Relatório: Desenvolvimento de Aplicações em Swing

Ricardo Rouco núm 39502 e Fábio Botelho núm 47031
Grupo 33

06/06/09

Conteúdo

1 Introdução **3**

2 Desenvolvimento da Aplicação **3**
2.1 Model 3
2.2 View 4
2.3 Controller 4
2.4 Conclusões 5

3 Camada de Persistência **5**
3.1 Testes 6
3.2 Conclusões 7

4 Bibliografia **7**

1 Introdução

Este relatório aborda a resolução das tarefas propostas na 3º fase do segundo projecto de LI3.

Para este foi desenvolvido uma aplicação utilizando Java Swing , abordando o modelo de engenharia de software MVC(Model-View-Controller). E também foram testados e comparados dois métodos diferentes soluções de persistência.

2 Desenvolvimento da Aplicação

Para este projecto , tentámos , tal como indicado por os docentes, implementar a aplicação baseando-nos no modelo MVC. De acordo com este modelo e as ferramentas utilizadas, a nossa aplicação traduz-se nos seguintes factores:

- A implementação da camada Model, que suportará a informação associada aos clientes. Esta será representada pela nossa classe Clientes , que oferece operações sobre os seus próprios dados, e é completamente isolada , isto é não oferece qualquer acção de input/output, nem têm em conta a implementação da interface com o utilizador.
- A implementação do View - o desenho da interface gráfica e os seus respectivos componentes
- A implementação do Controller , que detecta as acções do utilizador com a interface gráfica e responde comunicando com a camada Model

De seguida abordámos a implementação dos mesmos.

2.1 Model

Para o suporte da informação escolhemos uma classe Clientes, que suporta duas hashmap's - indexadas por nome e nif - de referências ao mesmo cliente.

A escolha desta estrutura deve-se ao facto da nossa familização com a hashing , pois este já foi utilizado no trabalho anterior em C , e na fase anterior também. Além disso sabemos que a hashmap é a estrutura mais rápida que existe para pesquisa , inserção e remoção.

A novidade desta classe , nesta fase , é a introdução da sua extensão á classe Observable. Esta extensão é necessário, pois de acordo com o modelo, é necessário que a camada Model , embora independente da implementação da interface com

o utilizador, possua mecanismos de notificação sobre alguém que deseje seguir as suas alterações. Para tal foram implementados os métodos de notificação dos seguidores, aquando a alteração de uma instância de clientes.

Em suma o modelo é uma classe normal , de java, que suporta operações de adição , pesquisa , remoção e a sua implementação é independente do meio escolhido para interagir com o utilizador.

2.2 View

O desenho da interface gráfica foi conseguido utilizando o IDE Netbeans. Mesmo assim o desenho da listagem de clientes que obedecesem a uma palavra chave (requisito 7 do enunciado) , foi conseguido através do desenho de uma tabela que foi escrita sem recurso a este.

O desenho que apresentámos é simples , e baseado nos próprios exemplos apresentados pelos docentos. Não foi prioridade , de forma alguma , o look and feel da aplicação , pois achámos que não era prioridade do trabalho. Em vez disso tentámos apresentar uma aplicação intuitiva e simples, tarefa esta que não é muito díficil visto as poucas operações que foram necessário implementar.

2.3 Controller

A camada de controle oferece mecanismos de reacção às acções do utilizador com a interface gráfica, sendo que é obrigação da mesma responder a estes.

Para tal , tivemos que , através do Swing, codificar as respostas ás acções do utilizador. O Swing oferece um mecanismo , baseado também em MVC, entre os elementos da interface gráfica e acções, estas podem ser associadas a um elemento contido na interface , e são invocadas (são métodos) quando o utilizador interage com o componente.

Nós apenas tivemos que codificar as acções apropriadas e associar as mesmas aos seus componentes, as Acções por sua vez invocam novos componentes da camada View para interagir com o utilizador , como jDialogs, e através do input do utilizador , comunicam com a camada Model.

2.4 Conclusões

Não existem grandes conclusões a tirar. Era lógico que a camada gráfica da aplicação deveria ser desenvolvida em separado da camada de dados, tal como o modelo MVC propõe. E foi o que realizámos.

No que diz respeito à abordagem do Swing ao modelo , não foi suficientemente explorada por nós para podermos concluir alguma coisa, tal como também não temos bases de comparação de desenvolvimento de aplicações com componentes gráficos , ou baseada em conceitos utilizados aqui , para concluirmos algo sobre a sua utilização

3 Camada de Persistência

Nesta fase explorou-se também a implementação de dois métodos distintos para assegurar a persistência da informação em Java: streams de texto e streams de objectos, através destes era possível salvaguardar o estado da nossa aplicação. As soluções implementadas deviam ser testadas , dentro dos parâmetros da fase anterior , isto é , os métodos envolvidos - leitura e escrita - foram registados os tempos necessários para vários patamares de clientes (5000, 1000, 15000 e 18000), e comparadas entre si.
A implementação da camada de persistência através de streams de texto , já tinha sido apresentada na fase anterior, ainda assim , realizámos de novo os testes visto que ocorreram várias alterações ao módulo/classe Cliente e Clientes. Estas justificam que se realize os testes novamente , pois , caso contrário , os resultados obtidos para a leitura/escrita de clientes através de streams de objectos não poderiam ser comparados com os resultados antigos, visto os objectos serem diferentes.

A implementação da leitura/escrita através de streams de objectos foi realizada através magia da interface java.util.Serializable, esta por detrás da cortina , implementa um algoritmo de serialização que permite que através dos métodos writeObject e readObject seja possível guardar/ler objectos que possuem referências a outros objectos, o único requerimento desta interface é que os atribuitos dos objectos sejam serializáveis - implementam java.util.Serializable - o que acontece com as nossas classes Clientes (HashMap é Serializável) e Cliente (todos os atribuitos simples são serializáveis). Isto permite que , por exemplo , no caso da nossa implementação de Clientes , que possui duas hashmaps que partilham uma referência para o mesmo objecto , seja possível , sempre preocupações nenhumas adicionais , utilizar o writeObject/readObject , e manter o mesmo objecto partilhado entre as duas hashmap's. Note-se que esta interface extremamente simples - desde

que não seja necessário o override dos métodos writeObject/readObject - é extramente poderosa , pois permite a partilha de Objectos entre máquinas virtuais java (JVM) de qualquer tido , ou mesmo entre processos diferentes dentro da mesma JVM sem qualquer tipo de preocupações como byte-ordering , sistema operativo ... etc, tornando-a um mecanismo poderoso de IPC (Inter Process Communication).

3.1 Testes

Os testes realizados à camada de persistência foram os seguintes : a leitura e escrita dos dois métodos - streams de textos e streams de objectos - para os vários patamares de clientes (5000,1000,15000,18000).

A implementação destes testes foi feito com recurso a uma classe de testes (testaPersistencia) , apresentada no código em anexo na pasta testesDePersistencia, ao blueJ e á sua interface para JUnit tests. Os testes foram realizados vários vezes (na ordem das dezenas) e foram escolhidos os cinco melhores resultados e finalmente calculada a média destes. Mais uma vez pode-se notar que os resultados apresentados , tal como na última fase , contém erros , certamente , mas apesar disto servem o objectivo desta fase - o da comparação dos dois métodos de persistência.

A implementação dos métodos writeObject() , writeText() foram realizadas dentro da classe dos clientes , ao contrário dos métodos de leitura que fazem mais sentido serem implementados fora da classe. Os tempos apresentados para os métodos de leitura têm em conta o tempo de output dos erros apanhados (a adição de clientes já existentes), este tempo não faz diferença porque é o mesmo tanto para o readObject() e readText().

Os testes foram realizados apenas uma vez , todos de seguida , na mesma máquina a correr o mesmo número de processos , sendo que o processo java foi atribuído prioridade máxima (realtime - Windows XP). As características da máquina de teste são as seguintes : Processador Intel Pentium Mobile 1Ghz , com 512Mb de Ram , disco rígido de 5200rpm's.

Resultados dos testes:

Tabela 1: Resultados dos testes

Patamar/Operação	Read Binary	Read Text	Write Binary	Write Text
5000	0.221593000	0.043185000	0.172478000	0.033074000
100000	0.499365000	0.158600000	0.349367000	0.066724000
150000	0.747844000	0.171373000	0.516158000	0.103622000
180000	0.747071000	0.205850000	0.633401000	0.291142000

3.2 Conclusões

Pelo aquilo que pudemos verificar - e que expusemos nesta secção - a utilização da streams de textos para a camada de persistência é mais rápida que a utilização do método de serialização. Isto no entanto já era óbvio antes dos testes realizados, basta pensar que , necessáriamente , exige bastante trabalho computacional envolvido no algoritmo de serialização utilizado por o Java.

No entanto pensámos, que a utilização de streams de objectos é muito mais vantajosa ao contrário das streams de textos. Isto porque , a simplicidade da implementação dos nossos métodos writeObjectToFile/readObjectFromFile é enorme quando comparada com os métodos de leitura e escrita dos campos de clientes para ficheiros de texto. Além disso , posto que a partilha de um ficheiro de texto entre jvm's diferentes pode trazer problemas - como a questão do byte-ordering , favorecemos a utilização de stream's de objectos para salvaguarda do estado da aplicação.

Em suma concluímos que os dois métodos devem ser considerados mediante a situação em causa. Obviamente os métodos de serialização são superiores quando considerados problemas de IPC (Inter Process Communication) ao contrário da utilização de texto.

4 Bibliografia

O'Reilly Java Swing , 2nd Edition - 2002 - Brian Cole, RObert Eckstein, James Elliott , Marc Lov , David Wood
Slides dos Docentes
Documentação online da Sun(Referenciada nos slides)