PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 15

Tema: Introdução à programação VI
Atividade: Orientação a objetos - Classes

01.) Contexto:

Uma empresa que comercializa tintas tem várias lojas distribuídas pela cidade. Para facilitar iniciar sua gestão de estoque unificado necessita de um sistema para integrar os estoques distribuídos entre suas diversas lojas.

02.) Objetivo do sistema:

Uma parte do sistema de estoque unificado é lidar com as transações diárias. Arquivos separados por operações para inserir, atualizar e remover dados do arquivo central com o estoque unificado são mantidos e consolidados diariamente. No arquivo para inserções estão lançadas todas as novas inclusões para serem atualizadas no estoque, com suas respectivas indicações de código, nome, quantidade e preço unitário de cada produto.
No arquivo para atualizações constam todas as baixas de produtos que precisarão ser alteradas no arquivo central.
No arquivo para remoções estão lançados todos os produtos, cujas indicações não mais deverão constar do estoque.

Requisitos do sistema

O subsistema para atualização do controle de estoque deverá, diariamente, consultar os arquivos relacionados acima, realizar as operações descritas e consolidar uma nova situação do arquivo central para início de trabalhos no dia seguinte.

O subsistema fará parte de um conjunto que permitirá um melhor planejamento da gestão do estoque unificado. Outros subsistemas serão desenvolvidos ao mesmo tempo e deverão manter características comuns a saber:

- implementações em C++, com classes para

  - tratamento de erros,
  - descrição de produto
  - unidades de testes

- desenvolvimento em bibliotecas separadas por classes

Modelos iniciais para as classes relacionadas:

```
/**
 * Classe para tratar erro.
 */

#ifndef  _ERRO_H_
#define _ERRO_H_

class Erro
{
  /*
   * tratamento de erro.
     Codigos de erro:
     0. Nao ha' erro.
   */

  /**
   * atributos privados.
   */
   private:
     int erro;

  /**
   * definicoes publicas.
   */
   public:
   /**
     * Destrutor.
     */
    ~Erro ( )
     { }

   /**
     * Construtor padrao.
     */
     Erro ( )
     {
      // atribuir valor inicial
         erro = 0;
     } // fim construtor padrao

  // ---------------------------------------- metodos para acesso

  /**
   * Funcao para obter o codigo de erro.
     @return codigo de erro guardado
   */
   int getErro ( )
   {
      return ( erro );
   } // end getErro ( )
```

```
  // ------------------------------------------- metodos para acesso restrito
    protected:

  /**
    * Metodo para estabelecer novo codigo de erro.
      @param codigo de erro a ser guardado
    */
    void setErro ( int codigo )
    {
       erro = codigo;
    } // end setErro ( )

}; // fim da classe Erro

#endif



/*
   Produto.hpp  - v0.0. - __ / __ / _____
   Author: _____________________

 */

// ------------------------------------------------ definicoes globais

#ifndef  _PRODUTO_H_
#define _PRODUTO_H_

// dependencias

#include <iostream>
using std::cin  ;          // para entrada
using std::cout;           // para saida
using std::endl;           // para mudar de linha

#include <iomanip>
using std::setw;           // para definir espacamento

#include <string>
using std::string;         // para cadeia de caracteres

#include <fstream>
using std::ofstream;       // para gravar arquivo
using std::ifstream;       // para ler    arquivo


// outras dependencias

#include "Erro.hpp"
```

```
// ------------------------------------------------  definicao de classe

/**
 * Classe para descrever produtos, derivada da classe Erro.
 */
class Produto : public Erro
{
 /**
  * atributos privados.
  */
  private:
    int       codigo;
    string  nome;
    int      quantidade;
    double preco;

 /**
  * definicoes publicas.
  */
  public:
  /**
   * Destrutor.
   */
  ~Produto ( )
   { }

  /**
   * Construtor padrao.
   */
   Produto (  )
   {
    // atribuir valores iniciais nulos/vazios
   } // fim construtor padrao

  /**
   * Construtor alternativo.
   */
   Produto ( int codigo_inicial, string nome_inicial, int quantidade_inicial, double preco_inicial )
   {
    // atribuir valores iniciais
   } // fim construtor

}; // fim da classe Contato

using ref_Produto = Produto*;  // similar a typedef Produto* ref_Produto;

#endif
```

Exercício:

DICAS GERAIS: Consultar o Anexo CPP 02 na apostila para outros exemplos.
Prever, realizar e registrar todos os testes efetuados.
Integrar as chamadas de todos os programas para testes em apenas um arquivo.

01.) Completar as definições de classes para atender aos requisitos do subsistema para atualizar controle de estoque unificado.

02.) Desenvolver a classe para atualização diária do arquivo central do estoque unificado.

03.) Definir as unidades de testes necessárias para validar as definições das classes e as operações sobre os dados.

Tarefas extras

E1.) Prever casos de uso para testar as operações básicas para inserir, atualizar e remover produtos no arquivo central para controle de estoque.

E2.) Desenvolver um módulo auxiliar para informar sobre produtos que estiverem em risco de desabastecimento e emitir um alerta sobre suas situações particulares.
DICA: Considerar o risco de desabastecimento todos os produtos que, ao final do dia, tiverem suas quantidades inferiores a um terço do dia anterior.