

Projeto Prático: Implementação de Sistema Orientado a Objetos em C++

Disciplina: Programação Orientada a Objetos

Submissão: somente o representante do grupo envia os Entregáveis

1. Objetivo

O objetivo deste projeto é aplicar os conceitos fundamentais de POO, incluindo abstração, encapsulamento, herança e polimorfismo, para resolver um problema real de engenharia ou gestão de dados. O foco deve estar na correção, simplicidade e organização do código.

Observação: Não é obrigatório a utilização de interface/elementos gráficos

2. Especificação Formal do Problema

Cada grupo deve definir um domínio de aplicação (ex: sistema de gestão hospitalar, simulador de tráfego, ou catálogo de sensores espaciais, etc.). A entrega deve conter:

- **Descrição do Problema:** O que o sistema resolve.
- **Mapeamento Código-Especificação:** Uma tabela ou lista relacionando cada requisito funcional à classe ou método correspondente no C++.
- **Diagrama de Classes Simples:** Identificando a hierarquia (Classes Base e Subclasses).

3. Requisitos Técnicos

A implementação deve seguir os princípios discutidos em aula e nas fontes:

A. Modelagem de Classes e Objetos

- **Encapsulamento:** Uso rigoroso de `public`, `private` e `protected`. Dados devem ser privados para manter a **invariante** da classe (garantia de valores válidos).
- **Herança e Polimorfismo:** Utilizar classes base (ex: `Shape` ou `Engine`) e criar subclasses que herdam propriedades e comportamentos.
- **Funções Virtuais:** Implementar o uso de `virtual` e `override` para garantir o despacho dinâmico (polimorfismo de tempo de execução).
- **Gerenciamento de Recursos (RAII):** Evitar o uso de "naked new/delete". Preferir o uso de **contêineres padrão** como `std::vector` ou ponteiros inteligentes (`unique_ptr`), garantindo que os recursos sejam liberados no destruidor.

B. Interface e Visualização

- **Método `render()` ou `visualizer()`**: Cada classe de objeto deve possuir um método para imprimir seu estado. (Observação: Não é obrigatório a utilização de interface/elementos gráficos)
- **Entrada e Saída (I/O)**: Utilizar `cin` e `cout` para interação, tratando os dados como fluxos (streams).

C. Testes e Robustez

- **Testes Automatizados**: Criar funções de teste (unit tests) que verifiquem se as funções retornam os resultados esperados para entradas diversas.
- **Tratamento de Erros**: Utilizar blocos `try-catch` e lançar exceções para lidar com argumentos inesperados ou estados inválidos.

4. Entregáveis e Prazos

1. **Código-Fonte**: Repositório (GitHub/GitLab) ou arquivo compactado.
2. **Vídeo de Demonstração (3–5 min)**: Explicação da arquitetura e demo do sistema rodando.
3. **Apresentação Presencial**: O grupo deve rodar o sistema e explicar a modelagem. **A presença é obrigatória para a avaliação.**