Lista de Exercícios — Herança em C++ (Abstração e Polimorfismo)

Exercício 1: Sistema de Pagamentos de Funcionários

Uma empresa precisa de um módulo de folha de pagamento para diferentes tipos de colaboradores: *Assalariado* (salário fixo mensal), *Horista* (pago por hora com horas extras) e *Comissionado* (percentual sobre vendas, com ou sem salário-base).

- Modele uma hierarquia com uma classe base `Funcionario` contendo dados comuns (id, nome, documentos) e um método para calcular a remuneração.
- Defina um método apropriado para cálculo de pagamento que permita **polimorfismo** (por exemplo, método virtual chamado via ponteiros/referências de `Funcionario`).
- Avalie e justifique se `Funcionario` deve ser uma **classe abstrata** (por exemplo, `calcularPagamento()` como virtual pura).
- Regras específicas: *Horista* recebe adicional de 50% nas horas acima de 44/semana; *Comissionado* possui teto de comissão; discuta possibilidade de salário-base.
- Inclua um método virtual para *gerar demonstrativo* (string com linhas formatadas), sobrescrito nas derivadas.
- Defina níveis de acesso e métodos de acesso quando necessário. Preveja um vetor/coleção heterogênea (por exemplo, `std::vector<std::unique_ptr<Funcionario>>`) para testes polimórficos.
- Extensões: adicional por produtividade; validações (percentual entre 0 e 100).

Exercício 2: Catálogo Multimídia com Itens Reproduzíveis

Deseja-se um catálogo que armazene itens de mídia: *Música* (duração, artista, bitrate), *Vídeo* (resolução, codec, duração) e *Podcast* (host, convidados, duração).

- Crie uma classe base `Midia` com dados comuns (título, ano, duração) e operações virtuais como `abrir()/reproduzir()` e `infoDetalhada()`; avalie torná-la **abstrata**.
- Especialize `Musica`, `Video` e `Podcast` com comportamentos distintos em `reproduzir()` e metadados específicos.
- Projete um método virtual `combina(filtro)` para pesquisa, com implementação própria em cada derivada (ex.: por artista, resolução, host).
- Crie um *gerenciador* que mantenha coleção heterogênea (`std::vector<std::unique_ptr<Midia>>`) e execute operações polimórficas (listar, abrir, filtrar).
- Garanta **polimorfismo** em tempo de execução e trate construção/remoção com **smart pointers**. Considere a **regra dos 5** quando necessário.

- Preveja extensão futura (ex.: *Audiolivro*) sem modificar o gerenciador, apenas adicionando novas classes (aberto para extensão).

Exercício 3: Frota de Veículos Autônomos

Uma empresa opera: *CarroAutonomo* (passageiros, autonomia), *CaminhaoAutonomo* (capacidade de carga, eixos) e *OnibusAutonomo* (lotação, acessibilidade).

- Modele `VeiculoAutonomo` com propriedades comuns (id, posição, bateria) e métodos virtuais para `planejarRota(destino)`, `mover()` e `relatorioStatus()`.
- Torne a base **abstrata** com ao menos um método virtual puro (ex.: `consumoPorKm()`), exigindo implementação nas derivadas.
- Sobrescreva comportamentos: caminhão evita rotas com restrição de peso; ônibus prioriza vias com paradas pré-definidas.
- Modele uma *estratégia de recarga* polimórfica (`estrategiaRecarga()`) que varie entre tipos (estações rápidas, docas específicas etc.).
- Integre **sensores**: um método virtual `detectarObstaculos()` que influencie `mover()` conforme o tipo.
- Crie um *Despachante* que, a partir de coleção heterogênea de `VeiculoAutonomo`, atribua rotas e gere **relatórios** unificados por **polimorfismo**.
- Discuta **herança vs. composição** para módulos como *Navegacao* e *Diagnostico*.

Exercício 4: Editor de Formas Vetoriais

Um editor vetorial manipula *Circulo* (centro, raio), *Retangulo* (canto, largura, altura) e *Poligono* (lista de vértices).

- Defina `Forma` com operações virtuais: `desenhar(canvas)`, `mover(dx,dy)`, `area()`, `perimetro()`, `clonar()`; avalie torná-la **abstrata**.
- Nas derivadas, sobrescreva os comportamentos, inclusive `clonar()` para duplicação polimórfica.
- Modele *Ferramentas* como possível hierarquia paralela: `Ferramenta` (talvez **abstrata**) com `aplicar(Forma&)` e derivadas como `FerramentaRedimensionar` e `FerramentaRotacionar`. Explique a interação polimórfica com `Forma`.
- Implemente um *Renderer* que receba coleção heterogênea (ponteiros para `Forma`) e invoque `desenhar()` polimorficamente. Considere uma camada de exportação (SVG/PNG) como serviço acoplado via **interfaces**.
- Delimine responsabilidades: cálculo geométrico nas formas; persistência/serialização em classes separadas (composição). Preveja pontos de extensão (novas formas/ferramentas) sem alterar o renderer.
- Regras: valide dimensões (raio > 0, polígonos simples) e trate exceções para operações inválidas (ex.: redimensionar com fator negativo) mantendo a interface polimórfica coesa.