Programação Orientada a Objetos

Exercício 1: Sistema de Desenho de Formas Geométricas

Descrição

Você foi contratado para desenvolver um sistema que calcula áreas e perímetros de diferentes formas geométricas para uma empresa de arquitetura. O sistema precisa gerenciar diferentes tipos de formas e realizar cálculos precisos para estimativas de materiais.

Requisitos

1. Classe Base `forma_geometrica`

- Implemente os seguintes atributos protected:
  - `cor`: string que representa a cor da forma
  - `nome`: string que identifica o tipo da forma
- Métodos públicos necessários:
  - Construtor que inicializa os atributos
  - `exibir_info()`: exibe todas as informações da forma
  - `double calcular_area()`: método para cálculo de área
  - `double calcular_perimetro()`: método para cálculo de perímetro

2. Classes Derivadas

Classe `retangulo`

- Atributos específicos (private):
  - `base`: double
  - `altura`: double
- Métodos:
  - Construtor apropriado usando construtor da classe base
  - `calcular_area()`: retorna base * altura
  - `calcular_perimetro()`: retorna 2 * (base + altura)
  - `bool eh_quadrado()`: verifica se base == altura

Classe `circulo`

- Atributos específicos (private):
  - `raio`: double
- Métodos:
  - Construtor apropriado usando construtor da classe base
  - `calcular_area()`: retorna π * raio²

- `calcular_perimetro()`: retorna 2 * π * raio
- `double obter_diametro()`: retorna 2 * raio

3. Implementação Principal

Crie um programa que:

1. Cadastre diferentes formas geométricas em um projeto arquitetônico
2. Calcule a área total do projeto
3. Gere um relatório detalhado com todas as formas e suas dimensões

---

Exercício 2: Sistema de Gestão Bancária

Requisitos

1. Classe Base `conta_bancaria`

- Atributos:
  - `protected double saldo`
  - `private string numero_conta`
  - `private vector<string> historico`
- Métodos públicos:
  - Construtor que inicializa número da conta e saldo
  - `bool sacar(double valor)`
  - `void depositar(double valor)`
  - `void consultar_saldo()`
  - `void imprimir_extrato()`
- Método protected:
  - `void registrar_operacao(const string& descricao)`

2. Classes Derivadas

Classe `conta_poupanca`

- Atributos privados:
  - `taxa_juros`: double
- Métodos:
  - Construtor usando construtor da classe base
  - `void aplicar_rendimento()`
  - `double calcular_rendimento_proximo_mes()`

Classe `conta_corrente`

- Atributos privados:
  - `limite`: double
  - `taxa_mensal`: double
- Métodos:

- Construtor usando construtor da classe base
- `bool sacar(double valor)` // Reimplementa considerando o limite
- `void cobrar_taxa_mensal()`

Exercício 3: Sistema de Gestão de Recursos Humanos

Requisitos

1. Classe Base `funcionario`

- Atributos protected:
  - `nome`: string
  - `salario_base`: double
  - `matricula`: string
  - `data_admissao`: string
- Métodos protected:
  - `double calcular_beneficios()` // Vale transporte e alimentação
  - `void registrar_atividade(const string& atividade)`
- Métodos públicos:
  - Construtor que inicializa todos os atributos
  - `double calcular_salario()`
  - `void exibir_dados()`
  - `void gerar_contracheque()`

2. Classes Derivadas

Classe `gerente`

- Atributos privados:
  - `bonus_mensal`: double
  - `departamento`: string
  - `vector<string> funcionarios_supervisionados`
- Métodos:
  - Construtor usando construtor da classe base
  - `double calcular_salario()` // Inclui bônus
  - `void adicionar_supervisionado(const string& matricula)`
  - `void remover_supervisionado(const string& matricula)`
  - `void relatorio_equipe()`

Classe `vendedor`

- Atributos privados:
  - `comissao`: double
  - `total_vendas`: double
  - `meta_vendas`: double
- Métodos:

- Construtor usando construtor da classe base
- `void registrar_venda(double valor)`
- `double calcular_salario()` // Inclui comissão
- `bool bateu_meta()`
- `void relatorio_vendas()`