

# Programação Orientada a Objetos

# EMPRESA DRINKEASY

Aplicação para gerenciar  
comandas

Alunos:

Elismara Rodrigues

Jonas Dantas

Luanna Bahia

Pedro Henrique Barreto



## Tópicos

**1** Introdução

---

**2** Contexto do Cenário

---

**3** Caso de uso em um Festival

---

**4** Diagrama de Classe

---

**5** Estrutura do código

---

**6** Demonstração do sistema

# Introdução



- Sistema DrinkEasy - Gerenciamento de Comandas.
- Objetivo: Criar uma aplicação para gerenciar comandas em estações de atendimento.

# Contexto do Cenário



## CENÁRIO 03: EMPRESA DRINKEASY

- Ter várias **estações**.
- As bebidas disponíveis são **chopp, vinho e refrigerante**.
- O consumo é registrado em **comandas individuais**.
- A aplicação **gerencia as comandas** e apresenta o **consumo final** de cada cliente.



## PROBLEMA A SER ROSOLVIDO

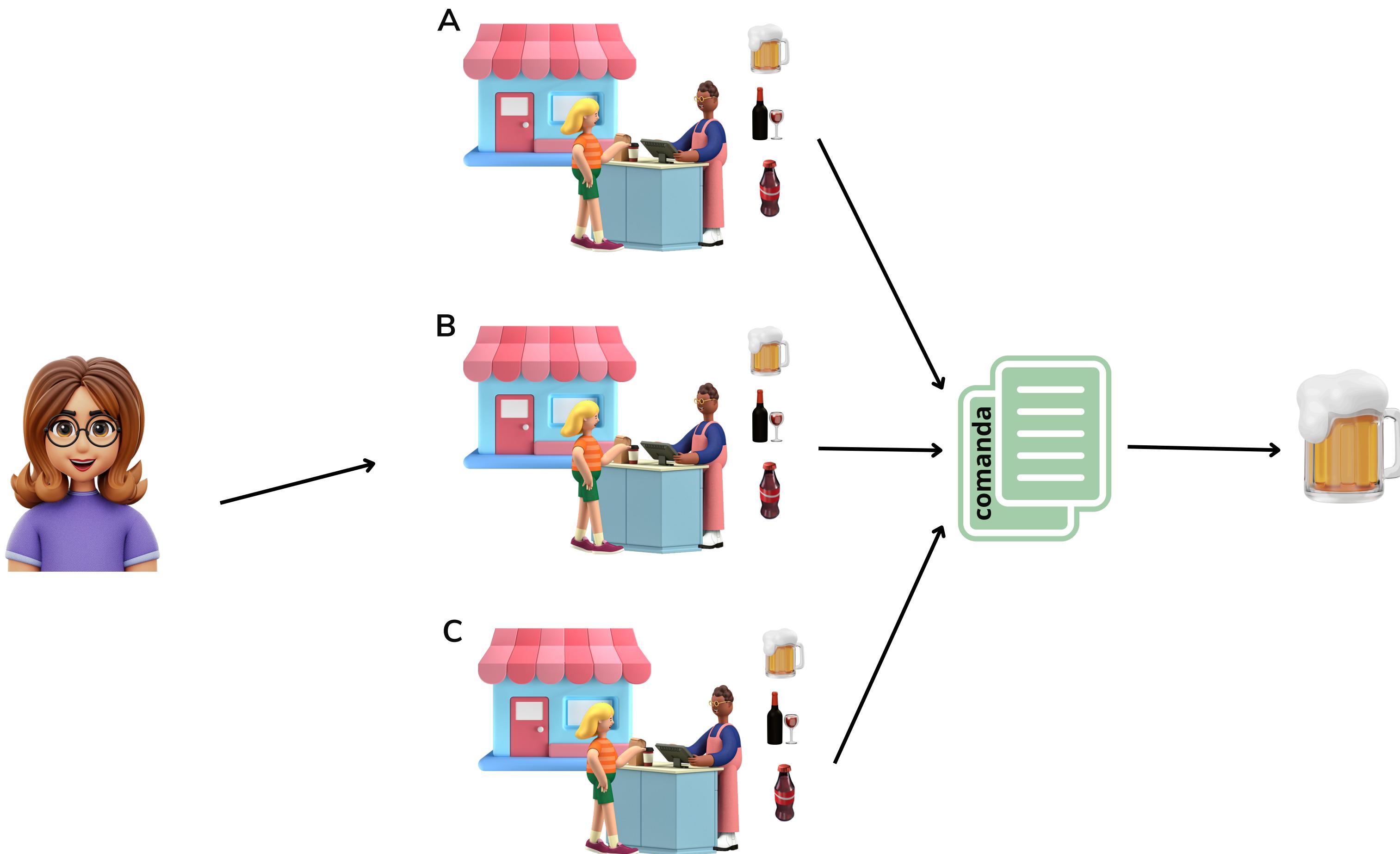
- Criar uma **Gestão de comandas e consumo** em **múltiplas estações**.

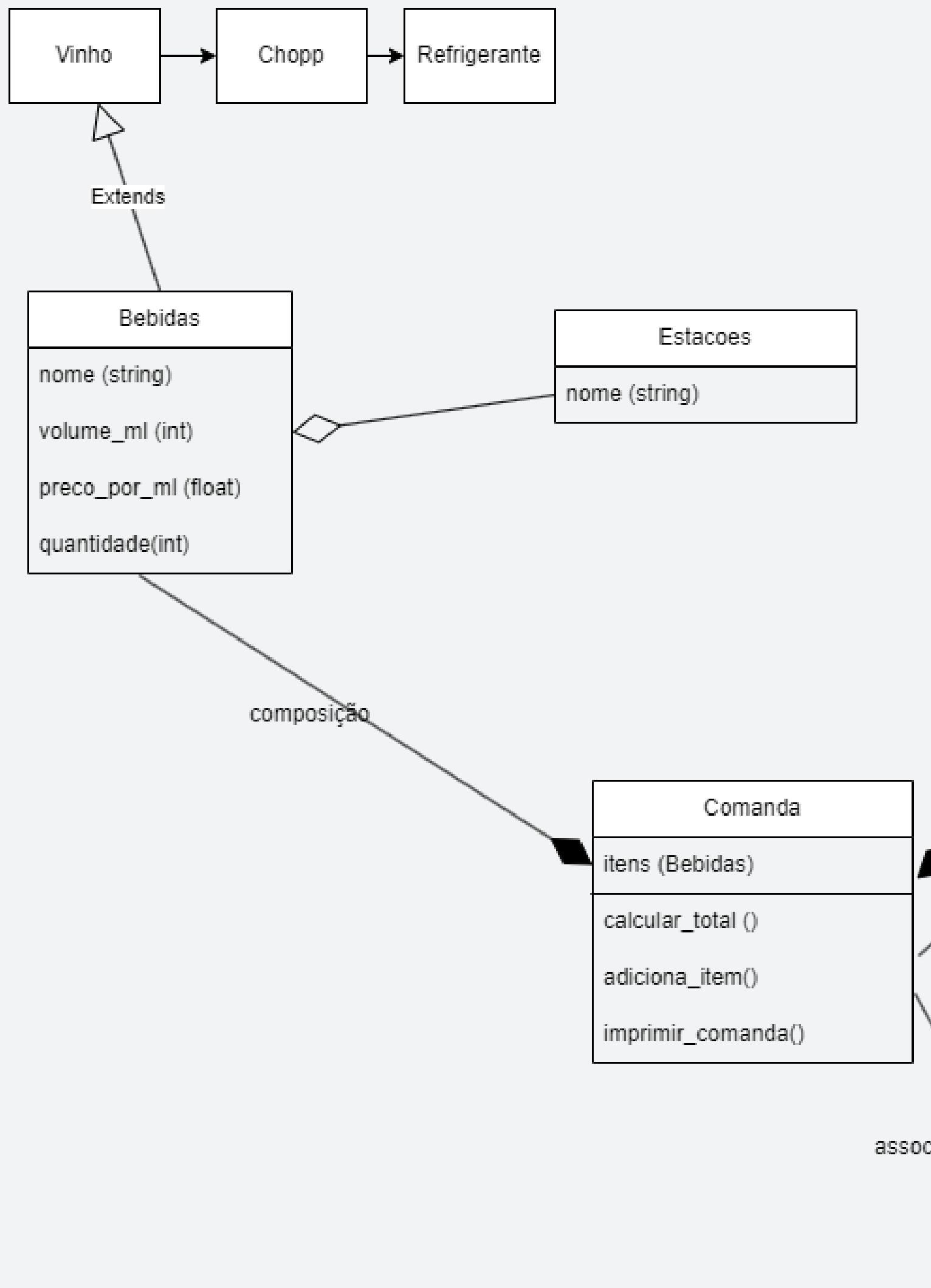


## SOLUÇÃO PROPOSTA

- Aplicação com funcionalidades específicas.
- Construir um **diagrama de classe**.
- Usar linguagem de programação: **Java ou Python**.
- Usar conceitos de POO: **herança, encapsulamento e polimorfismo**.

# Caso de uso em um Festival





# Diagrama de Classe

# Estrutura do código

Arquivos da aplicação



# Tabela de preços

Arquivos da aplicação

Tabela de preços		
CHOPP		
Pequeno mL / R\$	300mL	R\$ 9,00
Médio mL / R\$	400mL	R\$ 6,00
Grande mL / R\$	500mL	R\$ 7,50
VINHO		
Pequeno mL / R\$	300mL	R\$ 12,00
Médio mL / R\$	400mL	R\$ 16,00
Grande mL / R\$	500mL	R\$ 20,00
REFRIGERANTE		
Pequeno mL / R\$	300mL	R\$ 6,00
Médio mL / R\$	400mL	R\$ 8,00
Grande mL / R\$	500mL	R\$ 10,00



```
1 # Classe base Pessoa
2 class Pessoa:
3     def __init__(self, nome, cpf, telefone, endereco):
4         self.__nome = nome # Nome da pessoa
5         self.__cpf = cpf   # CPF da pessoa
6         self._telefone = telefone # Telefone da pessoa
7         self._endereco = endereco # Endereço da pessoa
8
9     @property
10    def nome(self):
11        return self.__nome # Getter para o nome
12
13    @property
14    def cpf(self):
15        return self.__cpf   # Getter para o CPF
16
17    def telefone(self):
18        return self._telefone # Getter para o telefone
19
20    def endereco(self):
21        return self._endereco # Getter para o endereço
22
23    #pessoas diferentes podem exibir a comanda, o que irá mudar é sua permissão
24    @property
25    def exibirComanda(self):
26        print("Nenhuma comanda")
```



```
1 # Classe Cliente que herda de Pessoa
2 class Cliente(Pessoa):
3     def __init__(self, nome, cpf, telefone, endereco, comandas):
4         super().__init__(nome, cpf, telefone, endereco)
5         self.comandas = comandas # Lista de comandas do cliente
6
7     def exibirComanda(self):
8         if not self.comandas:
9             print("Nenhuma comanda encontrada.")
10            return
11
12         for comanda in self.comandas:
13             print('\n')
14             print("-" * 60)
15             print("\bCOMANDA")
16             print(f"Funcionário: {comanda.funcionario().nome}") # Exibe o nome do funcionário
17             print(f"Cliente: {self.nome}")
18             print(f"CPF: {self.cpf}")
19
20             comanda.imprimirComanda()
21             print(f"Total: R${comanda.calcularTotal():.2f}")
22             print("-" * 60)
23
```



```
1 # Classe Funcionario que herda de Pessoa
2 class Funcionario(Pessoa):
3     def __init__(self, nome, cpf, telefone, endereco, cargo):
4         super().__init__(nome, cpf, telefone, endereco) # Chama o construtor da classe base
5         self.__cargo = cargo # Cargo do funcionário
6
7     @property
8     def cargo(self):
9         return self.__cargo # Getter para o cargo
10
11    #exibir todas as comandas
12    def exibirComanda(self):
13        for c in comandasGeral:
14            print('\n')
15            print('\bCOMANDAS EM GERAL')
16            print('-' * 60)
17            print("Cliente: " + c.cliente.nome)
18            c.imprimirComanda()
19            print('-' * 60)
20
```

```
21
22 # permiti ao funcionario gerencia o cliente
23 def gerenciarCliente(self, clientes):
24     # verifica se existe um cliente
25     if not clientes:
26         print("Nenhum cliente cadastrado. Crie um cliente primeiro.")
27         return
28
29     try:
30         cliente = pesquisar_cliente(clientes)
31
32         if not cliente.comandas:
33             print("Nenhuma comanda encontrada para o cliente.")
34             return
35
36         cliente.exibirComanda()
37
38         print('\nQuitar conta do cliente:\n1 - sim\n2 - não') #pergunta se deseja quitar a conta do cliente
39         opcao = int(input("\nSeleciono uma opcao: "))
40         comanda = cliente.comandas[0]
41         if opcao == 1:
42             cliente.comandas.remove(comanda)
43             print("\nConta quitada com sucesso")
44         else:
45             print("\n Não quitada")
46     except ValueError:
47         print("Opção inválida. Tente novamente.")
48
49     return
```



```
1 # Classe Estacao
2 class Estacao:
3     def __init__(self, nome):
4         self.__nome = nome # Nome da estação
5
6     @property
7     def nome(self):
8         return self.__nome # Getter para o nome da estação
```



```
1 # Classe Comanda
2 class Comanda:
3     def __init__(self, cliente, funcionario):
4         self.__itens = [] # Lista de itens na comanda
5         self.__cliente = cliente # Cliente associado à comanda
6         self.__funcionario = funcionario # Funcionário associado à comanda
7
8     def cliente(self):
9         return self.__cliente # Retorna o cliente associado à comanda
10
11    def funcionario(self):
12        return self.__funcionario # Retorna o funcionário associado à comanda
13
14    @property
15    def itens(self):
16        return self.__itens # Getter para a lista de itens na comanda
17
18    def calcularTotal(self):
19        total = 0
20        for item in self.__itens:
21            total += (item.preco * item.quantidade) # Calcula o total da comanda somando o preço dos itens multiplicado pela quantidade
22        return total
23
24    def adicionarItem(self, item):
25        self.__itens.append(item) # Adiciona um item à lista de itens na comanda
26
27    def imprimirComanda(self):
28        for item in self.__itens:
29            print("Estação: " + item.estacao.nome) # Imprime o nome da estação do item
30            print('Itens:')
31            print(str(item.quantidade) + 'x - ' + item.nome + ' - ' + item.volume + ' - R$' + str(item.preco)) # Imprime a quantidade, nome, volume e preço do item
```



```
1 # Classe base Bebida
2 class Bebida:
3     def __init__(self, nome, preco, volume, estacao, quantidade):
4         self.__nome = nome      # Nome da bebida
5         self.__preco = preco    # Preço da bebida
6         self.__volume = volume # Volume da bebida
7         self.__estacao = estacao # Estação associada à bebida
8         self.__quantidade = quantidade # Quantidade da bebida
9
10    @property
11    def nome(self):
12        return self.__nome # Getter para o nome
13
14    @property
15    def preco(self):
16        return self.__preco # Getter para o preço
17
18    @property
19    def volume(self):
20        return self.__volume # Getter para o volume
21
22 # diferentes tipos de bebidas
23 class Chopp(Bebida):
24     def __init__(self, nome, preco, volume, estacao, quantidade):
25         super().__init__(nome, preco, volume, estacao, quantidade) # Chama o construtor da classe base
26
27 class Vinho(Bebida):
28     def __init__(self, nome, preco, volume, estacao, quantidade):
29         super().__init__(nome, preco, volume, estacao, quantidade) # Chama o construtor da classe base
30
31 class Refrigerante(Bebida):
32     def __init__(self, nome, preco, volume, estacao, quantidade):
33         super().__init__(nome, preco, volume, estacao, quantidade) # Chama o construtor da classe base
```



```
1 comandasGeral = [] # Lista de comandas
2
3 # Função para inicializar estações com objetos
4 def estacoes():
5     estacao_sul = Estacao(" Estação do Sul") # Cria estação de chopp
6     estacao_centro = Estacao(" Estação do Centro") # Cria estação de vinho
7     estacao_norte = Estacao(" Estação do Norte") # Cria estação de refrigerante
8     return [estacao_sul, estacao_centro, estacao_norte] # Retorna lista de estações
9
10
11 # função para criar cliente
12 def criar_Cliente_teste(clientes):
13     if not clientes:
14         cliente = Cliente("Juca", "123.456.789-10", "99 99999999", "Rua maria knksdsds", comandas=[])
15         clientes.append(cliente)
16
17 # Função para criar administrador padrão
18 def criar_administrador_padrao(funcionarios):
19     if not funcionarios: # Verifica se a lista de funcionários está vazia
20         administrador = Funcionario("Adm", "000.000.000-00", "99 99999999", "Rua Maria asasas", "Administrador",) # Cria funcionário padrão
21         funcionarios.append(administrador) # Adiciona o administrador à lista
22         #print("Funcionário padrão (Admin) criado com sucesso!") # Mensagem de sucesso
23
```



```
1 # Função para carregar bebidas
2 def carregar_bebidas():
3     # Bebidas organizadas
4
5     bebidas = []
6
7     chops = []
8     chop = Chopp('Chopp', 9.00, '300ml', None, 0)
9     chops.append(chop)
10    chop = Chopp('Chopp', 6.00, '400ml', None, 0)
11    chops.append(chop)
12    chop = Chopp('Chopp', 7.50, '500ml', None, 0)
13    chops.append(chop)
14
15    bebidas.append(chops)
16
17    vinhos = []
18    vinho = Vinho('Vinho', 12.00, '300ml', None, 0)
19    vinhos.append(vinho)
20    vinho = Vinho('Vinho', 16.00, '400ml', None, 0)
21    vinhos.append(vinho)
22    vinho = Vinho('Vinho', 20.00, '500ml', None, 0)
23    vinhos.append(vinho)
24
25    bebidas.append(vinhos)
26
27    refrigerantes = []
28    refrigerante = Refrigerante('Refrigerante', 6.00, '300ml', None, 0)
29    refrigerantes.append(refrigerante)
30    refrigerante = Refrigerante('Refrigerante', 8.00, '400ml', None, 0)
31    refrigerantes.append(refrigerante)
32    refrigerante = Refrigerante('Refrigerante', 10.00, '500ml', None, 0)
33    refrigerantes.append(refrigerante)
34
35    bebidas.append(refrigerantes)
36    return bebidas # Retorna o a lista de bebidas
37
```

```
1 # Função para realizar nova venda
2 def nova_venda(clientes,funcionarios, estacoes):
3     if not clientes: # Verifica se a lista de clientes está vazia
4         print("Nenhum cliente cadastrado. Crie um cliente primeiro.") # Mensagem de erro
5         return
6
7     if not funcionarios: # Verifica se a lista de funcionários está vazia
8         print("Nenhum funcionário cadastrado. Crie um funcionário primeiro.") # Mensagem de erro
9         return
10
11    if not estacoes: # Verifica se a lista de estações está vazia
12        print("Nenhuma estação cadastrada. Crie uma estação primeiro.") # Mensagem de erro
13        return
14
15    print('\nNOVA VENDA') # Início do processo de nova venda
16
17    funcionario = pesquisar_funcionario(funcionarios)
18    cliente = pesquisar_cliente(clientes) # Pesquisa o cliente
19
20    if not cliente:
21        return
22
23    estacao_escolhida = escolherEstacao(estacoes) # escolha de estação
24
25    if not estacao_escolhida:
26        return
27
28    if not cliente.comandas:
29        comanda = Comanda(cliente,funcionario)
30        comanda.cliente = cliente
31        comandasGeral.append(comanda)
32        cliente.comandas.append(comanda) # Cria uma nova comanda para o cliente
33    else:
34        comanda = cliente.comandas[-1] # Utiliza a última comanda do cliente
35
36
37    print(f"\bEstação selecionada: {estacao_escolhida.nome}") # Confirmação da estação selecionada
38    bebida_escolhida = escolherBebida()
39
40    print(f"Bebida selecionada: {bebida_escolhida.nome}") # Confirmação da bebida selecionada
41    bebida_escolhida.estacao = estacao_escolhida
42    comanda.adicionarItem(bebida_escolhida)
43
44    print(f"Total da comanda: R${comanda.calcularTotal()}") # Exibe o total da comanda
45
```



```
1 #pesquisa e devolve o funcionario
2 def pesquisar_funcionario(funcionarios):
3     # Selecionar o vendedor
4     print("Selecione o vendedor:")
5     idx = 1 # Índice para listar os vendedores
6     for funcionario in funcionarios: # Percorre a lista de funcionários
7         print(f"{idx} - {funcionario.nome} (CPF: {funcionario.cpf})") # Lista de vendedores
8         idx += 1 # Incrementa o índice
9
10 try: # Tratamento de exceção
11     opcao_vendedor = int(input('Digite a opção desejada: ')) # Captura a opção do vendedor
12     if opcao_vendedor < 1 or opcao_vendedor > len(funcionarios): # Verifica se a opção é válida
13         print("Vendedor inválido. Tente novamente.") # Mensagem de erro
14     return
15     vendedor = funcionarios[opcao_vendedor - 1] # Seleciona o vendedor
16 except ValueError: # Tratamento de exceção
17     print("Opção inválida. Tente novamente.") # Mensagem de erro
18     return
19
20 print(f"Vendedor selecionado: {vendedor.nome} (CPF: {vendedor.cpf})") # Confirmação do vendedor selecionado
21 return vendedor # Retorna o vendedor selecionado
22
```



```
1 # Função para pesquisar cliente
2 def pesquisar_cliente(clientes):
3     print('\nPESQUISAR CLIENTE')
4     print('1 - Buscar por Nome')
5     print('2 - Buscar por CPF')
6     print('3 - Listar clientes')
7     print('4 - Voltar')
8     opcao = int(input('Digite a opção desejada: ')) # Captura a opção do usuário
9
10    if opcao == 1: # Buscar por nome
11        nome = input('Digite o nome do cliente: ')
12        for cliente in clientes: # Percorre a lista de clientes
13            if cliente.nome == nome: # Verifica se o nome do cliente é igual ao nome pesquisado
14                return cliente
15    elif opcao == 2: # Buscar por CPF
16        cpf = input('Digite o CPF do cliente: ')
17        for cliente in clientes: # Percorre a lista de clientes
18            if cliente.cpf == cpf: # Verifica se o CPF do cliente é igual ao CPF pesquisado
19                return cliente
20    elif opcao == 3: # Listar clientes
21        listarClientes(clientes) # Chama a função para listar clientes
22        voltar = input('Voltar? (s/n) ') # Pergunta se deseja voltar
23        if voltar.lower() == 's': # Verifica se deseja voltar
24            return pesquisar_cliente(clientes) # Chama a função para pesquisar cliente
25        else:
26            return pesquisar_cliente(clientes)
27    elif opcao == 4: # Voltar
28        return None # Retorna None
29    else: # Opção inválida
30        print('Opção inválida. Tente novamente.')
31        return None
32
33        print("Cliente não encontrado.")
34        return None
```



```

1 # função para escolher a bebida
2 def escolherBebida():
3     bebidas = carregar_bebidas()
4     try:
5         # pergunta que tipo de bebida o cliente prefere
6         print("\nEscolha a bebida: ")
7         print("Bebidas: \n1 - Chopp \n2 - Vinho \n3 - Refrigerante")
8         opcao = int(input("Digite a opção desejada: "))
9
10        if opcao == 1:
11            opcao_volume = 1
12            print("\nSelecione o tamanho: ")
13            # mostra a opção de tamanhos para a bebida selecionada
14            for c in bebidas[0]:
15                print(str(opcao_volume) + "-" + c.volume + " - " + str(c.preco))
16                opcao_volume += 1
17
18            # pergunta a quantidade que será comprada
19            opcao_escolhida_be = int(input("Digite a opção desejada: "))
20            quantidade = int(input("Digite a quantidade: "))
21
22            if quantidade < 1:
23                print("Quantidade inválida. Tente novamente.")
24                return
25
26            #atribui o a quantidade de bebida comprada
27            bebidas[0][opcao_escolhida_be - 1].quantidade = quantidade
28            return bebidas[0][opcao_escolhida_be - 1]
29

```

```

30        elif opcao == 2:
31            opcao_volume = 1
32            print("\nSelecione o tamanho: ")
33            # mostra a opção de tamanhos para a bebida selecionada
34            for c in bebidas[1]:
35                print(str(opcao_volume) + "-" + c.volume + " - " + str(c.preco))
36                opcao_volume += 1
37
38            # pergunta a quantidade que será comprada
39            opcao_escolhida_be = int(input("Digite a opção desejada: "))
40            quantidade = int(input("Digite a quantidade: "))
41
42            if quantidade < 1:
43                print("Quantidade inválida. Tente novamente.")
44                return
45
46            #atribui o a quantidade de bebida comprada
47            bebidas[1][opcao_escolhida_be - 1].quantidade = quantidade
48            return bebidas[1][opcao_escolhida_be - 1]
49
50        elif opcao == 3:
51            opcao_volume = 1
52            print("\nSelecione o tamanho: ")
53            # mostra a opção de tamanhos para a bebida selecionada
54            for c in bebidas[2]:
55                print(str(opcao_volume) + "-" + c.volume + " - " + str(c.preco))
56                opcao_volume += 1
57
58            # pergunta a quantidade que será comprada
59            opcao_escolhida_be = int(input("Digite a opção desejada: "))
60            quantidade = int(input("Digite a quantidade: "))
61
62            if quantidade < 1:
63                print("Quantidade inválida. Tente novamente.")
64                return
65
66            #atribui o a quantidade de bebida comprada
67            bebidas[2][opcao_escolhida_be - 1].quantidade = quantidade
68            return bebidas[2][opcao_escolhida_be - 1]
69
70    except ValueError:
71        print("Opção inválida. Tente novamente.") # Mensagem de erro
72        return

```



```
1 # função para escolher uma estação
2 def escolherEstacao(estacoes):
3     print("\nSelecione a estação:")
4     opcao_estacao = 1
5     for e in estacoes:
6         print(str(opcao_estacao) + " - " + e.nome)
7         opcao_estacao += 1
8
9     try:
10        opcao = int(input('Digite a opção desejada: ')) # Captura a opção do vendedor
11        if opcao < 1 or opcao > len(estacoes):
12            print("Opção inválida. Tente novamente.") # Mensagem de erro
13        return
14
15    return estacoes[opcao - 1]
16
17 except ValueError:
18    print("Opção inválida. Tente novamente.") # Mensagem de erro
19    return
20
```



```
1 # função para auxiliar no gerenciamento de clientes
2 def gerenciarCliente(clientes, funcionarios):
3     # verifica se existe um cliente
4     if not clientes:
5         print("Nenhum cliente cadastrado. Crie um cliente primeiro.")
6         return
7
8     try: # tratamento de exceção
9         cliente = pesquisar_cliente(clientes) # pesquisa o cliente
10
11    if cliente is None: # verifica se o cliente é nulo
12        print("Cliente não encontrado.") # mensagem de erro
13        return
14
15    if not cliente.comandas: # verifica se o cliente possui comandas
16        print("Nenhuma comanda encontrada para o cliente.")
17        return
18
19    cliente.exibirComanda() # exibe a comanda do cliente
20
21    print('\nQuitar conta do cliente:\n1 - sim\n2 - não') # pergunta se deseja quitar a conta do cliente
22    opcao = int(input("\nSelecione uma opção: "))
23    comanda = cliente.comandas[0] # seleciona a comanda do cliente
24    if opcao == 1: # verifica se a opção é 1
25        cliente.comandas.remove(comanda) # remove a comanda do cliente
26        print("\nConta quitada com sucesso")
27    else: # se não
28        print("\nConta não quitada")
29    except ValueError: # tratamento de exceção
30        print("Opção inválida. Tente novamente.")
31    return
```



```
1 # função para listar clientes
2 def listarClientes(clientes):
3     if not clientes: # Verifica se a lista de clientes está vazia
4         print("Nenhum cliente cadastrado.")
5     return
6
7     print("Lista de clientes:")
8     for cliente in clientes: # Percorre a lista de clientes
9         if isinstance(cliente, Cliente): # Verifica se o objeto é uma instância da classe Cliente
10            print(f"Nome: {cliente.nome}")
11
12
13 #funcao para auxiliar a exibir todas as coimandas
14 def mostrarComandas(funcionarios):
15     funcionario = pesquisar_funcionario(funcionarios)
16
17     if not funcionario: # se não existir funcionario retornar
18         return
19
20     if not comandasGeral: # se não existir comandas retornar
21         print("Nenhuma comanda encontrada.")
22     return
23
24     funcionario.exibirComanda() # exibir todas as comandas
25
```



```
1 # Menu principal
2 def menu():
3     estacoesRef = estacoes() # Carrega as estações
4     bebidas = carregar_bebidas() # Carrega as bebidas por estação
5     clientes = [] # Inicializa lista de clientes
6     funcionarios = [] # Inicializa lista de funcionários
7
8     criar_administrador_padrao(funcionarios) # Cria um administrador padrão
9     criar_Cliente_teste(clientes) # Cria um cliente teste
10    while True:
11        print('\nSISTEMA DRINKEASY') # Título do sistema
12        print('1 - Nova venda') # Opção para nova venda
13        print('2 - Cadastrar cliente') # Opção para cadastrar cliente
14        print('3 - Cadastrar funcionário') # Opção para cadastrar funcionário
15        print('4 - Criar nova estação') # Opção para criar uma nova estação
16        print('5 - Gerenciar cliente') # Opção para gerenciar a situação do cliente
17        print('6 - exibir comandas') # opcao para gerenciar todas as comandas
18        print('0 - Sair') # Opção para sair
19
20    try:
21        opcao = int(input('Digite a opção desejada: ')) # Captura a opção do usuário
22    except ValueError:
23        print("Opção inválida. Tente novamente.") # Mensagem de erro
24        continue
25
26    if opcao == 1:
27        nova_venda(clientes,funcionarios, estacoesRef) # Chama a função de nova venda
28
```

```
29 elif opcao == 2:  
30     print('\nCADASTRO DE CLIENTE') # Início do cadastro de cliente  
31     nome = input('Nome do cliente: ') # Captura o nome do cliente  
32     cpf = input('CPF do cliente: ') # Captura o CPF do cliente  
33     telefone = input('Telefone do cliente: ') # Captura o telefone do cliente  
34     endereco = input('Endereço do cliente: ') # Captura o endereço do cliente  
35     cliente = Cliente(nome, cpf, telefone, endereco, comandas = []) # Cria um novo cliente  
36     clientes.append(cliente) # Adiciona o cliente à lista  
37     print(f"Cliente {nome} cadastrado com sucesso!") # Mensagem de sucesso  
38  
39 elif opcao == 3:  
40     print('\nCADASTRO DE FUNCIONÁRIO') # Início do cadastro de funcionário  
41     nome = input('Nome do funcionário: ') # Captura o nome do funcionário  
42     cpf = input('CPF do funcionário: ') # Captura o CPF do funcionário  
43     telefone = input('Telefone do funcionário: ') # Captura o telefone do funcionário  
44     endereco = input('Endereço do funcionário: ') # Captura o endereço do funcionário  
45     cargo = input('Cargo do funcionário: ') # Captura o cargo do funcionário  
46     funcionario = Funcionario(nome, cpf, telefone, endereco, cargo) # Cria um novo funcionário  
47     funcionarios.append(funcionario) # Adiciona o funcionário à lista  
48     print(f"Funcionário {nome} cadastrado com sucesso!") # Mensagem de sucesso  
49  
50 elif opcao == 4:  
51     print('\nNOVA ESTAÇÃO') # Início do cadastro de nova estação  
52     nome = input('Nome da estação: ') # Captura o nome da estação  
53     estacao = Estacao(nome) # Cria uma nova estação  
54     estacoesRef.append(estacao) # Adiciona a estação à lista  
55     print('Estação cadastrada com sucesso!') # Mensagem de sucesso
```

```
56 elif opcao == 5:  
57     print('\nGERENCIAR CLIENTE') # Início do gerenciamento de cliente  
58     gerenciarCliente(clientes, funcionarios)  
59  
60 elif opcao == 6:  
61     print('\nEXIBIR COMANDAS') # Início do gerenciamento de  
62     mostrarComandas(funcionarios)  
63  
64 elif opcao == 0:  
65     print('Saindo...') # Mensagem de saída  
66     break # Encerra o loop  
67 else:  
68     print('Opção inválida. Tente novamente.') # Mensagem de erro  
69  
70 menu()
```

# Demonstração do código

Em ação



# Obrigado

