

Persistência de Dados em Arquivos

Exercicio

Vinicius Takeo Friedrich Kuwaki

Universidade do Estado de Santa Catarina

Seções

Exemplo

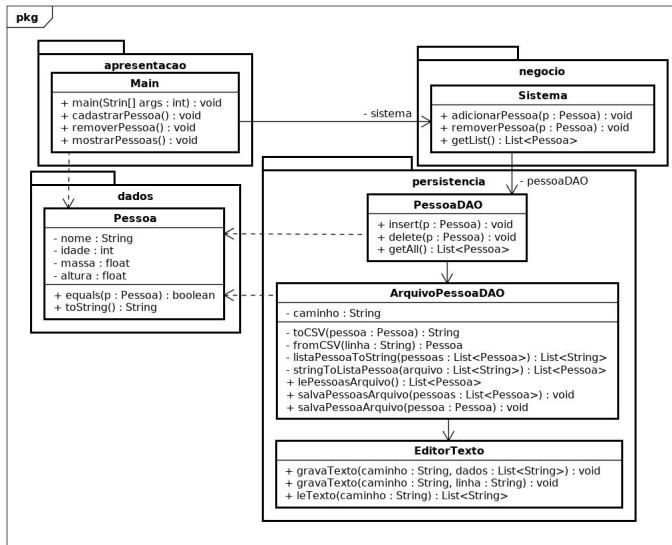
Resolução

Exercício

Exemplo

- Utilizando as classes Pessoa da Aula Prática 2: Orientação a Objetos em Java, implemente o diagrama a seguir:

Exemplo - Diagrama



Exemplo - Descrição Geral

- O sistema deve permitir o cadastro e a remoção de novas pessoas em um arquivo texto;
- Provendo uma interface via console com o usuário;
- Além do mais essa interface deve listar todas as pessoas que estão no arquivo, exibindo-as com suas respectivas informações (nome, idade, massa, altura e imc);

Exemplo - Descrição Pacote de Dados

- O pacote de dados possui apenas a classe Pessoa:
- A classe Pessoa possui os mesmos atributos e métodos da aula 2. Entretanto, para este exercício é necessário a implementação do método `bool equals(Object o)`;
- Esse método verifica a igualdade entre dois objetos quaisquer. O método retorna true caso sejam iguais e false caso não. Utilize Casting para converter o objeto do tipo Object para o tipo Pessoa.

Exemplo - Descrição Pacote de Persistência

- Esse pacote realiza a persistência dos dados em arquivos externos. Utilize arquivos .csv;
- A classe EditorTexto, possui três métodos:
 - **gravaTexto(String caminho, String linha)**: esse método deve adicionar uma nova linha ao final do arquivo sem sobrescreve-lo.
 - **gravaTexto(String caminho, List<String> dados)**: já esse método recebe uma List de Strings que irá salvar no arquivo cada String separada por uma quebra de linha, sobrescrevendo o arquivo antigo;
 - **leTexto(String caminho)**: esse método deve realizar a leitura de todas as linhas e retornar uma List de String, onde cada posição do List é uma linha do arquivo;
- A classe ArquivoPessoaDAO irá lidar com a conversão de objetos para string, de string para objeto e a leitura e gravação de objetos no arquivo texto;
- Já a classe PessoaDAO irá utilizar da classe ArquivoPessoaDAO para inserir, remover e obter os objetos do arquivo, lidando no nível de abstração de objetos;

Exemplo - Descrição Pacote de Persistência

- A classe `ArquivoPessoaDAO` possui os seguintes métodos:
 - **toCSV():** esse método recebe um objeto do tipo `Pessoa` e transforma ele em uma `String` contendo todos os atributos concatenados e separados por uma vírgula;
 - **fromCSV():** esse método faz o oposto do método `toCSV()`. Ele recebe uma `String` no formato gerado pelo método `toCSV()` e retorna um objeto do tipo `Pessoa` com as informações presentes na `String`;
 - **listaPessoaToString():** esse método recebe uma `List` de `Pessoa` e retorna uma `List` de `String`. Cada posição da `List<Pessoa>` deve ser transformada em uma `String` no formato CSV e armazenada dentro do objeto de retorno desse método.
 - **stringToListaPessoa():** esse método, novamente, é o contrário do descrito anteriormente. Dado uma `List<String>` contendo as informações dentro de `Strings`, esse método retorna uma `List` de `Pessoa` contendo as informações que estavam no formato `String`.

Exemplo - Descrição Pacote de Persistência

- A classe `ArquivoPessoaDAO` ainda possui os seguintes métodos:
 - **`lePessoasArquivo()`**: esse método utiliza os métodos descritos anteriormente e realiza o processo de obtenção das informações do arquivo texto e retornar uma `List` de `Pessoa` contendo as informações salvas no arquivo;
 - **`salvaPessoasArquivo()`**: já esse método recebe uma `List` de `Pessoa` e as salva em um arquivo texto. Novamente utilizando dos métodos descritos anteriormente;
 - **`salvaPessoaArquivo()`**: esse método recebe uma única `Pessoa` e a salva em um arquivo texto. Novamente utilizando dos métodos descritos anteriormente;

Exemplo - Descrição Pacote de Persistência

- A classe PessoaDAO utilizará dos métodos da classe ArquivoPessoaDAO e terá os métodos:
 - **insert()**: esse método recebe um objeto do tipo Pessoa e o salva no arquivo;
 - **delete()**: esse método lê os objetos que estavam salvos no arquivo, carrega-os para a memória e remove o objeto que foi passado como parâmetro (caso ele exista), salvando os objetos em memória no arquivo ao final do processo;
 - **getAll()**: esse método realiza a leitura dos objetos do arquivo e retorna uma List de Pessoa;

Exemplo - Descrição Pacote de Negócio

- Esse pacote contém a classe Sistema que realiza as funcionalidades de cadastrar, remover e listar pessoas;
- Ele possui os seguintes métodos:
 - **adicionarPessoa()**: esse método adiciona uma pessoa ao sistema e a armazena no arquivo csv;
 - **removerPessoa()**: esse método remove uma pessoa do sistema e a retira do arquivo csv;
 - **getPessoas()**: esse método retorna todos os objetos gerenciados pelo sistema que estão armazenados no arquivo csv;

Exemplo - Descrição Pacote de Apresentação

- Esse pacote contém a classe Main que realiza a interface via console com o usuário;
- Ela contém métodos que permitem acesso às funcionalidades que o pacote de negócio possui.

Seções

Exemplo

Resolução

Exercício

- Como a classe Pessoa já foi apresentada na resolução da Aula Prática 2, ela não será destrinchada aqui, apenas o método **equals()** será apresentado agora;
- Esse método é próprio da classe Object, ou seja, toda classe em Java pode sobreescrever esse método;
- Em nossa implementação não verificaremos todos os atributos para garantir que uma pessoa seja igual a outra;
- Basta que o nome dela seja igual, elas já serão consideradas iguais;
- Esse método então retorna *true* caso as duas pessoas tenham nomes iguais e *false* caso contrário;

Classe Pessoa

```
@Override
public boolean equals(Object o) {
    Pessoa p = (Pessoa) (o);

    if (p.getNome() == this.getNome()) {
        return true;
    }

    return false;
}
```

- Agora basta colocar a classe Pessoa dentro do pacote dados e adicionar a declaração de pacote para utilizarmos nos *imports* futuramente;

```
package dados;
```

Classe EditorTexto

- A classe EditorTexto pertence ao pacote persistência, então precisamos declará-lo;
- Iremos também importar as classes BufferedReader, FileReader e FileWriter para manipularmos o arquivo;
- Também importaremos as classes List e LinkedList;

```
package persistencia;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.util.LinkedList;  
import java.util.List;
```


Classe EditorTexto

- Iremos agora começar a implementação do método **gravarTexto()**;
- Nas aulas seguintes iremos estudar sobre o Tratamento de Exceções em Java;
- Basicamente, ela é uma classe que sinaliza que aconteceu algo inesperado na execução do código;
- No exemplo em questão, a exceção é lançada caso o arquivo esteja aberto ou não seja encontrado.
- Futuramente, aprenderemos como enviar o erro para a camada de apresentação e deixar o usuário decidir o que acontece.
- Esse processo ocorre dentro de um bloco estruturado try/catch;
- Ambos são palavras reservadas em Java;
- O comando **try** executa o trecho de código que pode lançar uma exceção;
- Caso aconteça a exceção o **catch** captura e trata a referida exceção;

Classe EditorTexto

```
public void gravaTexto(String caminho, List<String> dados) {  
    FileWriter arq;  
    try {  
        arq = new FileWriter(caminho);  
        for (int i = 0; i < dados.size(); i++) {  
            arq.write(dados.get(i) + "\n");  
        }  
        arq.close();  
    } catch (Exception e) {  
        System.err.println("Erro ao manipular o arquivo");  
        System.exit(0);  
    }  
}
```

Classe EditorTexto

- No momento não é necessário entender o mecanismo do bloco try/catch;
- O foco é a gravação das linhas no arquivo;
- Primeiro é instanciado um objeto do tipo **FileWriter**;
- Dentro do bloco **try** ele é instanciado passando o caminho onde está o arquivo (que veio como parâmetro para esse método);
- Caso o arquivo não seja encontrado, uma exceção é lançada e o bloco **catch** a captura;
- Caso o arquivo seja encontrado e aberto, um laço de repetição percorre a lista “dados” passada como parâmetro na chamada desse método;
- E para cada posição da lista, utiliza do método write da classe **FileWriter** para escrever no arquivo, adicionando uma quebra de linha ao final;
- Caso algum erro aconteça durante esse laço de repetição, uma exceção é lançada para o bloco **catch** a capturar;

- O bloco catch então captura um objeto do tipo Exception, dando o nome para ele de “e”;
- O erro é exibido no console através do método **err()** da classe System;
- Por fim, o console encerra a execução através do método **exit()** passando 0 como parâmetro;

- E agora vamos criar o método que adiciona uma linha ao arquivo;
- Para isso, o método **gravaTexto()** será sobrescrito, recebendo como parâmetro uma única String;
- Como diferença da implementação desse método para o anterior, ele não terá um laço de repetição para adicionar várias strings visto que ele recebe apenas uma única string como parâmetro;
- Ainda, o construtor da classe FileWriter deve indicar que iremos dar um *append* ao final do arquivo enviando true como parâmetro:

Classe EditorTexto

```
public void gravaTexto(String caminho, String linha) {  
    FileWriter arq;  
    try {  
        arq = new FileWriter(caminho, true);  
        arq.write(linha);  
        arq.close();  
    } catch (Exception e) {  
        System.err.println("Erro ao manipular o arquivo");  
        System.exit(0);  
    }  
}
```

Classe EditorTexto

- A lógica do método **leTexto()** é inversa a do **gravarTexto()**;
- Primeiro instanciamos a lista que será retornada;
- Além de declarar uma variável da classe **FileReader** que fará a leitura do arquivo e uma da classe **BufferedReader** que servirá de buffer para fazer a leitura;

```
public List<String> leTexto(String caminho) {  
    List<String> dados = new LinkedList<String>();  
  
    FileReader arq;  
    BufferedReader lerArq;
```

Classe EditorTexto

- Dentro do bloco try iremos instanciar os objetos das classes **FileReader** e **BufferedReader**;
- A classe **FileReader** vai receber no construtor o caminho que foi passado como parâmetro para esse método;
- E a **BufferedReader** vai receber a instância recém criada da **FileReader**;
- Também teremos uma String auxiliar “s”, que faz a leitura da primeira linha do arquivo;
- Após isso ela entra em um laço de repetição até que não hajam mais linhas a serem lidas;
- Para cada linha lida ela será inserida dentro da lista a ser retornada;
- Após terminar de usar o arquivo, precisamos fechá-lo utilizando o método **close()**;
- Observe o código do bloco try/catch

Classe EditorTexto

```
try {  
    arq = new FileReader(caminho);  
    lerArq = new BufferedReader(arq);  
    String s = lerArq.readLine();  
  
    while (s != null) {  
        dados.add(s);  
        s = lerArq.readLine();  
    }  
  
    arq.close();  
} catch (Exception e) {  
    System.err.println("Erro ao manipular o arquivo");  
    System.exit(0);  
}
```

Classe EditorTexto

- Dentro do bloco **catch** iremos capturar a exceção dando o nome a ela de “e” e repetindo o mesmo processo do método anterior;
- Caso uma exceção tenha sido capturada, o bloco **catch** vai exibir no console uma mensagem de erro e finaliza a execução;
- Passando pelo bloco try/catch a lista é retornada;

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

Classe ArquivoPessoaDAO

- A classe ArquivoPessoaDAO também pertence ao pacote de persistência;
- Ela também utiliza as classes List e LinkedList;
- Ela tem uma dependência com a classe Pessoa;

```
package persistencia;  
  
import java.util.LinkedList;  
import java.util.List;  
  
import dados.Pessoa;
```

Classe ArquivoPessoaDAO

- Teremos um atributo do tipo **final** para indicar o caminho do arquivo. Definimos ele como final para não ser modificado após ter seu valor definido;
- Crie uma pasta dentro do projeto (Eclipse, Netbeans, VSCode, etc...) para manter um arquivo csv e atribua o caminho dessa pasta ao valor desse atributo, junto com o nome do arquivo;
- Também teremos uma instância estática da classe EditorTexto;

```
public class ArquivoPessoaDAO {  
  
    private final String caminho = "files/pessoas.csv";  
    private static EditorTexto arquivo = new EditorTexto();  
}
```

Classe ArquivoPessoaDAO

- O primeiro método que criaremos é o método **toCSV()**;
- Esse método privado, retorna uma String contendo todos os atributos do objeto Pessoa passado como parâmetro concatenados e separados por vírgula (",");
- A implementação desse método é semelhante a do método **toString()**;
- O método utiliza de todos os getters do objeto Pessoa, e os separa por uma vírgula:

```
private String toCSV(Pessoa pessoa) {  
    String p = "";  
  
    p += pessoa.getNome() + ",";  
    p += pessoa.getIdade() + ",";  
    p += pessoa.getAltura() + ",";  
    p += pessoa.getMassa();  
  
    return p;  
}
```

Classe ArquivoPessoaDAO

- Já o método **fromCSV()** realiza o processo inverso;
- A partir de uma String igual a gerada pelo método anterior, ele retorna um objeto do tipo Pessoa;
- Utilizaremos do método **split()** da classe String para “quebrar” a String a cada incidência de uma vírgula;
- Esse método retorna um array de Strings;
- É necessário saber exatamente a ordem em que os atributos estão dispostos na String, ou seja, deve ser a mesma do método anterior.
- Como utilizaremos os métodos setters da classe Pessoa, não podemos esquecer de fazer as conversões de tipo necessárias:

Classe ArquivoPessoaDAO

```
private Pessoa fromCSV(String linhaCSV) {  
    String[] atributos = linhaCSV.split(",");  
  
    Pessoa pessoa = new Pessoa();  
    pessoa.setNome(atributos[0]);  
    pessoa.setIdade(Integer.parseInt(atributos[1]));  
    pessoa.setAltura(Float.parseFloat(atributos[2]));  
    pessoa.setMassa(Float.parseFloat(atributos[3]));  
  
    return pessoa;  
}
```

- Agora criaremos o método **listaPessoaToString()** que transforma uma List de Pessoa que é passado como parâmetro em uma List de String;
- Primeiro instanciamos uma nova List de String e depois percorremos a List de Pessoa;
- Para cada Pessoa da List chamaremos o método criado anteriormente **toCSV()** que retorna uma String;
- Após isso adicionamos essa String a lista;
- Após percorre a List de Pessoa, retornamos a lista de Strings;

Classe ArquivoPessoaDAO

```
private List<String> listaPessoaToString(List<Pessoa> pessoas) {  
    List<String> arquivo = new LinkedList<String>();  
    for (Pessoa pessoa : pessoas) {  
        arquivo.add(toCSV(pessoa));  
    }  
    return arquivo;  
}
```

- Terminado esse método iremos para o método **stringToListaPessoa()** que, a partir da lista de strings passada como parâmetro, retorna uma List de Pessoa;
- Primeiro alocamos uma nova List de Pessoa;
- Depois iremos percorrer a lista de Strings recebida como parâmetro e utilizaremos o método **fromCSV()** enviando a String em questão como parâmetro. Esse método vai retornar uma Pessoa que adicionaremos a lista de pessoas a ser retornada;
- Após realizar o processo, o método **stringToListaPessoa()** irá retornar a lista;

Classe ArquivoPessoaDAO

```
private List<Pessoa> stringToListaPessoa(List<String> arquivo) {  
    List<Pessoa> pessoas = new LinkedList<Pessoa>();  
    for (String linha : arquivo) {  
        pessoas.add(fromCSV(linha));  
    }  
    return pessoas;  
}
```

Classe ArquivoPessoaDAO

- Após criados os métodos que lidam com a transformação da informação texto-objeto, iremos criar os métodos **lePessoasArquivo()** e o **salvaPessoasArquivo()**;
- O **lePessoasArquivo()** chama o método **stringToListaPessoa()**, que recebe uma lista de Strings como parâmetro, por meio da chama do método **leTexto()** da classe EditorTexto, sendo que esse último recupera as linhas do arquivo cujo o nome é passado como parâmetro pela variável final “caminho”;

```
public List<Pessoa> lePessoasArquivo() {  
    return stringToListaPessoa(arquivo.leTexto(caminho));  
}
```

- Já o método **salvaPessoasArquivo()** faz o processo inverso ao **lePessoasArquivo()**;
- Ele chama o método **gravaTexto()** da classe EditorTexto, que pede como parâmetro o caminho e a lista de Strings. Iremos passar a nossa variável final caminho e uma chamada do método **listaPessoaToString()** que nos retorna exatamente uma lista de Strings;

```
public void salvaPessoasArquivo(List<Pessoa> pessoas) {  
    arquivo.gravaTexto(caminho, listaPessoaToString(pessoas));  
}
```

- Ainda, o método **salvaPessoaArquivo()** tem a mesma função que o método anterior, entretanto, ao invés de salvar uma lista inteira, ele salva apenas uma Pessoa;

```
public void salvaPessoaArquivo(Pessoa pessoa) {  
    arquivo.gravaTexto(caminho, toCSV(pessoa));  
}
```

Classe PessoaDAO

- Para finalizar o pacote de persistência, iremos criar a classe PessoaDAO;
- Ela contém as funções básicas de persistências para qualquer objeto de dados em qualquer meio de armazenamento;
- Nas próximas aulas iremos perceber que o seu padrão será utilizado também para a persistência em banco de dados;
- Ela possui uma dependência com a classe Pessoa e utiliza da interface List;
- Ainda, possui uma instância da classe ArquivoPessoaDAO;

```
package persistencia;  
  
import java.util.List;  
  
import dados.Pessoa;  
  
public class PessoaDAO {  
  
    private ArquivoPessoaDAO arquivoPessoaDAO = new ArquivoPessoaDAO();  
  
}
```

Classe PessoaDAO

- Ela contém os métodos que serão acessados pela classe Sistema para manipular os dados em arquivo;
- O primeiro deles é **insert()**, que recebe uma pessoa como parâmetro e adiciona ela ao final do arquivo;
- Utilizaremos o método **salvaPessoaArquivo()** da classe ArquivoPessoaDAO que recebe uma Pessoa como parâmetro;

```
public void insert(Pessoa pessoa) {  
    arquivoPessoaDAO.salvaPessoaArquivo(pessoa);  
}
```


Classe PessoaDAO

- O próximo é o método **delete()**;
- Assim como o **insert()**, esse método recebe uma pessoa;
- Entretanto, esse método lê todas as Pessoas do arquivo, transformando-o para uma List de Pessoa, para então remover dele a pessoa indicada pelo usuário;
- Esse método depende da implementação do método **equals()** da classe Pessoa, visto que é ele que a interface List utiliza para remover um objeto;

```
public void delete(Pessoa pessoa) {  
    List<Pessoa> pessoas = arquivoPessoaDAO.lePessoasArquivo();  
    pessoas.remove(pessoa);  
    arquivoPessoaDAO.salvaPessoasArquivo(pessoas);  
}
```

- Finalmente, o último método dessa classe, o **getAll()**;
- Esse método retorna uma List de Pessoa utilizando o método **lePessoasArquivo()** da classe ArquivoPessoaDAO;

```
public List<Pessoa> getAll() {  
    return arquivoPessoaDAO.lePessoasArquivo();  
}
```

Classe Sistema

- A classe Sistema concentra as funcionalidades de todo o sistema que está sendo desenvolvido;
- Para este exercício ela parece desnecessária, entretanto, para sistemas reais, onde a quantidade de tipos diferentes de objetos sendo armazenados é melhor, ela se torna fundamental;
- Criaremos a classe Sistema para gerenciar os dados do programa e enviá-los para o arquivo texto;
- Iremos então importar apenas as classes que serão utilizadas;
- Como teremos uma associação com a classe PessoaDAO e uma dependência com a classe Pessoa, iremos importar essas classes;
- Teremos um atributo do tipo PessoaDAO;
- Replicaremos os métodos de adicionar, remover e listar os objetos do tipo Pessoa;

Classe Sistema

```
package negocio;  
  
import java.util.List;  
  
import dados.Pessoa;  
import persistencia.PessoaDAO;  
  
public class Sistema {  
  
    private PessoaDAO pessoaDAO = new PessoaDAO();  
  
    public void adicionarPessoa(Pessoa p) {  
        pessoaDAO.insert(p);  
    }  
  
    public void removerPessoa(Pessoa p) {  
        pessoaDAO.delete(p);  
    }  
  
    public List<Pessoa> getLista() {  
        return pessoaDAO.getAll();  
    }  
  
}
```

Classe Main

- Por fim, criaremos a classe Main para servir como interface com o usuário e assim acessar as funcionalidades da classe Sistema;
- Utilizaremos a classe Scanner e importaremos a classe Sistema pela associação exemplificada no diagrama e a classe Pessoa pela dependência;

```
package apresentacao;  
  
import java.util.Scanner;  
  
import dados.Pessoa;  
import negocio.Sistema;
```

- Teremos então uma instância estática da classe Scanner e uma do Sistema;

```
public class Main {  
  
    private static Sistema sistema = new Sistema();  
    private static Scanner s = new Scanner(System.in);  
  
}
```

Classe Main

- Criaremos então um método **novaPessoa()**, que lê as informações da pessoa e retorna uma instância com os valores setados;

```
private static Pessoa novaPessoa() {  
    Pessoa p = new Pessoa();  
  
    System.out.println("Digite o nome:");  
    p.setNome(s.next());  
  
    System.out.println("Digite a idade:");  
    p.setIdade(s.nextInt());  
  
    System.out.println("Digite a altura:");  
    p.setAltura(s.nextFloat());  
  
    System.out.println("Digite a massa:");  
    p.setMassa(s.nextFloat());  
  
    return p;  
}
```

- Também criaremos um método para exibir todas as pessoas que a classe Sistema retorna, ou seja, as pessoas que estão salvas no arquivo texto;
- Esse método percorre a lista retornada e imprime os objetos um a um associando-os a um contador "i";

```
private static void mostrarPessoas() {  
    for (int i = 0; i < sistema.getList().size(); i++) {  
        System.out.println("Pessoa " + i);  
        System.out.println(sistema.getList().get(i));  
        System.out.println();  
    }  
}
```

- Utilizando desse método criaremos um para o usuário escolher uma pessoa presente nessa lista;
- Primeiro é chamado o método **mostrarPessoas()**;
- Então é requisitado ao usuário escolher o número associado a ela;
- Se esse valor digitado é um valor dentro da lista, a pessoa que possui esse número é retornada;
- Caso contrário o método retorna **null**;

Classe Main

```
private static Pessoa escolherPessoa() {  
    mostrarPessoas();  
    System.out.println("Escolha uma pessoa:");  
    int escolha = s.nextInt();  
    if (escolha < sistema.getLista().size()) {  
        return sistema.getLista().get(escolha);  
    } else {  
        return null;  
    }  
}
```

- Por fim, teremos um método para exibir um menu e outro para de fato realizar as funcionalidades do menu;

```
public static void imprimeMenu() {  
    System.out.println("Escolha uma opção:");  
    System.out.println("0 - Sair");  
    System.out.println("1 - Cadastrar pessoa");  
    System.out.println("2 - Remover pessoa");  
    System.out.println("3 - Mostrar pessoas");  
}
```

Classe Main

```
public static void menu() {  
    int opcao = -1;  
    while (opcao != 0) {  
  
        imprimeMenu();  
        opcao = s.nextInt();  
  
        switch (opcao) {  
            case 0:  
                break;  
            case 1:  
                sistema.adicionarPessoa(novaPessoa());  
                break;  
            case 2:  
                sistema.removerPessoa(escolherPessoa());  
                break;  
            case 3:  
                mostrarPessoas();  
                break;  
            default:  
                System.out.println("N mero inv lido!");  
                break;  
        }  
    }  
}
```

- O método main apenas chama o método **menu()**;

```
public static void main(String [] args) {  
    menu();  
}
```

- Todo o código-fonte está disponível nesse link;

Seções

Exemplo

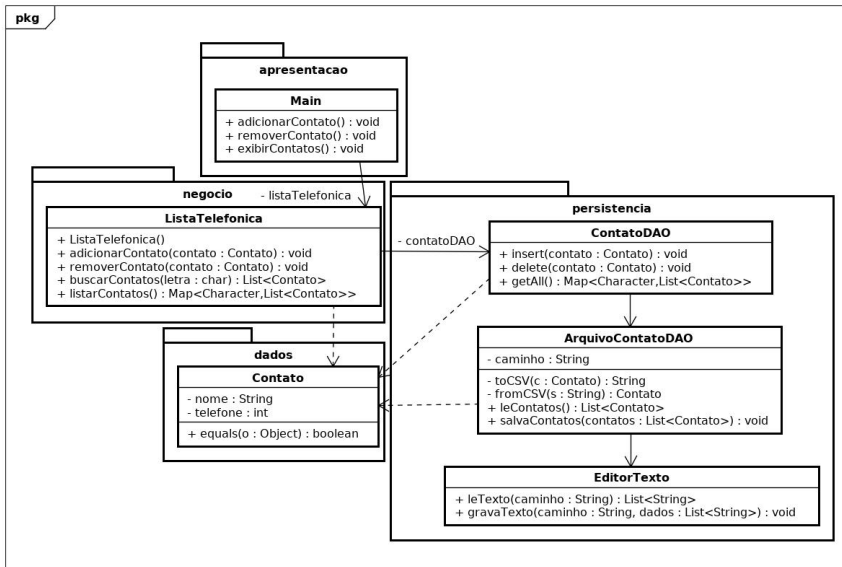
Resolução

Exercício

Exercício

A partir da resolução do exercício da Aula Prática 5: Utilização de Coleções em Java, implemente o Diagrama de Classes UML:

Exercício



A classe `ArquivoContatoDAO` realiza a persistência de dados em um arquivo externo. Observe a descrição dos métodos:


- **toCSV()**: esse método transforma todos um contato recebido como parâmetro em uma string pronta a ser salva no arquivo .csv e a retorna;
- **fromCSV()**: esse método transforma uma string passada como parâmetro em um objeto do tipo contato;
- **leContatos()**: esse método retorna uma `List<Contato>` contendo os contatos presentes no arquivo;
- **salvaContatos()**: esse método armazena no arquivo csv todos os contatos que ele recebe como parâmetro

Já a classe ContatoDAO, utilizando dos métodos da classe ArquivoContatoDAO e seguindo o mesmo caminho do Exemplo anterior, possui os seguintes métodos:

- **insert(Contato contato):** esse método adiciona o contato que foi passado como parâmetro ao final do arquivo texto;
- **delete(Contato contato):** esse método remove o contato que foi passado como parâmetro do arquivo texto;
- **getAll():** esse método retorna todos os contatos que estão no arquivo csv;

Exercício

O sistema deve carregar os contatos ao ser iniciado, isto é, ao iniciar a classe ListaTelefonica, todos os contatos devem ser puxados do arquivo texto. A cada inserção ou remoção de um contato o arquivo deve ser atualizado;

 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