
Salvar e Recuperar Transações em/de arquivo csv

Vamos modificar a classe GerenciadorFinanceiro para incluir os métodos de salvar e carregar.

1. Importando o Módulo csv

Primeiro, adicione esta linha no topo do seu arquivo financas.py:

Python

```
# financas.py
import csv
import os # O módulo 'os' nos ajuda a verificar se um arquivo existe
```

2. Atualizando a Classe GerenciadorFinanceiro

Agora, vamos adicionar os métodos salvar_transacoes e carregar_transacoes e modificar o construtor __init__ para chamar o carregamento.
Substitua a sua classe GerenciadorFinanceiro atual por esta versão atualizada:

Python

```
# controleFinanceiro.py (altere a classe ControleFinanceiro)

class ControleFinanceiro:
    """
    Gerencia uma coleção de transações financeiras,
    com capacidade de salvar e carregar de um arquivo CSV.
    """
    def __init__(self, nome_arquivo='transacoes.csv'):
        """
        Construtor da classe ControleFinanceiro.
        """
        self._transacoes = []
        self._nome_arquivo = nome_arquivo
        # O encapsulamento é aplicado aqui. A lista de transações e o nome do arquivo
```

```
    # são "protegidos" dentro do objeto.
    self.carregar_transacoes() # <-- NOVO: Carrega os dados ao iniciar

def adicionar_transacao(self, transacao: Transacao):
    """
    Adiciona uma nova transação à lista e salva no arquivo.
    """
    if isinstance(transacao, Transacao):
        self._transacoes.append(transacao)
        self.salvar_transacoes() # <-- NOVO: Salva a lista após adicionar
        print("Transação adicionada e salva com sucesso!")
    else:
        print("Erro: Apenas objetos do tipo Transacao podem ser adicionados.")

def listar_transacoes(self):
    """
    Exibe todas as transações registradas.
    """
    if not self._transacoes:
        print("\nNenhuma transação registrada.")
        return

    print("\n--- Extrato Financeiro ---")
    for transacao in self._transacoes:
        print(transacao)
    print("--------------------------")

def calcular_saldo(self) -> float:
    """
    Calcula e retorna o saldo total (receitas - despesas).
    """
    saldo = 0.0
    for transacao in self._transacoes:
        if transacao.tipo.lower() == 'receita':
            saldo += transacao.valor
        else:
            saldo -= transacao.valor
    return saldo

def salvar_transacoes(self):
    """
    Salva a lista de transações em um arquivo CSV.
    """
```

```
    try:
        with open(self._nome_arquivo, 'w', newline='', encoding='utf-8') as arquivo_csv:
            # O cabeçalho do nosso CSV
            cabecalho = ['descricao', 'valor', 'tipo']
            escritor = csv.DictWriter(arquivo_csv, fieldnames=cabecalho)

            escritor.writeheader() # Escreve o cabeçalho no arquivo
            for transacao in self._transacoes:
                # Usamos vars() para converter os atributos do objeto em um dicionário
                escritor.writerow(vars(transacao))
    except IOError as e:
        print(f"Erro ao salvar o arquivo: {e}")

def carregar_transacoes(self):
    """
    Carrega as transações de um arquivo CSV para a lista.
    """
    # Verifica se o arquivo existe antes de tentar abri-lo
    if not os.path.exists(self._nome_arquivo):
        print("Arquivo de transações não encontrado. Começando um novo.")
        return # Sai da função se o arquivo não existe

    try:
        with open(self._nome_arquivo, 'r', newline='', encoding='utf-8') as arquivo_csv:
            leitor = csv.DictReader(arquivo_csv)
            self._transacoes = [] # Limpa a lista atual antes de carregar
            for linha in leitor:
                # Recria o objeto Transacao a partir dos dados do CSV
                # Precisamos converter o valor de volta para float
                transacao = Transacao(
                    descricao=linha['descricao'],
                    valor=float(linha['valor']),
                    tipo=linha['tipo']
                )
                self._transacoes.append(transacao)
        print("Transações carregadas com sucesso!")
    except FileNotFoundError:
        # Esta exceção é um seguro extra, embora o os.path.exists já verifique
        print("Arquivo de transações não encontrado. Começando um novo.")
    except Exception as e:
        print(f"Erro ao carregar o arquivo: {e}")
        # Se o arquivo estiver corrompido, começamos do zero para evitar erros
        self._transacoes = []
```

Análise das Mudanças

1. **__init__(self, nome_arquivo='transacoes.csv')**:
    - O construtor agora aceita um nome_arquivo como argumento (com um valor padrão transacoes.csv).
    - Ele chama self.carregar_transacoes() assim que um objeto GerenciadorFinanceiro é criado.
2. **adicionar_transacao(...)**:
    - A única mudança é a chamada a self.salvar_transacoes() logo após adicionar uma nova transação à lista. Isso garante que cada nova entrada seja imediatamente salva no disco.
3. **salvar_transacoes(self) (Novo Método)**:
    - with open(...): Abre o arquivo em modo de escrita ('w'). O with garante que o arquivo seja fechado corretamente, mesmo que ocorram erros.
    - newline='' e encoding='utf-8': Parâmetros importantes para evitar problemas de formatação de linhas e garantir a compatibilidade com acentos e caracteres especiais.
    - csv.DictWriter: É um escritor de CSV que trabalha com dicionários. Isso é muito conveniente, pois podemos mapear diretamente os atributos do nosso objeto para as colunas do CSV.
    - escritor.writeheader(): Escreve a primeira linha do arquivo com os nomes das colunas (descricao, valor, tipo).
    - escritor.writerow(vars(transacao)): Para cada objeto transacao na nossa lista, a função vars() o converte em um dicionário (ex: {'descricao': 'Salário', 'valor': 5000.0, 'tipo': 'Receita'}). O writerow então escreve esses valores no arquivo.
4. **carregar_transacoes(self) (Novo Método)**:
    - os.path.exists(self._nome_arquivo): Antes de qualquer coisa, verificamos se o arquivo realmente existe. Se não, evitamos um erro e simplesmente começamos com uma lista vazia.
    - with open(..., 'r'): Abre o arquivo em modo de leitura ('r').
    - csv.DictReader: Lê o arquivo CSV e trata cada linha como um dicionário, usando o cabeçalho para criar as chaves. Isso facilita muito o acesso aos dados (linha['descricao'], linha['valor'], etc.).
    - self._transacoes = []: Limpamos a lista em memória para garantir que não haja duplicatas ao carregar do arquivo.
    - transacao = Transacao(...): Para cada linha lida do arquivo, nós recriamos um objeto Transacao e o adicionamos à lista self._transacoes. Note a conversão float(linha['valor']), pois os dados lidos de um arquivo são sempre texto.