ISEL | DEETC

Programação de Sistemas Computacionais

Serie 05 de Exercícios

Semestre de Verão de 2009/2010

Autor:
31401 – Nuno Cancelo

Indicie

Prefácio

Nesta série de exercícios pretende-se a aplicação de conhecimentos obtidos na aulas, nomeadamente o paradigma da programação orientado por objectos, fornecendo ferramentas para implementar princípios inerentes a linguagens de mais alto nível como seja a herança e polimorfismo. Estes princípios, complexos de entender ao início, são fundamentais na forma como os objectos interagem entre si.

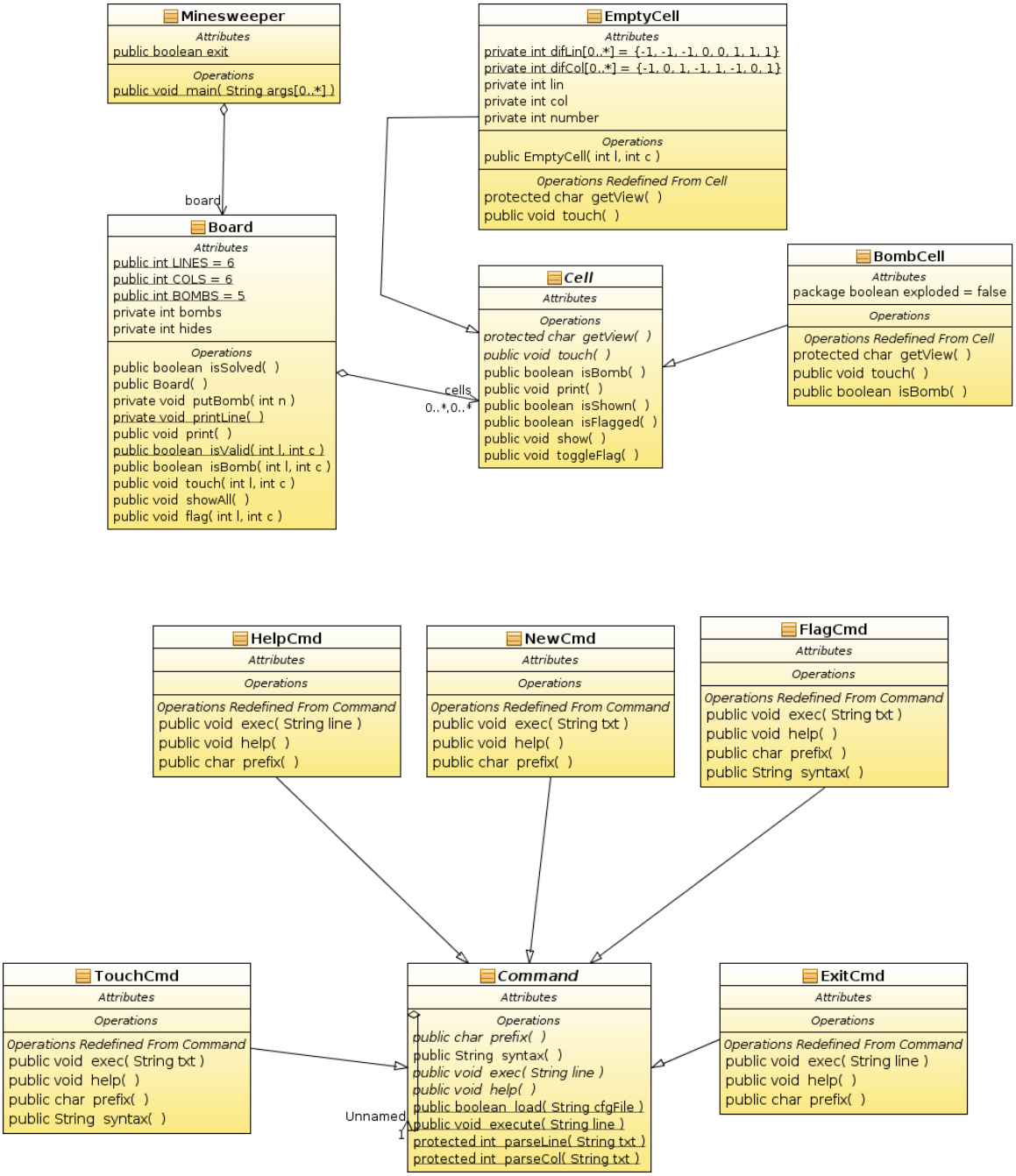
Outro dos objectivos é trabalhar bibliotecas dinâmicas que podem ser carregadas em tempo de execução, dando mais funcionalidades ao programa, sem ter que alterar uma linha de código no programa principal, no entanto é necessário que uma interface esteja estipulada para que o funcionamento seja “pacifico”.

O fornecimento do código fonte numa linguagem de alto nível(Java) permitiu obter conhecimentos extra nessa linguagem, como seja a forma como o carregamento bibliotecas.

Análise do Programa em Java

O código fonte disponibilizado continha 13 ficheiro incluindo o Makefile e o ficheiro de configuração com os comandos.

O primeiro passo foi abstracção completa do código para poder visualizar o problema de uma forma generalizada e para tal produziu-se o diagrama UML do programa.



Como é perceptível no diagrama, como faz sentido dada os objectivos do trabalho, o programa está separado em duas partes:

- O carregamento dinâmico de comandos
- O jogo em si, com a instanciação de um tabuleiro de jogo e o

tabuleiro constituído por várias células.

Implementação

Command e Descendentes

A partida para a tradução do código foi dada com a implementação dos comandos, uma vez que à primeira vista não estava directamente relacionada com a instância de MineSweeper.

Foram criados ficheiro de código(c) e respectivos header(h) equivalente a cada ficheiro de código java, no entanto ao longo do trabalho dadas algumas implementações que não me deixavam confortáveis foram alteradas dando origem a mais uns quantos ficheiros.

Seguindo esta linha de raciocínio, senti-me mais confortável em alterar a implementação de Command, separando em dois ficheiros:

- Command – Tem uma implementação abstracta de um comando genérico e que todos os comandos que estendem dele implementam os métodos em falta e redefinem a sua tabela de métodos para que executem a função correcta.
- Command_Loader - Tem a implementação dos métodos necessários para carregar e descarregar as bibliotecas dinâmicas.

Foi estipulada uma interface pública para que estes dois objectos pudessem comunicar sem que um estivesse dependente do outro, mas sim dependente da interface que implementam ou que usem para comunicação. Neste sentido foi criada a interface CommandInterface.h, que descreve um comando e um array de Comandos, dando espaço de criatividade para quem usa a interface, estabelecendo umas poucas regras.

Uma das regras que esta interface estabelece é a utilização de uma outra interface, a GameInterface.h, que irá servir como ponte de comunicação entre os comando e um jogo de MineSweeper. A necessidade desta interface surgiu quando na descrição de alguns métodos, havia chamadas ao jogo/tabuleiro principal de MineSweeper.

Admito que o objectivo seria a tradução do original em java para esta versão em C, mantendo as funcionalidades e afins, no entanto não me faz sentido ter código que será carregado dinamicamente dependente de algo, ou seja “agarrado” a uma implementação.

Após uma curta conversa com o docente, surgiu esta hipótese (válida, diga-se) que o código torna-se independente, podendo-se utilizar em outras implementações, desde que o mesmo implemente esta interface.

Assim desta forma, a implementação dos comandos definidos assim como os novos solicitados foram bastante simples, não havendo necessidade de mexer muito no código.

Board e Células

Caminhando para a segunda parte da implementação fui implementar o jogo "principal", começando pela implementação dos objectos Cell, BombCell, EmptyCell e por fim Board.

Como é visível no UML, BombCell e EmptyCell são Cell com características especiais que os tornam únicos. Também é evidente que Board é constituído por um conjunto de Cell (independentemente do Cell é na realidade).

À semelhança do que havia se passado nos comandos, senti a necessidade de isolar o programa principal da implementação do tabuleiro e respectivas células. Isto porque MineSweeper tem um tabuleiro constituído por células, mas as células não tem que conhecer a estrutura de MineSweeper, pois não lhes dizem respeito, mas têm que trabalhar com o tabuleiro a que pertencem.

Neste sentido, foi alterada a assinatura do comando touch das células para que lhes fosse indicada o endereço do Board a que pertencem, podendo desta forma isolar-se do mundo exterior e proceder à acção que lhes é destinada.

MineSweeper

MineSweeper tornou-se num objecto independente das outras implementações, podendo-se alterar a sua estrutura, acrescentar funcionalidades, mudar a forma de implementação tendo que respeitar as interfaces de comunicação de Board (para ter uma instância e aceder às suas funções), de Command_Loader (para ter acesso às funções para carregar/decarregar/executar os comandos) e de GameInterface (para que os comandos e o tabuleiro interajam).

GameInterface

Esta interface é constituída por uma estrutura que somente por tabela de métodos necessários para o funcionamento do jogo tornando, como já mencionei, independente a interacção dos objectos.

Concordo que o código de MineSweeper não é elegante, contudo é um mal necessário de forma a evitar a ter mais ficheiros com essa implementação.

Novos Comandos Load e Save

Para a implementação destes comandos houve a necessidade de implementar algumas funções no objecto Board:

- Export: que exporta num formato reconhecível um tabuleiro de jogo, para que o jogo seja guardado na forma que for desejável, sendo as únicas validações no intuito de evitar exportar dados quando um jogo já esteja terminado.
- Import: importa no formato que a função anterior disponibilizou e

tenta gerar um novo tabuleiro. Tenta criar um novo tabuleiro com o argumento recebido, se falhar desfaz o que alocou. Caso seja bem sucedido, limpa o tabuleiro existente no jogo e copia o novo tabuleiro.

- Foram criadas funções para a cópia do tabuleiro e processamento do tabuleiro de forma a indicar no tabuleiro o numero de bombas a que uma célula vazia e está “encostada”.
- Foi adicionada um atributo à estrutura de board a indicar se o jogo já acabou ou não, tornando a implementação de algumas funções mais rápidas.

O GameInterface foi alterado para suportar estas alteração, tendo as todas outras não sofrido alterações.

Os comandos save e load foram implementados de forma a gravar e ler um ficheiro binário, fazendo uma pequena alteração para tornar mais difícil a leitura directa do ficheiro.

Talvez não fosse necessário ter este trabalho todo se tivesse mantido um estrutura dependente entre si, mas penso que torno o código mais versátil desta forma, podendo num futuro disponibilizar mais funcionalidades pela interface de Board e de GameInterface, sem colocar em risco as funcionalidades já implementadas.

Conclusão

Ao longo deste trabalho fui-me apercebendo de vários aspectos da linguagem C e de programação Object Oriented nesta liguagem, que demonstra ainda mais potencialidade de uma linguagem “not oriented” em todos os aspectos de uma programação.

O facto de ser dado um “jogo” para traduzir, torna-se mais simpático e de certa forma evidente algumas alterações necessárias, para tornar este programa funcional no modo OO e (nesta implementação) isolado em blocos.

Vários aspectos foram compreendidos (digo eu) nomeadamente uma máxima de um guru que eu respeito imenso que foi deveras ilucidativa, que foi:

“Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. Os construtores não alocam memória. ...!” :D

Bibliografia

- Computer Systems, A Programmer's Perspective 2nd Edition
  - Randal E. Bryant, David R. O'Hallaron
    Editora: Prentice Hall
  - http://csapp.cs.cmu.edu/
- http://www.d.umn.edu/~gshute/C/ooC/ooC.html
- http://accu.informika.ru/acornsig/public/articles/oop_c.html
- http://ldeniau.web.cern.ch/ldeniau/html/oopc/oopc.html
- http://www.cs.rit.edu/~ats/
  - nomeadamente:
  - Objekt-orientierte Programmierung mit ANSI-C
  - 1994, Hanser, Munich, ISBN 3-446-17426-5 (Versão Englisa, claro está)