Programação Orientada pelos Objectos

2º Teste (Duração 2h)

LEI 2014/2015

Instruções:

- Antes de começar a resolver, **leia o enunciado do princípio até ao fim**.
  - As interfaces e classes do grupo de I e II têm mais métodos do que os que deverá implementar na resolução deste teste. Em cada grupo, tenha o cuidado de **ver com muita atenção quais os métodos que deve implementar**, para **não desperdiçar o seu tempo a implementar métodos que não lhe são pedidos**.
  - Disponibilizamos a descrição sumária de todos os métodos, incluindo os que não tem de implementar, para que os possa usar na sua resolução.
- **Pode** usar caneta ou lápis.
- Não é permitido consultar quaisquer elementos para além deste enunciado.

Introdução aos problemas para os grupos I, II e III:

Nestes 3 grupos vamos implementar parcialmente algumas das classes necessárias à construção de um serviço Web que permite criação e edição colaborativa de documentos (e.g. Google Docs). No primeiro grupo, faremos a implementação de uma classe que representa a colecção de documentos com uma lista. No segundo grupo, faremos a implementação de uma classe que representa a colecção de documentos com um ou vários mapas. Quer no primeiro, quer no segundo grupo, usamos as classes e interfaces especificadas na primeira parte enunciado. No terceiro grupo, realizamos alguns testes unitários e praticamos o uso de asserções.

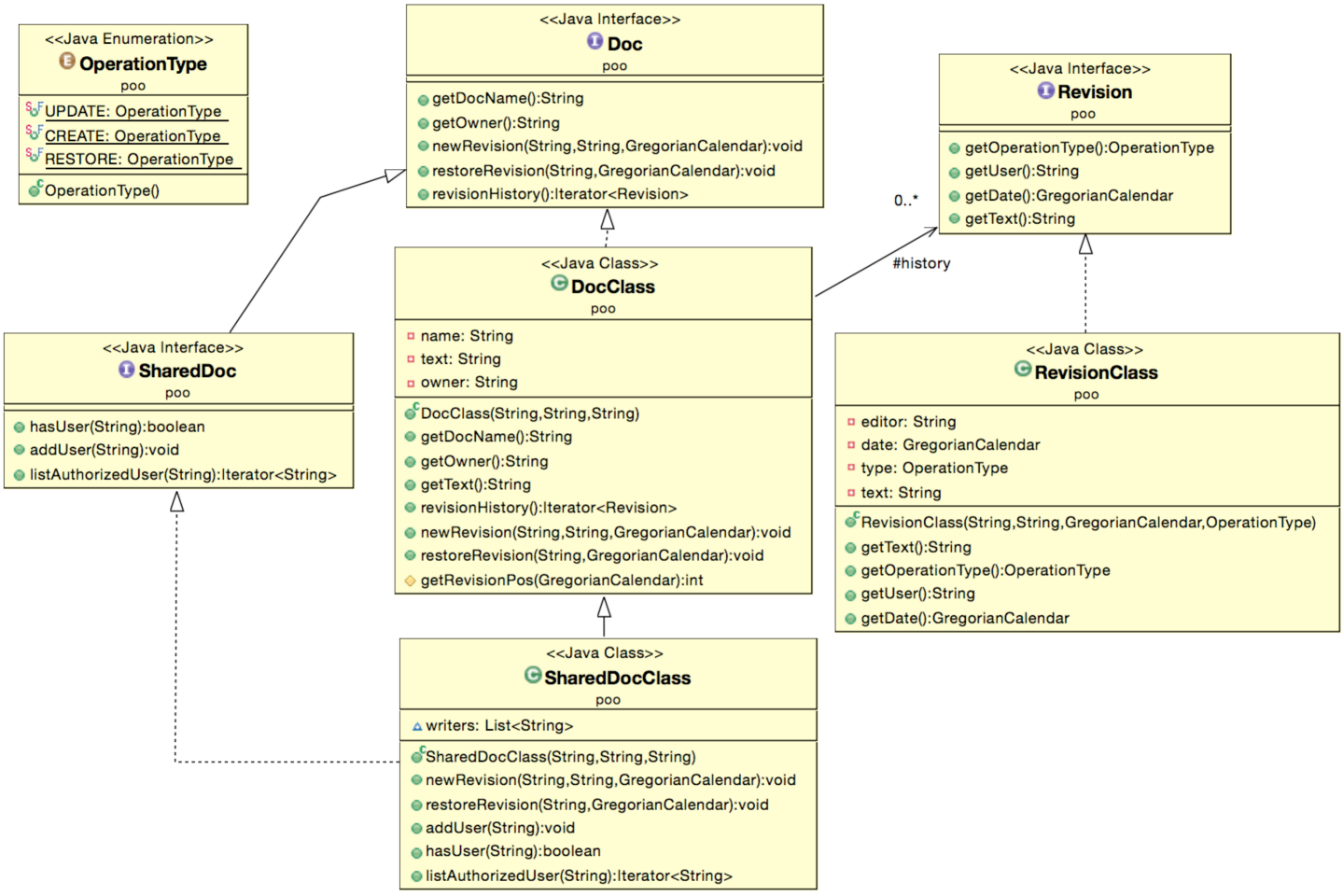

Figura 1: Documento de texto

A interface `Doc` representa um documento de texto não partilhado, ou seja, o documento só pode ser actualizado pelo utilizador que o criou. Note que não pode haver dois utilizadores diferentes com o mesmo nome. Os objectos desta classe mantêm uma lista (historial) de todas as operações efectuadas sobre o documento. Há três tipos de operações sobre documentos. A primeira operação é sempre a criação do documento (constante `CREATE`). As restantes operações são actualizações (constante `UPDATE`) ou a recuperação de uma versão antiga (constante `RESTORE`). Na interface `Doc`:

- `getDocName` devolve uma `String` com o nome do ficheiro (identificador único).
- `getOwner` devolve o nome do utilizador que criou o documento de texto.
- `getText` devolve a versão actual do texto.
- `revisionHistory` devolve a história de todas as actualizações desde a criação do documento (inclusive) por ordem de inserção.
- `newRevision` regista uma actualização do documento (operação do tipo `UPDATE`) para o utilizador `user`. Se esse utilizador não for o dono do documento não faz nada.
- `restoreRevision` recupera uma dada versão de acordo com a data recebida (operação do tipo `RESTORE`) e substitui o texto corrente por essa versão. Caso não exista uma versão com essa data ou o utilizador recebido não for o dono do documento, o método não tem efeito.

A classe `DocClass` implementa a interface `Doc`. O construtor desta classe recebe o nome e o texto associados ao documento, seguidos do username do dono, e cria um novo documento de texto com a

data actual. Os métodos implementados nesta classe obedecem à especificação já apresentada durante a descrição da interface.

A classe `DocClass` é especializada na classe `SharedDocClass`, que representa um documento partilhado para edição colaborativa. Os objectos desta classe mantêm uma lista de todos os utilizadores autorizados a editar o documento. Nesta classe as operações sobre o documento (`newRevision` e `restoreRevision`) têm um comportamento similar às operações da super-classe `DocClass`, sendo que não verificam apenas se o utilizador recebido é o dono documento mas se esse utilizador é um utilizador autorizado. A classe `SharedDoc` implementa o método `addUser` que adiciona um novo utilizador à colecção de utilizadores autorizados a editar o documento. Caso esse utilizador já exista não faz nada.
O método `hasUser` devolve `true` caso esse utilizador já exista e `false` caso contrário. Por último, o método `listAuthorizedUsers` devolve os utilizadores com acesso ao documento.

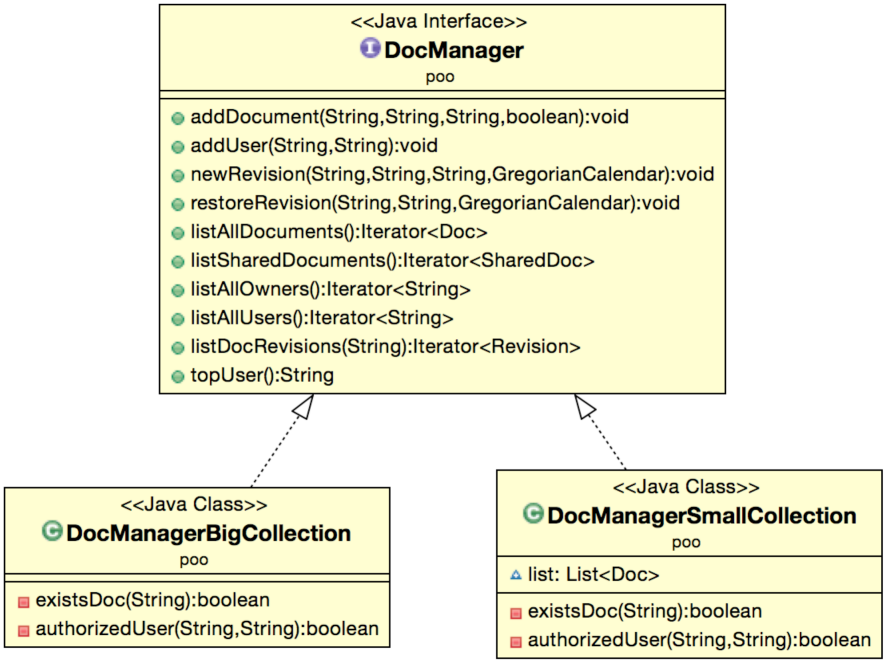

Figura 2: A interface DocManager

A Fig. 2 apresenta a interface e as classes necessárias à resolução dos grupos I e II. A interface `DocManager` representa a colecção de documentos de todos os utilizadores.

- O método `addDocument` adiciona um novo documento e recebe o nome `name`, o texto `text`, o utilizador do documento `owner` e um booleano `isShared` que indica se o documento é partilhado ou não. A operação lança a excepção `DocumentAlreadyExistsException` se já existir um documento com esse nome.
- O método `addUser` adiciona um novo utilizador à colecção de utilizadores autorizados a editar o documento, recebendo o nome do documento `name` e o utilizador a adicionar `user`. Esta operação lança a excepção `DocumentDoesNotExistException` no caso do documento não existir ou se não for um documento partilhado. Caso esse utilizador já exista não faz nada.
- O método `newRevision` regista uma operação de actualização sobre o documento com o nome `name` e texto `text` pelo utilizador `user` na data `date`. Esta operação lança as seguintes excepções: `DocumentDoesNotExistException` no caso do documento não existir; `NotAuthorizedUserException` se o utilizador não estiver autorizado a editar o documento (ou seja, se não for o dono ou um utilizador autorizado do documento).
- A operação `restoreRevision` regista uma operação de recuperação sobre o documento com o nome `name` pelo utilizador `user` na data `date`. Este método lança excepções nas mesmas situações que o método `newRevision`.
- O método `listAllDocuments` devolve um iterador para todos os documentos sem nenhuma ordem em particular. O método `listSharedDocuments` devolvendo um iterador para os documentos partilhados sem nenhuma ordem em particular.
- O método `listAllOwners` devolve um iterador para todos os utilizadores que são donos de pelo menos um documento. O método `listAllUsers` é similar, devolvendo um iterador para os utilizadores, sejam estes donos ou utilizadores autorizados a editar documentos. Estes iteradores percorrem os utilizadores por ordem alfabética.
- O método `listDocRevisions` devolve um iterador para a história de todas as revisões do documento recebido com parâmetro, não sendo necessário manter nenhuma ordem em particular. Esta operação lança a excepção `DocumentDoesNotExistException` no caso do documento não existir
- O método `topUser` devolve o utilizador com acesso a mais documentos (como dono ou utilizador autorizado). Em caso de empate é devolvido um dos top utilizadores. Caso não existam documentos o método devolver `null`.

Grupo I – Colecção pequena (`DocManagerSmallCollection`)

A classe `DocManagerSmallCollection` destina-se a situações com poucos documentos e utilizadores, pelo que vamos usar uma lista denominada `docs` para guardar a informação sobre estes artefactos. Nesta classe, implemente apenas:

a) O construtor de modo a que, ao ser criada a lista, esteja vazia. Apresente também a declaração da variável `docs` (pode colocar a declaração acima do construtor).

b) O modificador `void addDocument(String, String, boolean).`

c) O modificador `void newRevision(String, String, String, GregorianCalendar).`

d) O selector `Iterator<String> listAllOwners().`

e) O selector `Iterator<Revision> listDocRevisions(String).`

Grupo II – Colecção grande (`DocManagerBigCollection`)

Pretende-se implementar uma classe para a gestão de um número elevado de documentos e utilizadores. A implementação da classe deve ter em conta a eficiência das **pesquisas por nome de documento**, a eficiência de **todas as listagens** e a eficiência do **método `topUser`**. Implemente as seguintes operações da classe `DocManagerBigCollection`:

a) O construtor de modo a que, ao ser criado não existam documentos. Apresente também a declaração das variáveis que achar necessárias (pode colocar a declaração acima do construtor).

b) O modificador `void addDocument(String, String, boolean).`

c) O modificador `void addUser(String, String).`

d) O modificador `void restoreRevision(String, String, GregorianCalendar).`

e) O selector `Iterator<String> listAllUsers().`

f) O selector `String topUser(String).`

Grupo III – Testes unitários e asserções

a) Implemente uma operação de teste à operação `listAuthorizedUsers` da classe `SharedDocClass`, usando o JUnit. O seu teste deve construir um objecto `SharedDocClass`. De seguida deve inserir 3 utilizadores diferentes e por último um utilizador que já exista. Invoque o método `listAuthorizedUsers` e verifique que o iterador devolvido percorre todos os elementos pela ordem esperada.

b) Considere a seguinte classe:

```
public class AssertionsClass {

 public static void main(String[] args) {
  for(int i = 2; i < 4; i++)
        for(int j = 2; j < 4; j++)
              assert i != j : i;

  System.out.println("Fim.");
 }
}
```

Assumindo que o mecanismo de asserções está activado, qual é o resultado obtido na execução do programa acima? Justifique a sua resposta.

Algumas das interfaces abaixo reproduzidas poderão ser úteis na resolução deste teste: