LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA

2021/2022

MODULAÇÃO E PROGRAMAÇÃO

4º Trabalho (Parte A)

24/05/2022

Docente: Engº Carlos Júnior

Turma: LEIM23D

Realizado por:
Eduardo Marques A45977

Conteúdo

1 Introdução **2**

2 Conceito **3**

3 Funcionalidades **3**
3.1 Consultar stock . . . 3
3.2 Adicionar um Livro . . . 3
3.3 Vender um livro . . . 3

4 Classes necessárias e Diagrama UML **4**

5 Aprofundamento de cada Classe **5**
5.1 Categoria . . . 5
5.2 Nome . . . 5
5.3 Livro . . . 7
5.4 ConjLivros . . . 9
5.5 Stock . . . 10
5.6 AdicionarLivro . . . 10
5.7 VendaLivro . . . 10
5.8 Biblioteca . . . 11

6 Exemplo do protótipo do programa a funcionar **13**
6.1 Autenticação e menu com as escolhas . . . 13
6.2 Consulta do *stock* . . . 13
6.3 Adição de um livro ao *stock* . . . 14
6.4 Venda de um livro . . . 14
6.5 Mensagem de saída . . . 15

7 *WireFrame* **16**
7.1 Autenticação . . . 16
7.2 Menu principal . . . 16
7.2.1 *Stock* . . . 16
7.2.2 Adição ou Venda de um livro . . . 17

1 Introdução

Este trabalho pretende consolidar e avaliar os conhecimentos adquiridos em Modelação e Programação dando ao aluno a oportunidade de mostrar a sua criatividade enquanto aprofunda os conhecimentos adquiridos ao longo do semestre.

O objectivo do trabalho é desenvolvimento de uma aplicação usando a linguagem Java com interface gráfica (a ser introduzida posteriormente na Parte B deste trabalho).

Nesse sentido, o aluno deverá aplicar os conhecimentos adquiridos na disciplina para criar um modelo de objetos para o domínio da aplicação a que se propõe desenvolver. O aluno deverá desenhar e defender perante o docente da disciplina os objetivos da aplicação e apresentar o diagrama de classes UML com todas as entidades, abstracções, interfaces e relações entre entidades e as acções/comportamentos aplicáveis às instâncias dos objetos dessas mesmas classes.

A aplicação deverá também contemplar o armazenamento de dados (e.g. estado da aplicação) de forma persistente. Como tal, o aluno deverá também planear a gramática (DTD) e o exemplo de um documento *XML* que esteja válido de acordo com essa gramática.

2 Conceito

Pretende-se a criação de uma aplicação a ser ultilizada por uma biblioteca para o controlo de *stock* dos livros dentro da mesma.

Cada livro dirá o seu título, preço e as categorias às quais pertence, ou seja, se é um livro de ação, romance, etc...

3 Funcionalidades

Esta aplicação está protegida por uma palavra passe, desde modo o utilizador terá que introduzir a password correta com um máximo de 3 tentativas, caso contrário não irá conseguir autenticar-se.

Após o passo anterior ser superado, o utilizador irá deparar-se com 3 opções.

3.1 Consultar stock

Nesta opção não será necessária qualquer tipo de autenticação a mais, irá apenas mostrar quantos livros existem na biblioteca, preço total e uma lista com os livros todos disponíveis, cada um com título, preço e entre uma e oito categorias.

3.2 Adicionar um Livro

Nesta segunda opção o utilizador poderá adicionar um novo livro ao stock de livros existentes.

Para tal ocorrer, o utilizador terá que passar por uma outra autenticação, sendo esta o nome do trabalhador, ou seja, só poderá adicionar um livro ao stock o trabalhador da biblioteca.

Após a autenticação ser corretamente validada, iremos entrar na janela para adicionar um livro onde o utilizador terá que meter um titulo válido, um preço válido e, por fim, entre uma e oito categorias.

3.3 Vender um livro

Por fim, irá ter uma opção onde o utilizador pode vender um livro. À semelhança com a opção anterior, o utilizador terá que passar por uma outra autenticação, sendo esta o nome do trabalhador, ou seja, só o trabalhador da biblioteca poderá vender um livro. O utilizador terá também que escrever o nome do cliente que irá ser validado.

Após a autenticação ser corretamente validada, iremos entrar na janela para vender um livro onde o utilizador terá que meter um titulo válido, um preço válido e, por fim, entre uma e oito categorias. Se o livro constar do stock da biblioteca, o livro pode ser vendido, caso contrário irá aparecer uma mensagem a dizer que o livro que se quer vender não consta do stock.

4 Classes necessárias e Diagrama UML

Para o funcionamento idealizado iram ser necessárias cinco classes sendo elas a classe Livro, a classe ConjLivros, a classe AdicionarLivro, a classe VendaLivro e a classe principal, a classe Biblioteca.

Também serão necessárias duas classes abstratas sendo elas a Nome e Stock.

Existe ainda uma interface IInformacao e um enumerado Categoria.

Todas estas classes irão ser aprofundadas e detaladas mais à frente.

Ficamos assim com o diagrama UML seguinte:

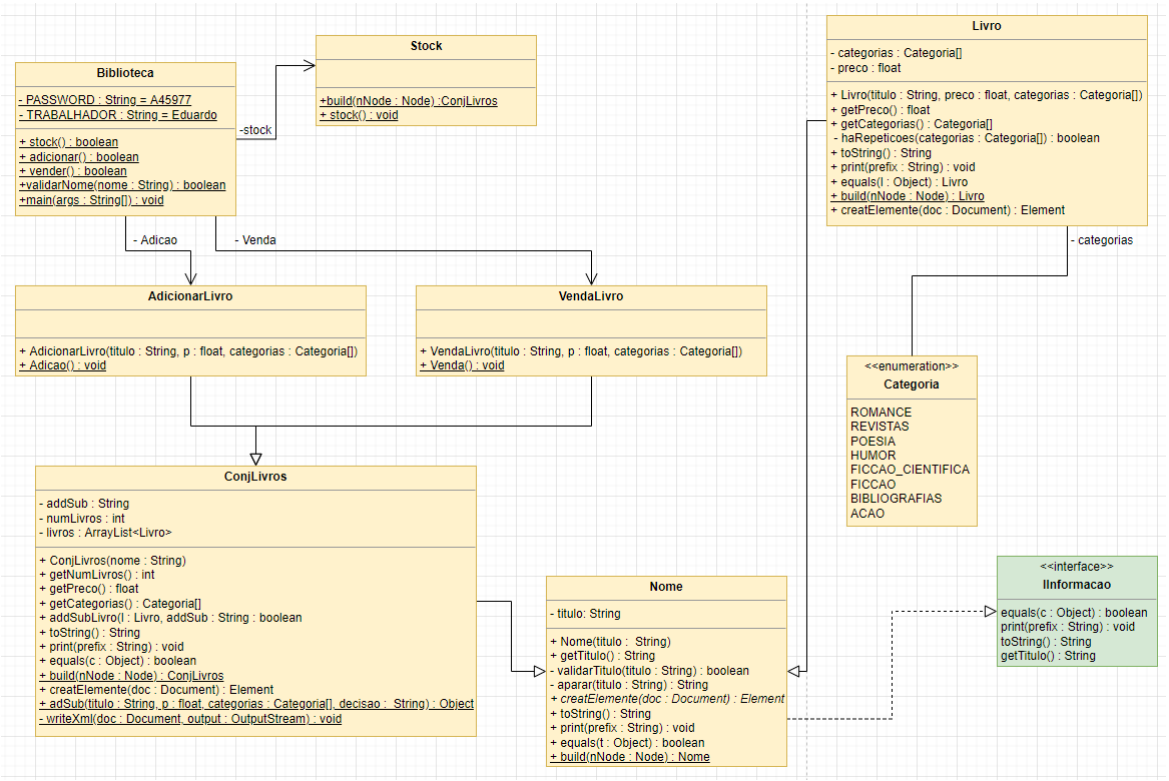

Figura 1: Diagrama UML

Enquanto o programa corre, caso haja um erro graças na introdução de qualquer variável, o utilizador irá ser informado na consola. Para tal ser sucedido irão ser implementados mecanismos que permitem mostrar o erro mas continuar a correr a aplicação sem qualquer problema.

5 Aprofundamento de cada Classe

5.1 Categoria

É uma classe enumerado. Ao ser utilizada, a biblioteca só aceita estas categorias de livros. Para serem utilizadas outras categorias, as mesmas teriam que ser adicionadas ao enumerado.

```
public enum Categoria {ACAO, BIBLIOGRAFIAS, FICCAO,
    FICCAO_CIENTIFICA, HUMOR, POESIA, REVISTAS, ROMANCE}
```

Figura 2: Valores do enumerado

5.2 Nome

É uma classe abstrata que tanto pode ser utilizada pela classe Livro como pela classe ConjLivros. Tem um contrutor para o nome, o que significa que irá validar o título por exemplo.

Tem também um método para validar o nome que entra no construtor, para aparar o mesmo, ou seja, se o título introduzido for ”Os Lusiadas ”, o método irá retirar os espaços a mais e vai devolver ”Os Lusiadas”. Tem também um método para comparar objectos.

```
public Nome(String titulo){
    /*Começamos por validar o titulo do livro/conjunto, porém em vez de o validar no construtor,
     *  irei criar um método especifico para o código ficar mais organizado*/
    if(!validarTitulo(titulo)){
        //se o titulo não for válido, envia a mensagem seguinte
        throw new IllegalArgumentException("O título introduzido não é válido");
    }
    //se for válido, instanciamos este livro/conjunto com este titulo
    this.titulo = aparar(titulo);//aparamos primeiro o titulo para retirar espeços a mais
}
```

Figura 3: Contrutor da Classe Nome

```java
private String aparar(String titulo) {
    //.trim() elimina espaços no inicio e no final de uma string, mas não elimina espaços no meio da string
    titulo = titulo.trim();

    //criação da StringBuilder
    StringBuilder sbTitulo = new StringBuilder(titulo);
    for(int i = 0; i < sbTitulo.length(); i++){
        if(Character.isWhitespace(sbTitulo.charAt(i)) && Character.isWhitespace(sbTitulo.charAt(i+1))){
            /*se a posição onde estivermos for um espaço bem como a posição seguinte tambem
             * for um espaço, então vamos retirar a posição onde estamos e vamos tirar uma
             * unidade a i, pois pode haver mais do que um espaço, o que significa que se
             * retirarmos o char seguinte em vez do char corrente, uma string com 3 espaços
             * iria passar a ter 2 espaços em vez de 1*/
            sbTitulo.deleteCharAt(i);
            i--;
        }
    }
    return sbTitulo.toString();
}
```

Figura 4: Metodo para aparar o título

```java
private boolean validarTitulo(String titulo) {
    //começamos por ver se o titulo é null ou igual a 0
    if(titulo == null || titulo.length() == 0){
        return false;
    }

    /*no fim da validação, a variavel nCaract tem de ser pelo menos 1
     * para o nome ser considerado válido*/
    int nCaract = 0;

    for(int i = 0; i < titulo.length(); i++){
        //variavel auxiliar para guardar o caracter na posição i
        char aux = titulo.charAt(i);
        //se o caracter não for uma letra nem um espeço nem um numero retorna false
        if(!Character.isWhitespace(aux) && !Character.isLetter(aux) && !Character.isDigit(aux)){
            return false;
        }
        //se o caracter for uma letra ou numero vai iterar nCaract
        if (Character.isLetter(aux) || Character.isDigit(aux)){
            nCaract++;
        }
    }
    //se houver pelo menos um caracter válido então retorna true
    if (nCaract >= 1){
        return true;
    }
    //caso contrário retorna false
    else return false;
}
```

Figura 5: Método para validar o título

Também nesta classe existe o método *build* que, como o nome indica, constrói informação a parit de um nó vindo de um ficheiro *XML*.

5.3 Livro

É uma classe que extende a classe anterior Nome.

Tanto para adicionar como para vender um livro, irá sempre passar por esta classe para o mesmo ser construído e validado. Por esta razão, o construtor além de ter uma *String* título que será validado através do uso do *super*, também irá ter um *float* com o preço e um *array* de categorias que utilização o enumerado.

```
public Livro(String titulo, float preco, Categoria[] categorias){
    //validação do titulo feita na classe abstrata
    super(titulo);
    //o preço não pode ser negativo
    if (preco < 0){
        throw new IllegalArgumentException("O preço não pode ser negativo");
    }
    this.preco = preco;

    //cada livro tem de ter pelo menos uma categoria existente
    if (categorias == null || categorias.length == 0){
        throw new IllegalArgumentException("O livro tem que ter pelo menos uma categoria");
    }
    //cada livro não pode ter categorias null
    for (int i = 0; i < categorias.length; i++){
        if (categorias[i] == null){
            throw new IllegalArgumentException("O livro não pode conter categorias nulas");
        }
    }
    /*cada livro não pode também ter categorias repetidas, ou seja, um livro não pode ter
    * as categorias [HUMOR, AÇÃO, HUMOR]*/
    if(haRepeticoes(categorias)){
        throw new IllegalArgumentException("O livro não pode ter categorias repetidas");
    }
    this.categorias = categorias;
}
```

Figura 6: Construtor do Livro

A classe Livro tem também um método *build* que constroi um novo livro a partir de um nó contendo as infomações lidas do documento *XML*, e um método *createElement* que cria um novo Elemento quer irá representar, no documento *XML*, o Livro associado ao Livro corrente.

```java
public static Livro build(Node nNode) {
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
        Element eLivro = (Element) nNode;

        String titulo = eLivro.getElementsByTagName("Titulo").item( index: 0).getTextContent();
        float preco = Float.parseFloat(eLivro.getElementsByTagName("Preco").item( index: 0).getTextContent());
        Element categorias = (Element) eLivro.getChildNodes();
        NodeList categoria = categorias.getElementsByTagName("Categoria");
        ArrayList<Categoria> c = new ArrayList<~>();

        int len = categoria.getLength();

        for (int j = 0; j < len; j++) {
            String a = categorias.getElementsByTagName("Categoria").item(j).getTextContent();
            c.add(Categoria.valueOf(a));
        }
        Categoria[] cat = c.toArray(new Categoria[c.size()]);
        Livro livro = new Livro(titulo, preco, cat);
        return livro;
    }
    return null;
}
```

Figura 7: *Build* do Livro

```java
public Element createElement(Document doc) {

    //começamos por criar o elemento Livro
    Element livro = doc.createElement( tagName: "Livro");

    Element titulo = doc.createElement( tagName: "Titulo");
    titulo.appendChild(doc.createTextNode(getTitulo()));
    livro.appendChild(titulo);

    Element preco = doc.createElement( tagName: "Preco");
    preco.appendChild(doc.createTextNode(String.valueOf(getPreco())));
    livro.appendChild(preco);

    Element categorias = doc.createElement( tagName: "Categorias");
    livro.appendChild(categorias);

    for (int i = 0; i < getCategorias().length; i++){
        Element categoria = doc.createElement( tagName: "Categoria");
        categoria.appendChild(doc.createTextNode(String.valueOf(getCategorias()[i])));
        categorias.appendChild(categoria);
    }
    return livro;
}
```

Figura 8: *Create Element* do Livro

Nas classes onde também são utilizados os métodos *Build* e *createElement*, a maneira como o código é feito é praticamente igual, pelo que não irei voltar a meter o seu código no relatório, para não ser repetitivo.

5.4 ConjLivros

Outra classe que extende a classe abstrata Nome. Esta classe irá servir para mostra o conjunto de livros, adicionar e vender livros.

É também nesta classe que iremos ler e escrever no ficheiro *XML*, que irá servir como base de dados sobre o *stock* de livros, este ficheiro irá ter a lista de livros com o respetivo, título, preço e categorias.

A leitura e escrita será efetuada na classe "*addSub*".

```
public static Object adSub (String titulo, float p, Categoria[] categorias, String decisao){
    try {
        File inputFile = new File( pathname: "C:/Users/.../tp4/Biblioteca.xml");

        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(inputFile);
        System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

        XPath xpath = XPathFactory.newInstance().newXPath();
        String expression = "/Biblioteca/TotalLivros/*";
        NodeList nList = (NodeList) xpath.evaluate(expression, doc, XPathConstants.NODESET);

        Node nNode = nList.item( index: 0);
        Nome nome = Nome.build(nNode);
        if (nome != null);

        Livro l = new Livro(titulo, p, categorias);
```

Figura 9: Método para ler e escrever no documento *XML* (Parte1)

```
if (nome instanceof ConjLivros){
    ConjLivros conj = (ConjLivros) nome;
    conj.addSubLivro(l, decisao);
    System.out.println(conj);
    Document newDoc = dBuilder.newDocument();


    Element rootElement = newDoc.createElement( tagName: "Biblioteca");
    rootElement.appendChild(conj.createElement(newDoc));


    newDoc.appendChild(rootElement);

    FileOutputStream output = new FileOutputStream( name: "C:/Users/.../tp4/Biblioteca.xml");
    writeXml(newDoc, output);
    conj.print("");
}
```

Figura 10: Método para ler e escrever no documento *XML* (Parte2)

5.5 Stock

É uma classe abstrata que serve apenas para ler o ficheiro *XML* e mostrar o *stock* existente na biblioteca, usando um método *build* igual ao da classe ConjLivros.

```
public static void stock(){
    try {
        File inputFile = new File( pathname: "C:/Users/.../tp4/Biblioteca.xml");

        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(inputFile);
        System.out.println("Root element :" + doc.getDocumentElement().getNodeName());

        XPath xpath = XPathFactory.newInstance().newXPath();
        String expression = "/Biblioteca/TotalLivros/*";
        NodeList nList = (NodeList) xpath.evaluate(expression, doc, XPathConstants.NODESET);

        Node nNode = nList.item( index: 0);
        ConjLivros nome = ConjLivros.build(nNode);
        if (nome != null) nome.print("");

    }catch (Exception e) {
        e.printStackTrace();
    }
}
```

Figura 11: Leitura do ficheiro *XML*

5.6 AdicionarLivro

É uma classe que extende a classe ConjLivros que irá servir para adicionar um livro.

Começa por pedir a *password* de autenticação. O utilizador tem 3 tentativas para acertar na mesma.

Nesta classe a aplicação irá pedir para o utilizador do livro adicionar as variáveis necessárias para adicionar o livro, ou seja, vai pedir o titulo, o preço do livro, o número de categorias e as respetivas categorias. Ao ter estas três variáveis, chama o método ”*adSub*”da Classe ”ConjLivros”que irá tratar de adicionar o livro e escrever no ficheiro *XML*.

5.7 VendaLivro

É uma classe que extende a classe ConjLivros que irá servir para vender um livro.

Faz o mesmo que a Classe anterior, com a diferença que em vez de adicionar, vende. Chama o mesmo método da classe ”ConjLivros”. A única diferença está no método que chama que irá, antes de remover, ver se o livro introduzido consta no *stock* da biblioteca.

5.8 Biblioteca

É a classe principal da aplicação. É nesta classe que o utilizador vai decidir o que quer fazer tendo em conta as três opções descritas, ou se vai querer fechar a aplicação.

Para cada uma das três opções, há um método correspondente.

Começando na opção para ver o *stock*, o seu método vai chamar o método *stock* da Classe *Stock*, que irá mostrar o stock existente na biblioteca.

```
public static boolean stock() {
    System.out.println("Consulta de Stock");
    Stock.stock();
    return true;
}
```

Figura 12: Método que é utilizado quando o utilizador seleciona a opção ”Consultar stock”

Se o utilizador selecionar a opção para adicionar um livro, o programa irá chamar o método ”adicionar”. Neste método é pedido o nome do utilizador para, como mencionado, só poder ser o trabalhador a adicionar livros.

```
public static boolean adicionar() {
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Adicionar um livro ao stock");
    System.out.println("Insira o nome do empregado que está a adiconar o livro");
    String nome = keyboard.nextLine();

    //vê se o nome do empregado é válido
    if (!nome.equalsIgnoreCase(TRABALHADOR)) {
        System.out.println("Esta pessoa não trabalha aqui, logo não pode adicionar um livro");
        return false;
    } else {
        AdicionarLivro.Adicao();
        System.out.println("\nLivro adicionado com sucesso");
        return true;
    }
}
```

Figura 13: Método que é utilizado quando o utilizador seleciona a opção ”Adicionar um Livro”

Se o utilizador selecionar a opção vender um livro o programa irá chamar o método ”vender”. Como no ponto anterior também pede o nome do trabalhar, porém neste método também é pedido o nome do cliente que está a comprar o livro.

Se o livro que o cliente quiser comprar não constar no *stock* da biblioteca, então aparece uma mensagem a dizer que não existe o livro.

```
public static boolean vender() {
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Vender um livro");
    System.out.println("Insira o nome do empregado que está a vender o livro");
    String nome = keyboard.nextLine();

    //vê se o nome do empregado é válido
    if (!nome.equalsIgnoreCase(TRABALHADOR)) {
        System.out.println("Esta pessoa não trabalha aqui, logo não pode adicionar um livro");
        return false;
    }

    System.out.println("Insira o nome do Cliente");
    String nome2 = keyboard.nextLine();

    //valida se o nome do cliente é válido
    if (!validarNome(nome2)) {
        System.out.println("O nome do cliente é inválido");
        return false;
    } else {
        VendaLivro.Venda();
        System.out.println("\nLivro vendido com sucesso");
        return true;
    }
}
```

Figura 14: Método que é utilizado quando o utilizador seleciona a opção ”Vender um Livro”

Por fim, se o utilizador selecionar a opção de saída, a aplicação lança uma mensagem de despedida e a aplicação fecha.

6 Exemplo do protótipo do programa a funcionar

Neste ponto será mostrado exemplos da aplicação a funcionar corretamente.

6.1 Autenticação e menu com as escolhas

```
Introduza o código de acesso, por favor: A45977

Bem Vindo!

Selecione o que pretende fazer das opções seguintes:
a - Consultar stock
b - Adicionar um Livro
c - Vender um livro
f - Terminar
Opção:
```

Figura 15: Autenticação e menu com as escolhas

6.2 Consulta do *stock*

```
Opção: a
Consulta de Stock
Root element :Biblioteca
Conjunto "Stock" com 4 livros e com preço final de 70.55€
   Livro "O Mercador de Livros", com valor de 17.3€ por livro e com categorias [ACAO, ROMANCE]
   Livro "Era da Inteligência Artificial", com valor de 13.75€ por livro e com categoria [BIBLIOGRAFIAS]
   Livro "1984", com valor de 15.5€ por livro e com categoria [FICCAO]
   Livro "Os Lusíadas", com valor de 24.0€ por livro e com categoria [POESIA]

Selecione o que pretende fazer das opções seguintes:
a - Consultar stock
b - Adicionar um Livro
c - Vender um livro
f - Terminar
Opção:
```

Figura 16: Consulta do *stock*

6.3 Adição de um livro ao *stock*

```
Opção: b
Adicionar um livro ao stock
Insira o nome do empregado que está a adiconar o livro
Eduardo
Adicionar um livro
Introduza o nome do livro:
O espião israelita
Introduza o preço do livro:
17.50
Introduza o número de Categorias do livro:
1
Introduza a(s) Categoria(s) do livro: 1)ACAO, 2)BIBLIOGRAFIAS, 3)FICCAO, 4)FICCAO_CIENTIFICA, 5)HUMOR, 6)POESIA, 7)REVISTAS, 8)ROMANCE ou 10 para terminar a adição
1
Root element :Biblioteca
Conjunto "Stock" com 5 livros e com preço final de 88.05€
Conjunto "Stock" com 5 livros e com preço final de 88.05€
   Livro "O Mercador de Livros", com valor de 17.3€ por livro e com categorias [ACAO, ROMANCE]
   Livro "Era da Inteligência Artificial", com valor de 13.75€ por livro e com categoria [BIBLIOGRAFIAS]
   Livro "1984", com valor de 15.5€ por livro e com categoria [FICCAO]
   Livro "Os Lusiadas", com valor de 24.0€ por livro e com categoria [POESIA]
   Livro "O espião israelita", com valor de 17.5€ por livro e com categoria [ACAO]

Livro adicionado com sucesso
```

Figura 17: Adição de um livro ao *stock*

6.4 Venda de um livro

```
Opção: c
Vender um livro
Insira o nome do empregado que está a vender o livro
Eduardo
Insira o nome do Cliente
Exemplo
Venda de um livro
Introduza o nome do livro:
Os Lusiadas
Introduza o preço do livro:
24
Introduza o número de Categorias do livro:
1
Introduza a(s) Categoria(s) do livro: 1)ACAO, 2)BIBLIOGRAFIAS, 3)FICCAO, 4)FICCAO_CIENTIFICA, 5)HUMOR, 6)POESIA, 7)REVISTAS, 8)ROMANCE ou 10 para terminar a adição
6
Root element :Biblioteca
Livro "Os Lusiadas", com valor de 24.0€ por livro e com categoria [POESIA]
Conjunto "Stock" com 4 livros e com preço final de 64.05€
Conjunto "Stock" com 4 livros e com preço final de 64.05€
   Livro "O Mercador de Livros", com valor de 17.3€ por livro e com categorias [ACAO, ROMANCE]
   Livro "Era da Inteligência Artificial", com valor de 13.75€ por livro e com categoria [BIBLIOGRAFIAS]
   Livro "1984", com valor de 15.5€ por livro e com categoria [FICCAO]
   Livro "O espião israelita", com valor de 17.5€ por livro e com categoria [ACAO]

Livro vendido com sucesso
```

Figura 18: Venda de um livro

6.5 Mensagem de saída

```
Selecione o que pretende fazer das opções seguintes:
a - Consultar stock
b - Adicionar um Livro
c - Vender um livro
f - Terminar
Opção: f

Até à próxima
```

Figura 19: Mensagem de saída

7 *WireFrame*

Por fim irei falar sobre a *wireframe* em mente.

7.1 Autenticação

A primeira janela a aparecer será a janela de autenticação, onde o utilizador terá que meter a *password* correta com o máximo de 3 tentativas.

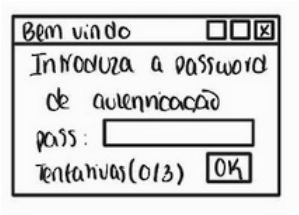

Figura 20: Janela de Autenticação

7.2 Menu principal

Após a autenticação ser validada corretamente, o utilizador irá deparar-se com o menu onde principal onde poderá escolher ver o *stock* e adicionar ou vender um livro.

Figura 21: Menu Principal

7.2.1 *Stock*

Se o utilizador escolher ver o *stock*, irá aparecer uma janela com o *stock* disponível de livros.

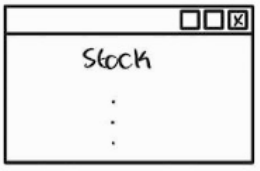

Figura 22: Janela com o *Stock* disponível

7.2.2 Adição ou Venda de um livro

Caso o utilizador decida adicionar ou vender um livro, irá aparecer uma janela em comum que é a janela da segunda autenticação, a autenticação referente ao nome do trabalhador.

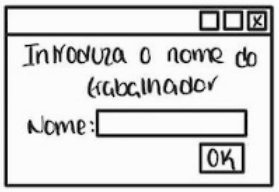

Figura 23: Janela com a Segunda Autenticação

Quando a autenticação for válida irão aparecer duas janelas semelhantes, uma referente à adição de livros e uma referente à venda de livros. A única diferença entre as duas janelas será o seu título e o espeço para meter o nome do cliente, que só irá aparecer na janela da venda do livro. Ambas as janelas irão ter um espaço para introduzir o título, o preço, o número de categorias do livro e as categorías.

Figura 24: Janela da Adição de um livro

Figura 25: Janela da Venda de um livro