UNIVERSIDADE FEDERAL DA PARAÍBA – CENTRO DE INFORMÁTICA

LPII - Programação Orientada a Objetos - 2024.2

Prof. Carlos Eduardo Batista

Exercício Prático - Prova 2

- **Entrega por email: bidu @ ci . ufpb . br**
- **Prazo: até 23h59 de 25/03/2025**
- O título do e-mail deve conter (substituir nome e matrícula): **"[POO-20242-E002] NOME DO ALUNO – MATRICULA"**.
- Arquivo de entrega deve anexar todos os códigos fonte em C++ dentro de um diretório nomeado **"MATRICULA_POO-20242-E002"** o qual deve ser comprimido em um arquivo ZIP (**"MATRICULA_POO-20242-E002.zip"**).
- O arquivo ZIP deve conter obrigatoriamente um arquivo de texto chamado **README.txt** (ou **README.md**) contemplando todas as instruções de compilação e execução, além de qualquer observação que se fizer necessária para correção.
- O NÃO ATENDIMENTO ÀS INSTRUÇÕES IMPLICARÁ NA NÃO CORREÇÃO DO EXERCÍCIO.
- TRABALHO INDIVIDUAL - plágio será punido com a não correção do exercício.
- **Pontuação: até 3,0 pontos (para a segunda prova)**.

Exercício de Programação Orientada a Objetos

Sistema de Controle de Dispositivos Industriais

Contexto

Na indústria moderna, linhas de produção são compostas por diferentes dispositivos que precisam ser monitorados e controlados de maneira centralizada. Cada dispositivo possui características próprias, mas compartilha funcionalidades comuns com outros dispositivos.

Objetivo

Desenvolver um sistema em C++ que modele diferentes dispositivos industriais usando conceitos de Programação Orientada a Objetos: herança, métodos virtuais, métodos virtuais puros e classes abstratas.

Especificação

1. Classe Abstrata: `dispositivo_industrial`

Esta classe servirá como base para todos os tipos de dispositivos e deverá ser definida como uma classe abstrata.

Atributos protegidos:

- `id` (string): identificador único do dispositivo
- `status` (bool): indica se o dispositivo está ligado ou desligado

- `temperatura` (double): temperatura atual do dispositivo

Métodos públicos:

- Getters e setters para os atributos
- Método virtual `verificar_seguranca()`: retorna um booleano indicando se o dispositivo está operando em condições seguras
- Método virtual puro `iniciar()`: coloca o dispositivo em funcionamento
- Método virtual puro `parar()`: interrompe o funcionamento do dispositivo
- Método virtual puro `gerar_relatorio()`: retorna uma string com informações sobre o estado atual do dispositivo
- Destrutor virtual para permitir a destruição adequada de objetos derivados

2. Classes Concretas

2.1 `sensor_temperatura`

Classe que representa um sensor de temperatura industrial.

Atributos específicos:

- `temperatura_maxima` (double): limite superior de temperatura aceitável
- `temperatura_minima` (double): limite inferior de temperatura aceitável

Métodos específicos:

- `alerta_temperatura()`: retorna true se a temperatura atual estiver fora dos limites aceitáveis
- Implementações dos métodos virtuais puros da classe base:
  - `iniciar()`: ativa o sensor e inicia a leitura da temperatura
  - `parar()`: desativa o sensor
  - `gerar_relatorio()`: fornece um relatório com os valores de temperatura atual, mínima e máxima

2.2 `controlador_motor`

Classe que representa um controlador de motor industrial.

Atributos específicos:

- `potencia` (int): potência nominal do motor em watts
- `rpm` (int): rotações por minuto atuais
- `horas_de_uso` (double): total de horas de funcionamento

Métodos específicos:

- `ajustar_velocidade(int nova_rpm)`: modifica a velocidade do motor
- `calcular_eficiencia()`: retorna um valor de eficiência baseado na potência e RPM
- Sobrescrita do método `verificar_seguranca()` para incluir verificações específicas de motor
- Implementações dos métodos virtuais puros da classe base:
  - `iniciar()`: liga o motor e inicia o contador de horas
  - `parar()`: desliga o motor e atualiza o contador de horas

- `gerar_relatorio()`: fornece um relatório de uso, incluindo horas de funcionamento e eficiência

2.3 `robo_manipulador`

Classe que representa um braço robótico manipulador usado em linhas de montagem.

Atributos específicos:

- `posicao_x`, `posicao_y`, `posicao_z` (double): coordenadas da posição atual
- `carga_atual` (double): peso do objeto sendo manipulado
- `carga_maxima` (double): capacidade máxima de carga

Métodos específicos:

- `mover_para(double x, double y, double z)`: move o braço para a posição especificada
- `agarrar_objeto(double peso)`: simula agarrar um objeto com determinado peso
- Sobrescrita do método `verificar_seguranca()` para verificar posição e carga
- Implementações dos métodos virtuais puros da classe base:
  - `iniciar()`: liga o robô e o coloca em posição inicial
  - `parar()`: para o robô na posição atual
  - `gerar_relatorio()`: fornece informações sobre posição atual e carga

3. Sistema de Gerenciamento: `sistema_controle`

Esta classe será responsável por gerenciar todos os dispositivos do sistema.

Atributos:

- `dispositivos`: um vetor de raw pointers para objetos do tipo `dispositivo_industrial`

Métodos:

- `adicionar_dispositivo(dispositivo_industrial* dispositivo)`: adiciona um dispositivo ao sistema
- `remover_dispositivo(const string& id)`: remove um dispositivo do sistema com base no ID
- `iniciar_todos()`: inicia todos os dispositivos cadastrados
- `parar_todos()`: para todos os dispositivos cadastrados
- `verificar_seguranca_geral()`: executa a verificação de segurança em todos os dispositivos
- `gerar_relatorio_completo()`: gera um relatório com informações de todos os dispositivos
- Destrutor que libera a memória alocada para todos os dispositivos

Instruções para Implementação

1. **Classe Abstrata e Polimorfismo**:
   - Implemente `dispositivo_industrial` como uma classe abstrata com métodos virtuais puros
   - Use polimorfismo para chamar os métodos específicos através de ponteiros para a classe base
2. **Herança e Sobrescrita**:
   - Implemente as três classes derivadas, cada uma herdando da classe base
   - Sobrescreva os métodos virtuais puros em cada classe derivada

    - Implemente comportamentos específicos para cada tipo de dispositivo

3. **Gerenciamento de Memória**:
    - Use raw pointers para representar os dispositivos
    - Gerencie a memória corretamente, liberando recursos no destrutor da classe `sistema_controle`

4. **Aplicação Principal**:
    - Crie um arquivo `main.cpp` que demonstre o uso das classes
    - Instancie diferentes tipos de dispositivos usando `new`
    - Adicione-os ao sistema de controle
    - Demonstre o uso de polimorfismo ao chamar métodos através da classe base

Estrutura de Arquivos

- `dispositivo_industrial.h/.cpp`
- `sensor_temperatura.h/.cpp`
- `controlador_motor.h/.cpp`
- `robo_manipulador.h/.cpp`
- `sistema_controle.h/.cpp`
- `main.cpp`

Entregáveis

1. Todos os arquivos fonte (.cpp) e cabeçalho (.h)
2. Relatório breve (máximo 2 páginas) explicando como os conceitos de POO foram aplicados

Critérios de Avaliação

- Implementação correta dos conceitos de POO (30%)
- Funcionalidade do sistema (30%)
- Organização e legibilidade do código (20%)
- Relatório (20%)
