

Trabalho **GRAU B**

ARTHUR KELLERMANN, ARTHUR RAMBOR,
EDUARDO DOS SANTOS, GIOVANI DE
SOUZA, ULISSES



Proposta

Montar um programa que simule o controle de estoque de um E-commerce (Ex.: Amazon e Mercado Livre).

Parâmetros

Para implementação desse programa seguimos as regras de:

- Vender 4 tipos distintos de produtos previamente determinados (Livros, Eletrodomésticos, Roupas, Eletrônicos);
- Montar o sistema do estoque e juntamente um esquema de carrinho de compras (funcional e não complexo);
- Abranger todos os métodos implementados com testes unitários JUnit (mínimo 2 testes por método);

Tópicos do Trabalho

APRESENTAÇÃO

1

2

3

4

5

6

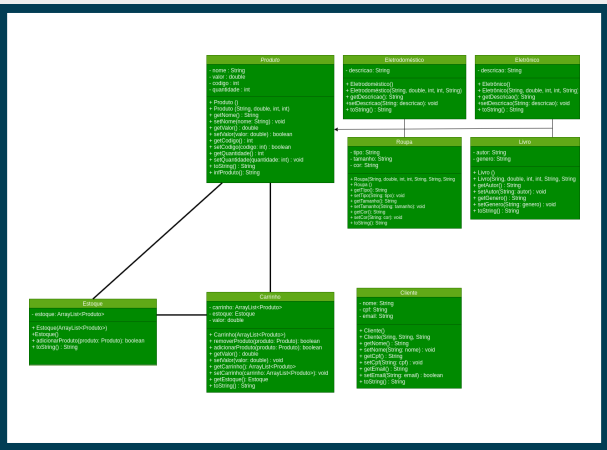
7

8

9

10

DIAGRAMA
UML



CÓDIGO

```
try {  
  
    String dirPath = "./Notinha";  
    File diretorio = new File(dirPath);  
  
    if (diretorio.mkdirs()) {  
        System.out.println("Pasta da notinha criada em: " + diretorio.getAbsolutePath());  
    }  
}
```

Diagrama UML

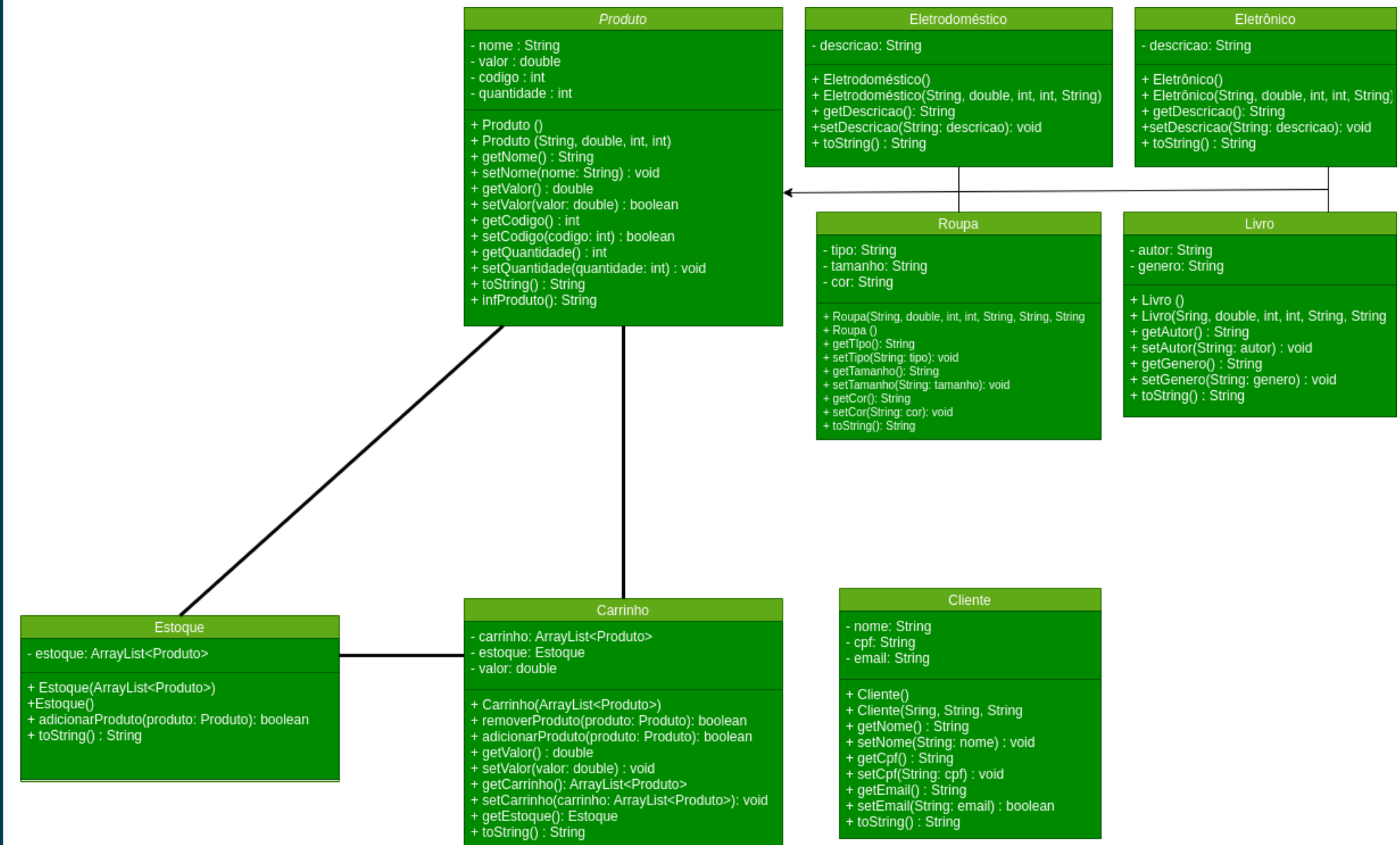
Diagrama padrão UML com classes detalhadas e indicadas as respectivas associações entre classe.

Superclasse: Produto

- Subclasses: Roupa, Eletrodoméstico, Livro e Eletrônico;

Classes compostas:

- Carrinho (contendo um ArrayList de produtos);
- Estoque (contendo um ArrayList de produtos);



Código

Produto

Como já mencionado, a classe produto é uma superclasse generalista que vai conter informações necessárias para as subcategorias de produto.

Nessa classe, além de possuir os métodos tradicionais (getters/setters) foi necessário realizar um filtro no momento de configurar o código e o valor, ou seja, proibindo valores abaixo de zero.

Principal método desta classe é o "public String infProduto()" que passa as informações de maneira simplificada sobre o produto, este método é utilizado no "main" para informar as características dos produtos selecionados (carrinho) de maneira resumida, outra utilização desse resumo de informações é no momento de imprimir uma nota fiscal.

```
public abstract class Produto {

    private String nome;
    private double valor;
    public int codigo;
    public int quantidade;

    /**
     * Produto é qualquer produto presente na loja.
     * @param nome É o nome de determinado produto.
     * @param valor O valor de determinado produto.
     * @param codigo Cada produto terá seu código específico.
     * @param quantidade Se refere a quantidade desse produto presente no estoque da empresa.
     */

    public Produto(String nome, double valor, int codigo, int quantidade) {
        this.nome = nome;
        this.valor = valor;
        this.codigo = codigo;
        this.quantidade = quantidade;
    }

    public Produto() {
    }

    public String getNome() { ...

    public void setNome(String nome) { ...

    public double getValor() { ...

    public boolean setValor(double valor) { ...

    public int getCodigo() { ...
    ...
}
```


Código

Subclasses

Classes derivadas da classe mãe Produto estendendo todos as características com o método "super" e criando novos conforme necessidade individual (Ex.: Livro: String gênero).

```
package grau8;

public class Livro extends Produto {

    private String autor;
    private String genero;

    public Livro() {
        super();
    }

    /**
     * Além dos parametros padrões, ele vai ter os atributos autor e genero.
     * @param nome
     * @param valor
     * @param codigo
     * @param quantidade
     * @param autor Se refere ao autor do livro.
     * @param genero Se refere ao gênero do livro.
     */
    public Livro(String nome, double valor, int codigo, int quantidade, String autor, String genero) {
        super(nome, valor, codigo, quantidade);
        this.autor = autor;
        this.genero = genero;
    }

    > public String getAutor() { ...
    > public void setAutor(String autor) { ...
    > public String getGenero() { ...
    > public void setGenero(String genero) { ...

    @Override
    public String toString() {
        return "\nLivro: " + super.toString() + " - Autor= " + autor + " - Gênero= " + genero;
    }
}
```

```
package grau8;

public class Roupa extends Produto {

    private String tipo;
    private String tamanho;
    private String cor;

    /**
     * Além dos parametros padrões, ele vai ter os atributos tipo, tamanho e cor.
     * @param nome
     * @param valor
     * @param codigo
     * @param quantidade
     * @param tipo Se refere ao tipo da roupa.
     * @param tamanho Se refere ao tamanho da roupa.
     * @param cor Se refere a cor da roupa.
     */
    public Roupa(String nome, double valor, int codigo, int quantidade, String tipo, String tamanho, String cor) {
        super(nome, valor, codigo, quantidade);
        this.tipo = tipo;
        this.tamanho = tamanho;
        this.cor = cor;
    }

    > public Roupa() { ...

    > public String getTipo() { ...
    > public void setTipo(String tipo) { ...
    > public String getTamanho() { ...
    > public void setTamanho(String tamanho) { ...
    > public String getCor() { ...
    > public void setCor(String cor) { ...
    > @Override
    > public String toString() { ...
}
```

```
package grau8;

public class Eletrodomestico extends Produto {

    private String descricao;

    /**
     * Além dos parametros padrões, ele vai ter o atributo descrição
     * @param nome
     * @param valor
     * @param codigo
     * @param quantidade
     * @param descricao Pode ser a respeito da marca ou características do produto.
     */
    public Eletrodomestico(String nome, double valor, int codigo, int quantidade, String descricao) {
        super(nome, valor, codigo, quantidade);
        this.descricao = descricao;
    }

    /**
     * Constructor padrao
     */
    > public Eletrodomestico() { ...

    /**#endregion

    /**#region get e set

    > public String getDescricao() { ...

    > public void setDescricao(String descricao) { ...

    @Override
    > public String toString() { ...

    /**#endregion
}
```

```
package grau8;

public class Eletrônico extends Produto {

    private String descricao;

    /**
     * Além dos parametros padrões, ele vai ter o atributo descrição
     * @param nome
     * @param valor
     * @param codigo
     * @param quantidade
     * @param descricao Pode ser a respeito da marca ou características do produto.
     */
    public Eletrônico(String nome, double valor, int codigo, int quantidade, String descricao) {
        super(nome, valor, codigo, quantidade);
        this.descricao = descricao;
    }

    > public Eletrônico() { ...

    > public String getDescricao() { ...

    > public void setDescricao(String descricao) { ...

    @Override
    > public String toString() { ...
}
```

Código

Estoque

Estoque foi definido como um ArrayList de produtos.

Principal método: adicionarProduto.

Método de adição de produtos, nele contém a verificação de quantidade de produtos, permitindo somente adicionar valores superiores a zero.

```
import java.util.ArrayList;

public class Estoque {

    private ArrayList<Produto> estoque = new ArrayList<Produto>();

    public Estoque() {
    }

    /**
     * O estoque é composto de um ArrayList de produtos
     * @param estoque
     */
    public Estoque(ArrayList<Produto> estoque) {
        this.estoque = new ArrayList<Produto>();
    }

    public Produto localizar(int codigo) {
        try {
            for (Produto p : estoque) {
                if (p.getCodigo() == codigo) {
                    return p;
                }
            }
            return null;
        } catch (Exception e) {
            return null;
        }
    }

    /**
     * É um método que adiciona produtos no estoque
     * @param produto Ele confere se a quantidade do produto adicionado é maior que 0, e em seguida adiciona ao estoque
     * @return
     */
    public boolean adicionarProduto(Produto produto) {
        try {
            if (produto.getQuantidade() <= 0) {
                return false;
            }

            return estoque.add(produto);
        } catch (Exception e) {
            return false;
        }
    }
}
```

Código

Carrinho

Carrinho foi definido como um ArrayList de produtos e tem como atributo principal o valor total do carrinho.

Principais métodos: removerProduto e o adicionarProduto.

Em suma, utilizam métodos presentes na classe ArrayList, ou seja, .add e .remove. Único trabalho que foi feito para diferenciar e ter uma finalidade diferente da tradicional, foi adicionar o produto de volta ao estoque quando ele é retirado do carrinho e quando ele é adicionado no carrinho ele desconta do estoque.

```
package grauB;
import java.util.ArrayList;

public class Carrinho {

    private ArrayList<Produto> carrinho = new ArrayList<Produto>();
    private Estoque estoque ;
    private double valor;

    public Carrinho() {
    }

    /**
     * O construtor é composto de um ArrayList de produtos.
     * @param carrinho É o nome dado ao ArrayList de produtos.
     */
    public Carrinho(ArrayList<Produto> carrinho) {
        this.carrinho = carrinho;
    }

    /**
     * É um método na qual remove um produto do carrinho.
     * @param produto Ele confere se o produto está presente no carrinho, caso sim, remove o mesmo além de diminuir o valor do carrinho,
     * @return
     */
    public boolean removerProduto(Produto produto) {
        try {
            if (carrinho.contains(produto)) {
                carrinho.remove(produto);
                valor -= produto.getValor();
                System.out.println("Produto: " + produto.getNome() + " removido do Carrinho.");
                produto.setQuantidade(produto.getQuantidade() + 1);
                return true;
            } else {
                System.out.println("Produto: " + produto.getNome() + " não encontra-se no carrinho.");
                return false;
            }
        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```


Código

Cliente

Classe criada para armazenar informações sobre o cliente. Vale ressaltar que as informações do cliente são solicitadas via teclado no início do "main" e depois são impressas na nota. Além dos getters e setters presentes na classe, acontece uma verificação se o email contém "@" no setEmail.

```
package grauB;

public class Cliente {

    private String nome;
    private String cpf;
    private String email;
    private String senha;

    // #region
    public Cliente() {
    }

    /**
     * Dados importantes para se a ligação entre o usuário e a loja.
     *
     * @param nome Nome do cliente.
     * @param cpf CPF do cliente.
     * @param email Email do cliente.
     */
    public Cliente(String nome, String cpf, String email, String senha) { ...

    // #endregion

    // #region get e set

    public String getNome() { ...

    public void setNome(String nome) { ...

    public String getCpf() { ...

    public void setCpf(String cpf) { ...

    public String getEmail() { ...

    public String getSenha() { ...

    public void setSenha(String senha) { ...

    /** ...

    public boolean confereSenha(String verificaSenha) { ...
    public boolean setEmail(String email) { ...

    @Override
    public String toString() { ...

    // #endregion
}
```

Código

Admin

Classe criada para armazenar informações sobre o o administrador da loja.

Classe herdeira de Cliente onde o administrador do estoque referente consegue controlar os produtos dispostos nesse, seja adicionando quantidade a produtos existentes ou criando novos produtos.

```
package grauB;

public class Admin extends Cliente{

    public Admin() {
        super();
    }

    public Admin(String nome, String cpf, String email, String senha) {
        super(nome, cpf, email, senha);
    }

    @Override
    public String toString() {
        return "Admin [getNome()=" + super.toString();
    }

}
```

Código

Main

Parte onde implementou-se o menu de escolha para o tipo de Conta, no início do código foram instanciados os produtos da classe abstrata Produto. Também, foram definidas os objetos empresa e usuário para preenchimento do Estoque da loja fictícia.

```
Run | Debug
public static void main(String[] args) {

    Scanner in = new Scanner(System.in);

    Produto camisaG = new Roupa(nome:"Camisa G", valor:60, codigo:101, quantidade:4, tipo:"Tecido", tamanho:"G", c
    Produto camisaM = new Roupa(nome:"Camisa M", valor:60, codigo:102, quantidade:16, tipo:"Tecido", tamanho:"M",
    Produto camisaP = new Roupa(nome:"Camisa P", valor:60, codigo:103, quantidade:10, tipo:"Tecido", tamanho:"P",
    Produto pp = new Livro(nome:"Pequeno Príncipe", valor:40, codigo:201, quantidade:2, autor:"Antoine de Saint-Ex
    Produto crepusculo = new Livro(nome:"Crepúsculo", valor:50, codigo:202, quantidade:5, autor:"Stephenie Meyer",
    Produto dk = new Livro(nome:"Dom Quixote", valor:30, codigo:203, quantidade:3, autor:"Miguel de Cervantes", ge
    Produto PC = new Eletrônico(nome:"PC Gamer", valor:3300, codigo:301, quantidade:3, descricao:"AMD Ryzen 5 4600
    Produto iphone = new Eletrônico(nome:"Iphone 14", valor:8000, codigo:302, quantidade:10, descricao:"Apple iPho
    Produto monitor = new Eletrônico(nome:"Monitor 23,5\"", valor:1400, codigo:303, quantidade:5, descricao:"Samsun
    Produto freezer = new Eletrodoméstico(nome:"Freezer", valor:3500, codigo:401, quantidade:4, descricao:"Horizon
    Produto geladeira = new Eletrodoméstico(nome:"Geladeira", valor:2800, codigo:402, quantidade:8, descricao:"Ele
    Produto fogao = new Eletrodoméstico(nome:"Fogão", valor:2000, codigo:403, quantidade:8, descricao:"Brastemp 4

    Estoque empresa = new Estoque(new ArrayList());

    empresa.adicionarProduto(camisaG);
    empresa.adicionarProduto(camisaM);
    empresa.adicionarProduto(camisaP);
    empresa.adicionarProduto(pp);
    empresa.adicionarProduto(crepusculo);
    empresa.adicionarProduto(dk);
    empresa.adicionarProduto(PC);
    empresa.adicionarProduto(iphone);
    empresa.adicionarProduto(monитор);
    empresa.adicionarProduto(freezer);
    empresa.adicionarProduto(fogao);
    empresa.adicionarProduto(geladeira);
```

Código

Main

Defina o tipo da sua conta: ADMIN ou CLIENTE. Cada tipo de conta tem funcionalidades diferentes:

- Admin: Localizar Produto, Adicionar Produto, Ver estoque
- Cliente: Adicionar ao carrinho os produtos e imprimir a notinha.

```
System.out.println(x:"Defina o tipo da sua conta (1/2)");
System.out.println(x:"1 - Admin\n2 - Cliente");
int tipoConta = in.nextInt();

if (tipoConta == 1) {
    Admin adm = Main.registerAdm();
    System.out.println("Olá " + adm.getNome());
    int opcAdm;

    do {
        System.out.println(x:"1 - Localizar produto\n2 - Adicionar produto\n3 - Ver estoque\n0 - Sair");
        opcAdm = in.nextInt();
        in.nextLine(); // Consumir a quebra de linha pendente

        switch (opcAdm) {
            case 1:
                System.out.print(s:"Digite o código do produto: ");
                int cod = in.nextInt();
                in.nextLine(); // Consumir a quebra de linha pendente

                if (empresa.localizar(cod) == null) {
                    System.out.println(x:"Produto não existe!");
                } else {
                    System.out.println(empresa.localizar(cod));
                }
                break;
            case 2:
                System.out.print(s:"Nome do produto:");
                String nome = in.nextLine();
                System.out.print(s:"Valor: ");
                double valor = in.nextDouble();
                System.out.print(s:"Código: ");
                int codigo = in.nextInt();
                System.out.print(s:"Quantidade: ");
                int quant = in.nextInt();

                System.out.println(x:"Digite o tipo do produto");
                System.out.println(x:"1 - Roupa\n2 - Livro\n3 - Eletrônico\n4 - Eletrodoméstico");
                int produto = in.nextInt();
                in.nextLine(); // Consumir a quebra de linha pendente
```

Código

Main

Métodos registerAdm e registerCliente são executados a partir da escolha do tipo de conta. Ambos retornam um objeto instanciado da classe Admin e outro da classe Cliente, respectivamente.

```
public static Admin registerAdm() {

    Scanner in = new Scanner(System.in);

    Admin adm = new Admin();
    System.out.println();
    System.out.println(x:"--CADASTRO ADMIN--");
    System.out.print(s:"Digite seu nome: ");
    adm.setNome(in.nextLine());
    System.out.print(s:"Digite seu CPF: ");
    adm.setCpf(in.nextLine());
    do {
        System.out.print(s:"Digite seu email: ");
        adm.setEmail(in.nextLine());
    } while (adm.getEmail() == null);

    System.out.print(s:"Digite a senha: ");
    adm.setSenha(in.nextLine());

    return adm;

}

public static Cliente registerCliente() {
    Scanner in = new Scanner(System.in);

    Cliente cliente = new Cliente();
    System.out.println();
    System.out.println(x:"--CADASTRO CLIENTE--");
    System.out.print(s:"Digite seu nome: ");
    cliente.setNome(in.nextLine());
    System.out.print(s:"Digite seu CPF: ");
    cliente.setCpf(in.nextLine());
    do {
        System.out.print(s:"Digite seu email: ");
        cliente.setEmail(in.nextLine());
    } while (cliente.getEmail() == null);

    System.out.print(s:"Digite a senha: ");
    cliente.setSenha(in.nextLine());

    return cliente;

}
```


Código

Main

Além do "public static void main(String[] args)", existe o "public static boolean notinha".

No método principal é realizado um menu com interações com o cliente, onde serão selecionados por ele quais produtos deseja do estoque, adicionando ou não no carrinho e, no momento do pagamento é chamado o método "notinha".

Nesse método, é impresso um arquivo .txt com BufferedWriter() com informações da compra realizada pelo cliente, simulando uma nota fiscal.

```
1 package grau8;
2
3 import java.io.BufferedWriter;
4 import java.io.File;
5 import java.io.FileWriter;
6 import java.io.IOException;
7 import java.util.ArrayList;
8 import java.util.Scanner;
9
10 public class Main {
11
12     /**
13      * Função para imprimir nota(arquivo txt) na pasta local do projeto
14      *
15      * @param cpfNaNota 1 = sim, 0 = não
16      * @param cpf recebe o cpf do cliente
17      * @param carrinho recebe o carrinho com a ArrayList de produtos
18      * @return true para certo e false para errado
19      */
20
21     public static boolean notinha(int cpfNaNota, String cpf, Carrinho carrinho) {
22         try {
23
24             String dirPath = "./Notinha";
25             File diretorio = new File(dirPath);
26
27             if (diretorio.mkdirs()) {
28                 System.out.println("Pasta da notinha criada em: " + diretorio.getAbsolutePath());
29             }
30
31             String path = diretorio.getAbsolutePath() + "\\notinha.txt";
32
33             try (BufferedWriter bw = new BufferedWriter(new FileWriter(path))) {
34
35                 System.out.println("Notinha impressa com sucesso!");
36
37                 if (cpfNaNota == 1) {
38                     bw.write("Cliente de CPF: " + cpf);
39                     bw.write(" | Produtos comprados: " + "\n");
40
41                     for (Produto produto : carrinho.getCarrinho()) {
42                         if (produto instanceof Produto) {
43                             Produto product = (Produto) produto;
44                             bw.write(product.infProduto() + "\n");
45                         }
46                     }
47
48                     bw.write("Valor total: R$ " + String.format(format: "%.2f", carrinho.getValor()));
49
50                 } else {
51                     bw.write("Produtos comprados: " + "\n");
52                     for (Produto produto : carrinho.getCarrinho()) {
53                         if (produto instanceof Produto) {
54                             Produto product = (Produto) produto;
55                             bw.write(product.infProduto() + "\n");
56                         }
57                     }
58                     bw.write("Valor total: R$ " + String.format(format: "%.2f", carrinho.getValor()));
59                 }
60
61             } catch (IOException e) {
62                 e.printStackTrace();
63             }
64             return true;
65
66         } catch (Exception e) {
67             System.out.println(e.toString());
68             return false;
69         }
70     }
71 }
```

Menus

Main

```
if (tipoConta == 2) {
    Cliente cliente = Main.registerCliente();

    System.out.println();
    System.out.println(x:"Bem vindo ao Mercado Livre, segue abaixo os produtos disponíveis em nosso estoque: ");

    System.out.println(empresa);

    Carrinho usuario = new Carrinho(new ArrayList<>());

    System.out.println(x:" ");
    System.out.println("Carrinho do " + cliente.getNome());

    int opcao = 1000;
    do {
        System.out.print(
            s:"Digite o código do produto que você quer adicionar ou digite 0 para finalizar o carrinho: ");
        opcao = in.nextInt();

        switch (opcao) {
            case 101:
                usuario.adicionarProduto(camisaG);
                break;
            case 102:
                usuario.adicionarProduto(camisaM);
                break;
            case 103:
                usuario.adicionarProduto(camisaP);
                break;
            case 201:
                usuario.adicionarProduto(pp);
                break;
            case 202:
                usuario.adicionarProduto(crepusculo);
```

cliente

```
do {
    System.out.println(x:"1 - Localizar produto\n2 - Adicionar produto\n3 - Ver estoque\n0 - Sair");
    opcAdm = in.nextInt();
    in.nextLine(); // Consumir a quebra de linha pendente

    switch (opcAdm) {
        case 1:
            System.out.print(s:"Digite o código do produto: ");
            int cod = in.nextInt();
            in.nextLine(); // Consumir a quebra de linha pendente

            if (empresa.localizar(cod) == null) {
                System.out.println(x:"Produto não existe!");
            } else {
                System.out.println(empresa.localizar(cod));
            }
            break;
        case 2:
            System.out.print(s:"Nome do produto:");
            String nome = in.nextLine();
            System.out.print(s:"Valor: ");
            double valor = in.nextDouble();
            System.out.print(s:"Código: ");
            int codigo = in.nextInt();
            System.out.print(s:"Quantidade: ");
            int quant = in.nextInt();

            System.out.println(x:"Digite o tipo do produto");
            System.out.println(x:"1 - Roupa\n2 - Livro\n3 - Eletrônico\n4 - Eletrodoméstico");
            int produto = in.nextInt();
            in.nextLine(); // Consumir a quebra de linha pendente

            switch (produto) {
                case 1:
                    System.out.print(s:"Tipo: ");
                    String tipo = in.nextLine();
                    System.out.print(s:"Tamanho");
                    String tamanho = in.nextLine();
                    System.out.print(s:"Cor: ");
                    String cor = in.nextLine();

                    Produto roupa = new Roupa(nome, valor, codigo, quant, tipo, tamanho, cor);
                    empresa.adicionarProduto(roupa);
                    break;
                case 2:
                    System.out.print(s:"Autor: ");
                    String autor = in.nextLine();
```

adm

Código

Main

Menu final onde os produtos escolhidos são exibidos, o valor final e a escolha de cpf na nota.
Usuário terá que confirmar a senha pelo método confereSenha() para realizar a compra

```
System.out.println(x:"\nProdutos selecionados:");
for (Produto produto : usuario.getCarrinho()) {
    if (produto instanceof Produto) {
        Produto product = (Produto) produto;
        System.out.println(product.infProduto());
    }
}
System.out.println("Valor Total: R$ " + String.format(format:"%.2f", usuario.getValor()));

in.nextLine();
System.out.print(s:"Digite sua senha novamente para confirmar a compra: ");

String senha = in.nextLine();

do {
    if (cliente.confereSenha(senha) == false) {
        System.out.print(s:"Senha incorreta!");
    }
} while (cliente.confereSenha(senha) == false);

System.out.println(x:"CPF na nota? (s/n)");
char cpfNaNota = in.next().charAt(index:0);

if (cpfNaNota == 's' || cpfNaNota == 'S') {
    notinha(cpfNaNota:1, cliente.getCpf(), usuario);
} else {
    notinha(cpfNaNota:0, cpf:null, usuario);
}

System.out.println("Obrigado pela confiança, " + cliente.getNome());
in.close();
}
}
```

Código

Testes JUnit

Testes dos métodos de classe da classe Produto.

Testes confirmando a validade dos códigos do produto somente quando código for positivo. E outros testes com intenção de confirmar que o setValor aceita apenas números positivos.

```
/**
 * verifica se a condicao esta certa inserindo um valor negativo e depois um positivo
 * @author ulisses953
 */
@Test
void testSetCodigo() {
    var livro = new Livro();

    assertFalse(livro.setCodigo(-10));

    assertTrue(livro.setCodigo(codigo:10));
}

/**
 * teste referente ao valor do produto
 * @author ulisses953
 */
@Test
void testGetValor(){
    var livro = new Livro();

    assertFalse(livro.setValor(-10));

    assertTrue(livro.setValor(valor:10));
}
```

Código

Testes JUnit

Testes dos métodos de classe da classe Estoque.

Testes confirmando que ao adicionar um produto válido ao Estoque, o método retorna "true" e, caso contrário, retorna "false".

Teste de localizar produto no estoque, retornando um produto caso exista na ArrayList de produtos ou retornando null.

```
/**
 * Teste que adiciona produto ao estoque e confirma retorno true ao adicionar
 */
@Test
void testAdiciona1() {
    Estoque est = new Estoque();
    Livro liv = new Livro(nome:"Crepúsculo", valor:30.00d, codigo:201, quantidade:4, autor:"Stephanie Meyer", genero:"Romance");
    assertTrue(est.adicionarProduto(liv));
}

/**
 * Teste que adiciona produto ao estoque e confirma retorno false ao não
 * adicionar
 */
@Test
void testAdiciona2() {
    Estoque est = new Estoque();
    Livro liv = new Livro(nome:"Crepúsculo", valor:30.00d, codigo:201, quantidade:0, autor:"Stephanie Meyer", genero:"Romance");

    assertFalse(est.adicionarProduto(liv));
}

@Test
void testAdicionarProduto() {
    var est = new Estoque();
    var p = new Eletrodomestico();
    p.setQuantidade(quantidade:10);

    assertTrue(est.adicionarProduto(p));
    p.setQuantidade(-10);
    assertFalse(est.adicionarProduto(p));
}

@Test
void testLocalizar() {
    var a = new Estoque();

    a.adicionarProduto(new Eletrodomestico(nome:null, valor:100, codigo:1, quantidade:10, descricao:null));

    assertNotNull(a.localizar(codigo:1));

    assertNull(a.localizar(codigo:0));
}
```


Código

Testes JUnit

Testes dos métodos da classe Carrinho.

Testes confirmando que ao adicionar um produto válido ao Carrinho, o método retorna "true" e, caso contrário, retorna "false".

```
void testAdiciona(){
    Estoque est = new Estoque();
    Livro liv = new Livro(nome:"Crepúsculo", valor:30.00d, codigo:201, quantidade:1, autor:"Stephanie Meyer", genero:"Romance");
    est.adicionarProduto(liv);

    Carrinho carrinho = new Carrinho(new ArrayList<>());
    assertTrue(carrinho.adicionarProduto(liv));
    assertFalse(carrinho.adicionarProduto(liv));
}

/**
 * Teste que remove produto ao carrinho e confirma retorno true ao remover
 * Teste que remove produto ao carrinho e confirma retorno false ao não remover
 */
@Test
void testRemove() {

    Estoque est = new Estoque();
    Livro liv = new Livro(nome:"Crepúsculo", valor:30.00d, codigo:201, quantidade:1, autor:"Stephanie Meyer", genero:"Romance");
    est.adicionarProduto(liv);

    Carrinho carrinho = new Carrinho(new ArrayList<>());
    carrinho.adicionarProduto(liv);
    assertTrue(carrinho.removerProduto(liv));
    assertFalse(carrinho.removerProduto(liv));
}
```

Código

Testes JUnit

Testes dos métodos de classe da classe Cliente.

Validação de email e senha.

Testes confirmando que ao adicionar um e-mail e uma senha válida ao perfil de cliente, o método retorna "true" e, caso contrário, retorna "false".

```
package test;

import grauB.*;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.jupiter.api.Test;

public class ClienteTest {

    /**
     * @autor ulisses953
     */
    @Test
    void testSetEmail() {
        Cliente a = new Cliente();
        assertFalse(a.setEmail("ulisseskranzdamotta"));
        assertTrue(a.setEmail("ulisseskranzdamotta@gmail.com"));
    }

    /**
     * @autor ulisses953
     */
    @Test
    void testConfereSenha() {
        var a = new Cliente();

        a.setSenha("123");
        assertTrue(a.confereSenha("123"));
        assertFalse(a.confereSenha("222"));
    }
}
```

Código

Testes JUnit

Testes dos métodos de classe da classe Main.

Testes confirmando que o método de impressão da notinha só é executado corretamente quando cliente opta por isso (retorno true) não executando quando cliente não desejar (retorno false).

```
@Test
void testNotinha1() {
    Main main = new Main();
    int cpfNota = 1;
    String cpf = "12345";
    Carrinho car = new Carrinho();
    assertTrue(main.notinha(cpfNota, cpf, car));
}

/**
 * Teste para validar retorno falso ao adicionar produto inválido na nota fiscal com m
 */
@Test
void testNotinha2() {
    Main main = new Main();
    int cpfNota = 4;
    String cpf = "12345";
    Carrinho car = new Carrinho();
    assertFalse(main.notinha(cpfNota, cpf, car));
}

}
```

Código

Testes JUnit

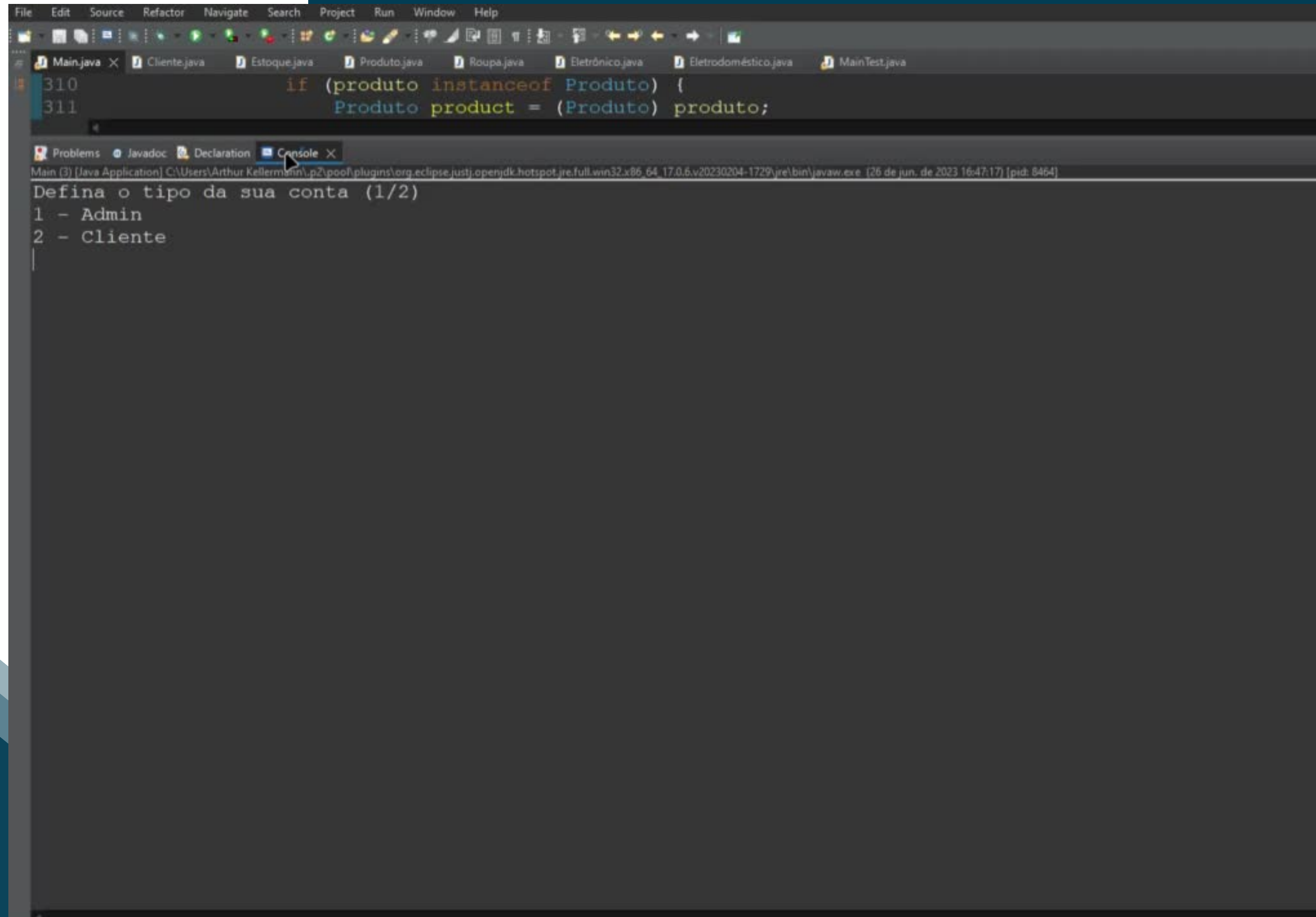
Testes unitários onde será certificado os retornos dos métodos da classe Main.

Testando o retorno de um usuário tipo cliente pelo método registerCliente. Confirmando que o único retorno correto dessa classe é um objeto cliente.

Testando o retorno de um usuário tipo administrador pelo método registerAdm. Confirmando que o único retorno correto dessa classe é um objeto administrador.

```
7
8
9  /*
10   * Teste que valida retorno de registro de Cliente e Admin
11   */
12  @Test
13  void testRegisterAdm() {
14      Admin expectedAdmin = new Admin(nome:"Arthur", cpf:"123456", email:"arthur@gmail.com", senha:"arthur123");
15      Admin actualAdmin = Main.registerAdm();
16
17      assertEquals(expectedAdmin, actualAdmin);
18  }
19
20  @Test
21  void testRegisterCliente() {
22      Cliente expectedCliente = new Admin(nome:"Arthur", cpf:"123456", email:"arthur@gmail.com", senha:"arthur123");
23      Cliente actualCliente = Main.registerCliente();
24
25      assertEquals(expectedCliente, actualCliente);
26  }
27  }
28
```

Exemplo: Cliente

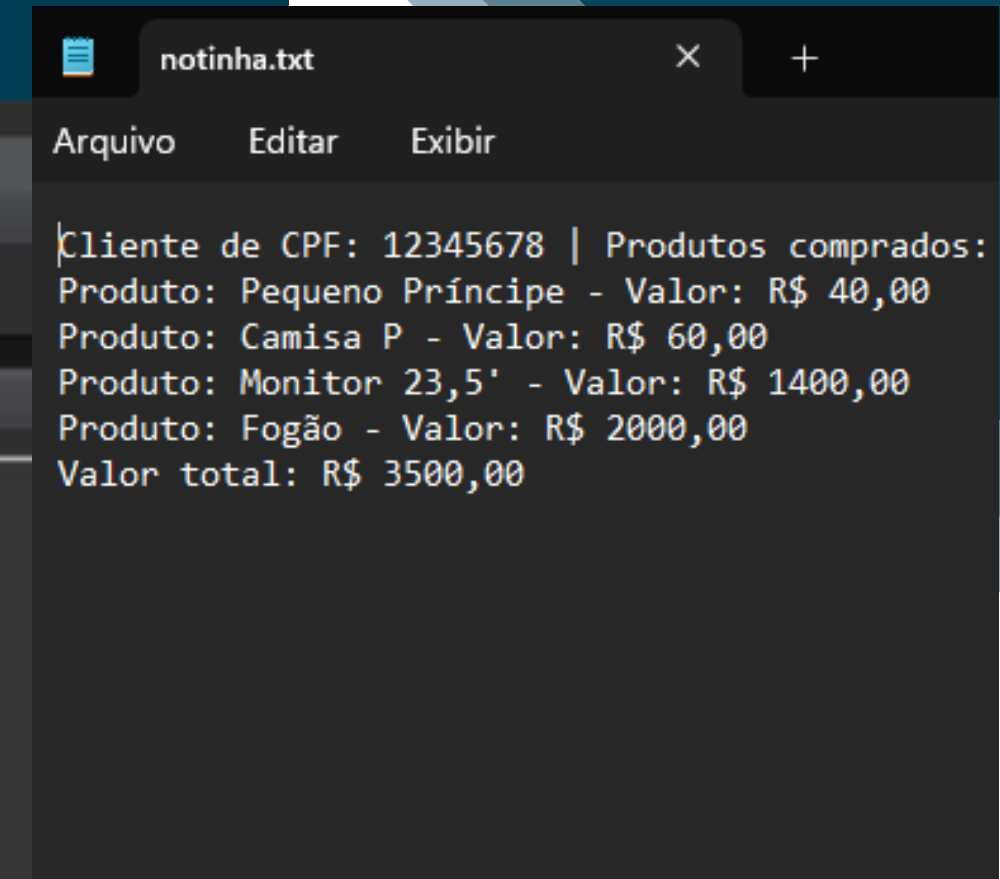


The screenshot shows the Eclipse IDE interface. The top menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the menu is a toolbar with various icons. The package explorer on the left shows a project named 'Exemplo' with several Java files: Main.java, Cliente.java, Estoque.java, Produto.java, Roupa.java, Eletrônico.java, Eletrodoméstico.java, and MainTest.java. The editor window displays the code in 'Cliente.java' at lines 310 and 311:

```
310         if (produto instanceof Produto) {  
311             Produto product = (Produto) produto;  
312         }
```

The console window at the bottom shows the following output:

```
Main (3) [Java Application] C:\Users\Arthur Kellermanin\p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\bin\javaw.exe [26 de jun. de 2023 16:47:17] [pid: 8464]  
Defina o tipo da sua conta (1/2)  
1 - Admin  
2 - Cliente  
|
```



The screenshot shows a text file named 'notinha.txt' with a menu bar containing 'Arquivo', 'Editar', and 'Exibir'. The text content of the file is as follows:

```
Cliente de CPF: 12345678 | Produtos comprados:  
Produto: Pequeno Príncipe - Valor: R$ 40,00  
Produto: Camisa P - Valor: R$ 60,00  
Produto: Monitor 23,5' - Valor: R$ 1400,00  
Produto: Fogão - Valor: R$ 2000,00  
Valor total: R$ 3500,00
```


Exemplo: Admin

