



**ATIVIDADE PRÁTICA DA DISCIPLINA METODOLOGIAS/  
MÉTODOS ÁGEIS  
ENGENHARIA DE SOFTWARE**

**RONNY WALLACE DE O. SOUZA – 4228692  
PROF<sup>a</sup>. MARIANE G B FERNANDES**

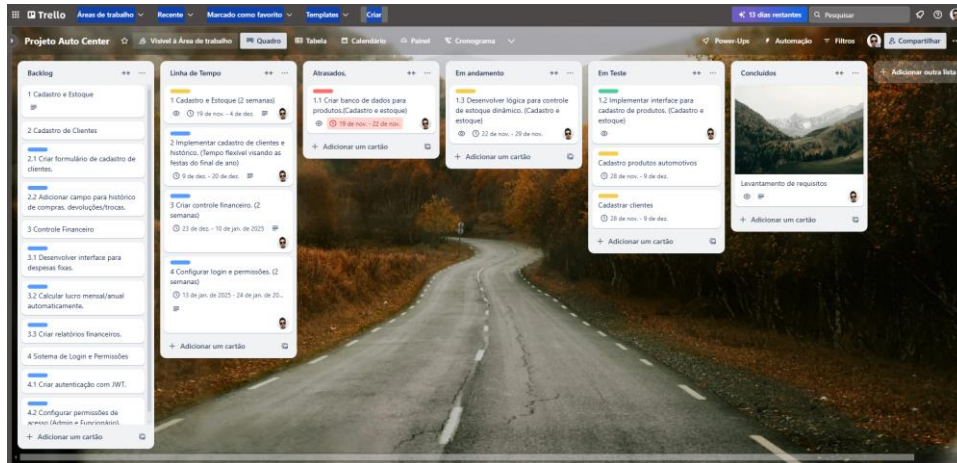
**VALINHOS - SÃO PAULO  
2024**

**História de Usuário:** O empresário Felipe Fernandes precisa realizar a automatização do sistema de sua startup AUTO CENTER FERNANDES. Atualmente o empresário disponibiliza em sua startup produtos automotivos de modo geral. Mas o empresário não tem nenhum software para realizar as seguintes funções: código do produto; marca do produto, quantidade dos produtos em estoque; valor unitário do produto; dados do cliente (nome, CPF, e-mail, contato, endereço e histórico de compras efetuadas e devoluções/trocas); impressão de notas fiscais das compras realizadas pelos clientes; Gastos mensais com funcionários; Gastos mensais básicos (energia e água); entrada/saída de produtos; e os lucros da empresa (mensal e anual). Além disso, Felipe precisará ter neste software dois tipos de login, um administrativo (terão acesso a todos os dados de sua startup e dos clientes) e outro login para seus funcionários (sem o demonstrativo de rendimentos que a startup ganha por dia/mês/ano e gastos gerais da empresa). Seu desafio é pensar como irá desenvolver futuramente um software que atenda a demanda do empresário Felipe para automatizar a startup AUTO CENTER FERNANDES.

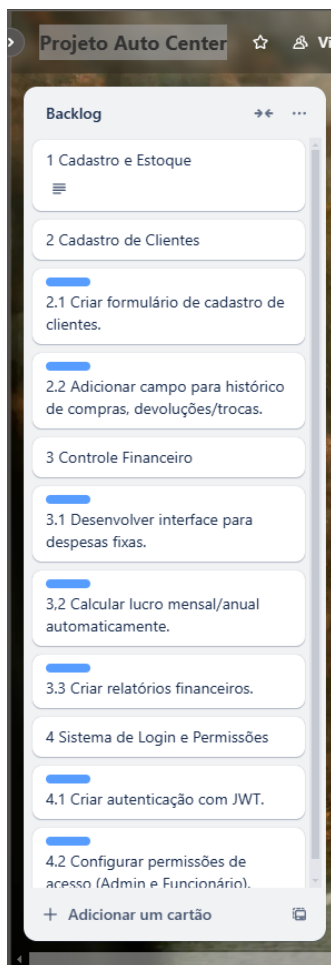
A partir da **HISTÓRIA DE USUÁRIO** responda as seguintes perguntas:

1. De acordo com Sutherland e Sutherland (2019, p. 17) “Scrum é uma metodologia ágil para gerenciar projetos complexos, em que não se conhece todas as etapas ou necessidades. Ela se baseia em valores, princípios e práticas que estimulam a colaboração, a criatividade e a adaptação às mudanças.” Posto isto, realize o gerenciamento do Método Scrum para a história de usuário da startup AUTO CENTER FERNANDES **utilizando a ferramenta TRELLO**. No seu gerenciamento você precisará mostrar os seguintes itens: Lista de Backlog; Linha de tempo (com o responsável pela tarefa e citar se tal tarefa está em andamento, realizada, em teste ou em atraso; Citar a quantidade de Sprints dentro de seu gerenciamento e com as devidas descrições). **NÃO SERÁ PERMITIDO ENTREGAR O LINK DO TRELLO.**

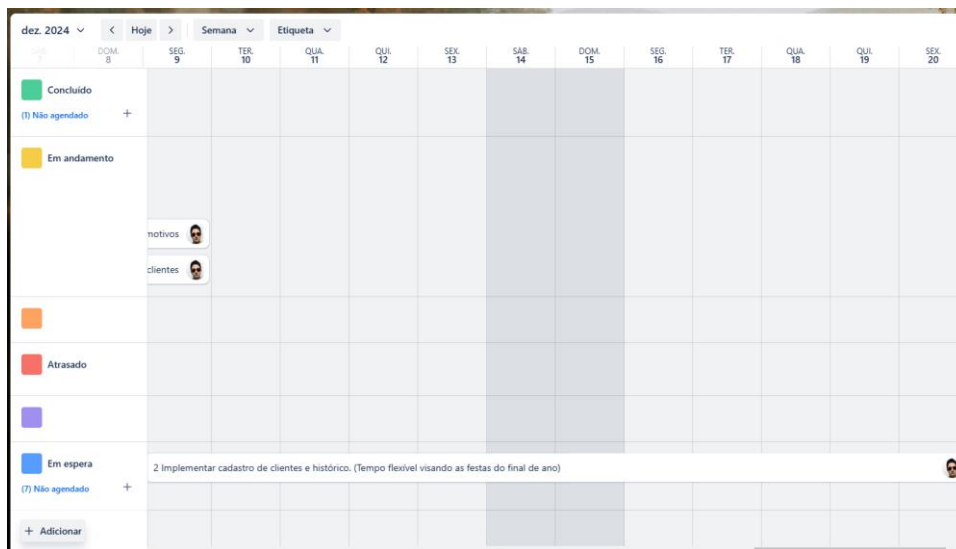
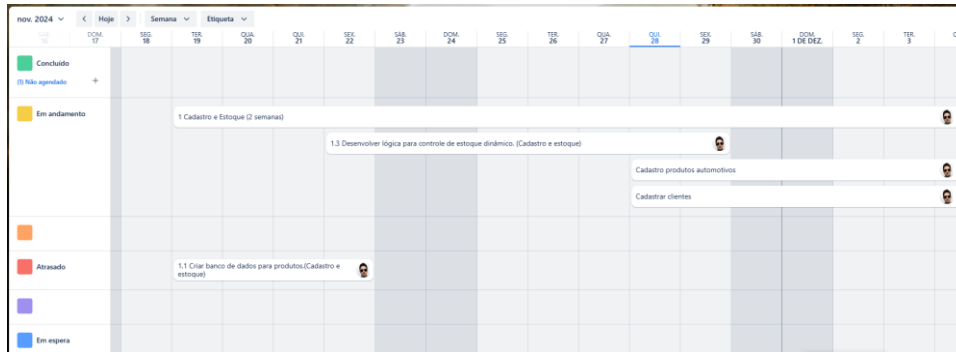
Visão do quadro geral:

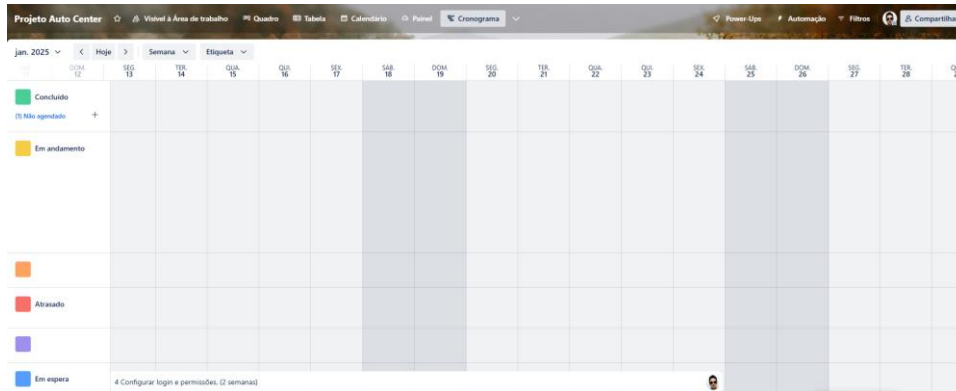


Backlog do projeto (Divido entre o objetivo principal de cada sprint e seus objetivos secundários):



Linha do tempo (Etiquetas de prazo à esquerda e a foto do membro responsável dentro da tarefa):





Sprints (primeira foto é a lista, depois a descrição com os objetivos de cada uma abaixo):

#### 1ª Sprint: Cadastro e Estoque de produtos

- Criar formulário para código, marca, quantidade, valor unitário.
- Implementar atualização automática do estoque.

#### 2ª Sprint: Implementar cadastro de clientes e histórico de compras

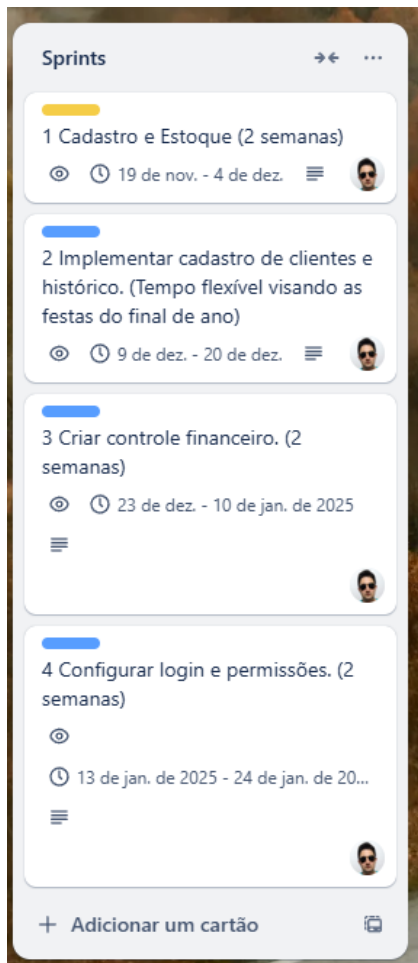
- Criar interface para cadastro de clientes (nome, CPF, e-mail, etc.).
- Desenvolver recurso para histórico de compras, trocas e devoluções.

#### 3ª Sprint: Criar controle financeiro

- Criar módulos para despesas fixas (água, energia, funcionários).
- Calcular lucros (mensal/anual) automaticamente.
- Implementar relatórios financeiros.
- Integrar API para emissão de notas fiscais.
- Implementar geração de PDF caso necessário.

#### 4ª Sprint: Configuração de login e permissões

- Criar autenticação (Admin e Funcionário).
- Implementar restrições baseadas nos perfis.



## 2 Implementar cadastro de clientes e histórico. (Tempo flexível visando as festas do final de ano)

na lista **SPRINTS**  

Membros



Etiquetas

Em espera



Notificações

 Seguindo 

Datas



9 de dez. - 20 de dez., 17:59 

### Descrição

Editar

#### Cadastro de Clientes (2 semanas)

- **Tarefas:**

- a. Criar formulário de cadastro de clientes.
- b. Adicionar campo para histórico de compras, devoluções/trocas.

## 3 Criar controle financeiro. (2 semanas)

na lista **SPRINTS**  

Membros



Etiquetas

Em espera



Notificações

 Seguindo 

Datas



23 de dez. de 2024 - 10 de jan. de 2025, 17:59 


### Descrição






Editar


#### Controle Financeiro (duração flexível devido ao fim de ano)


- **Tarefas:**

- a. Desenvolver interface para despesas fixas.
- b. Calcular lucro mensal/anual automaticamente.
- c. Criar relatórios financeiros.

 **4 Configurar login e permissões. (2 semanas)**  
na lista **SPRINTS** 

Membros   Etiquetas **Em espera**  Notificações  Seguindo 




Datas  
☐ 13 de jan. de 2025 - 24 de jan. de 2025, 17:59 




 **Descrição** Editar





**Sistema de Login e Permissões (2 semanas)**




- **Tarefas:**
  - a. Criar autenticação com JWT.
  - b. Configurar permissões de acesso (Admin e Funcionário)



Abaixo uma lista com algumas funcionalidades uma que está concluída e outras duas que estão em fase de teste:

**Em Teste**   

  
1.2 Implementar interface para cadastro de produtos. (Cadastro e estoque)  
 

  
Cadastro produtos automotivos  
  28 de nov. - 9 de dez. 

  
Cadastrar clientes  
 ☐ 28 de nov. - 9 de dez. 

**Este cartão deve ser entregue mais tarde.**  
 Adicionar um cartão 



2. “A conversão de requisitos em classes é uma etapa crucial no desenvolvimento de software orientado a objetos. Nesse processo, os requisitos funcionais e não funcionais identificados são analisados e transformados em classes, que são as unidades fundamentais de estrutura e comportamento do sistema”. Com base nisto, faça a conversão dos requisitos em classes, a partir da história de usuário da startup AUTO CENTER FERNANDES. Apresentar no mínimo 3 classes.

### **Classe produto**

#### Atributos da classe:

codigo; Integer

descricaoProduto; String

marca; String

quantidadeEmEstoque; Integer

valorUnitario; Double

### **Classe Cliente**

#### Atributos da classe:

nome; String

cpf; String

email; String

contato; String

endereco; String

historicoCompras; List<String>

## Classe Controle Financeiro

### Atributos da classe:

gastosMensaisFuncionarios; Double

gastosMensaisBasicos; Double

totalVendasMensal; Double

3. Para finalizar o projeto com sucesso, você precisará colocar em prática seus conhecimentos sobre a fase de teste, mais especificamente sobre **o Teste TDD**. Você foi destinado a testar duas classes programadas em Python, uma classe para o cadastro produtos automotivos e outra classe para cadastrar clientes. Além de desenvolver as DUAS classes, você precisará mostrar que as classes irão retornar com sucesso os dados, ou seja, colar o código indentado e com comentários. Por fim, colocar a imagem do terminar sendo executado sem erros com os dados correto das classes testadas.

### **Dicas para realizar o teste TDD em python:**

- ✓ Escreva testes iniciais: Comece escrevendo testes simples para cada funcionalidade que deseja implementar.
- ✓ Escreva o código mínimo: Implemente o código mínimo para fazer os testes passarem.
- ✓ Refatore (se necessário): Após os testes passarem, você pode refatorar o código para torná-lo mais limpo ou eficiente.
- ✓ Repita: Escreva mais testes para cobrir outros casos e continue iterando até que a funcionalidade seja completa.

Código completo aqui:

Classe Produto Automotivo:

```
class ProdutoAutomotivo:
    """
    Classe para representar produtos automotivos.
    """
    def __init__(self, codigo, descricao, marca, preco_unitario,
quantidade_estoque):
        self.codigo = codigo # Código único do produto
        self.descricao = descricao # Descrição do produto
        self.marca = marca # Marca do produto
        self.preco_unitario = preco_unitario # Preço unitário do produto
        self.quantidade_estoque = quantidade_estoque # Quantidade disponível em
estoque

    def atualizar_estoque(self, quantidade):
        """
        Atualizar o estoque com base na entrada/saída de produtos.
        """
        self.quantidade_estoque += quantidade
        return f"Estoque atualizado com sucesso. Quantidade atual:
{self.quantidade_estoque}"

    def calcular_valor_estoque(self):
        """
        Calcular o valor total dos produtos em estoque.
        """
        valor_total = self.quantidade_estoque * self.preco_unitario
        return f"Valor total em estoque: R${valor_total:.2f}"

    def exibir_detalhes_produtos(self):
        """
        Exibir os detalhes do produto.
        """
        return (f"Código: {self.codigo}\nDescrição: {self.descricao}\nMarca:
{self.marca}\n"
                f"Preço Unitário: R${self.preco_unitario:.2f}\nEstoque:
{self.quantidade_estoque}")
```

```
# Testando a Classe ProdutoAutomotivo no Console
if __name__ == "__main__":
    # Criando um produto
    produto = ProdutoAutomotivo(101, "Óleo Mineral 20W50", "Mobil", 27.99, 200)

    # Exibindo os detalhes do produto
    print("Detalhes do Produto:")
    print(produto.exibir_detalhes_produtos())
```

```
➡ Detalhes do Produto:
   Código: 101
   Descrição: Óleo Mineral 20W50
   Marca: Mobil
   Preço Unitário: R$27.99
   Estoque: 200
```

```
# Testando a Classe ProdutoAutomotivo no Console
if __name__ == "__main__":
    # Criando um produto
    produto = ProdutoAutomotivo(101, "Óleo Mineral 20W50", "Mobil", 27.99, 200)

    # Exibindo os detalhes do produto
    print("Detalhes do Produto:")
    print(produto.exibir_detalhes_produtos())

    # Atualizando o estoque (saída de 30 unidades)
    print("\nAtualizando Estoque (saída de 30 unidades):")
    print(produto.atualizar_estoque(-30))
```

```
➡ Detalhes do Produto:
   Código: 101
   Descrição: Óleo Mineral 20W50
   Marca: Mobil
   Preço Unitário: R$27.99
   Estoque: 200

   Atualizando Estoque (saída de 30 unidades):
   Estoque atualizado com sucesso. Quantidade atual: 170
```

```
# Testando a Classe ProdutoAutomotivo no Console
if __name__ == "__main__":
    # Criando um produto
    produto = ProdutoAutomotivo(101, "Óleo Mineral 20W50", "Mobil", 27.99, 200)

    # Exibindo os detalhes do produto
    print("Detalhes do Produto:")
```

```
print(produto.exibir_detalhes_produtos())

# Atualizando o estoque (saída de 30 unidades)
print("\nAtualizando Estoque (saída de 30 unidades):")
print(produto.atualizar_estoque(-30))

# Atualizando o estoque (entrada de 50 unidades)
print("\nAtualizando Estoque (entrada de 50 unidades):")
print(produto.atualizar_estoque(50))
```

```
➡ Detalhes do Produto:
Código: 101
Descrição: Óleo Mineral 20W50
Marca: Mobil
Preço Unitário: R$27.99
Estoque: 200

Atualizando Estoque (saída de 30 unidades):
Estoque atualizado com sucesso. Quantidade atual: 170

Atualizando Estoque (entrada de 50 unidades):
Estoque atualizado com sucesso. Quantidade atual: 220
```

```
# Testando a Classe ProdutoAutomotivo no Console
if __name__ == "__main__":
    # Criando um produto
    produto = ProdutoAutomotivo(101, "Óleo Mineral 20W50", "Mobil", 27.99, 200)

    # Exibindo os detalhes do produto
    print("Detalhes do Produto:")
    print(produto.exibir_detalhes_produtos())

    # Atualizando o estoque (saída de 30 unidades)
    print("\nAtualizando Estoque (saída de 30 unidades):")
    print(produto.atualizar_estoque(-30))

    # Atualizando o estoque (entrada de 50 unidades)
    print("\nAtualizando Estoque (entrada de 50 unidades):")
    print(produto.atualizar_estoque(50))

    # Calculando o valor total do estoque
    print("\nCalculando o Valor Total do Estoque:")
    print(produto.calcular_valor_estoque())
```

```
➡ Detalhes do Produto:  
Código: 101  
Descrição: Óleo Mineral 20W50  
Marca: Mobil  
Preço Unitário: R$27.99  
Estoque: 200  
  
Atualizando Estoque (saída de 30 unidades):  
Estoque atualizado com sucesso. Quantidade atual: 170  
  
Atualizando Estoque (entrada de 50 unidades):  
Estoque atualizado com sucesso. Quantidade atual: 220  
  
Calculando o Valor Total do Estoque:  
Valor total em estoque: R$6157.80
```

Classe Cliente:

```
class Cliente:  
    """  
    Classe para representar um cliente.  
    """  
  
    def __init__(self, nome, cpf, telefone, endereco):  
        self.nome = nome # Nome do cliente  
        self.cpf = cpf # CPF do cliente  
        self.telefone = telefone # Telefone de contato  
        self.endereco = endereco # Endereço do cliente  
        self.historico_compras = [] # Histórico de compras do cliente  
  
    def registrar_compra(self, produto, quantidade, valor_total):  
        """  
        Registrar uma compra no histórico do cliente.  
        """  
        compra = {  
            "produto": produto,  
            "quantidade": quantidade,  
            "valor_total": valor_total  
        }  
        self.historico_compras.append(compra)  
        return f"Compra registrada: {produto} (x{quantidade}) - Total:  
R${valor_total:.2f}"  
  
    def exibir_historico_compras(self):  
        """  
        Exibir o histórico de compras do cliente.  
        """  
        if not self.historico_compras:  
            return "O cliente não possui compras registradas."
```

```
        historico = "Histórico de Compras:\n"
        for i, compra in enumerate(self.historico_compras, start=1):
            historico += (f"{i}. Produto: {compra['produto']}, Quantidade: {compra['quantidade']}, "
                           f"Valor Total: R${compra['valor_total']:.2f}\n")
        return historico.strip()

    def exibir_informacoes(self):
        """
        Exibir as informações do cliente.
        """
        return (f"Nome: {self.nome}\nCPF: {self.cpf}\nTelefone: {self.telefone}\n"
                f"Endereço: {self.endereco}")

# Testando a Classe Cliente no Console
if __name__ == "__main__":
    # Criando um cliente
    cliente = Cliente("Ronny Wallace RU: 4228692", "123.456.789-10", "19-99579-7614", "Rua Zero, 123")

    # Exibindo informações do cliente
    print("Informações do Cliente:")
    print(cliente.exibir_informacoes())
```

```
➡ Informações do Cliente:
Nome: Ronny Wallace RU: 4228692
CPF: 123.456.789-10
Telefone: 19-99579-7614
Endereço: Rua Zero, 123
```

```
# Testando a Classe Cliente no Console
if __name__ == "__main__":
    # Criando um cliente
    cliente = Cliente("Ronny Wallace RU: 4228692", "123.456.789-10", "19-99579-7614", "Rua Zero, 123")

    # Exibindo informações do cliente
    print("Informações do Cliente:")
    print(cliente.exibir_informacoes())

    # Registrando compras
    print("\nRegistrando Compras:")
    print(cliente.registrar_compra("Óleo Mineral 20W50", 2, 55.98))
    print(cliente.registrar_compra("Filtro de Óleo", 1, 30.00))
```

```
➡ Informações do Cliente:  
Nome: Ronny Wallace RU: 4228692  
CPF: 123.456.789-10  
Telefone: 19-99579-7614  
Endereço: Rua Zero, 123  
  
Registrando Compras:  
Compra registrada: Óleo Mineral 20W50 (x2) - Total: R$55.98  
Compra registrada: Filtro de Óleo (x1) - Total: R$30.00
```

```
# Testando a Classe Cliente no Console  
if __name__ == "__main__":  
    # Criando um cliente  
    cliente = Cliente("Ronny Wallace RU: 4228692", "123.456.789-10", "19-99579-7614", "Rua Zero, 123")  
  
    # Exibindo informações do cliente  
    print("Informações do Cliente:")  
    print(cliente.exibir_informacoes())  
  
    # Registrando compras  
    print("\nRegistrando Compras:")  
    print(cliente.registrar_compra("Óleo Mineral 20W50", 2, 55.98))  
    print(cliente.registrar_compra("Filtro de Óleo", 1, 30.00))  
  
    # Exibindo histórico de compras  
    print("\nExibindo Histórico de Compras:")  
    print(cliente.exibir_historico_compras())
```

```
➡ Informações do Cliente:  
Nome: Ronny Wallace RU: 4228692  
CPF: 123.456.789-10  
Telefone: 19-99579-7614  
Endereço: Rua Zero, 123  
  
Registrando Compras:  
Compra registrada: Óleo Mineral 20W50 (x2) - Total: R$55.98  
Compra registrada: Filtro de Óleo (x1) - Total: R$30.00  
  
Exibindo Histórico de Compras:  
Histórico de Compras:  
1. Produto: Óleo Mineral 20W50, Quantidade: 2, Valor Total: R$55.98  
2. Produto: Filtro de Óleo, Quantidade: 1, Valor Total: R$30.00
```

Código e saída completos:

```
class ProdutoAutomotivo:
```



```
"""
Classe para representar produtos automotivos.
"""

def __init__(self, codigo, descricao, marca, preco_unitario,
quantidade_estoque):
    self.codigo = codigo # Código único do produto
    self.descricao = descricao # Descrição do produto
    self.marca = marca # Marca do produto
    self.preco_unitario = preco_unitario # Preço unitário do produto
    self.quantidade_estoque = quantidade_estoque # Quantidade disponível em
estoque

def atualizar_estoque(self, quantidade):
    """
    Atualiza o estoque com base na entrada/saída de produtos.
    """
    self.quantidade_estoque += quantidade
    return f"Estoque atualizado com sucesso. Quantidade atual:
{self.quantidade_estoque}"

def calcular_valor_estoque(self):
    """
    Calcula o valor total dos produtos em estoque.
    """
    valor_total = self.quantidade_estoque * self.preco_unitario
    return f"Valor total em estoque: R${valor_total:.2f}"

def exibir_detalhes_produtos(self):
    """
    Exibe os detalhes do produto.
    """
    return (f"Código: {self.codigo}\nDescrição: {self.descricao}\nMarca:
{self.marca}\n"
           f"Preço Unitário: R${self.preco_unitario:.2f}\nEstoque:
{self.quantidade_estoque}")

class Cliente:
    """
    Classe para representar um cliente.
    """

    def __init__(self, nome, cpf, telefone, endereco):
        self.nome = nome # Nome do cliente
        self.cpf = cpf # CPF do cliente
        self.telefone = telefone # Telefone de contato
        self.endereco = endereco # Endereço do cliente
        self.historico_compras = [] # Histórico de compras do cliente

    def registrar_compra(self, produto, quantidade, valor_total):
        """
```

```
        Registrar uma compra no histórico do cliente.
        """
        compra = {
            "produto": produto,
            "quantidade": quantidade,
            "valor_total": valor_total
        }
        self.historico_compras.append(compra)
        return f"Compra registrada: {produto} (x{quantidade}) - Total:
R${valor_total:.2f}"

    def exibir_historico_compras(self):
        """
        Exibir o histórico de compras do cliente.
        """
        if not self.historico_compras:
            return "O cliente não possui compras registradas."

        historico = "Histórico de Compras:\n"
        for i, compra in enumerate(self.historico_compras, start=1):
            historico += (f"{i}. Produto: {compra['produto']}, Quantidade:
{compra['quantidade']}, "
                        f"Valor Total: R${compra['valor_total']:.2f}\n")
        return historico.strip()

    def exibir_informacoes(self):
        """
        Exibir as informações do cliente.
        """
        return (f"Nome: {self.nome}\nCPF: {self.cpf}\nTelefone: {self.telefone}\n"
                f"Endereço: {self.endereco}")

# Testando a Classe ProdutoAutomotivo no Console
if __name__ == "__main__":
    # Criando um produto
    produto = ProdutoAutomotivo(101, "Óleo Mineral 20W50", "Mobil", 27.99, 200)

    # Exibindo os detalhes do produto
    print("Detalhes do Produto:")
    print(produto.exibir_detalhes_produtos())

    # Atualizando o estoque (saída de 30 unidades)
    print("\nAtualizando Estoque (saída de 30 unidades):")
    print(produto.atualizar_estoque(-30))

    # Atualizando o estoque (entrada de 50 unidades)
    print("\nAtualizando Estoque (entrada de 50 unidades):")
    print(produto.atualizar_estoque(50))
```

```
# Calculando o valor total do estoque
print("\nCalculando o Valor Total do Estoque:")
print(produto.calcular_valor_estoque())

# Testando a Classe Cliente no Console
if __name__ == "__main__":
    # Criando um cliente
    cliente = Cliente("Ronny Wallace RU: 4228692", "123.456.789-10", "19-99579-7614", "Rua Zero, 123")

    # Exibindo informações do cliente
    print("Informações do Cliente:")
    print(cliente.exibir_informacoes())

    # Registrando compras
    print("\nRegistrando Compras:")
    print(cliente.registrar_compra("Óleo Mineral 20W50", 2, 55.98))
    print(cliente.registrar_compra("Filtro de Óleo", 1, 30.00))

    # Exibindo histórico de compras
    print("\nExibindo Histórico de Compras:")
    print(cliente.exibir_historico_compras())
```



Detalhes do Produto:

Código: 101

Descrição: Óleo Mineral 20W50

Marca: Mobil

Preço Unitário: R\$27.99

Estoque: 200

Atualizando Estoque (saída de 30 unidades):

Estoque atualizado com sucesso. Quantidade atual: 170

Atualizando Estoque (entrada de 50 unidades):

Estoque atualizado com sucesso. Quantidade atual: 220

Calculando o Valor Total do Estoque:

Valor total em estoque: R\$6157.80

Informações do Cliente:

Nome: Ronny Wallace RU: 4228692

CPF: 123.456.789-10

Telefone: 19-99579-7614

Endereço: Rua Zero, 123

Registrando Compras:

Compra registrada: Óleo Mineral 20W50 (x2) - Total: R\$55.98

Compra registrada: Filtro de Óleo (x1) - Total: R\$30.00

Exibindo Histórico de Compras:

Histórico de Compras:

1. Produto: Óleo Mineral 20W50, Quantidade: 2, Valor Total: R\$55.98

2. Produto: Filtro de Óleo, Quantidade: 1, Valor Total: R\$30.00

