCadastradoException

- A classe ContatoJaCadastrado segue a mesma lógica, apenas com uma mensagem diferente;
- Ela estende a classe Exception:

```
public class ContatoJaCadastradoException extends Exception {  
    public ContatoJaCadastradoException() {  
        super("Contato ja cadastrado!");  
    }  
}
```

Resolução - Classe ContatoNaoCadastradoException

- Novamente, a classe ContatoNaoCadastradoException repete a mesma lógica:

```
public class ContatoNaoCadastradoException extends Exception {  
    public ContatoNaoCadastradoException() {  
        super("Contato nao cadastrado!");  
    }  
}
```

Resolução - Classe EditorTexto

- Vamos agora lançar essas exceções na classe EditorTexto;
- Começando pelo método **gravaTexto()**;
- Utilizaremos a palavra reservada **throws** com **'s'** ao final da declaração do método seguida do nome da exceção a ser lançada:

```
public void gravaTexto(String caminho, List<String> dados) throws  
    ErroNaGravacaoException {  
    . . .  
}
```

Resolução - Classe EditorTexto

- Na cláusula catch, iremos fazer alterações;
- Observe como era a implementação antes:

```
catch (Exception e) {  
    System.err.println("Erro ao manipular o arquivo");  
    System.exit(0);  
}
```

- Vamos agora utilizar a palavra reservada **throw** agora **sem 's'** para lançar uma exceção do tipo `ErroNaGravacaoException`;
- Primeiro vamos instanciá-la, como um objeto normal, e depois iremos setar o seu atributo caminho;
- Por fim, vamos lançá-la por intermédio da palavra reservada **throw**:

```
} catch (Exception e) {  
  
    ErroNaGravacaoException erro = new ErroNaGravacaoException();  
    erro.setCaminho(caminho);  
    throw erro;  
}
```

- Faça o mesmo para o método **leTexto()**;
- Entretanto, não esqueça que a exceção a ser lançada é a do tipo **ErroNaLeituraException**.

Resolução - Classe ArquivoContatoDAO

- Como o objetivo é tratar a exceção na interface com o usuário, a classe ArquivoContatoDAO apenas propaga / "re-lança" as exceções;
- Portanto, apenas na camada de apresentação utilizaremos o catch para capturarmos elas;
- Até lá, iremos apenas propagá-las;
- Utilizando a palavra **throws** ao final da declaração do método sucedida pela(s) exceção(ões) a serem propagadas;

```
public List<Contato> lerContatos() throws ErroNaLeituraException{  
    ...  
}
```

```
public void salvarContatos(List<Contato> contatos) throws  
    ErroNaGravacaoException{  
    ...  
}
```


Resolução - Classe ContatoDAO

- Na classe ContatoDAO deve-se seguir o mesmo processo;
- As IDE's costumam dar 'hints' a respeito das exceções que os métodos lançam, nesse caso é só seguir a sugestão e incluir as exceções logo após o **throws** na assinatura do método;
- O método **insert()**, pode lançar as seguintes exceções: ContatoJaCadastradoException, ErroNaLeituraException e ErroNaGravacaoException, então, colocaremos elas na assinatura do método.

```
public void insert(Contato contato) throws ContatoJaCadastradoException ,  
    ErroNaLeituraException , ErroNaGravacaoException {  
    ...  
}
```

Resolução - Classe ContatoDAO

- Esse método irá lançar uma exceção caso o contato que iremos cadastrar já esteja salvo;
- Precisamos primeiro recuperar a lista de contatos já salvos
- Faremos isso chamando o método **lerContatos()** da classe ArquivoContatoDAO.
- Após isso verificamos se o contato passado como parâmetro para o método se encontra nessa lista, caso sim, lançaremos uma exceção;
- Caso contrário inserimos o contato na lista e salvamos utilizando o método **salvarContatos()**, também da classe ArquivoContatoDAO;
- Observe a implementação:

Resolução - Classe ContatoDAO

```
public void insert(Contato contato) throws ContatoJaCadastradoException ,
    ErroNaLeituraException , ErroNaGravacaoException {

    List<Contato> contatos = arquivoContatoDAO.lerContatos();

    if (!contatos.contains(contato)) {

        contatos.add(contato);
        arquivoContatoDAO.salvarContatos(contatos);

    } else {

        throw new ContatoJaCadastradoException();

    }

}
```

Resolução - Classe ContatoDAO

- O método **delete()** repete a implementação do **insert()**
- A unica diferença é que a lógica é a inversa;
- Caso o contato **não** esteja na lista obtida, iremos lançar a exceção;
- Caso contrário podemos remover o contato;

```
public void delete(Contato contato) throws ContatoNaoCadastradoException ,
    ErroNaLeituraException , ErroNaGravacaoException {
    List<Contato> contatos = arquivoContatoDAO.lerContatos();
    if (contatos.contains(contato)) {
        contatos.remove(contato);
        arquivoContatoDAO.salvarContatos(contatos);
    } else {
        throw new ContatoNaoCadastradoException();
    }
}
```

- Para o método **getAll()** iremos apenas adicionar as exceções que ele pode vir a lançar na assinatura do método;
- Não iremos alterar a implementação:

```
public Map<Character , List<Contato>> getAll() throws ErroNaLeituraException{  
    ...  
}
```

Classe ListaTelefonica

- Vamos para o pacote de negócio;
- A classe ListaTelefonica faz o mesmo para todos os métodos:

```
public void adicionaContato(Contato contato) throws
    ContatoJaCadastradoException, ErroNaLeituraException, ErroNaGravacaoException
{
    contatoDAO.insert(contato);
}
```

```
public void removeContato(Contato contato) throws ContatoNaoCadastradoException
    , ErroNaLeituraException, ErroNaGravacaoException {
    contatoDAO.delete(contato);
}
```

Classe ListaTelefonica

```
public List<Contato> buscarContatos(char letra) throws ErroNaLeituraException {  
    return contatoDAO.getAll().get(letra);  
}
```

```
public Map<Character, List<Contato>> buscarContatos() throws ErroNaLeituraException  
{  
    return contatoDAO.getAll();  
}
```

Classe Main

- Agora na classe Main iremos tratar essas exceções que foram repassadas desde a camada de persistência, até chegarem aqui.
- Começando pelas lançadas pelo método **adicionaContato(contato)** dentro do case 1 do switch do método **main()**;
- Vamos criar as cláusulas **catch** para cada exceção específica que ele lança;
- Para cada exceção vamos exibir a sua mensagem;
- Para as exceções derivadas de `ErroNoArquivoException` vamos exibir também o caminho do arquivo que a exceção enviou:


```
case 1:
    Contato contato = novoContato();
    try {
        lista.adicionaContato(contato);
    } catch (ErroNaLeituraException e) {
        System.out.println(e.getMessage());
        System.out.println("Caminho informado: " + e.getCaminho());
    } catch (ErroNaGravacaoException e) {
        System.out.println(e.getMessage());
        System.out.println("Caminho informado: " + e.getCaminho());
    } catch (ContatoJaCadastradoException e) {
        System.out.println(e);
    }
    break;
```

- Note que deve existir um catch para cada exceção lançada pela chamada do método **adicionaContato()**;
- Cada qual com o seu tratamento específico;

- O método **exibirContatos()** também trata uma exceção, que é lançada pelo método **buscarContatos()**:

```
public static void exibirContatos() {  
    try {  
        lista.buscarContatos().forEach((chave, lista) -> {  
            System.out.println(chave + ":");  
            for (Contato contato : lista) {  
                System.out.println("    " + contato.toString());  
            }  
        });  
    } catch (ErroNaLeituraException e) {  
        System.out.println(e.getMessage());  
        System.out.println("Caminho informado: " + e.getCaminho());  
    }  
}
```

- O mesmo processo é feito para o método **exibirContatos()** que recebe uma letra como parâmetro
- Essa mudança não será mostrada aqui visto que é igual a anterior;
- Para o método **removerContato()** também é necessário envolver a chamada do método **removeContato()** com um bloco try/catch;
- Para cada exceção, trataremos ela em um catch específico;
- O processo é exatamente igual ao realizado no método **adicionarContato()**;
- Observe a implementação:

```
public static void removerContato() {  
  
    System.out.println("Escolha uma letra que deseja remover:");  
    String entrada = s.next().toUpperCase();  
  
    try {  
        if (lista.buscarContatos(entrada.charAt(0)).size() > 0) {  
            exibirContatos(entrada.charAt(0));  
            System.out.println("Escolha um contato para remover:");  
            int index = s.nextInt();  
            if (index < lista.buscarContatos(entrada.charAt(0)).size()) {  
                lista.removeContato(lista.buscarContatos(entrada.charAt(0)).get(  
index));  
            }  
        } else {  
            System.out.println("N o existem contatos para serem removidos");  
        }  
    } catch ...  
}
```

```
public static void removerContato() {  
    ...  
    try{  
        ...  
    } catch (ErroNaLeituraException e) {  
        System.out.println(e.getMessage());  
        System.out.println("Caminho informado: " + e.getCaminho());  
    } catch (ErroNaGravacaoException e) {  
        System.out.println(e.getMessage());  
        System.out.println("Caminho informado: " + e.getCaminho());  
    } catch (ContatoNaoCadastradoException e) {  
        System.out.println(e.getMessage());  
    }  
}
```

Seções

Exemplo

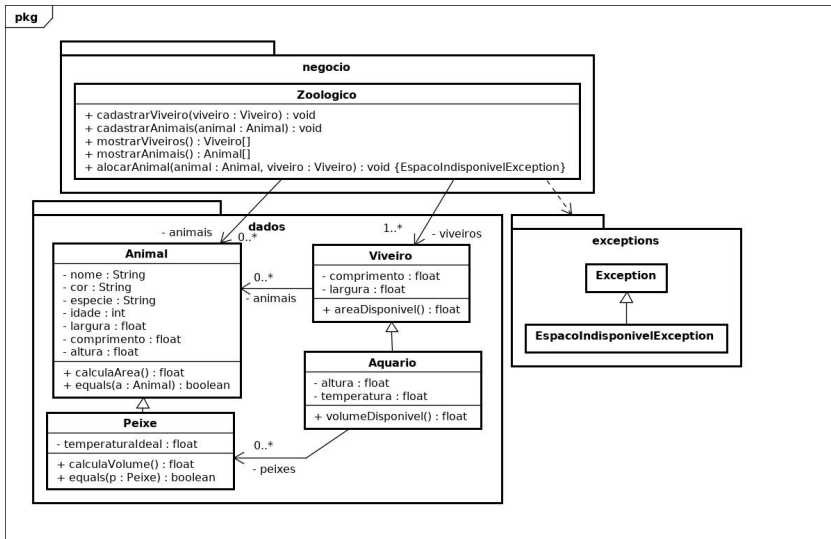
Resolução

Exercício

Exercício


- Continunado o exercício da Aula Prática 2: Classes e Objetos e a partir do Diagrama UML de Classes a seguir, faça o correto tratamento das exceções:

Exercício - Diagrama



Exercício - Descrição

- Continuando o exercício da Aula Prática 4: Herança em Java (diagrama a seguir). Lance uma exceção caso não seja possível alocar um animal em um viveiro;
- Trate corretamente essa Exceção na camada de apresentação;

 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:
Vinicius Takeo Friedrich Kuwaki
vinicius.kuwaki@edu.udesc.br
github.com/takeofriedrich



UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA