UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
PROFESSOR ANDRÉ FELIPE BÜRGER
PROGRAMAÇÃO ORIENTADA A OBJETOS

Trabalho Final

Implementar um programa, com interface gráfica de usuário (GUI) utilizando Java Swing, para ser utilizado no controle de estoque de uma loja de informática.
O sistema deve permitir registrar entradas e saídas de produtos do estoque, acompanhando a quantidade disponível e o valor total dos itens armazenados.

As entradas de produtos representam todos os itens que chegam ao estoque.

As saídas de produtos representam as vendas ou retiradas de itens do estoque e são classificadas em:

- vendas ao cliente,
- uso interno,
- devoluções a fornecedores, e
- outras saídas.

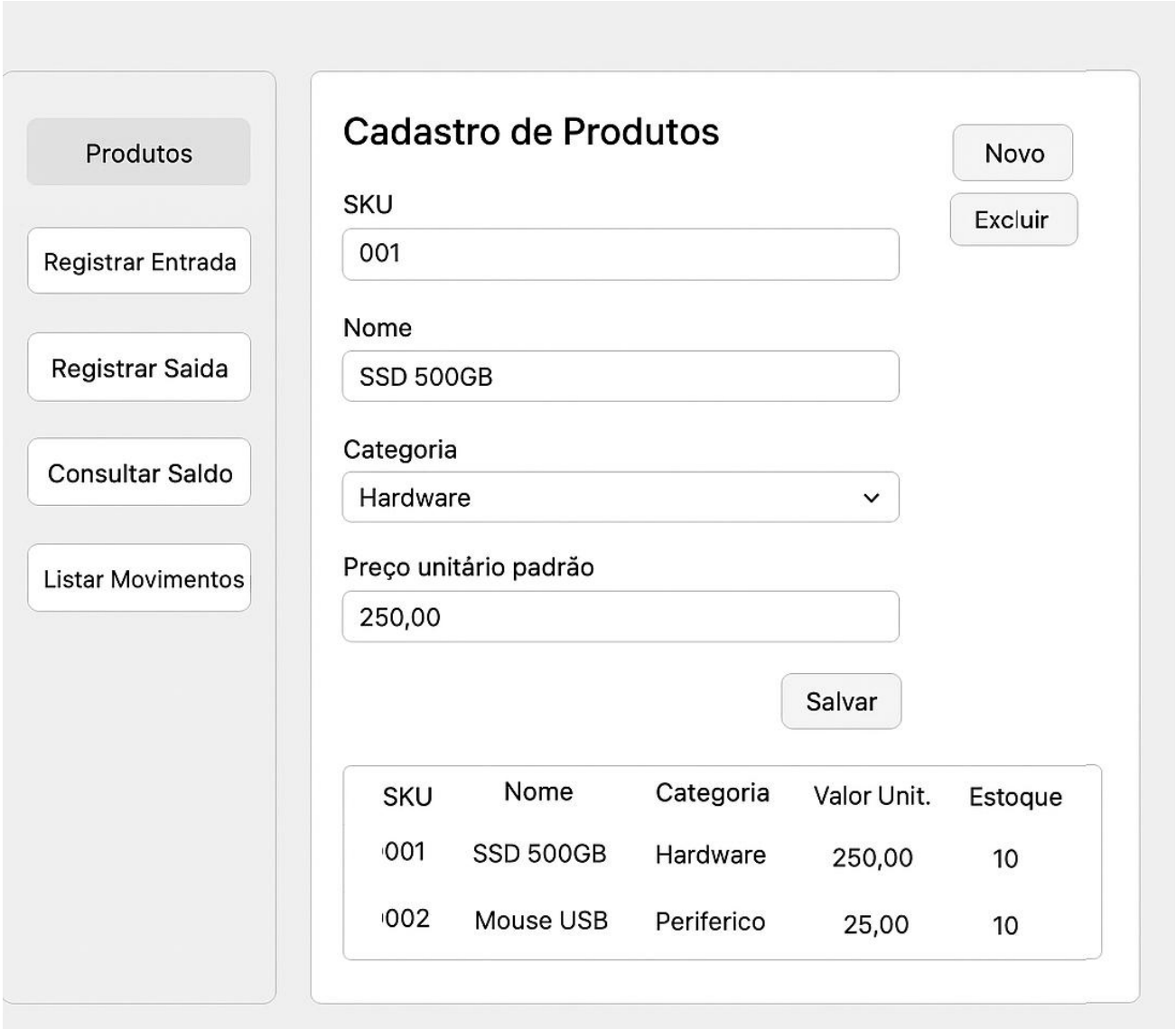
O programa deve atender aos seguintes requisitos funcionais (o programa deve permitir):

1. Cadastro de produtos contendo código, nome, preço unitário e quantidade em estoque e categorias:
   - componentes de hardware (como processadores, placas-mãe, memórias, SSDs, etc.),
   - periféricos (como teclados, mouses, monitores),
   - acessórios (como cabos, adaptadores, suportes) e
   - outros produtos.
2. Registrar entrada de produtos no estoque, informando o produto, data de entrada, quantidade e valor unitário;
3. Registrar saída de produtos, informando o produto, data da saída e quantidade retirada;
4. Consultar o saldo atual (quantidade e valor total) de produtos em estoque até a data atual;
5. Consultar o saldo total do estoque, no período informado;
6. Listar todas as entradas registradas;
7. Listar todas as saídas registradas;
8. Listar todos os movimentos de estoque (entradas e saídas) ordenados por data. A cada movimento, exibir como aquele lançamento impactou no saldo (quantidade e valor total), simulando um extrato de movimentação.

Também devem ser atendidos aos seguintes requisitos não funcionais:

- O sistema deve utilizar conceitos de herança e interfaces;
- Os dados submetidos pelo usuário devem ser gravados em disco e recuperados automaticamente quando o programa for iniciado. Os arquivos de dados devem ser formatados em arquivo CSV (.csv) ou binário.

Exemplo de interface gráfica



MÉTODO DE AVALIAÇÃO

É esperado que seja construído:

- **[2,0 pontos]** Diagrama de classes detalhando todas as classes e relacionamentos, com código fonte compatível com o diagrama. O diagrama de classes deve conter apenas as classes da camada de negócios. Utilizar a linguagem UML para desenhar o diagrama. O software para desenhar o diagrama de classes é de livre escolha da equipe, deixo a sugestão do draw.io.
- **[5,0 pontos]** Implementação do programa, atendendo aos requisitos funcionais e construído utilizando-se a arquitetura em duas camadas. As classes da camada de negócio devem ser devidamente documentadas com o estilo **javadoc**;
- **[1,0 ponto]** Construção de um plano de testes para validar as classes da camada de negócios;
- **[2,0 pontos]** Implementação do plano de testes através do framework JUnit. Todos os métodos públicos (que não são *getter* e *setter*) devem possuir testes.

APRESENTAÇÃO

Os trabalhos deverão ser obrigatoriamente apresentados à turma na última semana de aula.

Cada equipe terá um tempo máximo de 15 minutos para apresentar os seguintes itens:

- Diagrama de Classes
- Principais rotinas do código-fonte
- Plano de Testes
- Execução dos Testes
- Sistema funcionando

Durante a apresentação, deverão ser destacados os principais desafios enfrentados e as decisões tomadas na estruturação do código.

A nota da apresentação será individual, e será realizada uma média entre a nota da apresentação e a nota do trabalho, que comporá a nota final do trabalho.

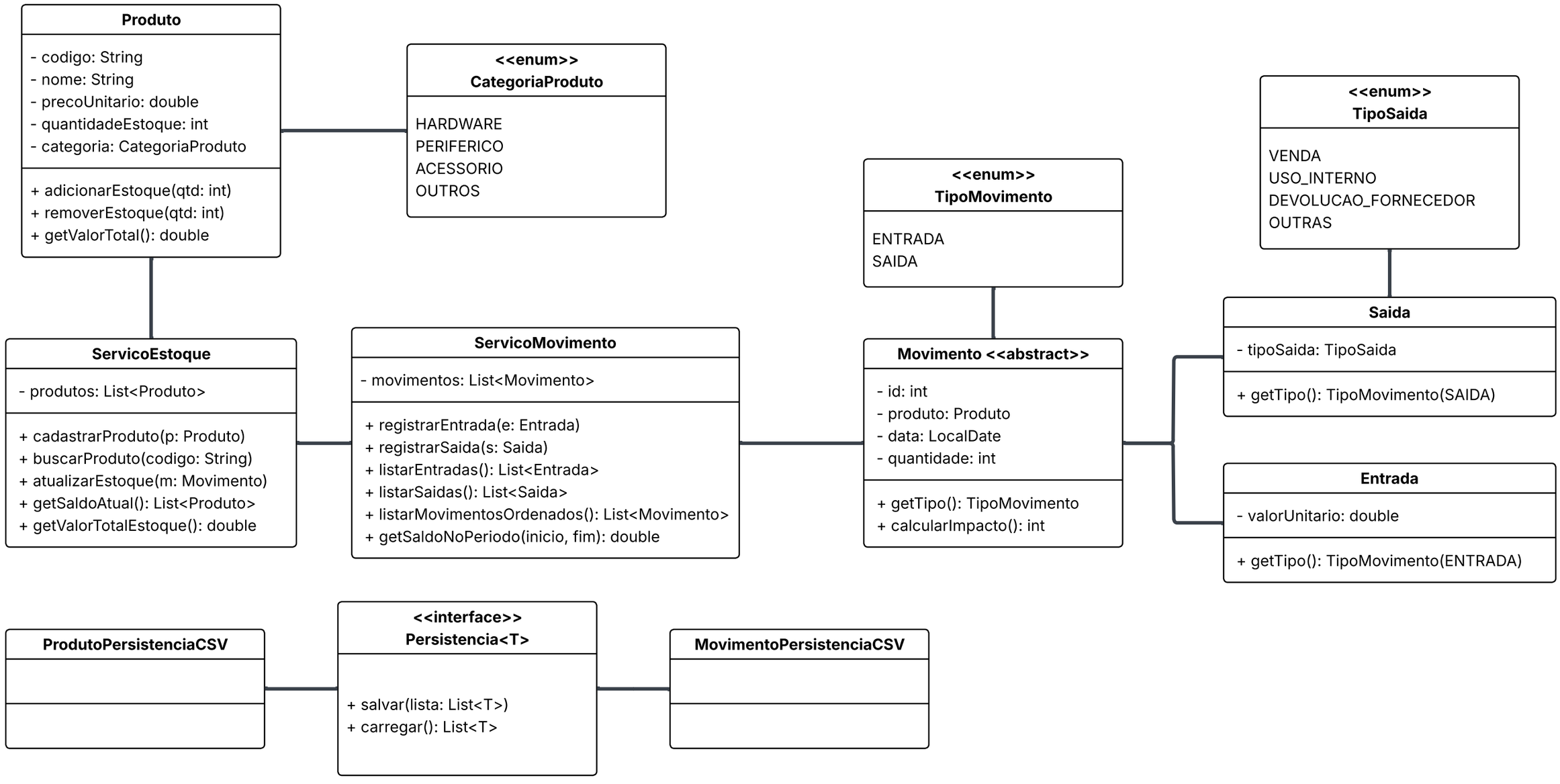
MATERIAL A SER SUBMETIDO NO AVA

No AVA deverá ser submetido:

- O projeto contendo os arquivos fontes. (Não utilizar bibliotecas de terceiros);
- O plano de testes;
- A implementação do plano de testes.
- O Diagrama de Classes UML (Enviar no formato pdf).

RESTRIÇÕES E PRAZO

- O trabalho deve ser feito em equipe de no máximo 4 pessoas.
- O trabalho deve ser entregue até o dia 23/11/2025 às 23:59.
- O trabalho deve ser apresentado para a turma na última semana de aula (26/11 e 27/11).


Produto
- codigo: String
- nome: String
- precoUnitario: double
- quantidadeEstoque: int
- categoria: CategoriaProduto
+ adicionarEstoque(qtd: int)
+ removerEstoque(qtd: int)
+ getValorTotal(): double
<<enum>>
CategoriaProduto
HARDWARE
PERIFERICO
ACESSORIO
OUTROS
<<enum>>
TipoMovimento
ENTRADA
SAIDA
<<enum>>
TipoSaida
VENDA
USO_INTERNO
DEVOLUCAO_FORNECEDOR
OUTRAS
ServicoEstoque
- produtos: List<Produto>
+ cadastrarProduto(p: Produto)
+ buscarProduto(codigo: String)
+ atualizarEstoque(m: Movimento)
+ getSaldoAtual(): List<Produto>
+ getValorTotalEstoque(): double
ServicoMovimento
- movimentos: List<Movimento>
+ registrarEntrada(e: Entrada)
+ registrarSaida(s: Saida)
+ listarEntradas(): List<Entrada>
+ listarSaidas(): List<Saida>
+ listarMovimentosOrdenados(): List<Movimento>
+ getSaldoNoPeriodo(inicio, fim): double
Movimento <<abstract>>
- id: int
- produto: Produto
- data: LocalDate
- quantidade: int
+ getTipo(): TipoMovimento
+ calcularImpacto(): int
Saida
- tipoSaida: TipoSaida
+ getTipo(): TipoMovimento(SAIDA)
Entrada
- valorUnitario: double
+ getTipo(): TipoMovimento(ENTRADA)
ProdutoPersistenciaCSV
<<interface>>
Persistencia<T>
+ salvar(lista: List<T>)
+ carregar(): List<T>
MovimentoPersistenciaCSV

A seguir está **um diagrama de classes completo da camada de negócio**, totalmente alinhado aos requisitos do trabalho.
Vou te entregar:

1. **Descrição completa das classes**
2. **Relacionamentos (herança, composição, associação)**
3. **Diagrama UML em texto** (você pode copiar para o *draw.io*)

Se quiser, posso gerar a imagem final em UML também.

✅ Diagrama de Classes – Camada de Negócios

1. Classes Principais

Produto

Representa qualquer item do estoque.

Atributos

- `codigo: String`
- `nome: String`
- `precoUnitario: double`
- `quantidadeEstoque: int`
- `categoria: CategoriaProduto`

Métodos

- `adicionarEstoque(qtd: int)`

- `removerEstoque(qtd: int)`
- `getValorTotal(): double`

CategoriaProduto (Enum)

- HARDWARE
- PERIFERICO
- ACESSORIO
- OUTROS

---

2. Movimentação de Estoque (Polimorfismo + Herança)

Movimento (classe abstrata)

Representa qualquer registro de entrada ou saída.

Atributos

- `id: int`
- `produto: Produto`
- `data: LocalDate`
- `quantidade: int`

Métodos abstratos

- `getTipo(): TipoMovimento`
- `calcularImpacto(): int`
  → Retorna quantidade positiva (entrada) ou negativa (saída)

Entrada extends Movimento

Atributos adicionais

- `valorUnitario: double`

Métodos

- `getTipo()` retorna ENTRADA

Saida extends Movimento

Atributos adicionais

- `tipoSaida: TipoSaida`

Métodos

- `getTipo()` retorna SAIDA

TipoMovimento (Enum)

- ENTRADA
- SAIDA

TipoSaida (Enum)

- VENDA
- USO_INTERNO
- DEVOLUCAO_FORNECEDOR
- OUTRAS

---

3. Serviços / Controladores de Negócio

EstoqueService

(Lida com produtos e alterações de estoque)

Atributos

- `produtos: List<Produto>`

Métodos

- `cadastrarProduto(p: Produto)`
- `buscarProduto(codigo: String)`
- `atualizarEstoque(movimento: Movimento)`
- `getSaldoAtual() : List<Produto>`
- `getValorTotalEstoque(): double`

MovimentoService

(Manipula entradas e saídas)

Atributos

- `movimentos: List<Movimento>`

Métodos

- `registrarEntrada(entrada: Entrada)`
- `registrarSaida(saida: Saida)`
- `listarEntradas()`
- `listarSaidas()`

- `listarMovimentosOrdenados()`
- `getSaldoNoPeriodo(dataInicial, dataFinal)`

---

4. Persistência (Interface para cumprir requisito não-funcional)

Persistencia (Interface)

Métodos

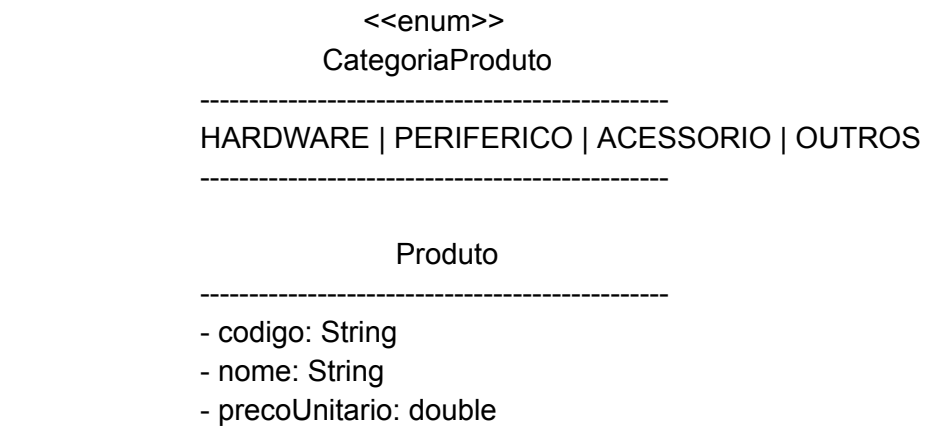
- `salvar(lista: List<T>)`
- `carregar(): List<T>`

ProdutoPersistenciaCSV implements Persistencia

MovimentoPersistenciaCSV implements Persistencia

(Atendem ao requisito de gravar CSV ou binário)

📘 UML – Diagrama em Texto (para colar no Draw.io)



- quantidadeEstoque: int
- categoria: CategoriaProduto

+ adicionarEstoque(qtd: int): void
+ removerEstoque(qtd: int): void
+ getValorTotal(): double

Movimento <<abstract>>

- id: int
- produto: Produto
- data: LocalDate
- quantidade: int

+ getTipo(): TipoMovimento
+ calcularImpacto(): int

Entrada ----|> Movimento

- valorUnitario: double

+ getTipo(): TipoMovimento(ENTRADA)

Saida ----|> Movimento

- tipoSaida: TipoSaida

+ getTipo(): TipoMovimento(SAIDA)

<<enum>> TipoSaida

VENDA | USO_INTERNO | DEVOLUCAO_FORNECEDOR | OUTRAS

<<enum>> TipoMovimento

ENTRADA | SAIDA

EstoqueService

- produtos: List<Produto>

+ cadastrarProduto(p: Produto)
+ buscarProduto(codigo: String)
+ atualizarEstoque(m: Movimento)

+ getSaldoAtual(): List<Produto>
+ getValorTotalEstoque(): double

MovimentoService

- movimentos: List<Movimento>

+ registrarEntrada(e: Entrada)
+ registrarSaida(s: Saida)
+ listarEntradas(): List<Entrada>
+ listarSaidas(): List<Saida>
+ listarMovimentosOrdenados(): List<Movimento>
+ getSaldoNoPeriodo(inicio, fim): double

Persistencia<T> <<interface>>

+ salvar(lista: List<T>)
+ carregar(): List<T>

ProdutoPersistenciaCSV ----|> Persistencia
MovimentoPersistenciaCSV ----|> Persistencia
