1 O projecto.

Com a divulgação massiva de informação através de formato digital, como acontece por exemplo com livros, músicas, vídeos e fotografias, as bibliotecas tradicionais, tal como as conhecemos, tendem a sofrer uma alteração profunda tanto na sua organização, como na relação com os seus utentes. Edifícios imponentes a abarrotar de livros; peregrinações à biblioteca para requisitar ou entregar um livro, ou para saber se determinado título já está disponível; estudar por livros estragados, riscados, ou sem a página das soluções que tanta falta nos fazia, são situações que estão a chegar ao fim.

Num futuro próximo, esperemos, será possível requisitar um título (livro, vídeo, etc.) interagindo com uma biblioteca virtual, que disponibiliza, de forma imediata, a informação que pretendemos. Claro que muitas regras continuaram como antes. Por exemplo, continua a ser necessário honrar os direitos de autor e não disponibilizar mais exemplares do que os que são detidos pela biblioteca; continua a existir um número máximo de exemplares que podem ser requisitados por cada utente; continua a ser necessário estabelecer um prazo para deter a requisição do livro; tal como continua a ser necessário tomar notas, marcar páginas ou sublinhar textos.

O trabalho que vos propomos é o de desenvolver um sistema para este novo conceito de biblioteca que pretendemos que se torne numa realidade. Imaginamos um sistema *cliente-servidor*, em que os utentes têm à sua disposição um programa *cliente* que lhes permite consultar os títulos disponíveis para aluguer, proceder à sua requisição, ler ou assistir à reprodução, anotar, marcar e devolver o título, entre outras funcionalidades. Além disso é fundamental consultar as notas e outra informação colocada pelo utente, mesmo depois da devolução do título. A parte *servidor* será responsável pelo catálogo dos itens alugáveis, pela gestão das contas dos utentes, pela gestão de requisições, só para enumerar algumas das funcionalidades que antevemos para a componente servidor.

Como o sistema é extenso, propomos, nesta primeira fase, efetuar o desenvolvimento da versão *cliente*. Numa segunda fase vamos proceder à análise e desenho da componente servidor (mas isto fica para mais tarde). Pretende-se desenvolver um sistema, com nome de código *LEI - Lending E-media Initiative*, que interaja com uma biblioteca virtual. Esta primeira versão (a 1.0) irá focar essencialmente nas funcionalidades básicas de requisição, organização, visualização e devolução de títulos digitais. Embora existam diversos formatos para armazenar documentos, desde `texto`, `html`, `kindle`, `chm`, etc., os formatos `texto` e `PDF` são sem dúvida o mais comuns.

Em versões ulteriores vislumbram-se outras potencialidades para este *software*, como por exemplo (1) a partilha de documentos para efetuar estudo em grupo, ou (2) a disponibilização das funcionalidades atrás descritas em equipamentos móveis (por exemplo, *tablets*).

A primeira fase do projecto consiste na construção de um organizador e visualizador de títulos em formato `texto` e `PDF` organizada de acordo com o princípio *model-view-controller*. As camadas *view* e *controller* já são fornecidas, cabendo-vos a tarefa de desenvolver a camada *model* e de estabelecer a ligação do *controller* com o *model*, que será feita via *delegação*.

A aplicação deverá oferecer as seguintes funcionalidades:

- *Acesso aos títulos disponíveis para aluguer.* Esta será a parte onde a aplicação cliente se liga remotamente a um servidor para obter os títulos passíveis de serem alugados. Como ainda não dispomos da versão servidor, a aplicação deve permitir que sejam adicionados títulos localmente (para efeitos de teste). Para tal a interface de utilizador (que já está desenvolvida) disponibiliza uma pasta com o nome *Library* para onde se podem "arrastar" documentos do vosso sistema de ficheiros. Para cada ficheiro terá de ser fornecido meta informação, da qual se destaca a localização do documento em disco (determinada de forma automática), o *Internet media type* do documento que identifica o seu conteúdo e o número de licensas disponíveis. Não deve ser possível adicionar documentos com a mesma localização.

- *Alugar títulos.* Permite selecionar e alugar um título da biblioteca. A interface de utilizador permite que se "arraste" o título pretendido da pasta *Library* para a pasta *My Rentals*. Ao alugar um título, o número de licensas disponíveis para alugar decresce, e é dada a possibilidade ao utente de aceder ao seu conteúdo. Durante esta fase é possível adicionar marcadores ao título para assinalar páginas (caso o título o permita) e dar ao leitor a hipótese de rapidamente aceder a uma página marcada. Outra facilidade do sistema consiste em registar anotações gerais ou associadas a páginas (caso o título o permita). Desta forma o leitor pode escrever comentários que mais tarde pode consultar, mesmo após o título ser devolvido.

- *Organização de documentos em estantes* Com o objetivo de organizar os títulos alugados pretende-se arruma-los em *estantes*. Uma estante permite agrupar títulos manualmente de acordo com um critério que convenha ao leitor. Uma *estante* é identificada por um nome e contém uma lista (não repetida) de documentos. Sobre uma estante podem ser adicionados ou removidos títulos. Ao adicionar um título a uma estante, este não deve ocupar mais espaço em disco. Assim, permite-nos ter múltiplas organizações de títulos, sem com isso ocupar espaço extra.

O sistema deve ainda permitir ao leitor criar estantes especiais, cujo conteúdo é automaticamente determinado a partir de uma condição especificada pelo leitor. Como exemplo pré-definido temos a estante das *últimas requisições* que contém (de forma automática) os documentos que foram requisitados nos últimos três dias. O utente pode criar as suas próprias estantes e especificar um critério de seleção que permite combinar critérios básicos (como a comparação com uma palavra-chave ou a comparação com um autor) de forma disjuntiva ou conjuntiva. Por exemplo, o sistema deve aceitar criar uma pasta com o seguinte critério de seleção:

author = Larman **and**
(palavra-chave = análise **or** palavra-chave = desenho)

A interface de utilizador para criar estantes especiais, nesta primeira iteração, não está ainda disponível. No entanto, devem ser programadas (e testadas) as classes que suportem a construção destes critérios.

- *Devolução de títulos* Deve ser possível devolver um título. Após a devolução não deve ser possível aceder ao conteúdo do título, mas deve ser dada a possibilidade de aceder às anotações e marcadores feitas pelo utente.

- *Visualização de títulos* A aplicação deve permitir a visualização em formato `texto` ou `PDF`, mas o sistema deve estar preparado para apresentar documentos em outros formatos, mediante a adição de novos visualizadores (*plug-ins*). Já se encontram desenvolvidos os visualizadores para imagens em formato *JPG*, *PNG* e *GIF* e documentos em formato *PDF*. Cabe-vos construir um visualizador para ficheiros em formato de texto. Como bonús a interface de utilizador permite a visualização de documentos em ecrã completo (fullscreen) e efetuar *slideshow*. Um *slideshow* consiste na apresentação de um título (se suportado) em que as páginas avançam ao fim de um tempo pre-definido pelo leitor.

2 Que devemos fazer?

O que devem fazer é implementar as classes da aplicação cliente de acordo com o modelo de implementação que é descrito a seguir. A aplicação permite a interação com o utilizador para efetuar o aluguer de média. Está organizada em torno do padrão *Model-view-controller* e faz uso extensivo do princípio *information hiding* e dos padrões de desenho GoF que apresentámos ao longo do semestre. Assim, a aplicação encontra-se organizada em cinco pacotes principais, a saber: `view`, que define uma interface de utilizador usando o pacote *javax.swing*. (Este pacote não deve ser alterado.);

controller , que intercepta os eventos gerados na interface de utilizador e os despacha para a aplicação que vão desenvolver. (Este pacote também não deve ser alterado.); model, que irá conter as classes a desenvolver, que implementam a aplicação; lei , que contem a classe de arranque do sistema (fornecida) e mais algumas classes de carácter global da aplicação (por exemplo, a classe com as propriedades da aplicação e a implementação das delegações que estabelecem a ligação entre a interface e a aplicação); services , que oferece serviços de visualização à aplicação.

2.1 Modelo de Implementação

Como anexo a este documento é fornecido o código fonte de um conjunto de classes Java que permitem a interação com o utilizador. Estas classes encontram-se nos pacotes view, controller e services e não têm necessidade de lhes mexer, a menos que queiram melhorar a interface gráfica fornecida e tirar *bugs* ;)

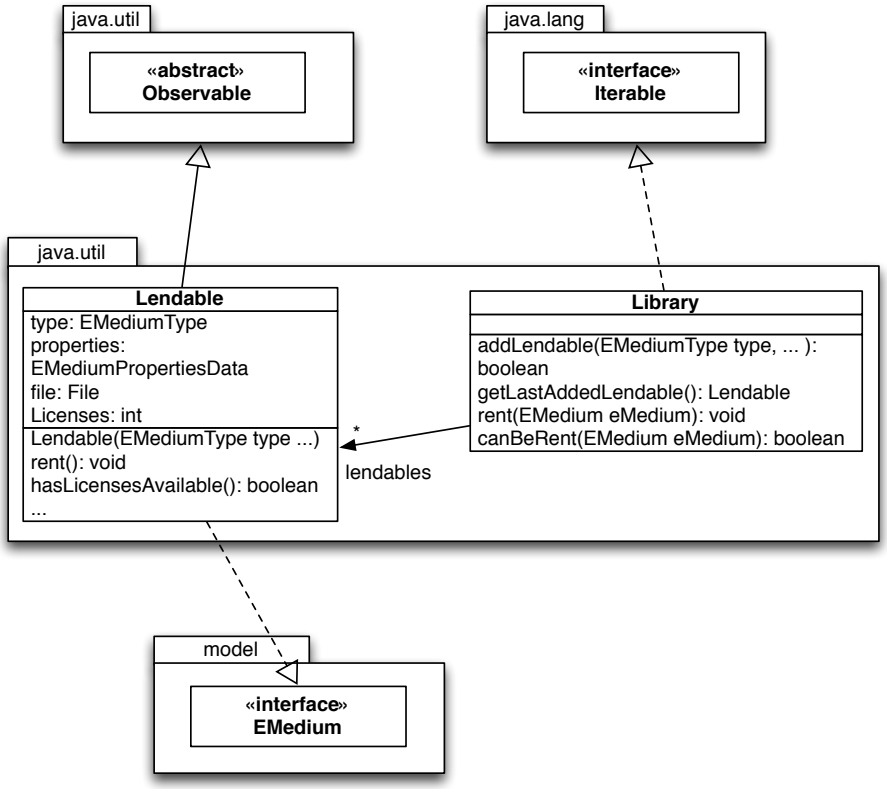
Vamos detalhar o pacote model que irá conter as classes que têm de desenvolver. O pacote está dividido em quatro sub-pacotes.

model O pacote model contem as classes e interfaces que são disponibilizados aos restantes pacotes. Conta com:

- uma interface EMedium que representa um item a alugar (da biblioteca) ou já alugado (fornecida);
- um enumerado EMediumAttribute que descreve os atributes que caracterizam um média (fornecido);
- uma classe EMediumPropertiesData que representa os atributos de um média. Esta classe deve ser clonável (parcialmente fornecida);
- um enumerado EMediumType que identifica os tipos de média que a aplicação trata (parcialmente fornecida);
- uma interface ILibraryFacade que especifica as operações que se podem fazer sobre a biblioteca de média (fornecida);
- uma interface IShelvesFacade que define as operações que se podem fazer sobre pastas e sobre os alugueres de média que guardam (fornecida);
- a classe LEI é a classe principal da aplicação. Deve ser esta classe que faz o arranque e estabelece o estado inicial da aplicação (fornecida (muito) parcialmente);

As interfaces e classes acima são (parcialmente) fornecidas. Devem consultar o *javadoc* destas interfaces e classes para obterem informação detalhada sobre o comportamento pretendido para cada método.

model.lendables O pacote model.lendables contém a interface com o sistema servidor onde estão armazenados os itens a alugar, a informação dos utilizadores, etc. Não é objeto deste trabalho desenvolver esta comunicação com o servidor, pelo que este módulo abstrair esta ligação. O diagrama de classes do módulo é o seguinte.



A classe Library deve oferecer os seguintes métodos (além dos obrigatórios devido às interfaces que implementam):

- public boolean addLendable (EMediumType type, EMediumPropertiesData properties), que adiciona um novo média à biblioteca para ser alugado. Esta operação não será possível realizar pelo utente no sistema final, mas nesta fase é a forma de evitar comunicações com o servidor central. A interface gráfica permite adicionar média à biblioteca e recolhe as propriedades associadas ao média. A biblioteca não permite itens repetidos, ou seja, itens que se refiram ao mesmo ficheiro em disco e que tenha o mesmo tipo;
- public Lendable getLastAddedLendable(), que devolve o último item adicionado à biblioteca;

- public **void** rent(EMedium eMedium), que aluga um item da biblioteca. Este método tem como pré-condição que só deve ser chamado sobre itens que podem ser alugados;

- public boolean canBeRent(EMedium eMedium), que permite averiguar se um item pode ser alugado. Não devem ser possível alugar itens que não tenham licensas disponíveis.

A classe Lendable representa um item que pode ser alugado. Esta classe deve manter informação sobre o tipo do média (EMediumType), as propriedades do média (EMediumPropertiesData), a informação do ficheiro em disco (File) e o número de licenças disponíveis (**int**). Deve oferecer os seguintes métodos (além dos definidos pelas interfaces implementadas):

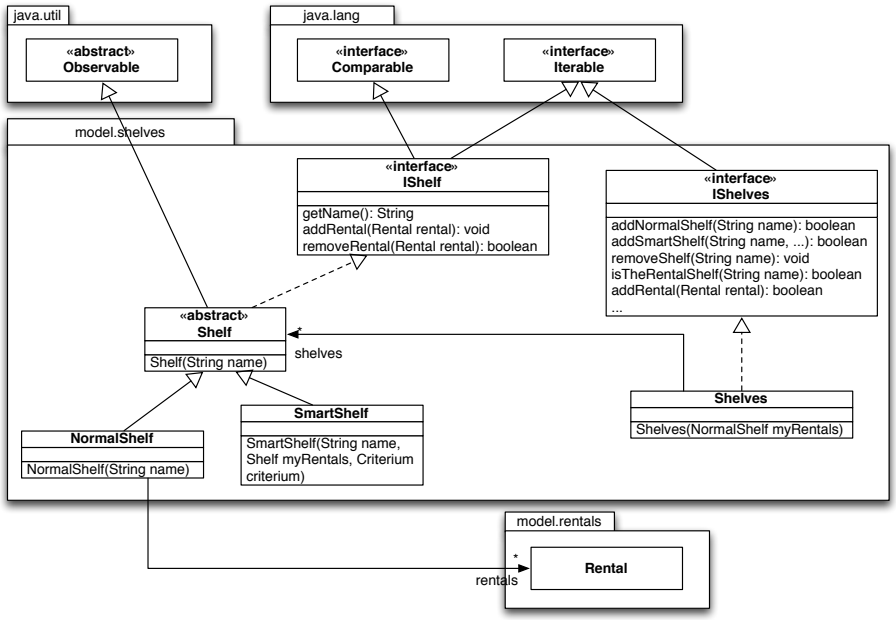
- public Lendable(EMediumType type, EMediumPropertiesData properties), que constrói um item a ser alugado dado o seu tipo e as propriedades que o caracterizam;

- public **void** rent(), que aluga um destes itens. Deve ser atualizado o número de licenças de forma conveniente. O método só deve ser chamado se o item tiver licenças disponíveis;

- public boolean hasLicensesAvailable (), que devolve o número de licenças disponíveis para este item.

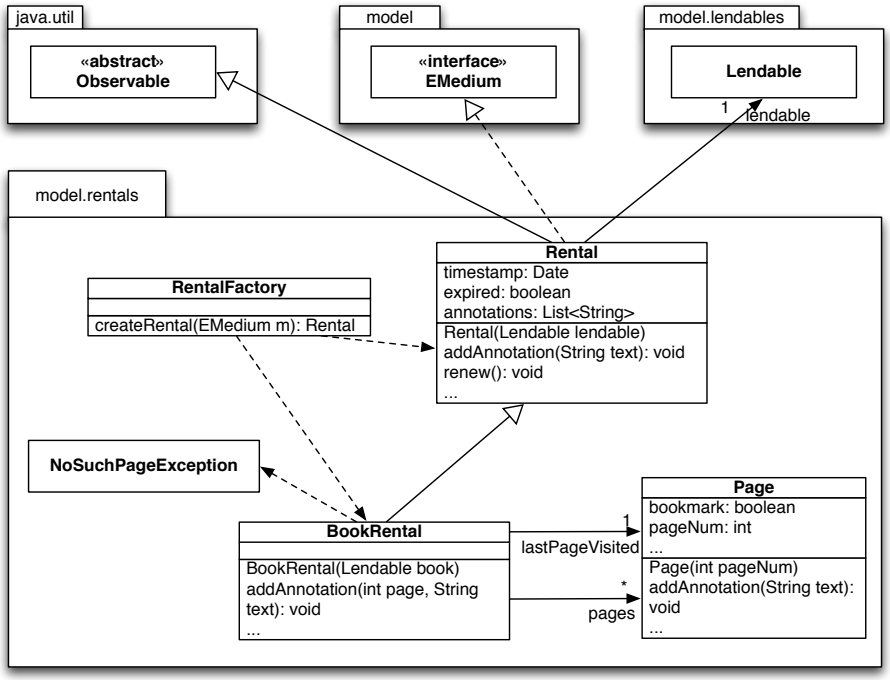
A fim de evitar a duplicação de itens, a classe Lendable deve implementar uma noção de igualdade considere iguais média com o mesmo tipo e que se referem ao mesmo ficheiro em disco. Para dispor de forma ordenada os itens, estes devem implementar uma noção de ordenação por título (estes objetos devem ser comparáveis por título).

model.shelves



A aplicação vem equipada com um sistema de pastas (*shelves*) para organizar os itens alugados. O sistema vem automaticamente carregado com duas pastas: a Library, que contem os itens (alojados num servidor) que podem ser alugados; e a pasta *MyRentals*, que mantém os itens alguma vez alugados pelo utente. Adicionalmente, o utente pode criar (e remover) pastas para o ajudarem a organizar os itens alugados. Estas pastas podem ser mantidas de forma manual (*normal shelf*) ou de forma automática (*smart shelf*). As primeiras permitem que o utente adicione e remova itens alugados. As segundas (as automáticas) filtram automaticamente o conteúdo da pasta *My Rentals* utilizando um critério de seleção. A aplicação fornecida já contém as interfaces IShelf e IShelves que representam, respetivamente, uma pasta e um catálogo de pastas. Devem implementar as classes Shelf, NormalShelf, SmartShelf e Shelves, de acordo com o seguinte diagrama de classes. O *javadoc* das interfaces contem informação do significado dos métodos a implementar. As decisões específicas sobre atributos das classes e estruturas de dados a utilizar (do pacote java.util) ficam a cargo da equipa de programação (vocês ;)).

model.rentals O pacote model.rentals contem as classes que registam o aluguer de média. De seguida apresentamos o modelos de classes deste pacote e as relações com outros pacotes.



Em detalhe, a classe Rental representa um aluguer e mantém informação sobre a que Lendable se refere o aluguer, o instante em que foi feito, se ainda está em vigor ou se já expirou e uma lista de anotações que o utilizador pode querer fazer associado ao média. Esta classe disponibiliza os métodos seguintes (além dos métodos que implementa):

- public Rental(Lendable lendable), que constrói um aluguer a partir do item que se pretende alugar;
- public Date getRentalTimestamp(), que devolve o instante em que foi alugado este item;
- public **void** addAnnotation(String text), que adiciona uma anotação (global) ao média;
- public **void** removeAnnotation(**int** n), que remove a n-ésima anotação do média. Como pré-condição sabe-se que n corresponde a uma anotação existente;
- public Iterable <String> getAnnotations() devolve as anotações associadas ao média;
- public **void** renew(), que faz a renovação de um aluguer;
- public boolean isExpired (), que indica se este aluguer ainda está em vigor. Um aluguer expirado não pode ser visualizado, mas devem ser preservadas as anotações feitas ao item.

A classe BookRental é uma especialização da classe Rental e destina-se a representar livros. A principal diferença é que um livro tem várias páginas e pode associar-se anotações a cada página ou adicionar marcadores, coisa que não é possível fazer para um média em geral. Nestes, como foi referido anteriormente, as anotações são associadas ao média em geral (e não tem o conceito de marcador de página). Adicionalmente, o sistema deve registar a última página acedida (lastPageVisited) para ao revisitar o livro, o sistema continue a visualização a partir desta página. Os métodos oferecidos por esta classe são:

- public BookRental(Lendable book), que cria o aluguer de um livro a partir do item a alugar;
- public **void** addAnnotation(**int** pageNum, String text), que permite adicionar uma anotação a uma página;
- public **void** removeAnnotation(**int** pageNum, **int** n), que remove a n-ésima anotação da página pageNum;
- public Iterable <String> getAnnotations(**int** pageNum), que obtém as anotações de uma página;
- public String getAnnotationText(**int** pageNum, **int** n), que obtém a n-ésima anotação de uma página;
- public boolean hasAnnotations(**int** n), que indica se a n-ésima página tem anotações;
- public boolean isBookmarked(**int** n), que indica se a n-ésima página está marcada;
- public List <Integer> getBookmarks (), que obtém a lista de páginas marcadas;
- public **void** toggleBookmark(**int** n), que marca a n-ésima página caso não esteja marcada, e desmarca-a caso contrário;
- public **int** getLastPageVisited (), que devolve a última página consultada pelo utilizador;
- public **void** setLastPageVisited (**int** lastPageVisited), que define a última página visitada.

Associado ao livro há a informação sobre as páginas: o número da página, se está marcada e as anotações da página. Esta informação é registada por objetos da classe Page. Os métodos da classe são:

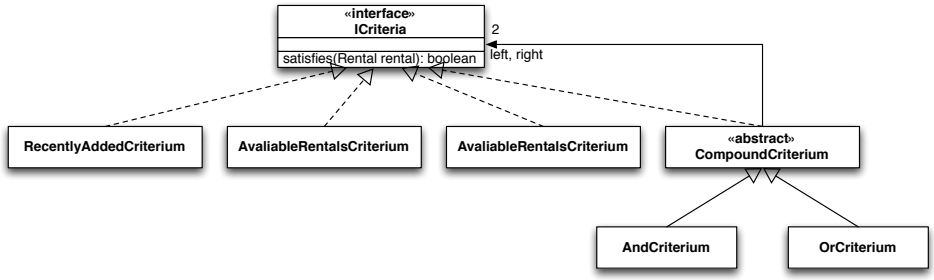
- public Page (**int** pageNum), que cria uma página dado o seu número;

- public **void** addAnnotation (String text), que adiciona uma anotação à página;
- public Iterable <String> getAnnotations (), que obtém as anotações da página;
- public String getAnnotationText (**int** n), que obtém a n-ésima anotação da página;
- public boolean isBookmark(), que indica se a página está marcada;
- public **void** toggleBookmark(), que muda o marcador da página: se não está marcada, marca-a, caso contrário remove o marcador;
- public **int** getPageNum(), que informa o número da página;
- public **void** removeAnnotation(**int** n), que remove a n-ésima anotação da página;
- public boolean hasAnnotations(), que indica se a página tem anotações;
- public boolean isBookmarked(), que indica se a página tem marcadores.

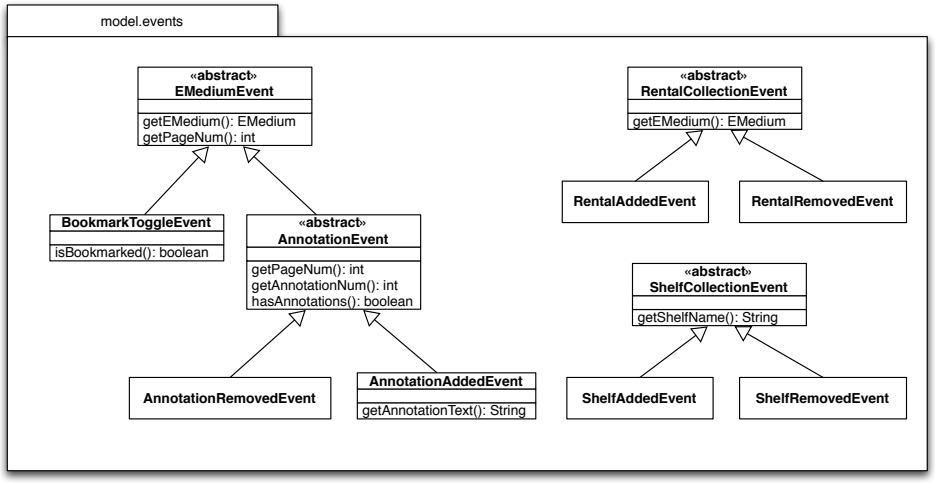
Quando é pedida informação sobre uma página que não existe no livro, é lançada a exceção NoSuchPageException.

Por último, a criação de objetos Rental ou BookRental depende do tipo de item a alugar. A classe RentalFactory é uma fábrica que disponibiliza o método public Rental createRental (EMedium rental) para criar um Rental a partir de um EMedium. As regras para criar um aluguer são as seguintes: se o EMedium é já um Rental, o objeto é devolvido; Senão, se o tipo EMediumType.DOCUMENT deve ser criado um BookRental, caso contrário um Rental geral.

model.shelves.criteria O pacote model.shelves.criteria contem os critérios para a definição de *smart shelves*. Deve ter uma interface ICriterion que descreve um critério, com um único método boolean satisfies (Rental rental), que dado um Rental verifica se este satisfaz a condição de ser seleccionado. Devem implementar as classes de acordo com o seguinte diagrama.



model.events O pacote model.events deve conter as classes de acordo com o diagrama seguinte. Os objetos destas classes servirão para sinalizar a interface de utilizador dos eventos ocorridos ao nível do modelo. Devem criar os construtores de acordo com as necessidades.



As únicas dependências entre o pacote view e o pacote *model* são já fornecidas no pacote model. Devem completar o seu desenvolvimento.

lei Para a interação com o pacote view devem completar, no pacote LEI, as classes LEIBookshelfUIDelegate, LEIEMediaUIDelegate e ainda a classe LEIEMediumMetadataUIDelegate que contém essencialmente os métodos correspondentes às operações que foram descritas e que são iniciadas no ambiente gráfico. Este ambiente invoca automaticamente cada um destes métodos. Estas classes contém também métodos protegidos que permitem a atualização do ambiente gráfico. Por exemplo, ver na super classe da classe LEIBookshelfUIDelegate, protected **void** addShelfTreeNode (String shelfName) é o método que adiciona uma nova pasta à árvore de pastas apresentadas pela interface de utilizador.

services.viewer Por último, o pacote services . viewer (fornecido) carrega dinamicamente visualizadores que estão no pacote *default*. Juntamente com a aplicação são fornecidos dos visualizadores. Devem desenvolver um visualizador SwingTextViewer para mostrar ficheiros do tipo text / plain .

O código é fornecido como um projeto *Eclipse* para facilitar a configuração dos parâmetros de compilação, assim como a dependência de pacotes externos. Devem simplesmente importar o projeto para o *Eclipse*. Se utilizarem outro *IDE* devem retirar as fontes das classes fornecidas e agir de acordo com os requisitos do *IDE* que utilizem. Notem que a leitura do ficheiros PDF baseia-se na biblioteca externa `PDFRenderer.jar` que é fornecida com o projeto e que tem de fazer parte do seu *classpath*.

2.2 Utilização de padrões de desenho

Este projeto apresenta imensas oportunidades para a utilização dos padrões de desenho GoF. De seguida dou alguns exemplos onde podem encontrar ou utilizar padrões GoF:

- *Composite*. A definição de critérios usa este padrão. Notem que um critério And ou Or junta dois critérios quaisquer e é tratado ele próprio como um critério.

- *Command*. Pode ser visto na implementação do ambiente gráfico na programação do comportamento dos elementos gráficos, por exemplo, para definir o comportamento de um item de um menu de contexto, passa-se um objeto (o comando) a ser executado assim que a opção for selecionada.

- *Observer*. Não só para atualização do ambiente gráfico, mas também para as pastas se notificarem umas às outras. Os diversos eventos são descritos no pacote model.events (ver acima).

- *Façade*. As classes do domínio são protegidas por fachadas: uma para operações sobre pastas outra para operações sobre a biblioteca de média a alugar.

- *Adapter*. A aplicação permite diversos visualizadores e, no futuro, poderá ser possível incluir outros formatos. Desenhem-na de forma a proteger esta variação futura previsível.

- *Strategy*. Os critérios usam o padrão estratégia para seleccionar os itens que devem ser mostrados em cada *smart shelf*.

- *Prototype*. Pretende-se que a aplicação carregue dinamicamente as classes que representam os visualizadores (como as referidas atrás). Embora a linguagem Java permita reflexão, na verdade, é este o padrão que utiliza quando se pretende carregar e criar classes/objetos dinamicamente.

- *Abstract Factory e Factory Method*. Podem utilizar a fabrica para criar os visualizadores, por exemplo.

- *Singleton*. As fábricas, por exemplo.

- *Template method*. Usado por exemplo, para fazer a delegação das ações recolhidas na interface de utilizador na fachada da aplicação.

- *Iterator*. A linguagem Java tem suporte para este padrão, através da classe *java.util.Iterator* e do ciclo *foreach*.

Além dos padrões GoF devem ver o princípio *Model-View-Controller* adequado ao desenvolvimento de aplicações que ofereçam uma interface de utilizador, a *delegação* e, em geral, o *information hiding* que se manifesta de muitas formas tanto no código fornecido como (deve manifestar-se) no código que vão implementar.

3 Como e o que entregamos?

Fazem um ficheiro com do vosso projeto Eclipse (ou de todos os ficheiros .java, caso não usem Eclipse) e entregam no sítio da disciplina (utilizando o mocho).