Universidade Tecnológica Federal do Paraná

Disciplina: Análise e Projeto Orientado a Objetos
Aluno: Jean Carlos Martins Miguel R.A: 1640593
Professor: Dr. Lucio Geronimo Valentin

1 Capítulo 3

1.1 Encapsulamento

O TAD é definido atravé de sua interface ou seja, a maneira pelo qual ele é manipulado,embora possam existir diferentes tipos de implementações para TAD, todos são utilizados da mesma forma,sua implementação não deve estar visívle ao usuário pois o mesmo irá acessá-la através de uma interface, essa maneira de "esconder"essas informações é definida como encapsulamento.

1.2 Controle de acesso - publico e private

Um dos objetivos da classe é enconder informações, então algumas restrições precisam ser impostas para uma classe ser manipulada, podendo ser definidos 3 níveis de permissão de acordo com o uso dos atributos:
Nos métodos da classe
A partir de objetos da classe
Métodos de classes derivadas
Cada um dos três contextos tem privilégios de acesso diferentes, ou seja, cada um tem uma palavra reservada associada, private, public e protected respectivamente. Atributos private são os mais restritos. Somente a própria classe pode acessar os atributos privados.

Declaração de classes com a palavra reservada class

"As classes podem ser declaradas usando-se a palavra reservada class no lugar de struct. A diferença entre as duas opções é o nível de proteção caso nenhum dos especificadores de acesso seja usado. Os membros de uma struct são públicos, enquanto que em uma class, os membros são privados". As declarações com a palavra class são mais usadas pois as interfaces das classes devem ser as menores possíveis e também dever ser explícitas.

Classes e funções friend

Existe a palavra reservada friend que serve para oferecer acesso especial à algumas classes ou funções dando-as acessos irrestritos.O recurso de classes e funções friend devem ser usados cautelosamente, pois isto é um furo no encapsulamento dos dados de uma classe.

1.3 Construtores e destrutores

Construtor é uma função que é utilizada para construir umobjeto de uma classe e é invocado logo depois que um objeto é criado, e os destrutores são invocados logo que os objetos são destruídos, destrutores geralmente são utilizados para liberar recursos alocados pelo objeto, como memória,arquivos etc.

Declaração de construtores e destrutores

Ambos são métodos especiais, nenhum dos tipos tem retorno, o construtor pode receber parâmetros mas o destrutor não, para declarar um construtor basta definir um método que tenha o mesmo nome da classe,então o nome do destrutor é o nome da classe precedido de $\sim$ .O construtor de uma classe A é declaro como um método de nome A, o nome do destrutor é $\sim$A e ambos não tem tipo de retorno A.Classes podem ter mais de um destrutor.

Chamada de construtores e destrutores

Construtores são chamados automaticamente sempre que um objeto de classe for criado Os construtores são chamados automaticamente sempre que um objeto da classe for criado, já os destrutores de objetos são chamados quando o objeto sai do seu escopo Os destrutores de objetos alocados na pilha são chamados quando o objeto sai do seu escopo.

Construtores com parâmetros

O objetivo dos construtores é o de garantir a consistência inicial dos objetos, então é necessário forncer um parâmetro sobre o qual será feita uma iteração por exemplo. A declaração de um construtor afirmae que os objetos só são criados através deles.

Construtores gerados automaticamente

Uma classe não ter construtor declarado não signifca que ela não tem construtores, pois o compilador gera alguns construtores automaticamente, o construtor vazio por exemplo ,que só é gerado se a classe não gerar nenhum construtor.Um outro construtor que o compilador gera é o construtor de cópia que é gerado mesmo se a classe declara algum construtor, o construtor de cópia receben como parâmetro uma referência para um objeto da própria classe.

Objetos temporários

"Assim como não é preciso declarar uma variável dos tipos primitivos sempre que se quer usar um valor temporariamente, podemos criar objetos temporários em C++. Uma utilização típica é a seguinte: quando uma função aceita como parâmetro um inteiro, e você quer chamá-la passando o valor 10, não é necessário atribuir 10 a uma variável apenas para chamar a função. O mesmo deve se aplicar a tipos definidos pelo usuário (classes)".

Conversão por construtores

Construtores que possuem apenas um parâmetro são como uma função de conversão do tipo do parâmetro par ao tipo da classe

Construtores privados

Construtores assim como métodos podem ser privados, já que construtores são chamados no momento da criação, então os objetos só podem ser criados com este construtor dentro de métodos da própria classe ou em funções e classes friend.

Destrutores privados

Destrutores também podem ser privados, significa que objetos desta classe só podem ser destruídos onde os destrutores podem ser chamados (métodos da própria classe, classes e funções friend).

Inicialização de campos de classes com construtores

No caso de objetos não ter construtor sem parâmetros, é necessário passar valores com os parâmetros,se o objeto for uma variável, é só definir os parâmetros na hora da declaração.

2 Capítulo 5

2.1 Aspectos de reutilização

Quando observa-se as repetições quando se trata de desenvolvimento de sistemas percebe-se as dificuldades técnicas de reutilização, embora os programadores escrevem os mesmos tipos de rotinas várias vezes, essas rotinas não são idênticas,pois alguns detalhes mudam de implementação para implementação Apesar das dificuldades, algumas soluções foram propostas com relativos sucessos, abaixo são citados pelo autor:

"Reutilização de código-fonte: Muito comum no meio científico. Muito da cultura UNIX foi difundida pelos laboratórios e universidades do mundo graças à disponibilidade de código-fonte ajudando estudadentes a estudarem, imitarem e estenderem o sistema. No entanto, esta não é a forma mais utilizada nos meios industrial e comercial além de que esta técnica não suporta ocultação de informação (information hiding)".

"Reutilização de pessoal: É uma forma muito comum na indústria. Consiste na transferência de engenheiros de software de projetos a projetos fazendo a permanência de know-how na compania e assegurando a aplicação de experiências passadas em novos projetos. Obviamente, esta é uma maneira não-técnica e limitada".

"Reutilização de design: A idéia por trás desta técnica é que as companhias devem acumular repositórios de idéias descrevendo designs utilizados para os tipos de aplicação mais comuns".

2.2 Requisitos para reuso

Temos alguns aspectos importantes para a reutilização de código, citados pelo autor:
"A noção de reuso de código-fonte lembra que software é definido pelo seu código. Uma política satisfatória de reutilização deve produzir programas (códigos) reutilizáveis.
A reutilização de pessoal é fundamental pois os componentes de software são inúteis sem profissionais bem treinados e com experiência para reconhecer as situações com possibilidade de reuso.
A reutilização de design enfatiza a necessidade de componentes reutilizáveis que estejam em um nível conceitual e de generalidade alto — não somente com soluções para problemas específicos. Neste aspecto, poderá ser visto como o conceito de classes nas linguagens orientadas por objetos pode ser visto como módulos de design e de implementação".

2.3 Variação no tipo

"Um módulo que implemente uma determinada funcionalidade deve ser capaz de fazê-lo sobre qualquer tipo a ele atribuído. Por exemplo, no caso da busca de um elemento x, o módulo deve ser aplicável a diferentes instâncias de tipo para x. É interessante a utilização do mesmo módulo para procurar um inteiro numa tabela de inteiros ou o registro de um empregado na sua tabela correspondente, etc".

2.4 Variação nas estruturas de dados e algoritmos

"No caso da busca de um elemento x, o modo de busca pode ser adaptado para diversos tipos de estruturas de dados e algoritmos de busca associados: tabelas seqüenciais (ordenadas ou não), vetores, listas, árvores binárias, B-trees, diferentes estruturas de arquivos, etc. Neste sentido, o módulo deve suportar variações nas estruturas a ele associado".

2.5 Existência de rotinas relacionadas

"De forma a fazer uma pesquisa em uma tabela, deve-se saber como esta é criada, como os elementos podem ser inseridos, retirados, etc. Deste modo, uma rotina de busca não é por si só suficiente; há a necessidade do acoplamento de diversas rotinas primitivas e relacionadas entre si".

2.6 Independência de representação

"O princípio da independência de representação não significa somente que mudanças na representação devem ser invisíveis para os clientes durante o ciclo de desenvolvimento do sistema: os clientes devem ser imunes também a mudanças durante a execução"

2.7 Semelhanças nos subcasos

Este item sobre reuso afeta o design e a construção dos módulos e não seus clientes, e é de extrema importância pois determina a possibilidade da construção de módulos bem construídos sem repetições indesejáveis. O aparecimento de repetições excessivas nos módulos compromete suas consistências internas e tornando-os difíceis de fazer manutenção.

2.8 Herança

Sem dúvidas herança é o recurso que torna o conceito de classes mais poderoso. "Em C++, o termo herança se aplica apenas às classes. Variáveis não podem herdar de outras variáveis e funções não podem herdar de outras funções. Herança permite que se construa e estenda continuamente classes desenvolvidas por você mesmo ou por outras pessoas, sem nenhum limite. Começando da classe mais simples, pode-se derivar classes cada vez mais complexas que não são apenas mais fáceis de debuggar, mas elas próprias são mais simples". "C++ permite não apenas herança simples, mas também múltipla, permitindo que uma classe incorpore comportamentos de todas as suas classes bases. Reutilização em C++ se dá através do uso de uma classe já existente ou da construção de uma nova classe a partir de uma já existente".

2.9 Classes derivadas

"Derivar uma classe de outra aumenta a flexibilidade a um custo baixo. Uma vez que já existe uma classe base sólida, apenas as mudanças feitas nas classes derivadas precisam ser depuradas".

2.10 O que não é herdado

"Nem tudo é herdado quando se declara uma classe derivada. Alguns casos são inconsistentes com herança por definição":

- Construtores
- Destrutores
- Operadores new
- Operadores de atribuição (=)
- Relacionamentos friend
- Atributos privados

Classes derivadas invocam o construtor da classe base automaticamente, assim que são instanciadas.

2.11 Membros de classes protected

Além dos especificadores public e private para ocultar atributos das classes, temos ainda o protected, que de fora da classe esse atributo funciona como um private, ou seja,

não é acessível fora da classe mas a diferença está na herança, já que um atributo private de uma classe base não é visível na classe derivada, já um protected é e continua sendo protected na classe derivada.

2.12 Construtores e Destrutores

"Quando uma classe é instanciada, seu construtor é chamado. Se a classe foi derivada de alguma outra, o construtor da classe base também precisa ser chamado. A ordem de chamada dos construtores é fixa em C++. Primeiro a classe base é construída, para depois a derivada ser construída. Se a classe base também deriva de alguma outra, o processo se repete recursivamente até que uma classe não derivada é alcançada. Desta forma, quando um construtor para uma classe derivada é chamado, todos os procedimentos efetuados pelo construtor da classe base já foram realizados".Os destrutores são chamados na ordem inversa dos construtores. Primeiro, os atributos mais especializados são destruídos, depois os mais gerais.

2.13 Herança pública x Herança privada

Os especificadores de acesso private e public podem ser usados na declaração de uma herança.As heranças são private por padrão.

Estes especificadores afetam o nível de acesso que os atributos terão na classe derivada:

Private: todos os atributos herdados (public, protected) tornam-se private na classe derivada.

Public: todos os atributos public são public na classe derivada, e todos os protected também continuam protected.