Sistema de Gestão para uma Smart Store

Fase 2
Licenciatura de Engenharia de Informática
Paradigmas da Programação

Paulo Enes
Valéria Pequeno

Turno Diurno
18/05/2019

Bruno Saraiva 20160782
Diogo Palos 30001058
Miguel Nunes 30000814
Ricardo Melo 30000486

Índice

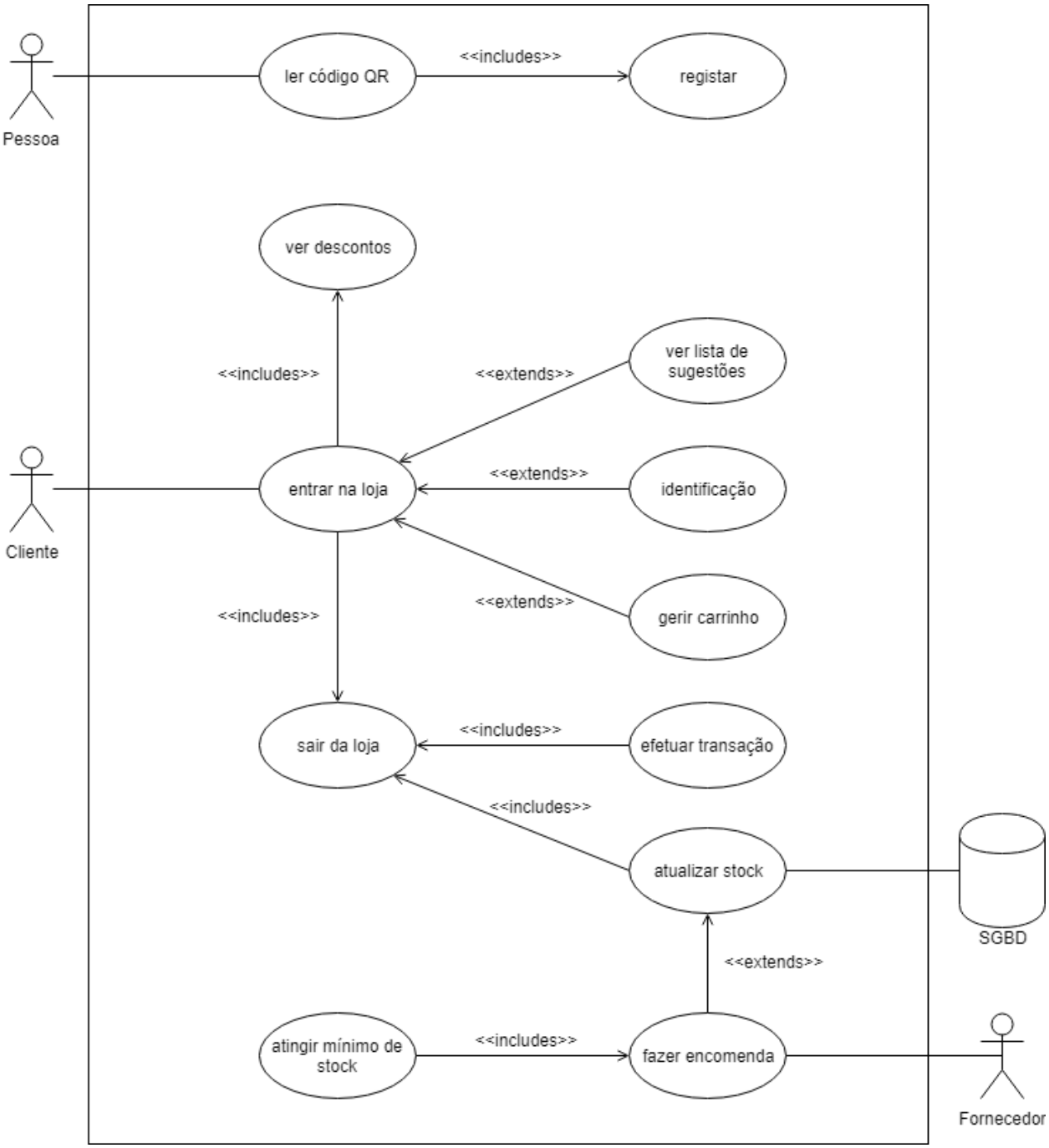
Introdução

1. Introdução

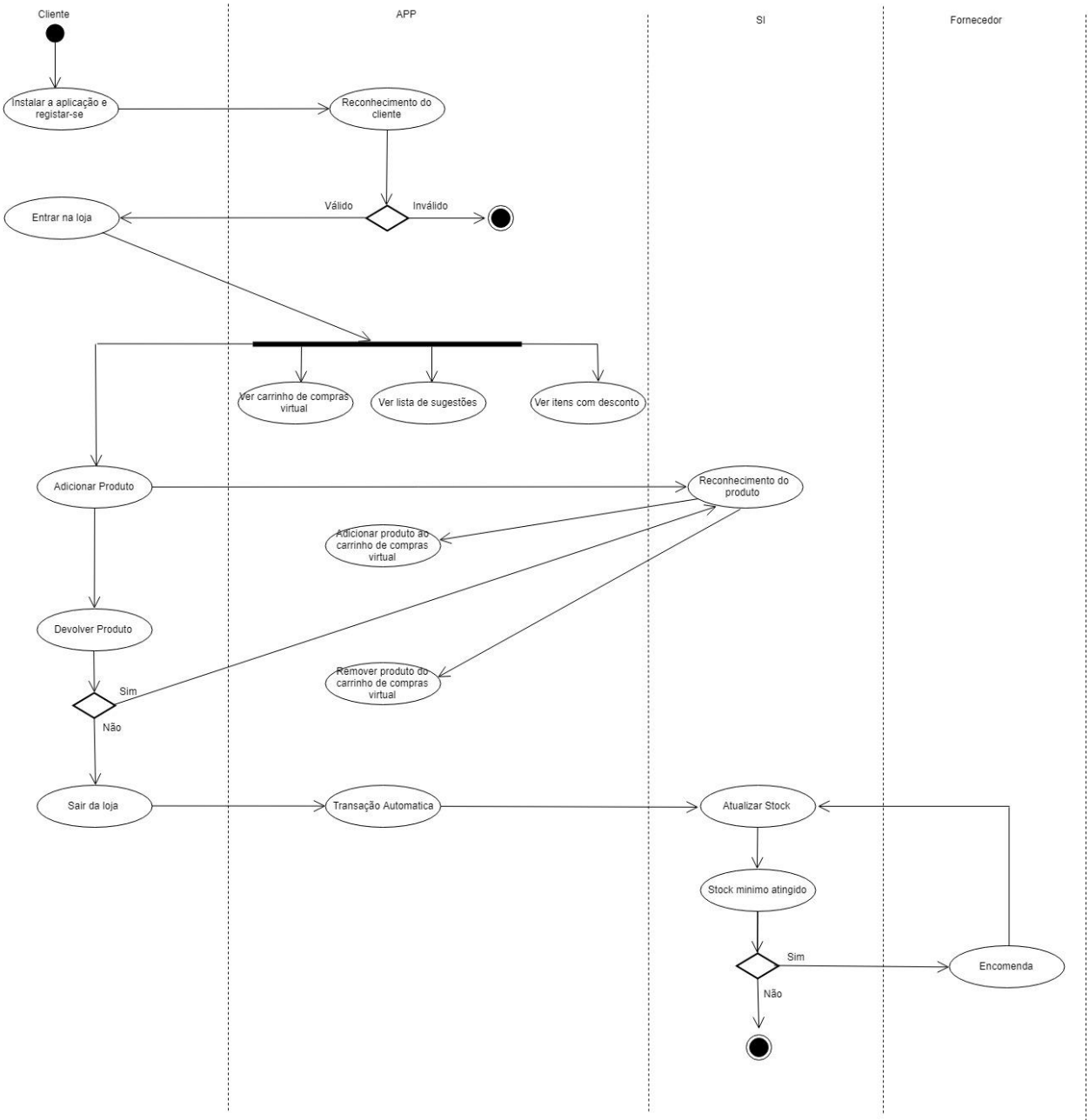
O objetivo deste trabalho é produzir uma aplicação em Java que funciona dentro de uma smart store. Esta aplicação vai acompanhar o que o utilizador coloca no seu carrinho, ou seja, o que adiciona e/ou remove de produtos no carrinho, atualizando a sua lista de compras e efetuando o controlo de stock da smart store no fim da compra do utilizador. Este controlo, consiste num alerta dado pela aplicação para se reporem em stock quantidades de um produto necessárias para satisfazer futuras compras por parte dos utilizadores da aplicação, sendo enviada uma requisição do produto para o fornecedor, contendo as quantidades, as referências e respetivos preços unitários. Ao sair da smart store o pagamento é efetuado automaticamente, e a aplicação vai enviar uma fatura para o utilizador.

2. Análise do Projeto

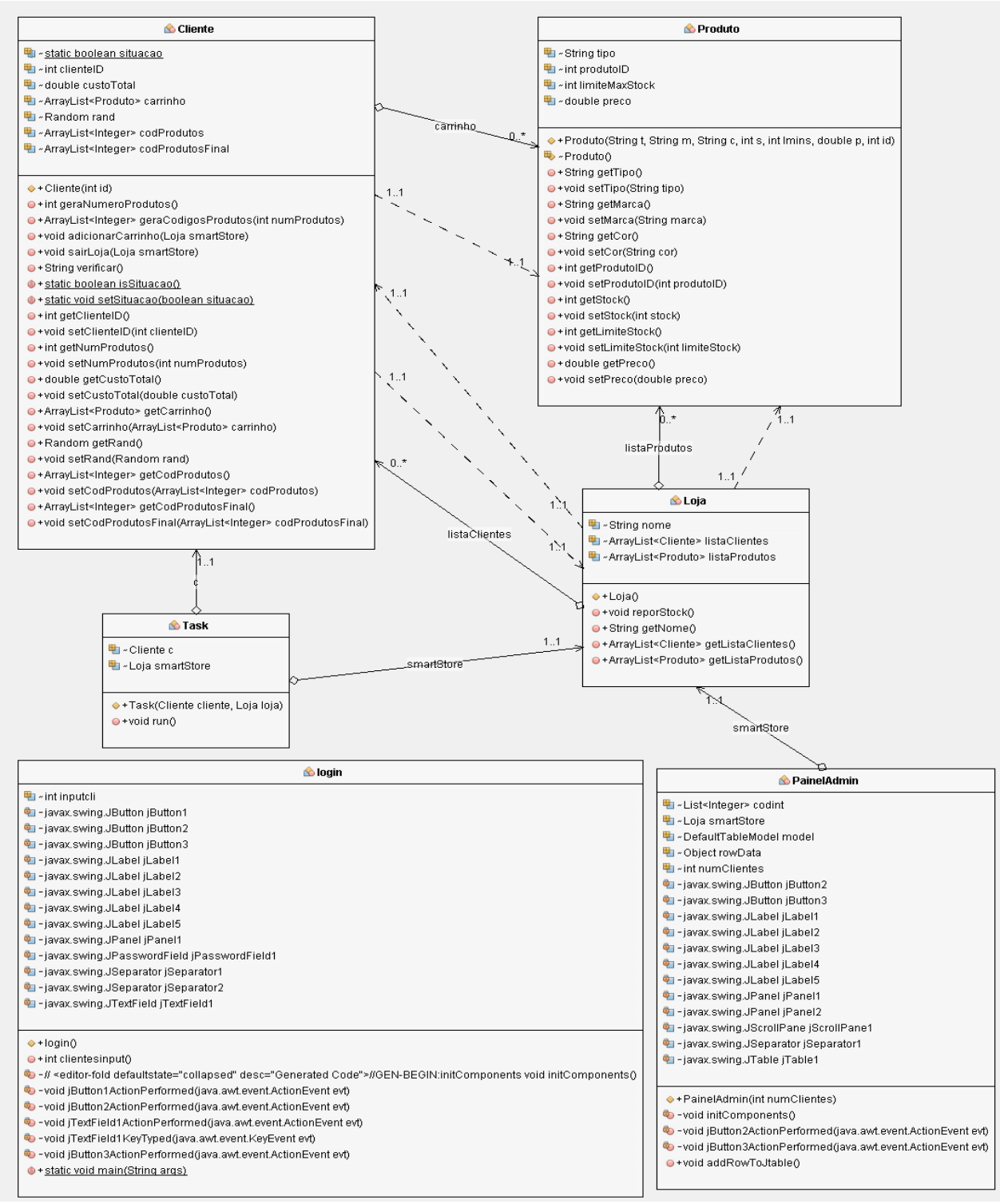
2.1 Diagrama de Casos de Uso



2.2 Diagrama de Atividades



2.3 Diagrama de Classes



3. Descrição das Opções Tomadas para os Componentes Introduzidos

3.1 Implementação de um GUI

Para o deployment do projeto em java tivemos quando

3.2 Exception-Handling

Para o deployment do projeto em java tivemos quando

3.3 Deployment do Projeto em Java

Para o deployment do projeto em java tivemos quando

4. Descrição Sumária do Código Implementado

4.1 Classes

4.1.1 Classe Cliente

Esta classe contém elementos como o carrinho de compras e o ID do cliente e possui ainda quatro métodos: um para gerar o número de produtos que vão ser comprados (\dos enquanto o cliente lá estiver; e por fim um método para sair da loja que, calcula também o valor total da compra (sairLoja).

4.1.2 Classe LerGravarDados

Esta classe contém dois métodos: um para gravar no ficheiro csv o id do cliente (gravarCliente) e outro para gravar os produtos comprados por esse mesmo cliente (gravarProduto).

4.1.3 Classe Loja

Esta classe contém as listas de clientes e o stock de produtos na loja e, contém ainda um método (reporStock) que, cada vez que um cliente sai da loja, verifica se algum produto atingiu a quantidade mínima de stock e, se for o caso, efetua uma encomenda para repor a quantidade de produtos em falta.

4.1.4 Classe Main

Esta classe, é a classe principal que, contém a criação dos objetos loja, clientes e produtos, assim como a sua colocação nas devidas listas (ArrayLists) e a chamada dos métodos e por fim, contém ainda a classe Task e a criação da ThreadPool e ExecutorService que criam as threads e lhes atribuem as suas devidas tasks (tarefas).

4.1.5 Classe Produto

Esta classe contém as informações (variáveis/atributos) de identificação e descrição do produto assim como os métodos getters e setters de cada um desses atributos para que estes possam ser guardados nos ficheiros CSV.

4.2 Implementação do Paradigma Concorrente

A implementação do paradigma concorrente foi feita através da utilização de um ExecuterService e a devida ThreadPool de tamanho fixo (tamanho igual ao input do utilizador), contendo esta uma thread para cada cliente, permitindo que a compra dos clientes (através da chamada dos respetivos métodos) seja feita em simultâneo.

5. Soluções Tomadas para os Componentes Introduzidos

5.1 Implementação de um GUI

Para o deployment do projeto em java tivemos quando

5.2 Exception-Handling

Para o deployment do projeto em java tivemos quando

5.3 Deployment do Projeto em Java

Para o deployment do projeto em java tivemos quando

6. Manual do Utilizador

O programa começa por perguntar o número de clientes para a simulação

```
run:
Bem vindo à Smart Store
Insira o número de clientes para a simulação:
```

Depois de inserido o número de clientes, o programa atribui aleatoriamente produtos para cada cliente e por fim, estes clientes, saem da loja.

```
O cliente 0 entrou na loja.
O cliente 1 entrou na loja.
[9, 10, 12]
[4, 1, 10, 15, 10, 7, 5]
Cliente 0  nº produtos: 3  cod Produtos: [9, 10, 12]
Cliente 1  nº produtos: 7  cod Produtos: [4, 1, 10, 15, 10, 7, 5]
Produto 9 adicionado ao carrinho do cliente 0
Produto 4 adicionado ao carrinho do cliente 1
Produto 10 adicionado ao carrinho do cliente 0
Produto 1 adicionado ao carrinho do cliente 1
Produto 12 adicionado ao carrinho do cliente 0
O cliente 0 saiu da loja
Produto 10 adicionado ao carrinho do cliente 1
Produto 10 devolvido.
Produto 15 adicionado ao carrinho do cliente 1
Produto 10 adicionado ao carrinho do cliente 1
Produto 7 adicionado ao carrinho do cliente 1
Produto 5 adicionado ao carrinho do cliente 1
O cliente 1 saiu da loja
O cliente 1 comprou os produtos: [4, 1, 15, 10, 7, 5]
O cliente 0 comprou os produtos: [9, 10, 12]
BUILD SUCCESSFUL (total time: 7 minutes 52 seconds)
```

7. Conclusão

7.1 Dificuldades Sentidas

A maior dificuldade sentida foi a incrementação das threads de acordo com o objetivo, ou seja, o uso dos executors para a distribuição de cada thread por cada cliente, analisando os documentos disponibilizados pelo docente e alguma pesquisa na internet, conseguimos ultrapassar esta dificuldade. Um aspeto a melhorar é corrigir a forma como são gravados os dados da fatura num ficheiro CSV, só conseguimos gravar numa linha o primeiro produto a ser comprado. Tivemos alguma dificuldade em elaborar o diagrama de classes.

7.2 Objetivos Atingidos

Em relação ao enunciado do projeto, quase todos os objetivos foram cumpridos, ou seja, o funcionamento total da aplicação está de acordo, as threads foram concebidas da maneira que o docente propôs e as faturas são guardadas num ficheiro CSV (incompletas).

7.3 Funções e Tarefas dos Elementos do Grupo

Para realizar o trabalho decidimos distribuir as tarefas de forma igual. A planificação do projeto foi realizada por todos os membros do grupo. O diagrama de casos de uso foi realizado por todos os elementos do grupo. O diagrama de atividades e de classes foi realizado por Diogo Palos. O código foi elaborado por todos. Miguel Nunes foi responsável pela implementação das threads. Bruno Saraiva foi responsável pela classe LerGravarDados. Diogo Palos foi responsável pela criação de restrições e correções no código. Ricardo Melo foi responsável pela construção das classes envolvendo a Loja. O relatório foi elaborado por todos. Cada tarefa foi revista por todos os membros do grupo de forma a corrigir qualquer erro.

8. Anexo

8.1 Classe Cliente

```
package projetopp;

import java.util.ArrayList;
import java.util.Random;

public class Cliente {

    int clienteID,numProdutos;
    double custoTotal;
    ArrayList<Produto> carrinho;
    Random rand = new Random();
    ArrayList<Integer> codProdutos = new ArrayList<>();   //contém os códigos dos produtos que vão ser colocados no carrinho
     ArrayList<Integer> codProdutosFinal = new ArrayList<>();   //contém os códigos dos produtos que vão ser comprados (após devoluções)

    public Cliente(int id){
        this.clienteID = id;
        carrinho = new ArrayList<>();
    }

    //gera número de produtos que o cliente vai comprar
    public int geraNumeroProdutos(){
        numProdutos = rand.nextInt(8);
        return numProdutos;
    }

    //gera os códigos dos produtos que o cliente vai comprar
    //gera um número de códigos igual ao int (parâmetro de entrada)
    public ArrayList<Integer> geraCodigosProdutos(int numProdutos){
        for(int i=0;i<numProdutos;i++){
            codProdutos.add((rand.nextInt(17)));  //17 = número de produtos que a loja tem (0-16)
        }
        return codProdutos;
    }

    //colocar produtos no carrinho se em stock e retirar do mesmo o produto adicionado
    //inclui chance de devolução
    public void adicionarCarrinho(Loja smartStore){
        for(int codProduto : codProdutos){
            if(smartStore.listaProdutos.get(codProduto).stock <= 0){
                System.out.println("Produto " + codProduto + "não está em stock.");
                break;
            }
            else{
```

```
                carrinho.add(smartStore.listaProdutos.get(codProduto));
                smartStore.listaProdutos.get(codProduto).stock -= 1;
                System.out.println("Produto " + codProduto + " adicionado ao carrinho do cliente " + this.clienteID);
                int chanceDevolucao = rand.nextInt(10);
                if(chanceDevolucao == 1){
                    carrinho.remove(smartStore.listaProdutos.get(codProduto));
                    smartStore.listaProdutos.get(codProduto).stock += 1;
                    System.out.println("Produto " + codProduto + " devolvido.");
                }
                else{
                    codProdutosFinal.add(smartStore.listaProdutos.get(codProduto).produtoID);
                }
            }
        }
    }

    //calcula custo total da compra e atualiza stock da loja
    public void sairLoja(Loja smartStore){
        for(Produto p : carrinho){
            custoTotal += p.preco;
        }
        smartStore.reporStock();
        System.out.println("O cliente " + this.clienteID + " saiu da loja");
        System.out.println("O cliente " + this.clienteID + " comprou os produtos: " + this.codProdutosFinal);
    }

}
```

8.2 Classe LerGravarDados

```
package projetopp;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class LerGravarDados {

    BufferedReader br;
    BufferedWriter bw;
    String linha;
    String delimitador;

    public LerGravarDados() {  // construtor
        br = null;
        linha = "";
        delimitador = ",";// ler ficheiros com dados separados por vírgulas
```

```
}

/*
public ArrayList<Produto> lerProduto(String fich)
{

   ArrayList<Produto> produto = new ArrayList<>();
   try {

      br = new BufferedReader(new FileReader(fich));
      while ((linha = br.readLine()) != null) {
         // usa vírgula como separador
         String[] dados = linha.split(delimitador);

         //como buscar os dados????

      }

   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   } finally {
      if (br != null) {
         try {
            br.close();
         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   }
   return produto;
}
*/


//guardar dados do produto
public void gravarProduto(String fich, Produto p)
{

   String s;
   //Produto p;
   try {
      bw = new BufferedWriter(new FileWriter(fich));
         s = p.tipo + ", " + p.marca + ", " + p.cor + ", " + p.stock + ", " + p.limiteMinStock + ", " + p.preco + ", " + p.produtoID;
         bw.write(s);
         bw.newLine();
```

```
    } catch (IOException e) {
       e.printStackTrace();
    } finally {
          try {
             bw.close();
          } catch (IOException e) {
             e.printStackTrace();
          }
    }

}


   //guardar dados do cliente
   public void gravarCliente(String fich, int cID) throws IOException{

      String id = "Cliente ";

      try {
         bw = new BufferedWriter(new FileWriter(fich));
         id += String.valueOf(cID);
            bw.write(id);
            bw.newLine();
         }

      catch (IOException e) {
         e.printStackTrace();
      }
      finally {
            try {
               bw.close();
            } catch (IOException e) {
               e.printStackTrace();
            }
      }
   }
```

8.3 Classe Loja

```
package projetopp;

import java.util.ArrayList;

public class Loja {

   String nome = "Smart Store";
   ArrayList<Cliente> listaClientes;
   ArrayList<Produto> listaProdutos;
```

```
    public Loja(){
        this.listaClientes = new ArrayList<>();
        this.listaProdutos = new ArrayList<>();
    }

    //verifica se os valores minimos de stock foram atingidos e repõe se necessário
    public void reporStock(){
        for(Produto p : listaProdutos){
            if(p.stock <= p.limiteMinStock){
                int encomenda = p.limiteMaxStock - p.stock;
                p.stock += encomenda;
                System.out.println("Foram encomendadas " + encomenda + " unidades do produto " + p.tipo + " " + p.marca + " " +
p.cor);
            }
        }
    }
```

8.4 Classe Main

```
package projetopp;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class Main {

    public static void main(String[] args) throws InterruptedException, IOException {

        Loja smartStore = new Loja();

        smartStore.listaProdutos.add(new Produto("T-shirt","FILA","azul",10,4,14.99,0));
        smartStore.listaProdutos.add(new Produto("T-shirt","Nike","Branco",9,3,19.99,1));
        smartStore.listaProdutos.add(new Produto("T-shirt","Nike","Preto",13,4,19.99,2));
        smartStore.listaProdutos.add(new Produto("T-shirt","Nike","Vermelho",20,4,14.99,3));
        smartStore.listaProdutos.add(new Produto("Calções","Nike","Azuis",18,4,13.99,4));
        smartStore.listaProdutos.add(new Produto("Calções","Adidas","Roxo",14,3,14.99,5));
        smartStore.listaProdutos.add(new Produto("T-shirt","Adidas","Branco",12,4,14.99,6));
        smartStore.listaProdutos.add(new Produto("T-shirt","Benfica","Vermelho",11,3,39.99,7));
        smartStore.listaProdutos.add(new Produto("T-shirt","Sporting","Verde",14,4,34.99,8));
        smartStore.listaProdutos.add(new Produto("T-shirt","Porto","Azul",23,4,39.99,9));
        smartStore.listaProdutos.add(new Produto("T-shirt","Braga","Vermelho",11,4,34.99,10));
        smartStore.listaProdutos.add(new Produto("Meias","Nike","Branco",11,4,5.99,11));
        smartStore.listaProdutos.add(new Produto("Meias","Adidas","Preto",11,4,4.99,12));
        smartStore.listaProdutos.add(new Produto("Bebida","RedBull","Vermelho",20,5,1.20,13));
        smartStore.listaProdutos.add(new Produto("Bebida","RedBull","Azul",25,5,1.10,14));
```

```
    smartStore.listaProdutos.add(new Produto("Suplemento","Whey Protein Prozis","Branco",15,5,25.99,15));
    smartStore.listaProdutos.add(new Produto("T-shirt","Quechua","Azul",15,5,4.99,16));

    //recolha de dados (número de clientes)
    Scanner input = new Scanner(System.in);
    System.out.println("Bem vindo à Smart Store \nInsira o número de clientes para a simulação: ");
    int numClientes = input.nextInt();

    //criação da threadPool cujo tamanho é igual ao número de clientes inserido
    ExecutorService service = Executors.newFixedThreadPool(numClientes);

    //criar clientes e adicionar à listaClientes da loja
    for(int i=0;i<numClientes;i++){
        smartStore.listaClientes.add(new Cliente(i));
        System.out.println("O cliente " + i + " entrou na loja.");
    }

    //correr task para cada cliente na loja
    for(Cliente c : smartStore.listaClientes){
        service.execute(new Task(c,smartStore));   //criar uma nova task a ser corrida por uma thread
    }

    LerGravarDados lgd = new LerGravarDados();
    File ficheiro = new File("faturas.csv");

    for(Cliente c : smartStore.listaClientes){
        lgd.gravarCliente("faturas.csv", c.clienteID);
        for(Produto p : c.carrinho){
            lgd.gravarProduto("faturas.csv", p);
        }
    }


    //terminar ExecutorService
    service.shutdown();

}

//classe task (tarefas das threads)
static class Task implements Runnable{

    Cliente c;
    Loja smartStore;

    public Task(Cliente cliente, Loja loja){
        this.c = cliente;
        this.smartStore = loja;
    }

    @Override
```

```
        public void run(){
            System.out.println(c.geraCodigosProdutos(c.geraNumeroProdutos()));
            System.out.println("Cliente " + c.clienteID + "  nº produtos: " + c.numProdutos + "  cod Produtos: " + c.codProdutos);
            c.adicionarCarrinho(smartStore);
            c.sairLoja(smartStore);
        }
    }

}
```

8.5 Classe Produto

```
package projetopp;

public class Produto {

    String tipo,marca,cor;
    int produtoID,stock,limiteMinStock;
    int limiteMaxStock = 25;
    double preco;

    public Produto(String t, String m, String c, int s, int lmins, double p, int id){
        this.tipo = t;
        this.marca = m;
        this.cor = c;
        this.stock = s;
        this.limiteMinStock = lmins;
        this.preco = p;
        this.produtoID = id;
    }

    Produto() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }
```

```
    public String getCor() {
        return cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public int getProdutoID() {
        return produtoID;
    }

    public void setProdutoID(int produtoID) {
        this.produtoID = produtoID;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }

    public int getLimiteStock() {
        return limiteMinStock;
    }

    public void setLimiteStock(int limiteStock) {
        this.limiteMinStock = limiteStock;
    }

    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }


}
```