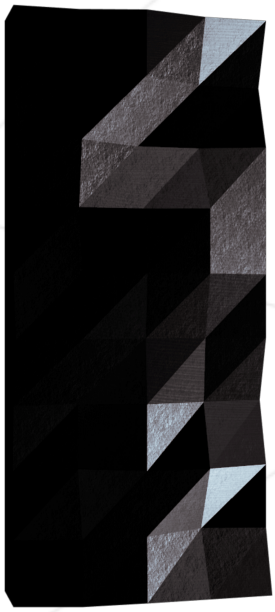

C++ - Módulo 04

Polimorfismo de subtipo, classes abstratas, interfaces

Resumo: Este documento contém o assunto do Módulo 04 dos módulos C++.

Versão: 10

Conteúdo

Capítulo I

Regras gerais

Para os módulos de C++, você usará e aprenderá apenas C++98. O objetivo é que você aprenda o básico de uma linguagem de programação orientada a objetos. Sabemos que o C++ moderno é muito diferente em muitos aspectos, então, se você quiser se tornar um desenvolvedor C++ proficiente, precisará do C++ padrão moderno mais tarde. Este será o ponto de partida da sua jornada em C++; cabe a você ir além após os 42 anos do Common Core!

- Qualquer função implementada em um cabeçalho (exceto no caso de modelos) e qualquer cabeçalho desprotegido significa 0 para o exercício.
- Cada saída vai para a saída padrão e será encerrada por uma nova linha, a menos que especificado de outra forma.
- Os nomes de arquivos impostos devem ser seguidos à risca, assim como nomes de classes, nomes de funções e nomes de métodos.
- Lembre-se: você está codificando emC++agora, não emCnão mais. Portanto:
    - As seguintes funções são PROIBIDAS e seu uso será punido com uma0, sem perguntas: *alocação, *printfelivre.
    - Você tem permissão para usar tudo na biblioteca padrão. NO ENTANTO, seria inteligente tentar usar as versões C++-ish das funções com as quais você está acostumado em C, em vez de se limitar ao que você já conhece, afinal, esta é uma linguagem nova. E NÃO, você não tem permissão para usar a STL até que seja necessário (ou seja, até o módulo 08). Isso significa que não há vetores/listas/ mapas/etc... ou qualquer coisa que exija um <algoritmo> de inclusão até então.
- Na verdade, o uso de qualquer função ou mecânica explicitamente proibida será punido com uma0,sem perguntas.
- Observe também que, salvo indicação em contrário, oC++palavras-chave "usando namespace"e "amigo"são proibidos. Seu uso será punido com uma -42,sem perguntas.
- Os arquivos associados a uma classe sempre serãoNomeDaClasse.hppeNomeDaClasse.cpp, a menos que especificado de outra forma.
- Os diretórios de entrega sãoex00/, ex01/, . . . , exn/.

- Você deve ler os exemplos atentamente. Eles podem conter requisitos que não são óbvios na descrição do exercício.
- Já que você tem permissão para usar oC++ferramentas que você aprendeu desde o início, você não tem permissão para usar nenhuma biblioteca externa. E antes que você pergunte, isso também significa que nãoC++11e derivados, nemImpulsionarou qualquer outra coisa.
- Talvez você precise entregar um número considerável de aulas. Isso pode parecer tedioso, a menos que você saiba programar seu editor de texto favorito.
- Leia cada exercício COMPLETAMENTE antes de começar! Faça-o.
- O compilador a ser usado éc++.
- Seu código deve ser compilado com os seguintes sinalizadores: -Parede -Wextra -Werror.
- Cada uma das suas inclusões deve poder ser incluída independentemente das outras. As inclusões devem conter todas as outras inclusões das quais dependem.
- Caso você esteja se perguntando, nenhum estilo de codificação é imposto duranteC++.Você pode usar o estilo que quiser, sem restrições. Mas lembre-se de que um código que seu avaliador não consegue ler é um código que ele não pode avaliar.
- Informações importantes agora: você NÃO será avaliado por um programa, a menos que explicitamente declarado na disciplina. Portanto, você tem certa liberdade na forma como escolhe fazer os exercícios. No entanto, esteja ciente das restrições de cada exercício e NÃO seja preguiçoso, pois você perderia MUITO do que eles têm a oferecer.
- Não há problema em ter alguns arquivos estranhos no que você entrega; você pode optar por separar seu código em mais arquivos do que o solicitado. Sinta-se à vontade, desde que o resultado não seja corrigido por um programa.
- Mesmo que o assunto de um exercício seja curto, vale a pena dedicar algum tempo a ele para ter certeza de que você entendeu o que é esperado de você e que o fez da melhor maneira possível.
- Por Odin, por Thor! Use seu cérebro!!!

Capítulo II

Exercício 00: Polimorfismo

	Exercício: 00
Polimorfismo	
Diretório de entrega:*ex*00*/*	
Arquivos para entregar:Makefile, main.cpp, todos os outros arquivos que você precisa	
Funções proibidas:Nenhum	

Para cada exercício, seu principal deve testar tudo.
Os construtores e destrutores de cada classe devem ter uma saída específica.
Crie uma classe base simples e completa, Animal.
A classe animal recebeu um atributo protegido:

- std::string tipo;

Crie uma classe Cachorro que herde de Animal.
Crie uma classe Gato que herde de Animal.
(para a classe animal, o tipo pode ser deixado em branco ou definido com qualquer valor). Cada classe deve colocar seu nome no campo Tipo, por exemplo: o tipo de classe Cachorro deve ser inicializado como "Cachorro".
Todo animal deve ser capaz de usar o método makeSound().
Este método exibirá uma mensagem apropriada nas saídas padrão com base na classe.

```
int principal()
{
    const Animal* meta = novo Animal();
    const Animal* j = novo Cachorro();
    const Animal* i = novo Gato();

    std::cout << j->getType() << " " << std::endl; std::cout
    << i->getType() << " " << std::endl; i->makeSound(); //
    irá emitir o som do gato! j->makeSound();

    meta->makeSound();
```

```
    ...
}
```

Isso deve gerar o makeSound específico da classe Dog e cat, não do animal.

Para ter certeza, você criará uma classe WrongCat que herda uma classe WrongAnimal que emitirá o comando WrongAnimal makeSound() quando testada nas mesmas condições.

Capítulo III

Exercício 01: Não quero incendiar o mundo

	Exercício: 01
Eu não quero incendiar o mundo	
Diretório de entrega:*ex*01/	
Arquivos para entregar:Makefile, main.cpp, além dos arquivos necessários para seus testes	
Funções proibidas:Nenhum	

Você reutilizará as classes Ex00.
Crie uma classe chamada Brain.
O cérebro conterá um array de 100 std::string chamado ideas.
Agora, Dog e cat terão um atributo Brain* privado.

Nem todo animal tem cérebro!

Durante a construção, Cão e Gato inicializarão seu Cérebro* com um novo Cérebro();
Durante a destruição, Cão e Gato excluirão seu Cérebro.

Seu personagem principal criará e preencherá um Array de Animais, metade dos quais será Cachorro e a outra metade Gato.

Antes de sair, seu principal executará um loop sobre este array e excluirá todos os Animais.
Você deve excluir diretamente Gato e Cachorro como Animais.

Uma cópia de um Gato ou Cachorro deve ser "profunda".
Seu teste deve mostrar que as cópias são profundas!

Construtores e destrutores de cada classe devem ter uma saída específica. Os destrutores apropriados devem ser chamados.

```
int principal()
{
	const Animal* j = novo Cachorro();
	const Animal* i = novo Gato();


	delete j;//não deve criar um vazamento
	delete i;

}
```

Capítulo IV

Exercício 02: classe abstrata

	Exercício: 02
Este código é impuro. PURIFIQUE-O!	
Diretório de entrega:*ex*02/	
Arquivos para entregar:Makefile, main.cpp, além de quaisquer arquivos necessários	
Funções proibidas:Nenhum	

Afinal, um Animal genérico não faz sentido algum. Por
exemplo, ele não emite nenhum som!
Para evitar erros futuros, a classe animal padrão não deve ser instanciável. Corrija a classe Animal para que ninguém a instancie por engano.

O resto deve funcionar como antes.

Capítulo V

Exercício 03: Interface e recapitulação

	Exercício: 03
Interface e recapitulação	
Diretório de entrega:*ex*03*/*	
Arquivos para entregar:Makefile, main.cpp, além de quaisquer arquivos necessários	
Funções proibidas:Nenhum	

Não há interface em C++98 (nem mesmo em C++20), mas é comum chamar uma classe abstrata pura de Interface. Então, para este último exercício, vamos testar interfaces e recapitular tudo!

Complete a definição do seguinteAMateriaclasse e implementar as funções de membro necessárias.

```
classe AMateria
{
        protegido:
                [...]

        público:
                AMateria(std::string const & tipo); [...]


                std::string const & getType() const; //Retorna o tipo de matéria

                virtual AMateria* clone() const = 0; virtual
                void use(ICharacter& target);
};
```

Crie os materiais concretosGeloeCura.O tipo deles será o nome deles em letras minúsculas ("ice" para Ice, etc...).

Delesclone()O método retornará, é claro, uma nova instância do tipo da Matéria real.

Em relação aousar(ICharacter&)método, ele exibirá:

- Gelo: "*dispara um raio de gelo em NOME*"
- Cura: "*cura as feridas de NAME*"

(É claro, substitua NOME pelo nome doPersonagem dado como parâmetro.)

Ao atribuir uma Matéria a outra, copiar o tipo não faz sentido...

Crie oPersonagemclasse, que implementará a seguinte interface:

```
classe ICharacter
{
        público:
                virtual ~ICharacter() {}
                std virtual::string const & getName() const = 0;
                vazio virtual equipar(AMateria* m) = 0;
                vazio virtual desequipar(int idx) = 0;
                virtual void use(int idx, ICharacter& target) = 0;
};
```

O personagem possui um inventário de no máximo 4 Matérias, vazio no início. Ele equipará as Matérias nos espaços 0 a 3, nesta ordem.

Caso tentemos equipar uma Matéria em um inventário completo, ou usar/desequipar uma Matéria inexistente, não faça nada.

Odesequiparo método NÃO deve excluir Matéria!

Ouse(int, ICharacter&)o método terá que usar a Matéria noidxslot e passealvo como parâmetro para oAMateria::usométodo.

Claro, você terá que ser capaz de suportar QUALQUER AMateria no inventário de um personagem.

SeuPersonagemdeve ter um construtor que receba seu nome como parâmetro. Cópia ou atribuição de umPersonagemdeve ser profundo, é claro. O velhoMatériade umPersonagem deve ser excluído. O mesmo ocorre com a destruição de umPersonagem .

Crie oFonte de Matériaclasse, que deverá implementar a seguinte interface:

```
classe IMateriaSource
{
        público:
                virtual ~IMateriaSource() {}
                vazio virtual learnMateria(AMateria*) = 0;
                AMateria virtual* createMateria(std::string const & tipo) = 0;
};
```

aprenderMateriadeve copiar a Matéria passada como parâmetro e armazená-la na memória para ser clonada posteriormente. Da mesma forma que paraPersonagem ,a Fonte pode conhecer no máximo 4 Matérias, que não são necessariamente únicas.

createMateria(std::string const &)retornará uma nova Matéria, que será uma cópia da Matéria (anteriormente aprendida pela Fonte) cujo tipo é igual ao parâmetro. Retorna 0 se o tipo for desconhecido.

Em resumo, sua Fonte deve ser capaz de aprender "modelos" de Matéria e recriá-los sob demanda. Você poderá então criar uma Matéria sem saber seu tipo "real", apenas uma string que a identifique.

Como de costume, aqui está um teste principal que você terá que melhorar:

```
int principal()
{
	IMateriaSource* src = new MateriaSource(); src-
	>learnMateria(new Ice());
	src->learnMateria(nova Cura());

	ICharacter* me = novo Personagem("eu");

	AMateria* tmp;
	tmp = src->createMateria("gelo"); me-
	>equip(tmp);
	tmp = src->createMateria("curar"); me-
	>equipar(tmp);

	ICharacter* bob = novo Personagem("bob");

	eu->usar(0, *bob);
	eu->use(1, *bob);

	apagar bob;
	me exclua;
	excluir src;

	retornar 0;
}
```

Saída:

```
$> clang++ -W -Wall -Werror *.cpp $> ./
a.out | cat -e
* atira um raio de gelo em Bob *$
* cura as feridas de bob *$
```