UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologia
Mestrado em Engenharia Informática
Disciplina: Engenharia de Software
Docente: Prof. Pedro Salgueiro

Test Driven Development

Segundo Trabalho Prático

Trabalho Elaborado Por:
João Aiveca, Nº 10712
Ricardo Dias, Nº11246
Marlene Oliveira, Nº11327

Ano Letivo 2013/2014

Índice

Introdução

Test Driven Development (ou TDD como é frequentemente abreviado) é um método usado no desenvolvimento de *software* no qual são criados testes automáticos antes das funcionalidades que se pretendem testar estarem implementadas. Inicialmente todos os testes escritos falham, visto que nada está implementado. Á medida que o processo de desenvolvimento avança, os testes vão passar ou falhar dependendo das funcionalidades (se estiverem a funcionar como pretendido o teste passa, caso contrário o teste falha). O código escrito inicialmente serve apenas para passar o teste, sendo que mais tarde vai ser reformulado de modo a atingir os patamares de qualidade pretendidos.

Na implementação de um *back-end* de um serviço de gestão documental efetuada neste trabalho utiliza-se TDD como metodologia principal. O sistema implementado permite fazer a gestão dos utilizadores do mesmo (criar, atualizar dados e eliminar utilizador e listar um ou vários utilizadores). Além disto, o sistema deve ainda permitir fazer a gestão dos documentos (carregar, substituir e eliminar utilizador) e listar e fazer pesquisas sobre a base de documentos existente no sistema. Os documentos são carregados em formato de texto e têm uma estrutura definida, que deve ser respeitada para que o documento seja aceite pelo sistema. O sistema irá depender de uma base de dados, onde serão armazenadas todas as informações relativas aos utilizadores e aos documentos.

O Sistema

A implementação do sistema é feita com recurso à linguagem Java, sendo os testes unitários criados com recurso ao JUnit 4. O sistema utiliza uma base de dados H2.

O Acesso Á Base de Dados

De modo a organizar melhor o código criado, os métodos que interagem com a base de dados encontram-se numa classe à parte (DBAccess.java). Os métodos existentes nesta classe são os seguintes:

- ***getConnection()*** – permite obter a ligação (objeto *Connection*) criada quando o objeto DBAcess foi criado. A ligação corresponde a uma ligação à base de dados H2 do sistema;
- ***initialize()*** – permite inicializar a base de dados com as tabelas necessárias para o bom funcionamento do sistema (uma tabela para os dados dos utilizadores e uma tabela para os documentos);
- ***runQuery(String query)*** – permite executar *querys* à base de dados. Este método é usado em várias operações (por exemplo, para adicionar os dados de um utilizador à base de dados ou para adicionar um documento à base de dados).

Pretende-se usar esta classe como uma API para acesso à base de dados, evitando a repetição de código em outros métodos sempre que se pretende executar uma *query* à base de dados. Objetos do tipo *DBAccess* são usados na grande maioria dos testes criados, visto que é frequente ser necessário interagir com a base de dados quando se testam funcionalidades do sistema.

Utilizadores

A classe na qual se efetuam as operações sobre a informação dos utilizadores é a classe Utilizador.java. Existem duas variáveis globais nas quais são armazenadas as informações do utilizador (ID e nome de utilizador) que são passadas como argumento de um dos construtores. O ID deve ser único e o nome do utilizador não pode ser *null*. O construtor vazio serve apenas para que seja possível criar objetos Utilizadores usados apenas para fazer listagens (por exemplo, para quando se pretende apenas listar um utilizador sem criar um novo).

Esta classe tem os seguintes métodos:

- ***String getName()*** – devolve o nome do utilizador que se encontra no objeto Utilizador;
- ***int getID()*** – devolve o ID do utilizador que se encontra no objeto Utilizador;
- ***void addUser(DBAccess dba)*** – este método permite adicionar um utilizador à base de dados do sistema. O objeto DBAccess fornecido como argumento permite aceder à base de dados e executar a *query* que adiciona o utilizador. Neste método começa por se verificar se o nome do utilizador não é nulo. Se isto se verificar, é lançada uma exceção. Caso contrário, a informação do utilizador é adicionada à base de dados;
- ***void UpdateUser(DBAccess dba, int to_upd, String n, int i)*** – este método permite atualizar a informação de um utilizador presente na base de dados. Para tal, o método tem como argumentos um objeto DBAccess que permite aceder à base de dados, uma *String* correspondente ao nome do utilizador e um *int* correspondente ao ID do utilizador cuja informação será atualizada. Caso o nome do utilizador seja *null*, é lançada uma exceção. Caso contrário, o nome do utilizador será atualizado na base de dados;
- ***void deleteUser(DBAccess dba, int it)*** – este método permite remover um utilizador da base de dados, usando para tal o ID do utilizador fornecido como argumento;
- ***String toString()*** – este método permite obter a informação relativa a um utilizador sob a forma de uma *String*;
- ***String listOneUser(DBAccess dba, int id)*** – método que permite listar um utilizador (ID e nome), cujo ID é fornecido como argumento;
- ***ArrayList<Utilizador> listSeveralUsers(DBAccess dba)*** – lista os utilizadores existentes na base de dados (ID e nome de todos).

A classe *UserTest.java* contém todos os testes efetuados para as operações sobre os utilizadores do sistema.

No geral, os testes seguem um padrão simples. Começa por se criar uma ligação à base de dados e inicializá-la. Seguidamente, cria-se um objeto Utilizador (ou vários) que vai ser usado no teste. De seguida executa-se a operação que se pretende testar e, finalmente, faz-se uma outra chamada à base de dados para obter a informação lá armazenada, comparando-se os valores obtidos com os esperados. Se os valores obtidos e esperados forem iguais, o teste passa. Caso contrário, o teste falha. Existem testes em que se pretende testar se são lançadas exceções e outros em que só se pretende verificar se os atributos de um utilizador estão a ser processados de modo correto.

Os testes que fazem verificações de integridade são os seguintes:

- ***testNameInObj()*** - verifica se o nome dado é o mesmo que se encontra no objeto Utilizador;
- ***testIDInObj()*** - verifica se o ID dado é o mesmo que se encontra no objeto Utilizador;

- ***testInsertDuplicate()*** - verifica se é lançada uma exceção quando é inserido um utilizador duplicado na base de dados;
- ***testAddUserNullName()*** - verifica se é lançada uma exceção quando é inserido um utilizador cujo nome tem o valor *null*;
- ***testUpdateUserNullName()*** - verifica se é lançada uma exceção quando se atualiza um nome de utilizador para o valor *null*.

Os testes que verificam se as operações sobre os utilizadores estão a funcionar corretamente são os seguintes:

- ***testAddUserName()*** - verificar se o nome de utilizador atribuído foi inserido corretamente na base de dados;
- ***testAddUserID()*** - verificar se o ID de utilizador atribuído foi inserido corretamente na base de dados;
- ***testUpdateName()*** - verificar se a atualização do nome de utilizador foi efetuada com sucesso (se o nome foi atualizado na base de dados). Não se atualiza o ID do utilizador, visto que este é único e atualizar o mesmo poderia gerar problemas de integridade na base de dados;
- ***testDeleteUser()*** - verificar se um utilizador foi removido com sucesso da base de dados;
- ***testListUser()*** - verifica se a informação obtida quando se lista um utilizador é a correta;
- ***testListSeveralUsers()*** - verifica se a informação obtida quando se listam vários utilizadores é a correta.

Documentos

A classe encarregue da criação dos documentos é *Documento.java*. Esta classe representa num objeto um documento, com campos para cada um dos componentes básicos de um documento (título, corpo, data de criação, data de modificação, ID do documento, ID do utilizador).

A classe *Documento.java* pode ser instanciada de duas formas: fornecendo cada um destes parâmetros individualmente, ou fornecendo um caminho de ficheiro do tipo *String* (que dispensa os argumentos *body* e *title*, visto lê-los do ficheiro).

Qualquer que seja a forma de instanciar, as seguintes funções estão disponíveis para um Documento:

- ***String getTitle()*** – devolve o título do documento que se encontra no objeto Documento;
- ***String getBody()*** – devolve o corpo do documento que se encontra no objeto Documento;
- ***int getID()*** – devolve o ID do documento que se encontra no objeto Documento;
- ***int getUser()*** – devolve o ID do utilizador que está associado ao documento que se encontra no objeto Documento;
- ***Timestamp getD_criacao()*** – devolve a data de criação do documento que se encontra no objeto Documento;
- ***Timestamp getD_alteracao()*** – devolve a data de alteração do documento que se encontra no objeto Documento;
- ***void addDoc(DBAccess dba)*** – este método permite adicionar um documento à base de dados do sistema. O objeto DBAccess fornecido como argumento permite aceder à base de dados e executar a *query* que adiciona o documento. Neste método verifica-se se o título e

corpo do documento não são nulos. Verifica-se também se não foi introduzido um ID de utilizador e se não foi introduzida a data de criação. Em caso afirmativo é lançada uma exceção. Caso contrário, a informação do documento é adicionada à base de dados;

- ***void updateDocTitle(DBAccess dba, String n, int id,Timestamp d_alteracao)*** – este método permite atualizar o titulo de um documento presente na base de dados. Para tal, o método tem como argumentos um objeto DBAccess que permite aceder à base de dados, uma *String* correspondente ao novo título do documento e um *int* correspondente ao ID do documento cuja informação será atualizada. Caso o título do utilizador seja *null*, é lançada uma exceção. Caso contrário, o título do documento será atualizado na base de dados assim como a data de alteração do título do documento;
- ***void updateDocBody(DBAccess dba, String n, int id,Timestamp d_alteracao)*** – este método permite atualizar o corpo de um documento presente na base de dados. Para tal, o método tem como argumentos um objeto DBAccess que permite aceder à base de dados, uma *String* correspondente ao novo corpo do documento e um *int* correspondente ao ID do documento cuja informação será atualizada. Caso o corpo do utilizador seja *null*, é lançada uma exceção. Caso contrário, o corpo do documento será atualizado na base de dados assim como a data de alteração do título do documento;
- ***void updateDocId_user(DBAccess dba, int id_user, int id, Timestamp d_alteracao)*** – este método permite atualizar o utilizador associado a um documento presente na base de dados. Para tal, o método tem como argumentos um objeto DBAccess que permite aceder à base de dados, um ID do utilizador correspondente ao novo utilizador associado ao documento e um int correspondente ao ID do documento cuja informação será atualizada. Caso o id do utilizador seja invalido, é lançada uma exceção. Caso contrário, o novo utilizador associado ao documento será atualizado na base de dados assim como a data de alteração do novo utilizador associado ao documento;
- ***void deleteUser(DBAccess dba, int it)*** – este método permite remover um documento da base de dados, usando para tal o ID do documento fornecido como argumento;
- ***String toString()*** – este método permite obter a informação relativa a um documento sob a forma de uma *String*.

A classe *DocumentoTest.java* contém todos os testes efetuados para as operações de criação, inserção e atualização dos documentos no sistema.

Nestes testes é necessária uma ligação à base de dados e a sua inicialização. É necessário criar um objeto Utilizador e adicioná-lo na base de dados de modo a obter um utilizador para associar a um documento. De seguida cria-se um objeto Documento para ser usado no teste e é executada a operação em que o teste incide. Por fim obtém-se a informação armazenada na base de dados e os valores obtidos são comparados com um valor esperado. Se os valores comparados forem iguais o teste passa. Caso contrário, o teste falha. Existem também testes com o propósito de verificar se as exceções são lançadas. Estas exceções ocorrem quando os atributos não são validos ou existem inconsistências ao nível da base de dados.

Os testes que fazem verificações de integridade são os seguintes:

- ***testFilepathWrongContent()*** - verifica se é lançada uma exceção quando é dado um documento com conteúdo errado;
- ***testFilepathConstructor()*** - verifica se é lançada uma exceção quando é dado um documento da extensão errada;
- ***testFilepathConstructorNullFilepath()*** - verifica se é lançada uma exceção quando é dado um documento tem o valor *null*;

- ***testFilepathConstructorWrongDocumentName()*** - verifica se é lançada uma exceção quando é dado um documento que não existe no disco;
- ***testTitleInObj()*** - verifica se o título dado é o mesmo que se encontra no objeto documento;
- ***testIDInObj()*** - verifica se o ID dado é o mesmo que se encontra no objeto documento;
- ***testBodyInObj()*** - verifica se o corpo dado é o mesmo que se encontra no objeto documento;
- ***testID_userInObj()*** - verifica se o ID do utilizador associado ao documento dado é o mesmo que se encontra no objeto documento;
- ***testTitleNullObj()*** - verifica se é lançada uma exceção quando é criado um documento cujo título tem o valor null;
- ***testBodyNullObj()*** - verifica se é lançada uma exceção quando é criado um documento cujo corpo tem o valor null;
- ***testD_criacaoNullObj()*** - verifica se é lançada uma exceção quando é criado um documento cujo Timestamp de criação tem o valor null;
- ***testInsertDuplicate()*** - verifica se é lançada uma exceção quando é inserido um documento duplicado na base de dados;
- ***testUpdateDocNullTitle()*** - verifica se é lançada uma exceção quando se atualiza um título de um documento para o valor null.
- ***TestUpdateDocNullBody()*** - verifica se é lançada uma exceção quando se atualiza um corpo de um documento para o valor null.
- ***testUpdateDocNullId_user()*** - verifica se é lançada uma exceção quando se atualiza um ID do utilizador associado ao documento para o valor inválido.

Os testes que verificam se as operações sobre os documentos estão a funcionar corretamente são os seguintes:

- ***testAddDocTitle()*** - verificar se o título do documento atribuído foi inserido corretamente na base de dados;
- ***testAddDocBody()*** - verificar se o corpo do documento atribuído foi inserido corretamente na base de dados;
- ***testAddDocID()*** - verificar se o ID do documento atribuído foi inserido corretamente na base de dados;
- ***testAddDocTimestamp()*** - verificar se o *Timestamp* de criação do documento atribuído foi inserido corretamente na base de dados;
- ***testAddDocID_user()*** - verificar se o ID do utilizador associado ao documento atribuído foi inserido corretamente na base de dados;
- ***testUpdateTitle()*** - verificar se a atualização do título do documento foi efetuada com sucesso na base de dados;
- ***testUpdateBody()*** - verificar se a atualização do corpo do documento foi efetuada com sucesso na base de dados;
- ***testUpdateID_user()*** - verificar se a atualização do ID do utilizador associado ao documento foi efetuada com sucesso na base de dados;
- ***testUpdateD_alteracao()*** - verificar se a atualização do *Timestamp* da data de alteração do documento foi efetuada com sucesso na base de dados;
- ***testDeleteDoc()*** - verificar se um documento foi removido com sucesso da base de dados.

Pesquisas

A classe encarregue das pesquisas é DocSearch.java. São utilizadas como variáveis globais *allDocs*, uma *ArrayList* que inclui todos os documentos da BD, e *sourceDatabase*, a DBAccess sobre a qual se pretendem obter dados. Nesta classe estão implementados métodos que pesquisam sobre o conjunto de documentos que estiver presente na base de dados passada como argumento, como apresentados de seguida:

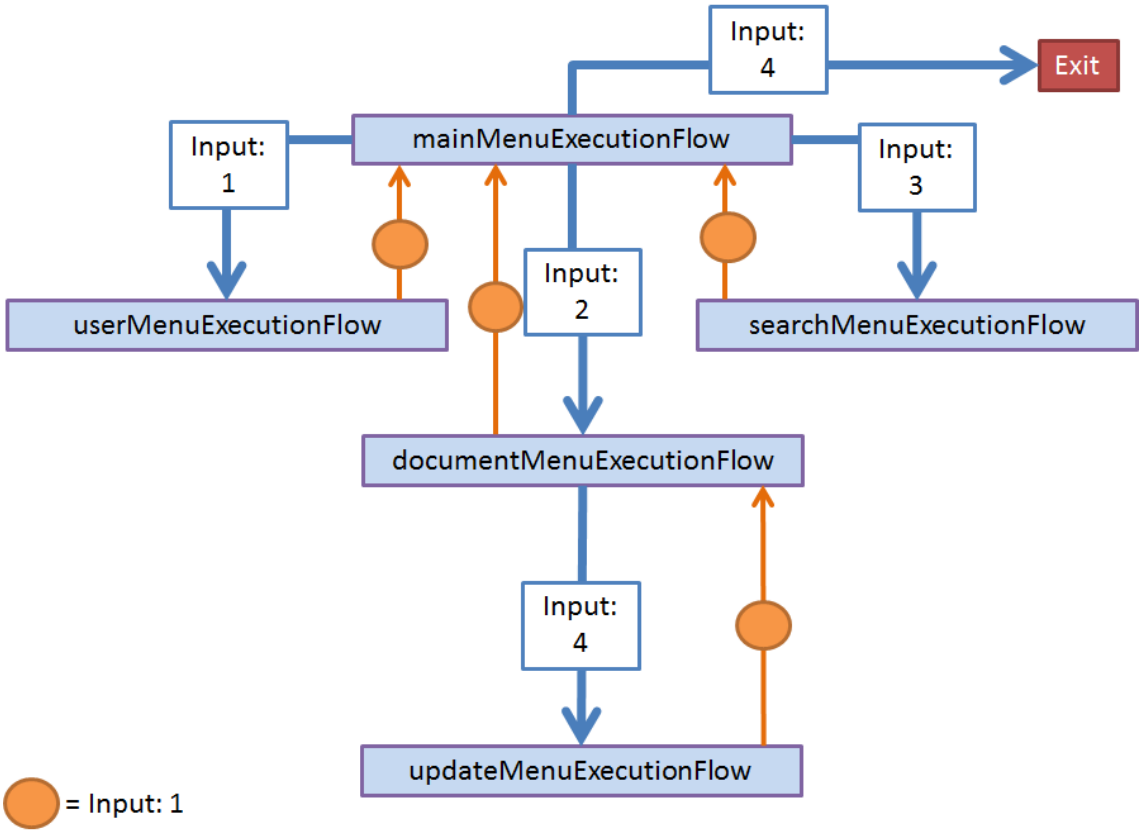
- ***updateDB()*** – este método encarrega-se de atualizar a *ArrayList* de documentos. Deve ser chamado após ações de inserção, remoção ou alteração efetuadas sobre uma base de dados para garantir que as pesquisas estão na versão correta.
- ***ArrayList<Integer> searchTitles(String toSearch)*** - este método devolve uma *ArrayList* com os ids dos documentos que satisfaçam a procura (*toSearch* está, portanto, incluído algures no seu título). Não é permitido pesquisar valores *null* (lança uma exceção).
- ***ArrayList<Integer> searchBodies(String toSearch)*** - semelhante à pesquisa por títulos, este método devolve uma *ArrayList* com os ids dos documentos que satisfaçam a procura. Também não é permitido pesquisar valores *null*.
- ***ArrayList<Integer> searchGeneric(String toSearch)*** - faz uma pesquisa semelhante à de searchTitles e searchBodies, mas verifica em ambos os campos (tendo em consideração que, para palavras que apareçam tanto no campo de título como no campo de corpo, não deve ser repetida a entrada. Também não é permitido pesquisar valores *null)*.
- ***public ArrayList<String> searchGeneric(int toSearch)*** - um *overload* do método anterior, neste caso o *toSearch* é um id do documento (um inteiro), e a *ArrayList* devolvida tem elementos do tipo *String*, contendo os títulos do documento. Dado que os ids são únicos, o resultado deste método deve ser sempre de tamanho total 1.
- ***ArrayList<String> searchByDate(Timestamp toSearch)*** - este método retorna os títulos de todos os documentos cuja data de criação é mais recente que a data fornecida como argumento. O argumento tem o formato *Timestamp*.

A nível de testes a base de dados de teste inclui, na maioria das funções, três documentos: dois semelhantes (com IDs de documento distintos), e um terceiro carregado de um ficheiro.

A abordagem genérica para testar cada método é a de recolher um documento que responda à pesquisa (isto é, procurar por uma *String* que apareça apenas em um dos documentos da base de dados), pesquisar uma *String* que apareça em diversos documentos e testar a reação à passagem de argumentos nulos. Destaca-se, contudo, o caso da pesquisa por data: dado que a criação dos documentos depende da data do sistema (*System.currentTimeMillis()*), a sua execução pode implicar que os *Timestamps* são todos idênticos, quando não é este o objetivo. Para evitar esta situação, o *Timestamp* usado num dos documentos foi alterado para ser diferente dos demais (foram-lhe somados 50ms).

Main

A classe de demonstração (Main.java), através de *inputs* do utilizador (*java.io.Scanner*) proporciona uma representação gráfica de uma possível implementação da base de dados. Ao receber um conjunto de *inputs* numéricos, o menu adequado é mostrado, como esquematizado abaixo:



Cada uma das operações pede ao utilizador o *input* adequado; por exemplo, uma atualização ao título de um documento pede qual o ID do documento e qual o novo título.

mainMenuExecutionFlow

Input: 1 – userMenuExecutionFlow
Input: 2 – documentMenuExecutionFlow
Input: 3 – searchMenuExecutionFlow
Input: 4 – Exit

updateMenuExecutionFlow

Input: 1 – Return to previous menu
Input: 2 – Update Document Title
Input: 3 – Update Document Body
Input: 4 – Update User ID

userMenuExecutionFlow

Input: 1 – Return to previous menu
Input: 2 – Add User
Input: 3 – Remove User
Input: 4 – Update User
Input: 5 – List Users

documentMenuExecutionFlow

Input: 1 – Return to previous menu
Input: 2 – Add Document
Input: 3 – Remove Document
Input: 4 – Update Document

searchMenuExecutionFlow

Input: 1 – Return to previous menu
Input: 2 – Search by title
Input: 3 – Search by body
Input: 4 – Search by title or body
Input: 5 – Search by creation date
Input: 6 – Search by document ID

Note-se que, na implementação, existem algumas limitações:

- Dado o uso de Scanner, aplicam-se algumas restrições sobre os *inputs*, nomeadamente quanto aos espaços vazios (títulos ou corpos de documentos passados como argumentos devem começar num caracter e terminar num espaço em branco (espaço normal, *tab*, nova linha));
- Não estão garantidas todas as exceções possíveis (por exemplo, na pesquisa por data, é possível pesquisar datas como 32 de Fevereiro de 2090);
- Dado que os *updates* não causam uma exceção se ocorrerem sobre utilizadores que não existem, é permitido fazer *updates* sobre utilizadores fantasma.

Conclusão

A utilização de *Test Driven Development* tem como principal vantagem ganhos a médio e longo prazo. O acréscimo de volume de código durante a fase inicial de programação permitiu prever erros comuns que, durante a utilização do programa, poderiam causar problemas. A implementação foi mantida tão genérica quanto possível.

Na implementação do sistema existe uma classe (DBAccess.java) que funciona como API para acesso à base de dados da qual o sistema depende.

A classe Utilizador.java permite representar um objeto Utilizador, no qual podem ser armazenadas as informações de um utilizador. Este objeto permite ainda a listagem de um ou vários utilizadores. Além disto, esta classe permite efetuar algumas operações sobre os utilizadores do sistema (adicionar novo, remover ou atualizar um utilizador existente).

A classe Documento.java pode ser utilizada como representação de um objeto Documento (lido da base de dados) ou como representação de um ficheiro presente no disco. Foram implementadas as funções básicas ao funcionamento da base de dados, e ainda acrescentada a funcionalidade de pesquisa por data sobre os documentos. De notar que são apenas considerados documentos com tamanho máximo de 1000 caracteres no body e 255 no título; esta simplificação é conveniente, mas deveria ser substituída por um formato mais adequado numa implementação real.

A classe DocSearch.java cria um índice de todos os documentos na base de dados, pronto a ser pesquisado, e disponibiliza métodos para efetuar pesquisas por título, corpo, data de criação ou id do documento.

A classe de demonstração permite, através de *inputs* do utilizador na linha de comandos, exemplificar um caso de uso da implementação que foi criada. É usado *java.io.Scanner* para obter o *input* do utilizador e às classes Documento, DBAccess, Utilizador e DocSearch para a implementação.

Referências

Sommerville, I. (2010). *Software Engineering*. Harlow, England: Addison-Wesley. ISBN: 978-0-13-703515-1