Prova de Programação Orientada a Objetos

Adolfo Gustavo Serra Seca Neto

26 de junho de 2003

1 Questão I

1.1 Como implementar Software IC's em linguagens orientadas a objetos? Vantagens e desvantagens

De acordo com Brad Cox, um Software IC é um componente de software plugável, o equivalente em software aos circuitos integrados no hardware. Com eles, pode-se construir uma aplicação juntando vários Software IC's de diferentes fornecedores. E pode-se atualizar uma aplicação trocando um Software IC velho por um novo. A programação orientada a objetos permite que os desenvolvedores criem objetos reutilizáveis que se comunicam, e portanto torna possível a criação de Software IC's.

Toda linguagem orientada a objetos apresenta características do próprio paradigma (herança, encapsulamento e outras, como veremos depois) que permitem a implementação de Software IC's. Além disso, as três linguagens orientadas a objetos abaixo apresentam, cada uma, vantagens e desvantagens particulares para a implementação de Software IC's:

1.1.1 Java

As principais vantagens apresentadas por Java são:

- Java não implementa o conceito de herança múltipla, mas implementa o conceito de interface. Uma interface é um conjunto de métodos. Se duas classes diferentes implementam uma interface, então existe um subconjunto de métodos que ambas implementam. Deste modo, em Java, tipo é igual a classe ou interface. Isto é, pode-se dizer que um certo método espera um objeto que (i) é instância de uma certa classe ou que (ii) é instância de uma classe que implementa uma certa interface.

- Existem várias tecnologias (Java Beans e Entreprise Java Beans, entre outras) para a construção de componentes em Java. Além disso, Java vem com um conjunto rico de bibliotecas de classe padrão. Outras bibliotecas podem ser obtidas ou adquiridas.

- Java é uma linguagem multiplataforma. Portanto, uma classe escrita em Java pode ser utilizada em qualquer ambiente computacional com suporte a Java, sem necessidade de recompilação.
- Java é uma linguagem bastante popular – logo, existe um bom mercado para quem queira vender software IC's. E as classes ou bibliotecas de classe podem ser entregues sem o código-fonte.

A principal desvantagem de Java é que a checagem de tipos é estática, feita em tempo de compilação. Isso limita um pouco o polimorfismo (apesar de o conceito de interfaces atenuar este problema). Por outro lado, os "engenheiros de software" tradicionais gostam disso pois diminui a possibilidade de erros por um objeto não entender uma mensagem enviada a ele.

1.1.2 C++

C++ é uma linguagem bastante utilizada, com diversos compiladores disponíveis. Portanto, existe um mercado amplo para o desenvolvimento de Software IC's (SI's). Diversas ferramentas estão disponíveis para auxiliar esse desenvolvimento. E a biblioteca padrão de classes de C++ (Standard C++ Library), da qual faz parte a Standard Template Library (STL), já possui muitas classes para facilitar o desenvolvimento de Software IC's. STL, por exemplo, fornece diversas estruturas de dados prontas para usar. O próprio conceito de template facilita a criação dos SI's pois permite a parametrização de classes e funções.

As desvantagens de C++ são que o conceito de tipo é igual ao de classe, o que limita o polimorfismo. A checagem de tipos é estática, feita em tempo de compilação, o que, como já vimos, pode ser considerado tanto uma vantagem como uma desvantagem. C++ não é multiplataforma e o código-fonte só pode ser recompilado em outras plataformas se seguir o padrão ANSI ou algum outro padrão proprietário.

Outra característica de C++ que pode ser considerada tanto uma vantagem como uma desvantagem é a herança múltipla. Se por um lado ela permite maior flexibilidade por outro pode deixar a hieraquia de classes muito confusa.

1.1.3 Smalltalk

Smalltalk é uma linguagem orientada a objetos pura. Isso é uma vantagem pois aumenta a produtividade do programador após ele ter aprendido a programar orientado a objetos. Por outro lado, isso pode atrasar um pouco o domínio da linguagem. O excelente ambiente de desenvolvimento e a rica hierarquia de classes que já vem com o Smalltalk facilitam bastante o desenvolvimento de SI's.

Em Smalltalk, não há checagem de tipos. Aqui, um tipo é apenas o conjunto de métodos que um objeto pode executar. Isso permite flexibilidade mas ao mesmo tempo aumenta a chance de um erro em tempo de execução devido a um objeto não entender uma mensagem. Por outro lado, o mercado para SI's em Smalltalk é menor, por ser uma linguagem menos utilizada.

1.2 O relacionamento entre Software IC's e outros conceitos de orientação a objetos

1.2.1 Software IC's e Padrões de Desenho

Uma vez que padrões de desenho representam boas soluções de projeto, eles devem ser usados no desenvolvimento de SI's. Por exemplo, o padrão Singleton pode ser utilizado no desenvolvimento de um SI PrinterSpooler para controlar a quantidade de instâncias do Printer Spooler. E com certeza já devem existir padrões específicos para o desenvolvimento de SI's. Portanto, esses conceitos são complementares.

1.2.2 Software IC's são exclusividade de orientação a objetos?

SI's podem ser feitos como tipos abstratos de dados e portanto não estão restritos ao mundo orientado a objetos.

1.2.3 Software IC's e as principais características das linguagens orientadas a objetos

As características das linguagens orientadas a objetos facilitam muito o desenvolvimento de SI's. Em primeiro lugar, há possibilidade de criação de classes, que são fábricas de objetos. Em segundo lugar, os objetos se comunicam através de mensagens. A herança permite que se escreva um SI a partir de outros SI's e outras classes, estendendo, adaptando, modificando ou unindo. O encapsulamento e o escondimento de informação diminuem a área de superfície de um SI, isto é, a quantidade de coisas que se precisa saber para usar um SI. A delegação também pode ser usada para "encaixar" um SI em um sistema.

1.3 Dificuldades para implementar Software IC's

1.3.1 C++

Em C++, quando se pretende que uma subclasse possa redefinir um método da superclasse, é preciso que este método esteja definido na superclasse como virtual. Isso limita o polimorfismo e a liberdade de extensão de classes.

1.3.2 C

C não é orientada a objetos. Esta é a principal limitação. Além disso, assim como C++, C não é multiplataforma e sua sintaxe é complicada.

1.3.3 Pascal

Pascal também não é orientada a objetos, apesar de já existirem extensões orientadas a objetos. Além disso, Pascal é uma linguagem voltada para o ensino, cuja sintaxe dificulta a produtividade do programador.

2 Questão II

2.1 Quais conceitos estudados foram aplicados no projeto?

Diversos conceitos estudados foram aplicados no projeto, conforme veremos abaixo:

2.1.1 Conceitos de orientação a objetos

Os conceitos mais básicos de orientação a objetos (classe, herança, mensagem, método) foram utilizados desde a fase de análise. Todas as entidades do sistema foram vistas como objetos e classes. Por exemplo, na fase 2 tínhamos uma classe Docente. Cada professor era um objeto desta classe. Quando queríamos ver o histórico acadêmico de um professor (a lista de disciplinas que ele havia ministrado), precisávamos enviar uma mensagem historicoAcademico para este objeto. Mas para que algo fosse feito era necessário haver algum item na lista de itens de carreira do docente. E esta lista é atualizada pelo método notas da classe Turma: cada vez que são atribuídas notas para os alunos de uma turma, cria-se um item de carreira para um professor, constando que o professor ensinou aquela disciplina para uma certa quantidade de turmas e o total de alunos daquelas turmas.

O conceito de herança foi utilizado na fase 1 para definir que um instrutor é um usuário. Isso permitiu polimorfismo no momento da criação de empréstimos: um empréstimo poderia ser feito tanto para um usuário simples quanto para um instrutor, mas os instrutores recebiam tratamento diferente (tempo de empréstimo maior). O conceito de encapsulamento foi utilizado, por exemplo, quando se criava um aluno apenas passando nome, rg, cpf, endereço e senha. O usuário do sistema não precisa saber que a partir daí pode ser criada uma instância da classe Pessoa, pois isso é representação interna.

A classe Pessoa foi usada também para fazer composição e delegação, em vez de herança. As classes Docente e Aluno da fase 2 não eram subclasse de Pessoa, mas sim continham uma instância da classe Pessoa. Deste modo, pode haver um aluno que compartilha a mesma pessoa com um docente, isto é, uma pessoa que é ao mesmo tempo aluno e docente.

2.1.2 Conceitos de desenvolvimento de sistemas

Neste sistema, procuramos minimizar a "área de superfície". Havia uma interface mínima que cada classe deveria satisfazer: o resto era opcional. Tentamos tornar mínima a comunicação entre as classes. Também procuramos fazer testes sistemáticos e, sempre que possível, escrever os testes antes de escrever o código.

2.1.3 Padrões de desenho

Utilizamos o padrão Strategy para a classe PolíticaDeMatrícula e suas subclasses: MatrículaPorOrdem e MatrículaPorSorteio. O padrão Strategy permite que uma família de algoritmos seja definida e que um destes algoritmos seja escolhido.

O padrão Facade poderá ser utilizado na fase 3 para definir uma interface unificada de alto nível para um conjunto de interfaces do sistema. Assim o uso do sistema pode ser facilitado.

O padrão Observer, que define uma dependência um-para-muitos entre objetos de tal modo que quando um objeto muda seu estado todos os seus dependentes são notificados e atualizados automaticamente, será utilizado na interface gráfica da fase 3.

2.2 O que auxiliaria a implementação em C++?

2.2.1 A UML do projeto

A UML do projeto já está pronta. Muito pouco precisará ser modificado para a implementação em C++. Teremos que alterar os nomes dos métodos, pois o padrão do Smalltalk é diferente. Alguns métodos podem desaparecer e outros irão surgir.

2.2.2 Os testes já escritos

Já foram escritos testes para a versão Smalltalk. Estes testes devem servir como base para os testes da versão C++. Como vimos no curso, é uma boa técnica de desenvolvimento escrever os testes antes de implementar. E a partir daí tentar desenvolver o sistema incrementalmente, o que com certeza é mais fácil em Smalltalk.

2.2.3 Os padrões de desenho utilizados e a STL

Os padrões de desenho que utilizamos podem nos auxiliar pois o próprio livro de padrões fornece dicas para a implementação de classes baseadas nos padrões tanto em Smalltalk quanto em C++ (para um bom exemplo, ver a definição do padrão Singleton). Além disso, podemos estudar outros padrões e tentar ver se mais algum se aplica ao projeto.

A STL (biblioteca de classes template padrão) deve ser utilizada para implementar as coleções usadas extensivamente em nosso projeto. Com certeza o código vai ser bem menos claro e intuitivo do que o código em Smalltalk e, para quem não domina a STL, vai ser mais difícil descobrir como fazer as mesmas coisas em C++.

2.3 Avaliação da integração

2.3.1 Integração da classe Pessoa

A única integração feita foi a da classe Pessoa. Com a fase 2 foi criada a classe Pessoa, que servia para armazenar dados comuns a Docentes e Alunos. Foi observado que os objetos das classes Usuário e Instrutor também contém informações da classe Pessoa. Portanto, estas classes passaram a conter também uma instância da classe Pessoa. Feito isto, para evitar a repetição e a inconsistência de informações, os objetos dessas quatro classes não devem criar novas instâncias da classe Pessoa a menos que se trate realmente de uma nova pessoa.

2.3.2 Integração do resto do sistema

A integração do resto do sistema se deu apenas com a colocação de todas as classes numa mesma categoria. O próximo passo será criar uma interface gráfica (a fase 3 do projeto). Outra possível integração é perguntar ao cliente se um instrutor é um docente e, em caso afirmativo, unir essas duas classes. Além disso, começamos também a desenvolver um sistema de pagamento para funcionários (e um docente é um funcionário - logo, haverá necessidade de integração).