Disciplina SI300 - Programação Orientada a Objetos I

Faça todos os códigos em C++.

Use std::*cout* e std::cin e não *printf* ou *scanf* em todos os exercícios da disciplina. É recomendável usar a declaração de uso do espaço de nomes para evitar repetição de 'std::'.

1. Escreva um programa C++ do tipo "hello-world", para certificar-se da configuração do seu ambiente de trabalho. Compile e rode o programa, garantindo que não há erros ou avisos do compilador.

2. Escreva um programa C++ para ler e imprimir um número inteiro, um número em ponto flutuante e uma *string*.

3. Crie uma classe C++ com único atributo inteiro encapsulado em modo privativo (*private*) e os respectivos métodos de escrita e leitura (*setters & getters*). Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. Escreva uma função main() que use a classe, exercitando todos os seus métodos.

4. Crie uma classe C++ com único atributo inteiro encapsulado em modo privativo (*private*) e os respectivos métodos *getters, setters,* construtores padrão (*default*) e parametrizado. Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. Escreva uma função main() que use a classe, exercitando todos os seus métodos.

5. Crie uma classe C++ com único atributo *string* encapsulado em modo privativo (*private*) e os respectivos métodos *getters, setters* e construtor padrão (*default*) com parâmetro padrão (*default parameter*) para que, em caso de omissão, a *string* seja preenchida com o valor “Unicamp”. Se disponível, use o recurso de geração automática de *getters* e *setters* do seu IDE. Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. Escreva uma função main() que use a classe, alocando dinamicamente (new) os objetos, exercitando todos os seus métodos e liberando a memória (delete).

6. Crie uma classe Tabuada1 em C++, com único atributo inteiro encapsulado em modo privativo (*private*) e os respectivos métodos *getters, setters* e construtor parametrizado. Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. A classe deve ter quatro métodos de serviço, correspondentes às operações de adição, subtração, multiplicação e divisão. Uma vez invocado qualquer um desses métodos, ele deve imprimir a respectiva tabuada (1-10), considerando o número gravado no atributo. Caso o atributo tenha valor 0, o método da tabuada de divisão deve informar uma mensagem de erro. Escreva uma função main() que use a classe, alocando dinamicamente (new) os objetos, exercitando todos os seus métodos e liberando a memória (delete).

7. Crie uma enumeração OPERACAO, com os símbolos referentes a adição, subtração, multiplicação e divisão. Codifique uma classe Tabuada2 em C++, com um atributo inteiro e um atributo OPERACAO encapsulados em modo privativo (*private*) e os respectivos métodos *getters, setters* e construtor parametrizado (assuma valores padrão = { 1, SOMA }) . Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. A classe deve ter um método de serviço que, uma vez invocado, imprima a respectiva tabuada (1-10), considerando o número e a operação gravados nos atributos. Considere usar métodos privativos para a execução de cada tabuada (operação) individual. Caso os atributos sejam { 0, DIVISAO }, o método deve informar uma mensagem de erro. Escreva uma função main() que use a classe, alocando dinamicamente (new) os objetos, exercitando todos os seus métodos e liberando a memória (delete).

8. Crie uma classe Forno em C++, com único atributo real (*float*) temperatura encapsulado em modo privativo (*private*) e os respectivos métodos *getters, setters* e construtor parametrizado (*default* = { 0.0f }). O método *setTemperatura*(float) deve garantir que nunca o valor seja negativo (vai para 0.0f) ou maior que 280 graus (fica em 280 C). Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro. Escreva um método de serviço chamado *getStatus()* que imprima as condições de operação do Forno (ex: Forno cozinhando a xx graus). Da classe Forno, derive

publicamente três classes: forno elétrico, forno a gás e forno à lenha. Para cada uma dessas classes, inclua pelo menos um atributo distinto próprio e forneça um construtor personalizado apropriado. Em cada uma delas, sobrescreva o método de serviço *getStatus*, para que apresente também o novo atributo na tela. Caso ache conveniente, invoque o método *getStatus* da classe base para realizar parte da tarefa. Na mensagem, cuide de especificar qual é a classe que está imprimindo (ex: Forno elétrico cozinhando a xx graus). A partir da classe forno elétrico, derive forno de indução e forno resistivo, repetindo a ideia de criar atributos próprios e sobrescrever o método de serviço. Faça o mesmo em relação ao forno a gás, criando forno a gás de petróleo e forno a gás natural. Escreva uma função main() que use todas as classes, alocando dinamicamente (new) os objetos, exercitando todos os seus métodos e liberando a memória (delete).

9. Crie uma classe Potencia em C++, com atributos adequados encapsulados em modo privativo (*private*) e somente um método de serviço sobrecarregado, denominado *calcula(),* com as seguintes assinaturas*:* double calcula(int base, int expoente), double calcula(int base, double expoente) e double calcula(double base, double expoente). Para cada um, forneça uma implementação que execute e retorne o cálculo da potência, e imprima uma mensagem identificando o método invocado. Escreva uma função main() que use a classe, alocando dinamicamente (new) os objetos, exercitando todos os seus métodos e liberando a memória (delete).

10. Crie uma classe HotDog em C++, com atributos adequados encapsulados em modo privativo (*private*) e somente o respectivo método construtor sobrecarregado. Um *hotdog* tem pão, salsicha e *pode ter*: molho de tomate, mostarda, batata frita, maionese, picles, pimenta e outras especiarias. O construtor padrão deve dar conta do pão e da salsicha. Cada um dos demais construtores sobrecarregados deve permitir adição de um novo item, na ordem listada anteriormente, ao sanduíche. Considere usar valores booleanos para a indicação da presença ou não do item. Crie também um método *deliveryHotDog*, que apresente a configuração do sanduíche. Use o ponteiro *this* dentro dos métodos para diferenciar atributo e parâmetro.

Escreva uma função main() que use a classe, alocando dinamicamente (new) vários objetos (hotdog), com diferentes configurações, exercitando todos os seus métodos e liberando a memória (delete).

11. Escreva um programa C++ que utilize a classe Forno e todas as suas derivadas. O programa deve criar uma estrutura de dados (vetor/*array* ?) e colocar ao menos um objeto de cada classe nessa estrutura. Para que isso seja possível, o tipo de dados da estrutura deve ser ponteiro para a classe mais ancestral (Forno *). Inicialize cada objeto individualmente de acordo com os respectivos construtores e seus parâmetros. Utilizando uma estrutura de repetição, solicite a cada um dos elementos da estrutura para que imprima seu status. Note que não será necessário nenhuma estrutura de decisão, como *if-else* ou *switch*. Desaloque individualmente a memória de cada objeto antes de encerrar o programa.

12. Crie uma classe SerVivo em C++, com único atributo string nome encapsulado em modo privativo (*private*) e os respectivos métodos *getters*, *setters*. Faça esses métodos serem encapsulados em modo protegido (*protected*) de forma a não precisar repetí-los nas classes derivadas. Faça que os métodos sejam *virtuais*. Inclua um construtor com o único parâmetro, de modo que a classe não possa ser inicializada com o atributo vazio ou inconsistente. Use o método *getNome()* para as operações de demonstração do programa, como método de serviço. Derive ao menos uma dúzia de classes, em diferentes níveis, criando uma hierarquia de classes. Faça um esquema ilustrativo dessa hierarquia para ter uma boa visualização do resultado final. Forneça os métodos construtores apropriados a cada classe derivada, mas não escreva *getters* e *settets* (vai recebê-los prontos da classe ancestral comum). Crie uma estrutura de dados capaz de conter ao menos um objeto de cada uma das classes. Inicialize cada objeto individualmente, a partir de seu construtor específico. A seguir, usando uma estrutura de repetição, peça que cada objeto da estrutura imprima a sua identificação, usando o método *getNome()*. Use como tipo de dados da estrutura um ponteiro para a classe SerVivo. Não use estruturas de decisão para selecionar entre os métodos a serem executados. Desaloque individualmente a memória de cada objeto antes de encerrar o programa.