Pontifícia Universidade Católica de Minas Gerais
Bacharelado em Engenharia de Software - Campus Lourdes
Algoritmos e Estruturas de Dados II (Laboratório)
Profs. Ariane Carla e Luiz Henrique - **Semestre** 2/2025
Oficina: revisão e nivelamento (Dados em arquivos-texto)

Este documento é um guia da atividade desenvolvida em conjunto no laboratório de AED-II. Nesta atividade utilizamos as classes previamente desenvolvidas para carregar e armazenar dados de forma simples em um arquivo texto semiestruturado. A atividade é individual e seu conteúdo servirá como base para atividades avaliativas futuras. Portanto, atenção ao que será desenvolvido em aula e procure fazer sua parte na tarefa, enviando as atualizações para o repositório GitHub da atividade.

Tema: Armazenando dados de Produtos

Temos as classes básicas para que o nosso cliente possa registrar vendas de produtos perecíveis e não perecíveis. Porém, ainda está nos faltando a lógica do sistema principal. Além disso, nos falta também uma maneira simples de fazer os dados serem ***persistentes****, ou seja, que eles sejam armazenados antes da finalização do sistema e carregados no momento de sua inicialização. Foi definido que os arquivos de dados terão a seguinte estrutura:*
N
tipo; descrição;preçoDeCusto;margemDeLucro;[dataDeValidade]

*A primeira linha contém um número inteiro N > 0, indicando quantos produtos vêm a seguir. Na sequência, uma linha com a informação de cada produto. O tipo é "**1**" para não perecíveis e "**2**" para perecíveis. A informação de data de validade só existe nos produtos perecíveis e estará no formato dd/mm/aaaa.*

Preparação:
Na classe abstrata Produto, crie o método abaixo para permitir saber se dois produtos são iguais por meio de sua descrição:

```
/**
 * Igualdade de produtos: caso possuam o mesmo nome/descrição.
 * @param obj Outro produto a ser comparado
 * @return booleano true/false conforme o parâmetro possua a descrição igual ou não a este produto.
 */
@Override
public boolean equals(Object obj){
  Produto outro = (Produto)obj;
  return this.descricao.toLowerCase().equals(outro.descricao.toLowerCase());
}
```

Tarefa 2 (em aula):
Criar métodos nas classes existentes para criar objetos a partir das linhas de dados especificadas, bem como para gerar linhas de dados a partir de um objeto.

Para realizar essa tarefa, a **classe abstrata Produto** deverá possuir mais dois métodos, além dos já existentes, desenvolvidos na Oficina anterior. O primeiro método consiste apenas na assinatura do método abstrato abaixo:

```
/**
 * Gera uma linha de texto a partir dos dados do produto
 * @return Uma string no formato "tipo; descrição;preçoDeCusto;margemDeLucro;[dataDeValidade]"
 */
public abstract String gerarDadosTexto();
```

Já o segundo método que a classe abstrata Produto terá, deverá ser desenvolvido por você, seguindo a estrutura abaixo:

```
/**
 * Cria um produto a partir de uma linha de dados em formato texto. A linha de dados deve estar de acordo com a formatação
 * "tipo; descrição;preçoDeCusto;margemDeLucro;[dataDeValidade]"
 * ou o funcionamento não será garantido. Os tipos são 1 para produto não perecível e 2 para perecível.
 * @param linha Linha com os dados do produto a ser criado.
```

```
    * @return Um produto com os dados recebidos
    */
   static Produto criarDoTexto(String linha){
      Produto novoProduto = null;
      /*Você deve implementar aqui a lógica que separa os dados existentes na String linha, verifica se o produto é do tipo 1 ou 2 e constrói o objeto adequado, com os dados fornecidos de acordo com seu tipo. O objeto construído é retornado pelo método*/
      return novoProduto;
   }
```

Feito isso, você precisará implementar os métodos gerarDadosTexto() nas duas classes filhas (ProdutoNaoPerecivel e ProdutoPerecivel), como na estrutura abaixo:

```
   /**
    * Gera uma linha de texto a partir dos dados do produto. Preço e margem de lucro vão formatados com 2 casas decimais.
    * @return Uma string no formato "1; descrição;preçoDeCusto;margemDeLucro"
    */
   @Override
   public String gerarDadosTexto() {
      /*Você deve implementar aqui a lógica que monta a String com os atributos do objeto ProdutoNaoPerecivel, respeitando o formato do arquivo de dados. */
   }

   /**
    * Gera uma linha de texto a partir dos dados do produto. Preço e margem de lucro vão formatados com 2 casas decimais.
    * Data de validade vai no formato dd/mm/aaaa
    * @return Uma string no formato "2; descrição;preçoDeCusto;margemDeLucro;dataDeValidade"
    */
   @Override
   public String gerarDadosTexto() {
      /*Você deve implementar aqui a lógica que monta a String com os atributos do objeto ProdutoPerecivel, respeitando o formato do arquivo de dados. */
   }
```

Tarefa 3 (em aula):
Criar um **programa principal** que consiga carregar vários Produtos (de qualquer tipo) para um vetor. O programa deve permitir localizar produtos e armazenar os produtos no arquivo ao final de sua execução.

Dessa forma, sua classe principal deverá ter uma estrutura parecida com a que se segue.

```
public class Comercio {
   /** Para inclusão de novos produtos no vetor */
   static final int MAX_NOVOS_PRODUTOS = 10;

   /** Nome do arquivo de dados. O arquivo deve estar localizado na raiz do projeto */
   static String nomeArquivoDados;

   /** Scanner para leitura do teclado */
   static Scanner teclado;

   /** Vetor de produtos cadastrados. Sempre terá espaço para 10 novos produtos a cada execução */
   static Produto[] produtosCadastrados;

   /** Quantidade produtos cadastrados atualmente no vetor */
   static int quantosProdutos;
```

```
/** Gera um efeito de pausa na CLI. Espera por um enter para continuar */
static void pausa(){
   System.out.println("Digite enter para continuar...");
   teclado.nextLine();
}

/** Cabeçalho principal da CLI do sistema */
static void cabecalho(){
   System.out.println("AEDII COMÉRCIO DE COISINHAS");
   System.out.println("==========================");
}

/** Imprime o menu principal, lê a opção do usuário e a retorna (int).
 * Perceba que poderia haver uma melhor modularização com a criação de uma classe Menu.
 * @return Um inteiro com a opção do usuário.
*/
static int menu(){
   cabecalho();
   System.out.println("1 - Listar todos os produtos");
   System.out.println("2 - Procurar e listar um produto");
   System.out.println("3 - Cadastrar novo produto");
   System.out.println("0 - Sair");
   System.out.print("Digite sua opção: ");
   return Integer.parseInt(teclado.nextLine());
}

/**
 * Lê os dados de um arquivo texto e retorna um vetor de produtos. Arquivo no formato
 * N  (quantiade de produtos) <br/>
 * tipo; descrição;preçoDeCusto;margemDeLucro;[dataDeValidade] <br/>
 * Deve haver uma linha para cada um dos produtos. Retorna um vetor vazio em caso de problemas com o arquivo.
 * @param nomeArquivoDados Nome do arquivo de dados a ser aberto.
 * @return Um vetor com os produtos carregados, ou vazio em caso de problemas de leitura.
 */
static Produto[] lerProdutos(String nomeArquivoDados) {
   Produto[] vetorProdutos;
      /*Ler a primeira linha do arquivoDados contendo a quantidade de produtos armazenados no arquivo. Instanciar o vetorProdutos com o tamanho necessário para acomodar todos os produtos do arquivo + o espaço reserva MAX_NOVOS_PRODUTOS. Após isso, ler uma após a outra o restante das linhas do arquivo, convertendo, a cada leitura de linha, seus dados em objetos do tipo Produto (utilizar o método criarDoTexto()). Cada objeto Produto instanciado será armazenado no vetorProdutos.*/
   return vetorProdutos;
}

/** Lista todos os produtos cadastrados, numerados, um por linha */
static void listarTodosOsProdutos(){
      /*Percorrer o vetor de produtosCadastrados, escrevendo na tela os dados de cada um (utilizar o método toString() já implementado).*/
}

/** Localiza um produto no vetor de cadastrados, a partir do nome (descrição), e imprime seus dados.
 *  A busca não é sensível ao caso.  Em caso de não encontrar o produto, imprime mensagem padrão */
static void localizarProdutos(){
      /*Ler do teclado a descrição (nome) do produto que o usuário deseja localizar, procurar no vetor de produtosCadastrados o produto em questão (utilizar o método equals() já implementado na classe Produto) e imprimir na tela seus dados.*/
}

/**
 * Rotina de cadastro de um novo produto: pergunta ao usuário o tipo do produto, lê os dados correspondentes,
```

```
   * cria o objeto adequado de acordo com o tipo, inclui no vetor. Este método pode ser feito com um nível muito
   * melhor de modularização. As diversas fases da lógica poderiam ser encapsuladas em outros métodos.
    * Uma sugestão de melhoria mais significativa poderia ser o uso de padrão Factory Method para criação dos objetos.
   */
  static void cadastrarProduto(){
        /*Implementar a sub-rotina de exibir o novo menu para cadastro de novo produto, ler os dados necessários conforme o tipo desejado, criar o objeto correspondente, salvando-o no vetor de produtosCadastrados e incrementando a variável de controle da quantidade de produtos.*/
  }

  /**
   * Salva os dados dos produtos cadastrados no arquivo csv informado. Sobrescreve todo o conteúdo do arquivo.
   * @param nomeArquivo Nome do arquivo a ser gravado.
   */
  public static void salvarProdutos(String nomeArquivo){
        /*Você deve implementar aqui a lógica que abrirá um arquivo para escrita com o nome informado no parâmetro, percorrerá um por um todos os produtos existentes no vetor de produtosCadastrados, gerando uma linha de texto com os dados de cada objeto Produto, escrevendo-a no arquivo.*/
  }

  public static void main(String[] args) throws Exception {
    teclado = new Scanner(System.in, Charset.forName("ISO-8859-2"));
    nomeArquivoDados = "dadosProdutos.csv";
    produtosCadastrados = lerProdutos(nomeArquivoDados);
    int opcao = -1;
    do{
      opcao = menu();
      switch (opcao) {
        case 1 -> listarTodosOsProdutos();
        case 2 -> localizarProdutos();
        case 3 -> cadastrarProduto();
      }
      pausa();
    }while(opcao !=0);

    salvarProdutos(nomeArquivoDados);
    teclado.close();
  }
}
```

Tarefa 4 (a ser feita até 25/08):
Incluir, no programa principal, as opções de listar todos os produtos cadastrados e de cadastrar novos produtos.

Instruções e observações:

- O projeto deve estar hospedado na tarefa correspondente do GitHub Classroom. Endereço para aceitar a tarefa: https://classroom.github.com/a/m3QV2JU_
- As atividades pontuadas da disciplina podem depender direta ou indiretamente dos códigos desenvolvidos nas aulas, portanto, é essencial o comprometimento no acompanhamento das atividades semanais;
- Para a correção das atividades pontuais, serão considerados todo os *commits/pushes* realizados ao longo das semanas, não somente o último com a resposta final do exercício.