

LAB05: Derivação Pública vs Privada, Herança Múltipla

INSTRUÇÕES INICIAIS:

Nota: Os exercícios devem ser executados na ordem apresentada, pois o nível de dificuldade é crescente.

1. Obtenha o arquivo **lab05.zip** (disponível no ensino aberto);
2. Descompacte esse arquivo no seu diretório de trabalho (workspace do Eclipse);
3. Crie um novo projeto Java no Eclipse. Clicar em File > New > Java Project, manter o lugar padrão e escrever como nome do projeto exatamente o nome da pasta descompactada.
4. Ao final do lab, compacte a **pasta raiz do projeto**, em um arquivo com extensão **.zip**, e suba no ensino aberto no seu portfólio*, disponibilizando-o para os formadores. Suba apenas um arquivo. **Outros formatos não serão avaliados, implicando em nota 0 (zero) na atividade.**

5. Suba um arquivo de texto com a resposta para todas as questões conceituais **dentro** da pasta compactada. Serão aceitos os formatos **.txt** e **pdf**. **Coloque seu nome e RA neste arquivo.**

***obs:** criar apenas um item no portfólio com o arquivo zip, com nome Lab05. Não criem pastas no portfólio.

Pacote `br.unicamp.ic.mc302.termostato`:

Abra o arquivo `Pilha.cpp`. Este arquivo contém um exemplo em C++ de uma Pilha de inteiros implementado por herança simples de uma lista ligada.. Compile o arquivo e observe sua execução.

Obs.: Para compilar o arquivo, utilize o comando: `c` Para executá-lo, utilize o comando: `./Pilha`

Semelhante ao exercício 1 do Laboratório 04, o resultado não respeita o comportamento padrão de uma pilha. Nesse exercício, a solução foi introduzir uma relação de agregação entre as classes. No entanto em C++ podemos conseguir o efeito desejado através da derivação Privada:

- Em uma derivação pública, todos os atributos e operações públicos de uma classe tornam-se atributos e operações públicos de uma outra classe (subclasse).
- Em uma derivação privada, todos os elementos públicos de uma classe tornam-se elementos privados de uma outra classe (subclasse).

1. Modifique o tipo da derivação entre as classes `Pilha` e `Lista` e compile o arquivo. O que acontece? Comente.

Abra o arquivo `Termostato.cpp`. Este arquivo contém a implementação em C++ do exemplo do `termostato` de herança múltipla . Compile o arquivo e observe sua execução.

Igualmente, para compilar o arquivo, utilize o comando: `g++ Termostato.cpp -o Termostato`

Para executá-lo, utilize o comando: `./Termostato`

2. Modifique o tipo da variável `t` da função `main` para `Chave`. Compile o arquivo modificado. O que acontece? Modifique o tipo de `t` para `Termometro` e compile. O que acontece?

3. Abra os arquivos `Termostato.java`, `Termometro.java` e `Chave.java`. Eles correspondem à solução de herança múltipla em Java utilizando agregação. Defina uma classe `ExemploTermostato` que cria um objeto do tipo `Termostato` e chame os métodos deste objeto (como `setTempRequerida()` e `fazerMonitoramento()`, por exemplo). Compile e execute essa classe.

4. Na classe `ExemploTermostato`, modifique o tipo da variável anteriormente criada para `Chave` e compile. O que acontece? E o que acontece quando se modifica o tipo para `Termometro`? Por que isso ocorre?

5. Outra solução possível de herança múltipla envolve utilizar herança simples e agregação em conjunto. Implemente um *refactoring* do código Java seguindo essa solução com `Termostato` como subclasse de `Termometro` e `Termostato` agregando `Chave`.

6. Na classe `ExemploTermostato`, modifique o tipo da variável anteriormente criada para `Chave` e compile. O que acontece? E o que acontece quando se modifica o tipo para `Termometro`? Por quê?

7. Abra o arquivo `Veiculo.cpp`. Este arquivo ilustra o chamado “Problema do Diamante”. Compile-o (comando similar ao do item 1). O que acontece? Uma forma de solucionar este problema em C++ é adicionando o modificador “virtual” nas derivações públicas. Modifique o exemplo dado de tal forma que o modificador “virtual” seja aplicado à hierarquia de classe `Veiculo` de forma adequada para resolver o problema. Compile e execute o arquivo. Comente os resultados.

8. Edite o arquivo `Veiculo.cpp` para incluir uma chamada ao método `display`. Compile e comente.

9. Implemente um programa Java que cria uma classe `Pessoa` com atributos, como por exemplo, nome, rg, cpf, nome do pai, nome da mãe, etc. Crie a subclasse `EstudanteUniversitario` a partir da classe `Pessoa` com atributos como, por exemplo, RA, nome do curso, créditos concluídos, etc. Implemente a classe `Docente` que herda de `Pessoa`, com operações para calcular tempo de aposentadoria, atribuir disciplina, etc. Implemente a classe `PosGraduando` que herda de `EstudanteUniversitario` com operações para cálculo de média de notas baseada em conceitos (A,B,C,D,E). Crie a classe `PED` que representa alunos de pós-graduação que atuam como docentes em disciplinas de graduação. Implemente um *refactoring* adequado para Java e instancie um objeto do tipo `PED`.

EXERCÍCIO EXTRA:

10. Reimplemente o exercício 7 usando a linguagem Java. Escolha um “*refactoring*” adequado para sua solução.