UTFPR - UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Campus Campo Mourão

[1]Alexandre Aparecido Scrocaro Junior R.A.: 2135485

[1]alexandre.2001@alunos.utfpr.edu

RESENHA DO CAPÍTULO 1 DO LIVRO "Programação orientada a objetos com c++" de Renato Borges e André Luiz Clinio

1. QUALIDADE DO SOFTWARE

Aqui analisa-se o que é necessário para produzir um programa de qualidade. Porém, não é uma tarefa simples atingir tal qualidade, há uma série de fatores a serem cumpridos. Para tal, o programa deve ser "rápido, confiável, modular, estruturado, modularizado e etc.". Isso dito, define-se que o software deve respeitar alguns elementos de qualidade externa - eficiência, facilidade de uso, extensibilidade, etc - bem como de qualidade interna - legibilidade, modularidade, etc.

1.1 QUALIDADE EXTERNA

Fatores que devem ser satisfeitos:

- Corretude
    - O programa é capaz de responder adequadamente aos comandos conforme sua especificação;
- Robustez
    - O programa funciona corretamente mesmo sob condições não previstas, anormais;
- Extensibilidade
    - O quão fácil é para fazer mudanças na especificação de um programa
    - Aqui, tem-se dois principais pontos: a simplicidade de design - arquitetura simples para ser facilmente modificada - e a descentralização - módulos autônomos para, quando necessária a modificação, menos módulos são alterados.
- Capacidade de reuso
    - Como diz o nome, é a capacidade do programa de ser reutilizado, totalmente ou em partes.
    - Isso ocorre quando se percebe que há um padrão específico numa nova aplicação. Com isto, pode-se utilizar antigas soluções para novos programas com antigos problemas.
- Compatibilidade
    - É a facilidade com que o programa é combinado com outros.
    - Normalmente, há uma interação entre diversos programas na qual o resultado de um é utilizado para entrada noutro.

- Outros
    - Além dos supracitados, existem outros aspectos também importantes:
        - Eficiência: É o bom aproveitamento dos recursos computacionais como processadores, memória, dispositivos de comunicação, etc.
        - Portabilidade: É a facilidade com que um programa pode ser transferido de uma plataforma
        - Facilidade de uso: É a facilidade de aprendizagem de como o programa funciona, sua operação, etc.

1.2 QUALIDADE INTERNA

Internamente há a modularização, e é graças à ela que se atinge a qualidade externa. Dificilmente encontra-se uma forma de especificar o que é a modularidade, mas pode-se dizer que ao utilizar esta técnica deve-se produzir módulos autônomos, coerentes e organizados em uma estrutura robusta.

A seguir se encontra cinco critérios para criar a modularidade do design:

- Decomposição
    - Deve-se decompor o problema em subproblemas, facilitando, assim, a resolução do problema como um todo.
- Composição
    - Aqui, o programa deve ser produzido pensando numa possível combinação com outros para produzir novos sistemas. Está diretamente relacionada com a reutilização: pedaços do programa podem ser utilizados em outros contextos.
- Entendimento
    - Módulos são compreendidos quando separados, assim a manutenção torna-se simples
- Continuidade
    - Uma pequena mudança na especificação do problema resulta em alterações em um único ou poucos módulos. Tal alteração não tem reflexos na arquitetura geral do sistema.
- Proteção
    - Sob condições anormais - tempo de execução, falta de espaço em disco, falhas de hardware, etc - seus efeitos ficam restritos àquele módulo ou pelo menos se propaga a poucos módulos vizinhos. Não se considera a correção de erros, mas sua propagação.

Já com os critérios em mente, pode-se, agora, discutir os princípios para obtenção da modularização de um programa:

- Linguística modular
    - Módulos devem respeitar a sintaxe da linguagem utilizada no programa.

- Poucas interfaces
    - Um módulo se comunica o mínimo possível com outro.
- Pequenas interfaces
    - Este está relacionado com o tamanho da interface: os canais da comunicação intermodular devem ser limitados.
- Interfaces explícitas
    - Impõe limitações no número de módulos que se comunicam e na quantidade de informações trocadas, além de ter que explicitar esta comunicação.
- Ocultação de informação
    - Se todo módulo é conhecido através de uma descrição oficial e explícita: toda informação sobre um módulo deve ser privada para outro a não ser que seja “pública”. Assim, se um módulo precisa ser alterado, outros que o utilizam não sofrem interferência.

2. CALCULADORA RPN EM C

Neste tópico, o autor mostra um código de uma calculadora desenvolvida em C, utilizando pilha para armazenamento, também mostra como a pilha é utilizada.

```c
#include <stdlib.h>
#include <stdio.h>
#include "stack-c.h"

int getop(struct Stack* s, int* n1, int* n2)
{
    if (empty(s))
    {
        printf("empty stack!\n");
        return 0;
    }
    *n2 = pop(s);
    if (empty(s))
    {
        push(s, *n2);
        printf("two operands needed!\n");
        return 0;
    }
    *n1 = pop(s);
    return 1;
}

int main(void)
{
    struct Stack* s = createStack();
    while (1)
    {
        char str[31];
        int i;
        printf("> ");
        gets(str);
        if (sscanf(str, " %d", &i)==1) push(s, i);
        else
        {
            int n1, n2;
            char c;
            scanf(str, "%c", &c);

            switch(c)
            {
                case '+':
                    if (getop(s, &n1, &n2)) push(s, n1+n2);
                    break;
                case '-':
                    if (getop(s, &n1, &n2)) push(s, n1-n2);
                    break;
                case '/':
                    if (getop(s, &n1, &n2)) push(s, n1/n2);
                    break;
                case '*':
                    if (getop(s, &n1, &n2)) push(s, n1*n2);
                    break;
                case 'q':
                    return 0;
                default:
                    printf("error\n");
            }
        }
        {
            int i;
            for (i=0; i<s->top; i++)
            printf("%d:%6d\n", i, s->elems[i]);
        }
    }
}
```

Fonte: capítulo 1 do livro de Borges e Clínio (há também a implementação da pilha nele, porém considerei desnecessária para a resenha).

3. TIPOS ABSTRATOS DE DADOS

Para a abstração de dados adequada a este propósito os objetos devem ser classificados de acordo com o seu comportamento esperado.

Para que se possa abstrair dados de forma adequada ao seu propósito, os objetos devem ser classificados de acordo com o comportamento esperado para tal.

- Classes em C++
    - Esta é a base de POO, é uma extensão de uma estrutura, que passa a ter também funções dentro. Assim, tipos abstratos de dados são definidos pelas operações. No exemplo da calculadora, a pilha é um candidato natural a se tornar uma classe.

- O paradigma da orientação a objetos
    - Seus componentes são objeto, mensagem, classe, instância e método; sendo definidos como:
        - Objeto: é uma abstração encapsulada que tem um estado interno dado por uma lista de atributos cujos valores são únicos para o objeto.
        - Mensagem: é representada por um identificador deixando clara a ação que o objeto deve tomar.
        - Classe: é um modelo para a criação de um objeto. Incluindo: nome para o tipo de objeto, lista de atributos e lista de mensagens com os métodos que o objeto desta classe sabe responder.
        - Instância: é um objeto que tem suas propriedades definidas na descrição da classe. Os valores dos atributos são propriedades únicas.
        - Método: é uma lista de instruções que define como um objeto responde a uma mensagem. Toda mensagem em uma classe tem um método correspondente.

    - Logo, no c++:
        - classes = estruturas
        - objetos = variáveis do tipo de alguma classe (instância de alguma classe)
        - métodos = funções de classes
        - enviar uma mensagem para um objeto = chamar um método de um objeto.

    - Resumindo: “Objetos são instâncias de classes que respondem a mensagens de acordo com os métodos e atributos, descritos na classe.”