

LISTA DE EXERCÍCIOS XV

Leia atentamente as instruções gerais:

- No Eclipse crie um novo projeto chamado **br.edu.mpo0.listaXV.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** sistema, contendo todas as respostas da lista.
- A lista envolve questões práticas e conceituais.

Fique atento!

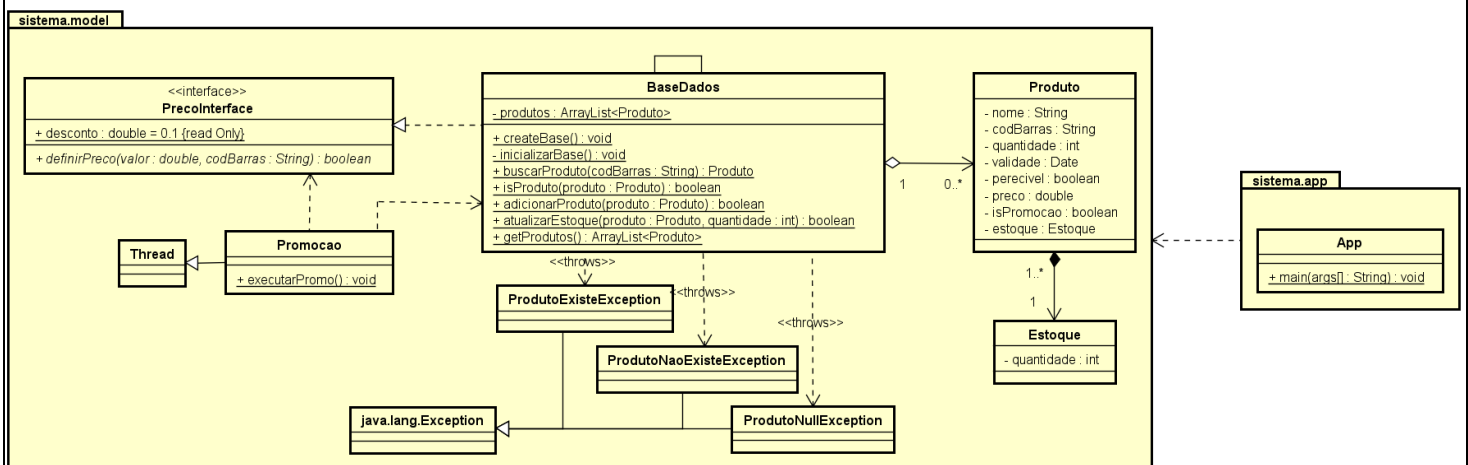
Prezado aluno, nesta lista de exercícios discutiremos os conceitos de "Threads" e "Entrada e Saída".

Desafio

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar novos problemas a partir de conhecimentos de POO.



- Uma Bodega que vende produtos em uma cidade pequena procurou a empresa "MPOO Development Systems" para que fosse desenvolvido um sistema para gerenciar seus produtos e realizar promoções sobre esses produtos. A empresa de desenvolvimento solicitou ao ser programador "O Furão" (codinome para *mustela putorius furo*) que implementasse o model do sistema a partir do diagrama de classes.



O Scrum Master da empresa explicou a "O Furão" que:

- BaseDados é classe que contém uma estrutura de dados para Produtos e métodos de manipulação dessa estrutura de dados;
- codBarras é chave primária de produto;
- Produto contém Estoque;
- PrecoInterface possui comportamento de definição de preço que deve ser implementado por classe que a realize. Também possui o valor de desconto que pode ser aplicado a um produto;
- Deve-se fazer uso de tratamento de exceção para os métodos de BaseDados. Deve-se tratar por throws/throw a exceção ProdutoNullException quando um produto for null, ProdutoExisteException se um produto já pertencer a base (isProduto(produto)==true) ou ProdutoNaoExisteException caso contrário. Deverá ter o retorno definido para cada método quando uma exceção não for levantada.
- O sistema possui a Thread Promoção que a cada 24 horas (86400000ms) executa uma promoção aos produtos que tem uma promoção ativa. Uma vez aplicada uma promoção, o produto deixa ter uma promoção ativa. Uma vez aplicada uma promoção, o preço de um produto só retornará ao valor antes da promoção se modificado por uma GUI futura.

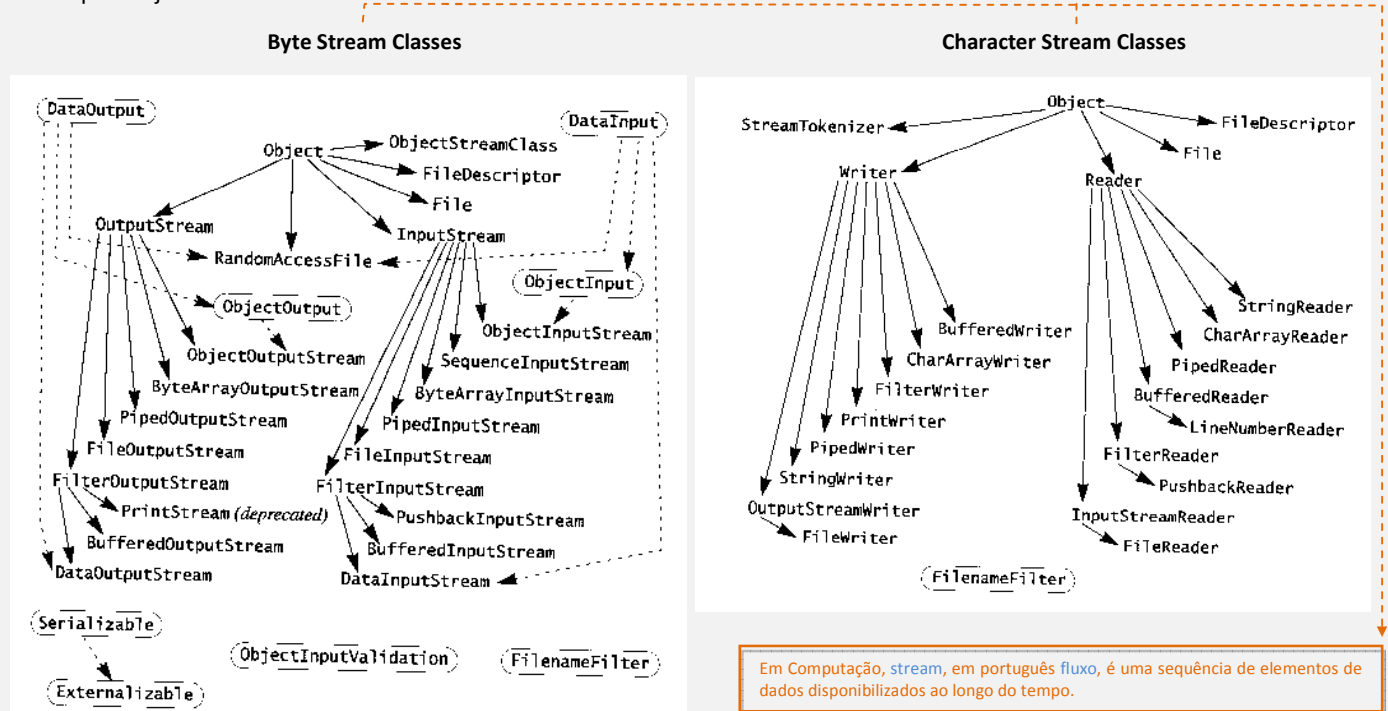
Sendo assim, ilustre a execução da Thread Promoção aplicando descontos a produtos que possui promoção ativa (isPromocao = true).

Saiba Mais!

Em desenvolvimento de software, de uma forma ou de outra, manipulam-se [arquivos](#) para, por exemplo, gerar relatórios, ler configurações ou para persistir dados formando uma base de dados de uma aplicação, como, por exemplo: .txt, .xml, .csv; etc.

A manipulação de arquivos em Java é uma maneira simples de armazenar dados em ROM quando não há necessidade de usar um Sistema Gerenciador de Banco de Dados (SGBD).

Há diversas classes pertencentes ao pacote [java.io](#) para manipular um arquivo, nas quais se deve levar em consideração se a operação é do tipo leitura, escrita ou ambas, bem como o tipo de dado a ser armazenado (caracteres ou binário). Vejamos a hierarquia Object:



Nessa arquitetura podemos observar quatro categorias de Classes:

- Classes de stream que representam origens ou destinos de bytes ou caracteres.
 - `InputStream` e `OutputStream` são classes abstratas que declaram métodos para ler bytes de uma fonte específica (para gravar bytes em um destino específico).
 - `FileReader` e `FileWriter` para fluxos de caracteres que operam com arquivos, etc.
- Classes de stream que representam várias maneiras de processar os fluxos, como:
 - obtendo bytes ou caracteres em um buffer: `BufferedInputStream`, `BufferedOutputStream`, `BufferedReader`, `BufferedWriter`
 - para estruturar em tipos de dados(floats, por exemplo): `DataInputStream` e `DataOutputStream`
 - para filtrar de alguma forma: `FilterReader` e `FilterWriter`. Um fluxo de filtro é construído em outro fluxo (o fluxo subjacente).
- Conversão entre stream de bytes e caracteres:
 - `InputStreamReader` converte de bytes em caracteres,
 - `OutputStreamWriter` converte de caracteres em bytes.
 - Há também a classe `PrintStream` obsoleta, que recebe bytes e imprime caracteres.
- Classes que encapsulam localizações de arquivos (diretórios, nomes) e propriedades (por exemplo, poder ser lido, gravável, etc): `File` e `RandomAccessFile`

Conheça as características das classes do pacote java.io: <https://docs.oracle.com/javase/8/docs/api/java/io/package-frame.html>

Investigue e pratique a implementação de I/O acessando o tutorial de JavaSE da Oracle: <https://docs.oracle.com/javase/tutorial/essencial/io/index.html>

Vejamos alguns exemplos de I/O:

Verificação de arquivo ou diretório por File

```
Arquivo.java
1 package io;
2
3 import java.io.File;
4 import java.util.Scanner;
5
6 public class Arquivo {
7     public void analisarPath(String path){
8         File nome = new File(path);
9         if (nome.exists()){
10             System.out.println( nome.getName() + " existe" + "\n" +
11                 (nome.isFile() ? "é um arquivo" : "não é um arquivo") + "\n" +
12                 (nome.isDirectory() ? "é um diretório" : "não é um diretório") + "\n" +
13                 (nome.isAbsolute() ? "é um caminho absoluto" : "não é um caminho absoluto") + "\n" +
14                 "Last modified: " + nome.lastModified() + "\n" +
15                 "Tamanho: " + nome.length() + "\n" +
16                 "Caminho: " + nome.getPath() + "\n" +
17                 "Caminho Absoluto: " + nome.getAbsolutePath() + "\n" +
18                 "Diretório pai: " + nome.getParent());
19         }
20         if (nome.isDirectory()){
21             String diretorio [] = nome.list();
22             System.out.println("\nConteúdo do Diretório:");
23             for (String diretorioNome : diretorio)
24                 System.out.println(diretorioNome);
25         }
26         else{
27             System.out.println(path + " não existe");
28         }
29     }
30
31     public static void main(String [] args){
32         Scanner in = new Scanner (System.in);
33         Arquivo aplicacao = new Arquivo();
34         System.out.println("Digite o nome de um arquivo ou diretório");
35         aplicacao.analisarPath(in.nextLine());
36         in.close();
37     }
38 }
```

Exemplos de analisarPath(String path):

```
Console [Arquivo [Java Application]]
<terminated> Arquivo [Java Application]
Digite o nome de um arquivo ou diretório
src/io
io existe
não é um arquivo
é um diretório
não é um caminho absoluto
Last modified: 1651288566508
Tamanho: 4096
Caminho: src\io
Caminho Absoluto: D:\Programacao\workspaceMPOOSI\br.edu.mpoo.listaX.SeuNomeSobrenome\src\io
Diretório pai: src

Conteúdo do Diretório:
Arquivo.java
EscreverArquivo.java
LerArquivoTexto1.java
LerArquivoTexto2.java
URLReader.java
```

```
Console [Arquivo [Java Application]]
<terminated> Arquivo [Java Application]
Digite o nome de um arquivo ou diretório
src/io/Arquivo.java
Arquivo.java existe
é um arquivo
não é um diretório
não é um caminho absoluto
Last modified: 1651367579572
Tamanho: 1272
Caminho: src\io\Arquivo.java
Caminho Absoluto: D:\Programacao\workspaceMPOOSI\br.edu.mpoo.listaX.SeuNomeSobrenome\src\io\Arquivo.java
Diretório pai: src\io
src/io/Arquivo.java não existe
```

Leitura de Arquivo por BufferedRead com FileReader

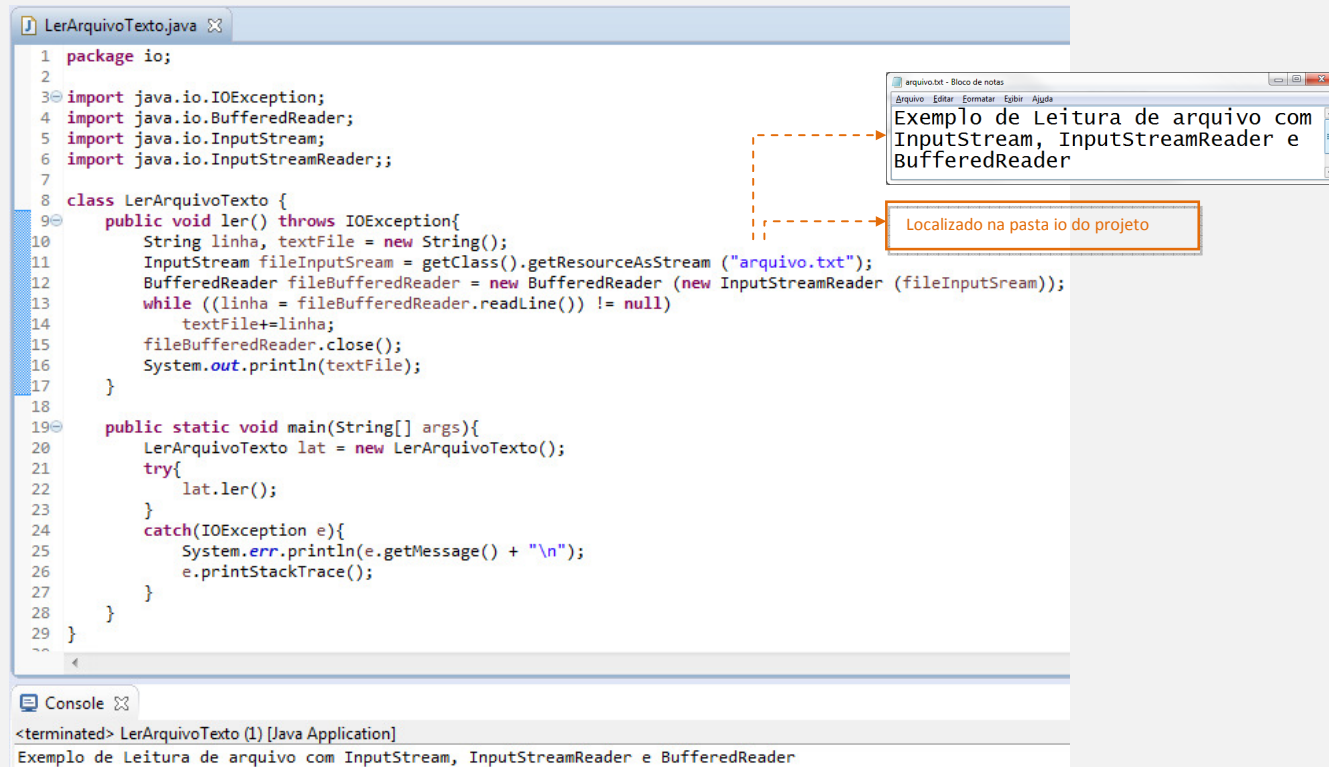
```
LerArquivoTexto.java
1 package io;
2
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5
6 public class LerArquivoTexto{
7     public static void main(String [ ] args){
8         String line, text = new String();
9         try {
10             BufferedReader in = new BufferedReader( new FileReader("arquivo.txt"));
11             while((line = in.readLine()) != null)
12                 text += line+ "\n";
13             in.close();
14         }
15         catch (Exception e) {
16             System.err.println (e.getMessage() + "\n");
17             e.printStackTrace();
18         }
19         System.out.println(text);
20     }
21 }
```

Exemplo de Leitura de arquivo com
BufferedReader e FileReader

Localizado na pasta do projeto

```
Console [LerArquivoTexto (1) [Java Application]]
<terminated> LerArquivoTexto (1) [Java Application]
Exemplo de Leitura de arquivo com BufferedReader e FileReader
```

Leitura de Arquivo por BufferedReader com InputStreamReader e InputStream



The screenshot shows an IDE with a file named `LerArquivoTexto.java`. The code is as follows:

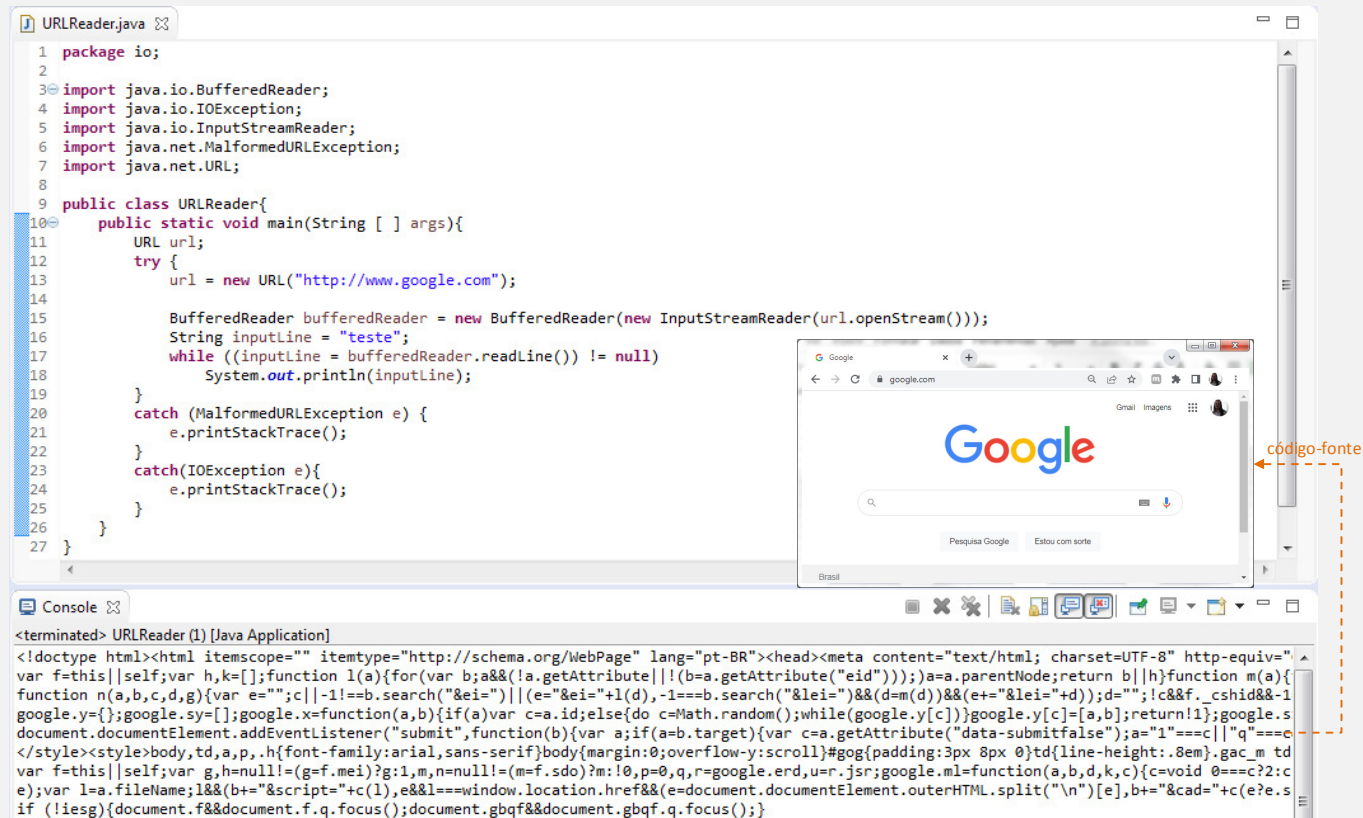
```
1 package io;
2
3 import java.io.IOException;
4 import java.io.BufferedReader;
5 import java.io.InputStream;
6 import java.io.InputStreamReader;
7
8 class LerArquivoTexto {
9     public void ler() throws IOException{
10         String linha, textFile = new String();
11         InputStream fileInputStream = getClass().getResourceAsStream("arquivo.txt");
12         BufferedReader fileBufferedReader = new BufferedReader(new InputStreamReader(fileInputStream));
13         while ((linha = fileBufferedReader.readLine()) != null)
14             textFile+=linha;
15         fileBufferedReader.close();
16         System.out.println(textFile);
17     }
18
19     public static void main(String[] args){
20         LerArquivoTexto lat = new LerArquivoTexto();
21         try{
22             lat.ler();
23         }
24         catch(IOException e){
25             System.err.println(e.getMessage() + "\n");
26             e.printStackTrace();
27         }
28     }
29 }
```

Annotations in the image:

- A dashed orange box highlights the package `io` and the `main` method, with a callout box stating "Exemplo de Leitura de arquivo com InputStream, InputStreamReader e BufferedReader".
- A dashed orange box highlights the `io` package, with a callout box stating "Localizado na pasta io do projeto".

The console window shows the output: `<terminated> LerArquivoTexto (1) [Java Application]` and the text "Exemplo de Leitura de arquivo com InputStream, InputStreamReader e BufferedReader".

Leitura de Arquivo (Site Google) por BufferedReader com InputStreamReader



The screenshot shows an IDE with a file named `URLReader.java`. The code is as follows:

```
1 package io;
2
3 import java.io.BufferedReader;
4 import java.io.IOException;
5 import java.io.InputStreamReader;
6 import java.net.MalformedURLException;
7 import java.net.URL;
8
9 public class URLReader{
10     public static void main(String [ ] args){
11         URL url;
12         try {
13             url = new URL("http://www.google.com");
14
15             BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(url.openStream()));
16             String inputLine = "teste";
17             while ((inputLine = bufferedReader.readLine()) != null)
18                 System.out.println(inputLine);
19         }
20         catch (MalformedURLException e) {
21             e.printStackTrace();
22         }
23         catch(IOException e){
24             e.printStackTrace();
25         }
26     }
27 }
```

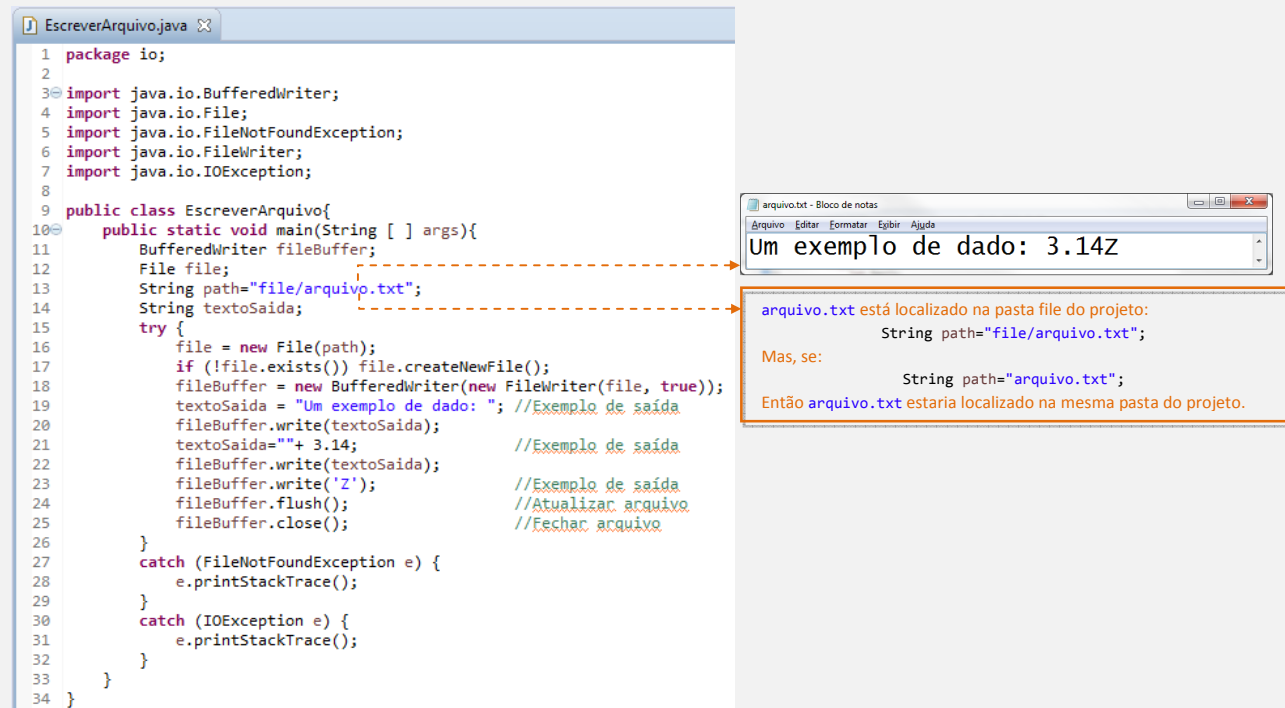
Annotations in the image:

- A dashed orange box highlights the `URL` class and the `main` method, with a callout box stating "código-fonte".

The console window shows the output: `<terminated> URLReader (1) [Java Application]` and the HTML source code of the Google homepage.

A browser window in the background shows the Google homepage.

Escrita de Arquivo por BufferedWriter com FileWriter e File



```
1 package io;
2
3 import java.io.BufferedWriter;
4 import java.io.File;
5 import java.io.FileNotFoundException;
6 import java.io.FileWriter;
7 import java.io.IOException;
8
9 public class EscreverArquivo{
10     public static void main(String [ ] args){
11         BufferedWriter fileBuffer;
12         File file;
13         String path="file/arquivo.txt";
14         String textoSaida;
15         try {
16             file = new File(path);
17             if (!file.exists()) file.createNewFile();
18             fileBuffer = new BufferedWriter(new FileWriter(file, true));
19             textoSaida = "Um exemplo de dado: "; //Exemplo de saída
20             fileBuffer.write(textoSaida);
21             textoSaida += " 3.14; //Exemplo de saída
22             fileBuffer.write(textoSaida);
23             fileBuffer.write('Z'); //Exemplo de saída
24             fileBuffer.flush(); //Atualizar arquivo
25             fileBuffer.close(); //fechar arquivo
26         }
27         catch (FileNotFoundException e) {
28             e.printStackTrace();
29         }
30         catch (IOException e) {
31             e.printStackTrace();
32         }
33     }
34 }
```

arquivo.txt - Bloco de notas

Um exemplo de dado: 3.14Z

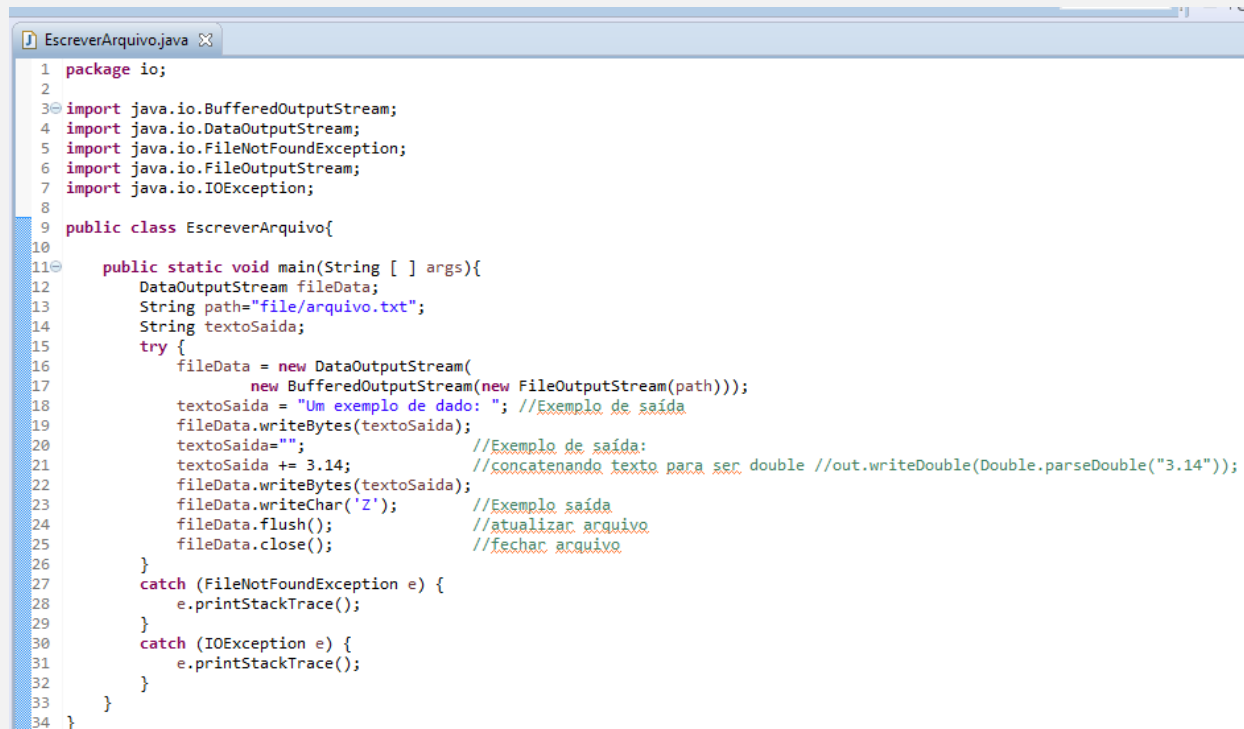
arquivo.txt está localizado na pasta file do projeto:
String path="file/arquivo.txt";

Mas, se:
String path="arquivo.txt";

Então arquivo.txt estaria localizado na mesma pasta do projeto.

Vejamos o mesmo exemplo de escrita de arquivo utilizando outras classe de java.io

Escrita de Arquivo por DataOutputStream com BufferedOutputStream e FileOutputStream



```
1 package io;
2
3 import java.io.BufferedOutputStream;
4 import java.io.DataOutputStream;
5 import java.io.FileNotFoundException;
6 import java.io.FileOutputStream;
7 import java.io.IOException;
8
9 public class EscreverArquivo{
10
11     public static void main(String [ ] args){
12         DataOutputStream fileData;
13         String path="file/arquivo.txt";
14         String textoSaida;
15         try {
16             fileData = new DataOutputStream(
17                 new BufferedOutputStream(new FileOutputStream(path)));
18             textoSaida = "Um exemplo de dado: "; //Exemplo de saída
19             fileData.writeBytes(textoSaida);
20             textoSaida = ""; //Exemplo de saída:
21             textoSaida += " 3.14; //concatenando texto para ser double //out.writeDouble(Double.parseDouble("3.14"));
22             fileData.writeBytes(textoSaida);
23             fileData.writeChar('Z'); //Exemplo saída
24             fileData.flush(); //atualizar arquivo
25             fileData.close(); //fechar arquivo
26         }
27         catch (FileNotFoundException e) {
28             e.printStackTrace();
29         }
30         catch (IOException e) {
31             e.printStackTrace();
32         }
33     }
34 }
```


Escrita de Arquivo por PrintWriter

```

1 package io;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.PrintWriter;
6
7 public class EscreverArquivo{
8
9     public static void main(String [ ] args){
10         PrintWriter filePrintWriter;
11         File file;
12         String path="file/arquivo.txt";
13         String textoSaida;
14         try {
15             file = new File(path);
16             filePrintWriter = new PrintWriter(file);
17             textoSaida = "Um exemplo de dado: "; //Exemplo de saída
18             filePrintWriter.print(textoSaida); //println para quebra de linha
19             textoSaida=""+3.14; //Exemplo de saída
20             filePrintWriter.print(textoSaida);
21             filePrintWriter.print('Z'); //Exemplo de saída
22             filePrintWriter.flush(); //Atualizar arquivo
23             filePrintWriter.close(); //Fechar arquivo
24         } catch (FileNotFoundException e) {
25             e.printStackTrace();
26         }
27     }
28 }

```

PrintWriter possui descarga automática, forçando a gravação de dados quando `print()` ou `println()` for chamado :

```

filePrintWriter = new PrintWriter(new FileWriter(path), true);
ou
filePrintWriter = new PrintWriter(new BufferedOutputStream(new
FileOutputStream(path)));
Com isso, não é necessário atualizar o arquivo a cada gravação:
filePrintWriter.flush();

```

Mas, atenção: `PrintWriter` não levanta nenhuma exceção de entrada / saída, ou seja, `IOException`! Para isso utiliza-se o método `checkError()` para encontrar qualquer erro.

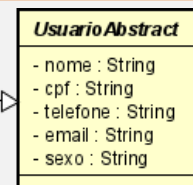
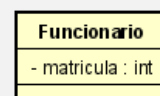
É características das classes de **java.io**:

DataOutputStream - permite a inserção de novos dados mesmo após o arquivo ser fechado.

BufferedWriter e **PrintWriter** – se reaberto um arquivo fechado, uma nova escrita sobrescreverá o conteúdo, apagando dados anteriores. Para isso é preciso recuperar os dados antes de escrevê-lo



Mas e se desejássemos em um sistema armazenar os dados de funcionários em um arquivo de texto?



Como seria a organização interna de `BaseDados.txt`? `/n` ou `;` ou alguma outra separação?

```

José Silva
111.111.111-11
(87)99999-9999
zesilva@gmail.com
masculino
1010

```

```

José Silva;111.111.111-11;
(87)99999-9999;zesilva@gmail.com;
masculino;1010

```

E se a base de dados (`BaseDados.txt`) possuir diversos indivíduos? Como identificar os atributos dos indivíduos?

```

José Silva
111.111.111-11
(87)99999-9999
zesilva@gmail.com
masculino
1010
João Santos
222.222.222-22
(81)98888-8888
js@gmail.com
masculino
1020

```

```

José Silva;111.111.111-11;(87)99999-
9999;zesilva@gmail.com;masculino;1010
João Santos;222.222.222-22;(81)98888
-8888;js@gmail.com;masculino;1020

```

O Java não impõe nenhuma estrutura a um arquivo — noções, como registros, não fazem parte da linguagem Java. Portanto, você deve estruturar os arquivos para satisfazer os requisitos dos seus aplicativos. No exemplo a seguir, veremos como impor uma estrutura de registro chaveado a um arquivo.



```
// UsuarioAbstract.java
public abstract class UsuarioAbstract {
    private String nome;
    private String cpf;
    private String telefone;
    private String email;
    private String sexo;

    public UsuarioAbstract(String nome, String cpf, String telefone, String email, String sexo) {
        this.nome = nome;
        this.cpf = cpf;
        this.telefone = telefone;
        this.email = email;
        this.sexo = sexo;
    }

    public String getNome() {return nome;}
    public void setNome(String nome) {this.nome = nome;}

    public String getCpf() {return cpf;}
    public void setCpf(String cpf) {this.cpf = cpf;}

    public String getTelefone() {return telefone;}
    public void setTelefone(String telefone) {this.telefone = telefone;}

    public String getEmail() {return email;}
    public void setEmail(String email) {this.email = email;}

    public String getSexo() {return sexo;}
    public void setSexo(String sexo) {this.sexo = sexo;}

    @Override
    public String toString() {
        return "UsuarioAbstract [nome=" + nome + ", cpf=" + cpf + ", telefone="
            + telefone + ", email=" + email + ", sexo=" + sexo + "];"
    }
}
```

```
// Funcionario.java
public class Funcionario extends UsuarioAbstract{
    private int matricula;

    public Funcionario(String nome, String cpf, String telefone, String email,
        String sexo, int matricula) {
        super(nome, cpf, telefone, email, sexo);
        this.matricula = matricula;
    }

    public int getMatricula() {return matricula;}

    public void setMatricula(int matricula) {this.matricula = matricula;}

    @Override
    public String toString() {
        return "Funcionario [matricula=" + matricula + ", toString()="
            + super.toString() + "];"
    };
}
```



```
// CreateBaseTextFile.java
import java.io.FileNotFoundException;
import java.util.Formatter;
import java.util.FormatterClosedException;

public class CreateBaseTextFile{
    private Formatter output;

    public void openFile(String nameFile){
        try{
            output = new Formatter( nameFile ); // abre o arquivo
        }
        catch ( SecurityException securityException){
            System.err.println("Você não tem acesso de gravação a este arquivo!");
            System.exit( 1 ); // termina o programa
        }
        catch ( FileNotFoundException fileNotFoundException ){
            System.err.println("Erro ao abrir ou criar arquivo!");
            System.exit( 1 );
        }
    }

    public void addFuncionario(Funcionario funcionario) throws FormatterClosedException{
        output.format("%s\n%s\n%s\n%s\n%s\n%s\n%d\n",
            funcionario.getNome(),
            funcionario.getCpf(),
            funcionario.getTelefone(),
            funcionario.getEmail(),
            funcionario.getSexo(),
            funcionario.getMatricula()
        );
    }

    public void closeFile(){ if (output!=null) output.close(); }
}
```

Formatter é um interpretador para formatar Strings em diferentes layouts no estilo printf. Para saber mais acesse:
<https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.htm>

```
// FuncionarioFileReader.java
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class FuncionarioFileReader {
    BufferedReader fileBufferedReader;

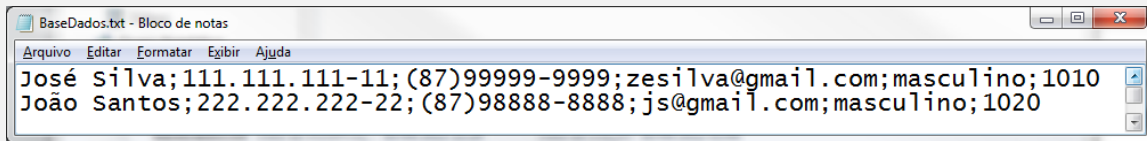
    public ArrayList<Funcionario> readFile(String path) throws IOException{
        ArrayList<Funcionario> funcionarios = new ArrayList<Funcionario>();
        ArrayList<String> dadosFuncionario = new ArrayList<String>();
        String linha="";
        InputStream is = new FileInputStream(path);
        fileBufferedReader = new BufferedReader( new InputStreamReader(is) );
        int cont=0;
        while (linha!=null){
            linha = fileBufferedReader.readLine();
            dadosFuncionario.add(linha);
            cont++;
            if (cont==6){
                cont=0;
                funcionarios.add(new Funcionario(dadosFuncionario.get(0), dadosFuncionario.get(1),
                    dadosFuncionario.get(2), dadosFuncionario.get(3),
                    dadosFuncionario.get(4),
                    Integer.parseInt(dadosFuncionario.get(5)))
                );
                dadosFuncionario.clear();
            }
        }
        fileBufferedReader.close();
        return funcionarios;
    }
}
```

```
BaseDados.txt - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda

José Silva
111.111.111-11
(87)99999-9999
zesilva@gmail.com
masculino
1010
João Santos
222.222.222-22
(81)98888-8888
js@gmail.com
masculino
1020
```

Pode-se usar:
 getClass().getResourceAsStream(path);
 Mas atenção: o file apontado por path deve estar na mesma pasta de FuncionarioFileReader.java

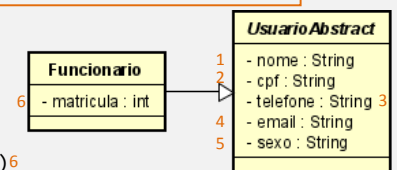
Mas e se estrutura de BaseDados.txt fosse a do arquivo abaixo?



Vejam os uma solução fazendo uso de StringTokenizer no método readFile:

```
public ArrayList<Funcionario> readFile(String path) throws IOException{
    ArrayList<Funcionario> funcionarios = new ArrayList<Funcionario>();
    String linha="";
    StringTokenizer st;
    InputStream is = new FileInputStream(path);
    fileBufferedReader = new BufferedReader (new InputStreamReader (is));
    linha = fileBufferedReader.readLine();
    do {
        st = new StringTokenizer(linha, ";");
        funcionarios.add(new Funcionario(st.nextToken(),1,
                                         st.nextToken(), 2,
                                         st.nextToken(), 3,
                                         st.nextToken(), 4,
                                         st.nextToken(), 5,
                                         Integer.parseInt(st.nextToken()))6
    );
    }while((linha = fileBufferedReader.readLine())!=null);
    fileBufferedReader.close();
    return funcionarios;
}
```

Separador dos dados em BaseDados.txt



// AppCreateBase.java

```
import java.util.FormatterClosedException;
import javax.swing.JOptionPane;
```

```
public class AppCreateBase {
    public static void main(String[] args) {
        Funcionario funcionario = new Funcionario("José Silva", "111.111.111-11", "(87)99999-9999",
                                                  "zesilva@gmail.com", "masculino", 1010);

        CreateBaseTextFile file = new CreateBaseTextFile();
        file.openFile("BaseDados.txt");
        try {
            file.addFuncionario(funcionario);
        } catch (FormatterClosedException e) {
            JOptionPane.showMessageDialog(null, "Erro de formatação ao adicionar Funcionário!");
        }
        file.closeFile();
    }
}
```

// AppReadBase.java

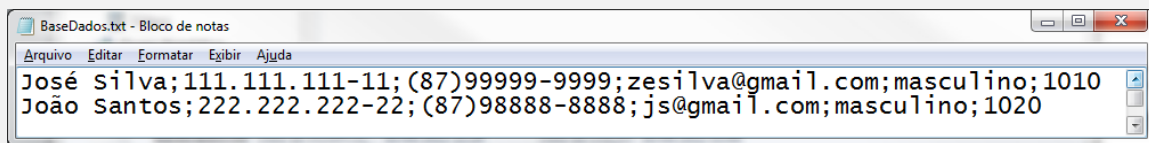
```
import java.io.IOException;
import java.util.ArrayList;
import javax.swing.JOptionPane;

public class AppReadBase {
    public static void main(String[] args) {
        FuncionarioFileReader funcionarioFileReader = new FuncionarioFileReader();
        try {
            ArrayList<Funcionario> funcionarios = funcionarioFileReader.readFile("file/BaseDados.txt");
            for (Funcionario funcionarioCurrent:funcionarios)
                System.out.println(funcionarioCurrent.toString());
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Erro ao carregar dados!");
        }
    }
}
```

Você Sabia?

Na seção “**Saiba Mais!**” vimos como utilizar arquivos TXT para armazenamento de dados. Mas a principal problemática é ter uma codificação dedicada à estrutura do arquivo, ou seja, a leitura dos dados obedece a estrutura da sequência das informações (atributos de uma classe), por exemplo:

```
//..
funcionarios.add(new Funcionario(st.nextToken(), //1º dado é o atributo nome
                                st.nextToken(), //2º dado é o atributo cpf
                                st.nextToken(), //3º dado é o atributo telefone
                                st.nextToken(), //4º dado é o atributo email
                                st.nextToken(), //5º dado é o atributo sexo
                                Integer.parseInt(st.nextToken()) //6º dado é o atributo matricula
                                );
//..
```



Uma [solução dedica](#) à estrutura do arquivo, e esse arquivo dificilmente será utilizado por outras aplicações o que inviabiliza a [interoperabilidade](#) entre sistemas.

Mas se para o projeto é escolhida a utilização de arquivo ao invés de um SGBD então se pode utilizar [XML](#) ao invés de [TXT](#).

No Brasil, por exemplo, o XML da nota fiscal de uma compra obedece ao padrão nacional de escrituração fiscal e pode ser utilizado com segurança em todo o país.

XML é a sigla para “eXtensible Markup Language” em inglês, que é basicamente um formato de arquivo universal usado para criar documentos com dados organizados, facilitando o compartilhamento de dados. É um formato que não depende das plataformas de hardware ou de software em que os dados são organizados de forma hierárquica através de **tags**.

Vejamos como seria o arquivo XML para a BaseDados.txt demonstrada:

Observe que a identificação dos atributos é dada por tags, não importando a ordem!

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <Funcionarios>
3   <Funcionario>
4     <nome>José Silva</nome>
5     <cpf>111.111.111-11</cpf>
6     <telefone>(87)99999-9999</telefone>
7     <email>zesilva@gmail.com</email>
8     <sexo>masculino</sexo>
9     <matricula>1010</matricula>
10  </Funcionario>
11  <Funcionario>
12    <matricula>1020</matricula>
13    <sexo>masculino</sexo>
14    <email>js@gmail.com</email>
15    <telefone>(87)98888-8888</telefone>
16    <cpf>222.222.222-22</cpf>
17    <nome>João Santos</nome>
18  </Funcionario>
19 </Funcionarios>
```

Para utilizar XML em um projeto Java é preciso a utilização de **bibliotecas**, como, por exemplo, [xstream](#), [dom4j](#) e [jdom](#).

Vejamos a exemplificação da seção “**Saiba Mais!**” utilizando XML:

```

// Funcionario.java
import com.thoughtworks.xstream.annotations.XStreamAlias;

@XStreamAlias("Funcionario")
public class Funcionario extends UsuarioAbstract{
    private int matricula;

    public Funcionario(String nome, String cpf, String telefone, String email,
        String sexo, int matricula) {
        super(nome, cpf, telefone, email, sexo);
        this.matricula = matricula;
    }

    public int getMatricula() {return matricula;}
    public void setMatricula(int matricula) {this.matricula = matricula;}

    @Override
    public String toString() {
        return "Funcionario [matricula=" + matricula + ", toString()="
            + super.toString() + "];"
    };
}

// CreateBaseXMLFile.java
import java.io.BufferedOutputStream;
import java.io.BufferedWriter;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.Dom4JDriver;

public class CreateBaseXMLFile{
    private XStream xStream;
    private BufferedWriter fileBuffer; //opção1
    // private DataOutputStream fileData; //opção2
    // private PrintWriter filePrint; //opção3

    public void createFile(String path) throws FileNotFoundException, IOException{
        xStream = new XStream(new Dom4JDriver());
        xStream.alias("Funcionarios", ArrayList.class);
        xStream.processAnnotations(Funcionario.class);

        //opção1
        try {
            File file = new File(path);
            if (!file.exists()) file.createNewFile();
            fileBuffer = new BufferedWriter(new FileWriter(file, true));
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
            System.err.println("Erro ao abrir ou criar arquivo!");
            throw new FileNotFoundException();
        }
        catch (IOException e){
            e.printStackTrace();
            System.err.println("Você não tem acesso de gravação a este arquivo!");
            throw new IOException();
        }
        //fim opção1
    }
}

```

//continua na próxima página...

```

        //opção 2
        try {
            //
            //      fileData = new DataOutputStream(
            //          new BufferedOutputStream(new FileOutputStream(path)));
            //
            //
            //      catch (FileNotFoundException e) {
            //          e.printStackTrace();
            //          System.err.println("Erro ao abrir ou criar arquivo!");
            //          throw new FileNotFoundException();
            //
            //      }
            //fim opção2

            //opção3
            //
            //      File file = new File(path);
            //      filePrint = new PrintWriter(file);
            //      //fim opção3
        }

        public void closeFile(){
            //opção1
            if (fileBuffer!=null)
                try {
                    fileBuffer.close();
                } catch (IOException e) {
                    e.printStackTrace();
                    System.err.println("Erro ao fechar arquivo!");
                    System.exit( 1 );
                }
            //fim opção1

            //opção2
            //
            //      if (fileData!=null)
            //          try {
            //              fileData.close();
            //          } catch (IOException e) {
            //              e.printStackTrace();
            //              System.err.println("Erro ao fechar arquivo!");
            //              System.exit( 1 );
            //          }
            //      //fim opção2

            //opção3
            //
            //      if (filePrint!=null) filePrint.close();
            //      //fim opção3
        }

        public void addFuncionario(ArrayList<Funcionario> funcionarios) throws IOException{
            //opção1:
            try {
                fileBuffer.write(xStream.toXML(funcionarios));
                fileBuffer.flush();
            } catch (IOException e) {
                e.printStackTrace();
                System.err.println("Você não tem acesso de gravação a este arquivo!");
                throw new IOException();
            }
            //fim opção1

            //opção2
            //
            //      try {
            //          fileData.writeBytes(xStream.toXML(funcionarios));
            //          fileData.flush();
            //      } catch (IOException e) {
            //          e.printStackTrace();
            //          System.err.println("Você não tem acesso de gravação a este arquivo!");
            //          throw new IOException();
            //      }
            //      //fim opção2

            //opcao3
            //
            //      filePrint.println(xStream.toXML(funcionarios));
            //      filePrint.flush();
            //      //fim opção3
        }
    }

```

```

//FuncionarioFileReader.java
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.Dom4JDriver;

public class FuncionarioFileReader {

    BufferedReader fileBufferedReader;
    InputStream fileInput;

    public void openFile(String path) throws FileNotFoundException, IOException{
        fileInput = new FileInputStream(path);
        fileBufferedReader = new BufferedReader(new InputStreamReader(fileInput));
    }

    public ArrayList<Funcionario> readFile() throws FileNotFoundException, IOException{
        XStream xStream = new XStream(new Dom4JDriver());
        xStream.alias("Funcionarios", ArrayList.class);
        xStream.processAnnotations(Funcionario.class);
        return (ArrayList) xStream.fromXML(fileBufferedReader);
    }
}

//AppCreateBase.java
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import javax.swing.JOptionPane;

public class AppCreateBase {
    public static void main(String[] args) {
        ArrayList<Funcionario> funcionarios = new ArrayList<Funcionario>();
        funcionarios.add(new Funcionario("José Silva", "111.111.111-11", "(87)99999-9999",
"zesilva@gmail.com", "masculino", 1010));
        funcionarios.add(new Funcionario("João Santos", "222.222.222-22", "(87)98888-8888",
"js@gmail.com", "masculino", 1020));
        CreateBaseXMLFile file = new CreateBaseXMLFile();
        try {
            file.createFile("file/BaseDados.xml");
            file.addFuncionario(funcionarios);
            file.closeFile();
        } catch (FileNotFoundException e1) {
            JOptionPane.showMessageDialog(null, "Erro ao abrir ou criar arquivo!");
        } catch (IOException e1) {
            JOptionPane.showMessageDialog(null, "Você não tem acesso de gravação a este arquivo!");
        }
    }
}

//AppReadBase.java
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import javax.swing.JOptionPane;

public class AppReadBase {
    public static void main(String[] args) {
        FuncionarioFileReader funcionarioFileReader = new FuncionarioFileReader();

        try {
            funcionarioFileReader.openFile("file/BaseDados.xml");
            ArrayList<Funcionario> funcionarios = funcionarioFileReader.readFile();
            for (Funcionario funcionarioCurrent:funcionarios)
                System.out.println(funcionarioCurrent.toString());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Erro ao abrir o arquivo!");
        }
        catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Erro ao carregar dados!");
        }
    }
}

```

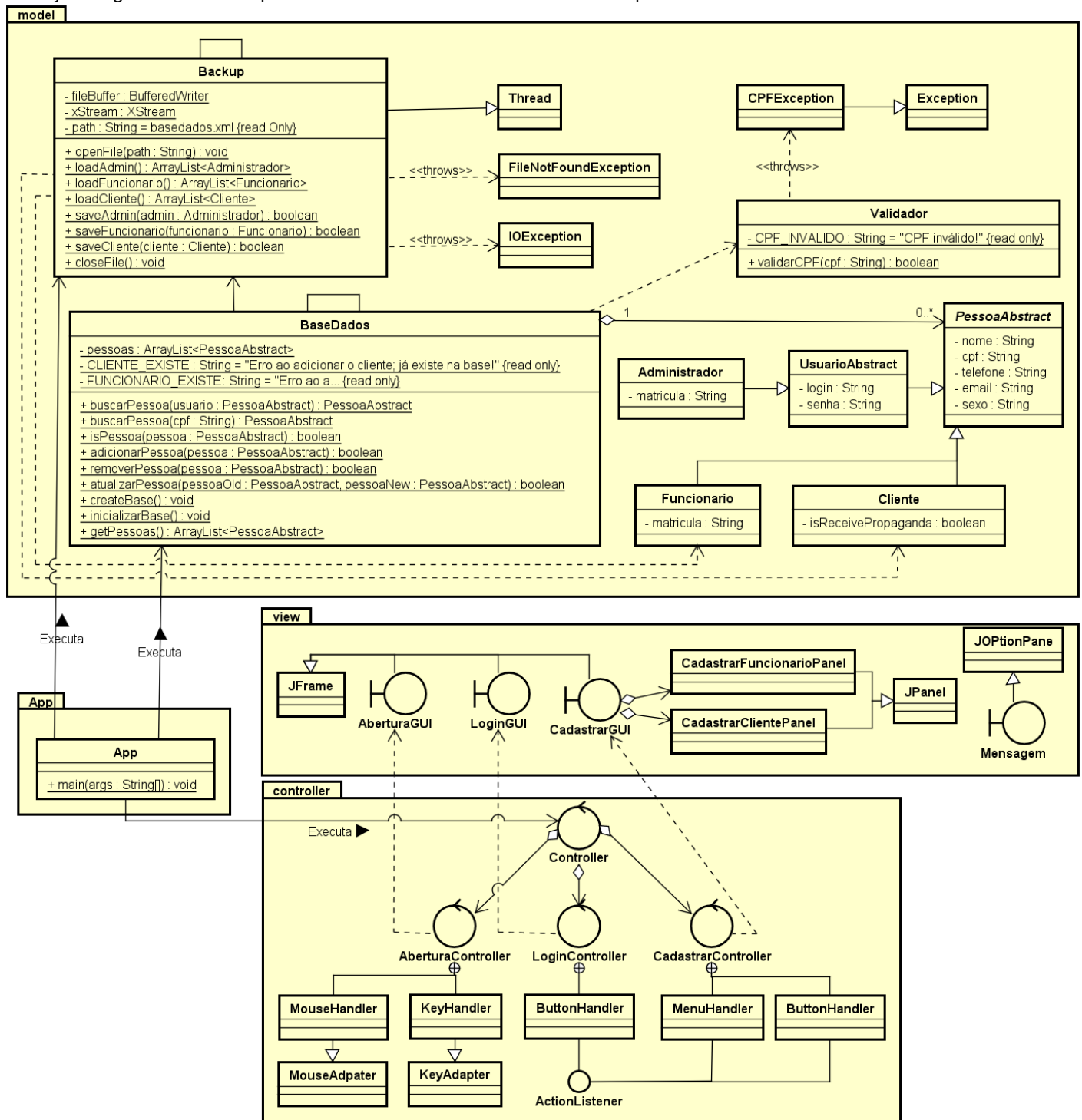
Mão na Massa!

2) O sistema de “MPOO Market” precisa ser desenvolvido. Ajude ao programador “O Furão” (codinome para *mustela putórius furo*) apresentando uma solução para persistência dos dados de cadastros de clientes, funcionários e administradores em uma base de dados disposta em ROM. Para isso utilize o arquivo basedados.xml para persistir BaseDados.. E sempre que uma nova pessoa for cadastrada irá utilizar os métodos de BaseDados, que por sua vez, utilizarão os métodos de Backup.

Também deve haver uma Thread que salva a cada 5 minutos faz um backup dos dados nos arquivos basedadosbackup.txt e basedadosbackup.xml.

Lembre-se de utilizar GUI's do sistema (Apêndice A) já desenvolvidas em listas anteriores.

Veja o diagrama de classes que o Scrum Master de MPOOSoftware LTDA apresentou a “O Furão”:



São Regras de negócios:

- RN01 – As operações na base devem verificar se uma pessoa e seu cpf são válidos.
- RN02 – A codificação deve aproveitar comportamentos já definidos, evitando a duplicidade de programação;
- RN03 – Apenas administradores poderão logar no sistema;
- RN04 – A liberação do botão “Adicionar” da tela de cadastro é dada apenas quando todos os campos forem preenchidos; e
- RN05 – Confirmações devem exibir mensagens para o usuário, por exemplo:



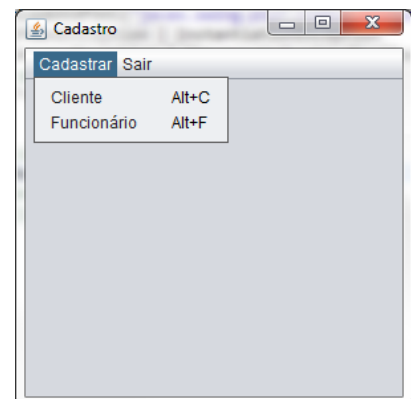
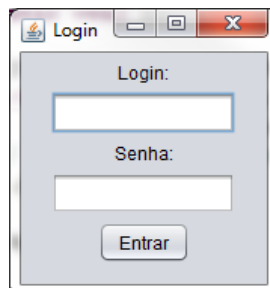
Para validação de cpf utilize o método `validarCPF(String CPF)` apresentado nas listas de exercícios VIII e IX.

Problema: Quando o sistema for inicializado devem-se carregar as pessoas já cadastradas na base de dados (basedados.xml). Para isso defina pelo menos três pessoas, sendo um cliente, um funcionário e um administrador.

Utilize o gerador de cpf para garantir dados válidos: <https://www.geradordecpf.org/>

APÊNDICE

Telas do Sistema



Para entrar na tela de Login é preciso que clicar com o mouse ou pressionar a tecla Enter na tela de abertura. A tecla Esc fecha o sistema. A partir da seleção das opções do menu Cadastrar (Cliente ou Funcionário) tem-se as telas abaixo. Realize os tratamentos de eventos para as interações do usuário conforme o diagrama de classes do sistema.

Observe o campo nome. Quando o usuário clicar no campo nome a dica “Nome Completo” deve desaparecer, mas caso o usuário não digite nada ela deverá continuar.

