Programa 3 (valendo nota) – *Biblioteca, em partes*

Uma Biblioteca precisa ter controle sobre o cadastro de usuários e acervo de livros para empréstimo. Desenvolva uma aplicação que permita criar um cadastro de usuários e de livros, acessar o cadastro quando necessário e registrar empréstimo e devoluções quando solicitado. O diagrama de Figura 1 apresenta o diagrama de classes geral do sistema.

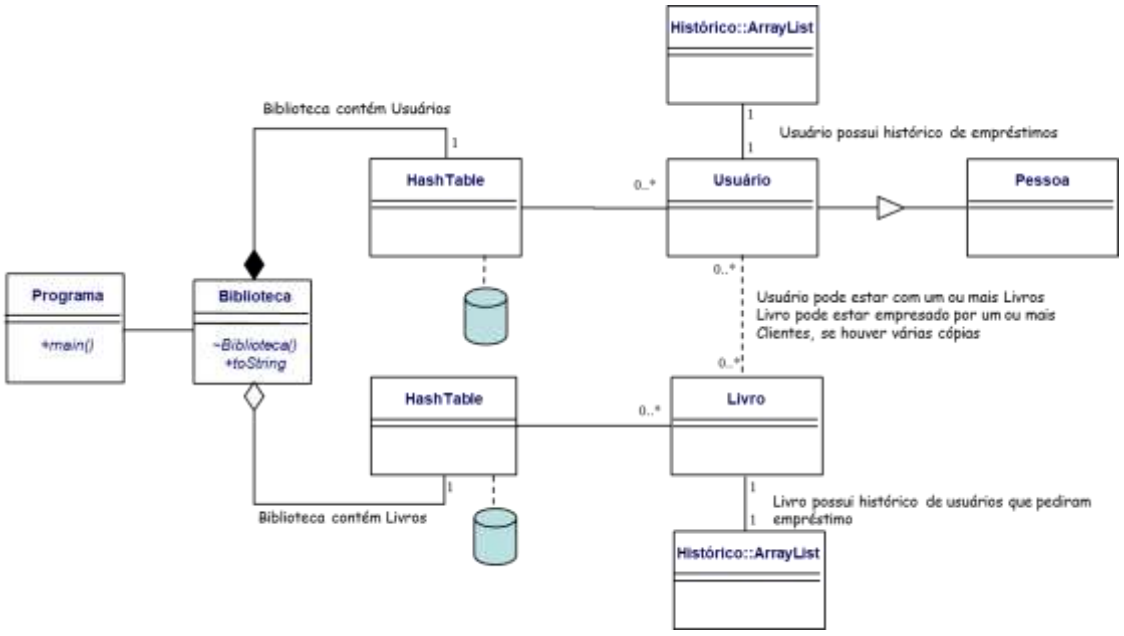

Figura 1: Modelo da Biblioteca

Requisitos:

Todas as classes, exceto a classe principal, devem pertencer a um pacote chamado lp2gXX.biblioteca (onde XX é o número do seu usuário na máquina virtual).

I) Vamos reusar a classe *Pessoa* do P1n

II) Crie uma classe *Usuario*, que estende *Pessoa*, e possua um campo *String endereço*. O construtor da classe *Usuário* deve inicializar todos os seus campos.

A classe *Usuario* também deve ter um campo *hist* da classe *ArrayList*. Este campo deve armazenar objetos da classe *Emprest.*

A classe *Usuário* deve ter um método *addLivroHist()*, que recebe a data de locação e o código do *Livro* emprestado (um *int*), cria um objeto *Emprest* com estas informações e adiciona o mesmo ao histórico (campo *hist*).

Opcionalmente acrescente um campo para contar os livros que estão emprestados para o usuário. Isso ajudar a verificar rapidamente se existem livros que devem ser devolvidos.

III) Crie a classe *Emprest*. A classe *Emprest* deve conter a data do empréstimo, data de devolução, ambos da classe *GregorianCalendar*, e o código do *Livro* solicitado para

empréstimo. Observe que, a data de devolução deveria registrar o dia em que o livro foi devolvido. Reforçando os campos com data devem ser um objeto da classe *GregorianCalender* (como foi feito na idade da classe *Pessoa*).

Observe também que você deve manter a informação de que, até que o livro seja devolvido, a data de devolução não está preenchida. Não coloque uma estimativa de data de devolução, pois isso vai confundir. Ao contrário, se a data de devolução não existe, na hora de exibir, pode colocar "Pendente" (isso é fácil verificar e acertar no próprio *toString*) ... e até acrescentar a data limite de entrega (dependendo dos parâmetros estabelecidos na classe principal).

IV) Crie uma classe *Livro*, com campos para:

- Código do Livro (*int, de 1 a 999*)
- Título do Livro (*String*)
- Categoria (que pode ser Aventura, Ficção, Romance, etc.) – pode ser String ou um enumerate, não vamos limitar.
- Quantidade de cópias deste título disponíveis no acervo (*int*)
- Emprestados, número de cópias deste título emprestadas (*int*)

Um **construtor** que inicializa todos os seus campos

Também deve ter os seguintes métodos:

- *empresta*, que não recebe parâmetros e acerta o campo *Emprestados*. Caso todas as cópias estejam emprestadas, deve levantar a exceção confirmada *CopiaNaoDisponivelEx*;
- *devolve*, que não recebe parâmetros e acerta o campo *Emprestados*. Caso nenhuma cópia tenha sido emprestada, deve levantar a exceção *NenhumaCopiaEmprestadaEx*.

Observação: crie as exceções do enunciado como confirmadas (*cheked*)

A classe *Livro* também deve ter um campo *hist* da classe *ArrayList*. Este campo deve armazenar objetos da classe *EmprestPara.*

A classe *Livro* deve ter um método *addUsuarioHist()*, que recebe a data de locação, data de devolução e o CPF do *Usuário* que pediu o livro emprestado, cria um objeto *EmprestadoPara* com estas informações e adiciona o mesmo no *Histórico*.

Observe que no caso de um empréstimo, o programa principal vai chamar empresta() e em seguida, se não houve exceção, *addUsuarioHist().* A mesma observação em relação à data de devolução vale aqui. Não faz muito sentido ter uma data se o livro ainda não foi devolvido. Mas é possível deixar uma "marca" (por exemplo, com null, para registra isso (que ainda não foi devolvido).

V) Implemente a classe *EmprestPara.* A classe *EmprestPara* deve conter a data do empréstimo, data de devolução e o CPF *do usuário* que pegou o livro emprestado. As datas devem usar a classe *GregorianCalendar*.

VI) Implemente a classe *Biblioteca* com:

- um campo de instância para o cadastro de usuários contendo um objeto da classe java.util.*Hashtable*
- um campo de instância para o cadastro de livros contendo um objeto da classe java.util.*Hashtable.*

A classe *Biblioteca* deve possuir dois construtores:

- um que inicialize os campos, zerando os "bancos de dados", ou seja criando uma nova instância de HashTable;
- um construtor que carregue o cadastro de usuários e o cadastro de livros salvos em dois arquivos distintos. O nome dos arquivos deve ser fornecido pelo usuário. Não precisa deixar fixo. O seu código deve administrar a forma de associar os nomes aos elementos corretos (campos e arquivos de usuários ou livros).

A classe *Biblioteca* deve possuir os seguintes métodos:

- *cadastraUsuário*: de retorno *void*, que recebe como parâmetro um objeto da classe *Usuário* e o armazena no objeto *Hashtable correspondente*. O CPF do usuário deve ser utilizado como chave;

  Não deve ser permitido cadastrar dois usuários com o mesmo CPF, sobrepondo o usuário anterior. Pode usar exceções, se quiser. Usar exceções não é obrigatório, mas sobrepor um usuário não deve acontecer.

- *cadastraLivro*: de retorno *void*, recebe como parâmetro um objeto da classe *Livro* e o armazena no objeto *Hashtable correspondente*; O código do *Livro* deve ser usado como chave; O mesmo vale aqui, não pode cadastrar dois livros com o mesmo código.
- *salvaArquivo*: de retorno *void*, recebe como parâmetros um objeto da classe *Hashtable (que pode ser o cadastro de usuários ou o acervo de livros)* e um objeto da classe *String* contendo o nome do arquivo onde o outro parâmetro será salvo;

  Atenção, é para salvar o objeto HashTable. Com uma chamada salva-se todo o objeto com todos os objetos colecionados. Não é para salvar registro a registro, objeto a objeto.

  Atenção. É possível usar um mesmo método para salvar o objeto HastTable de usuários e de livro. O seu código deve controlar os nomes e campos usados. Mas se você achar melhor, pode usar dois métodos salvaArqUsu e salvaArqLiv, com objetivos específicos.

- *lêArquivo*: de retorno *void*, recebe como parâmetros um objeto da classe *String* contendo o nome do arquivo a ser lido (como o construtor, mas que pode ser chamado a qualquer hora, e lê somente o acervo de livros ou o cadastro de usuários);

  O mesmo aqui. Se você salvou a HashTable, de usuários, por exemplo, uma única leitura lê a HashTable salva no arquivo, que deve ser atribuída, obviamente, a uma referência a objeto da classe HashTable. Exatamente como vimos em sala de aula para a classe Coin.

  No programa principal, você deve controlar o vínculo do cadastro de usuários e de arquivos ao respectivo nome. Não "amarre". Permita o usuário salvar com o nome

que ele quiser... Pode, no entanto, concatenar com um padrão seu, para identificar facilmente se o arquivo contém a HashTable de usuários ou dos livros.

Aqui, novamente, registramos que é possível usar um único método e fazer o casting para a HastTable usando o "generics" adequado. Mas, se você achar melhor, pode desenvolver dois métodos: leArqUsu e leArqLiv

Colocamos o retorno *void*, pois esta operação vai necessariamente atribuir o objeto resgatado do arquivo a um campo da classe Biblioteca. Isso pode ficar interno ao código do método. Mas, se você fizer questão, retorne o objeto da classe *Object* lido do arquivo e depois atribua ao campo de *HashTable* correspondente ao cadastro de usuário ou livro e faça o casting adequado.

- *emprestaLivro*: recebe como parâmetros a referência a um objeto Usuário e a referência a um objeto da classe *Livro*. (as referências já devem ter sido validadas – obtidas através dos métodos *getLivro* e *getUsuário* – veja observação abaixo). Chama o método *empresta* no objeto *Livro* e atualiza o histórico no livro emprestado chamando *addUsuarioHist()*, e no objeto *Usuário* chamando *addLivroHist()*. A data do empréstimo é obtida consultando o relógio/calendário do sistema no momento da operação e/ou usando o *GregorialCalendar*;
- *devolveLivro*: o mesmo definido para o método anterior, só que chama o método *devolve* no objeto *Livro*. Aqui, no entanto, se você implementou a personalização direito, pode ser verificado se o usuário está devolvendo o livro com atraso e avisar de uma multa. Use a data do empréstimo e a data da devolução;

  Atenção. Precisa, necessariamente, atualizar REGISTRO no histórico, tanto do usuário quanto do livro, atualizando a data de devolução. Seria o mesmo que "carimbar" o cartão. Para isso, é necessário criar um método setDataDevol(), nas respectivas classes de "Emprestado", Emprest e EmprestadoPara para registar a data de devolução, que antes estava como null, por exemplo. Isso não estava claro no enunciado.

- *String imprimeLivros()*: Devolve uma *string* com a lista de livros cadastrados, ordenados pelo código do livro;
- *String imprimeUsuários()*: Devolve uma *string* com a lista de usuários cadastrados, ordenados pelo CPF;
- *Livro getLivro (int cod)*: Recebe o código do livro e obtém o objeto *Livro* da *Hashtable* correspondente. Se o livro não estiver cadastrado, deve gerar a exceção *LivroNaoCadastradoEx*;
- *Usuário getUsuário (long CPF)*: Recebe o CPF do usuário e obtém o objeto *Usuário* da HashTable correspondente. Se o usuário não existir na *Hashtable*, deve gerar a exceção *UsuárioNaoCadastradoEx*.

Obs.: Além dos métodos listados, verifique se as classes *Pessoa*, *Usuário*, *Livro* e *Biblioteca*, devem ter, para cada campo, um método *get<NomeDoCampo>()* que retorna o conteúdo do campo.

Obs.: Para todas estas classes, exceto *Biblioteca*, sobrescrever o método *toString()* para mostrar todas as informações sobre o objeto [no caso de *Usuario* e *Livro*, deve exibir inclusive o histórico].

Reforçando eu todas as classes até aqui devem pertencer a um pacote *chamado lp2qXX.biblioteca* (onde XX é o número do usuário).

VII) Desenvolver o programa principal.

No início, antes de carregar a biblioteca, o programa deveria entender se o bibliotecário quer iniciar como o cadastro zerado, ou se ele deseja executar o programa lendo as *HastTables* de arquivos. Faça isso num diálogo, ou de esta opção ao usuário de alguma forma através dos menus. Ou seja, não temos como adivinhar o que o usuário quer: deixe que ele decida.

O programa que deve ter os seguintes módulos (O conceito de módulo é abstrato aqui. Entenda módulo como bloco, rotina, característica, parte, etc. Se você modularizar o código, vai ficar mais próximo disso. Pense em opções de menu de um primeiro nível, que vão ter sub-opções - módulos):

- *Manutenção*, que cria, abre e salva os arquivos (dois) que contém, cada um, uma *Hashtable* (livros e usuários); Também dê liberdade ao bibliotecário para salvar com outro nome e iniciar a biblioteca com nomes específicos de arquivo.

- *Cadastro*, que cadastra usuários e livros. Deve exibir um menu com opções para: cadastrar usuários, cadastrar livros e salvar (logo) em arquivo. A opção de salvamento em arquivo deve exibir um sub-menu para que o usuário escolha se deseja salvar o cadastro de usuários ou o cadastro de livros. O usuário deve poder escolher os nomes dos arquivos, até para fazer uma nova versão, como se fosse um bkp. Utilize a mesma rotina que executa o mesmo no módulo de Manutenção. A ideia é que ao fazer o cadastro o usuário pode pedir para salvar logo, sem entrar em um outro nível de menu.

- *Empréstimo*: que executa a locação ou a devolução de um livro para um usuário. Um menu com as opções de exibir o cadastro de livros, fazer um empréstimo, ou uma devolução deve ser exibido. Ao fazer um empréstimo, devem ser obtidos os objetos *Livro* e *Usuário*. As exceções (livro não existe, usuário não cadastrado, cópia não disponível, etc.) devem ser tratadas. Quando um empréstimo for bem-sucedido os dados devem ser escritos na tela, o número de livros disponíveis atualizados, bem como os históricos do livro e do usuário atualizados. O mesmo para a devolução.

  Os dados a serem exibidos depois do empréstimo ou devolução servem como um "recibo" de transação. O toString pode ser usado e combinado para ficar coerente.

- *Relatório*, que permite listar o acervo de livros, o cadastro de usuários ou detalhes de um usuário ou livro específico (com seu histórico).

Observação. No programa principal, o programador deve customizar a política da biblioteca (por isso as classes de suporte estão em um pacote e o programa principal não está).

Por exemplo, pode restringir o número máximo de livros que um usuário pode emprestar de uma só vez e o número de dias que um usuário pode ficar com o livro sem pagar multa.

A forma de implementar esta customização é livre. Pode carregar informações e limites de um arquivo, ou de um arquivo *.properties*. Pode solicitar ao bibliotecário logo no início da execução, no mesmo momento quando ele vai decidir como iniciar o programa. Obviamente isso afeta as rotinas de empréstimo e devolução.

Os atrasos devem apenas ser sinalizados no momento em que o usuário vai devolver o livro, ou até no momento em que ele vai fazer um empréstimo. Por exemplo, não pode emprestar livro se a devolução de outro, já emprestado, estiver em atraso. A forma de implementar isso também é livre.

VII) Desenvolver dois *scripts* (um para Windows e outro para Linux) para inicializar o programa. Os *scripts* podem (devem) usar argumentos e/ou variáveis de ambiente para carregar a JVM e classes adequadamente (isso deve tornar os pacotes "independentes de localização).

VIII) Na entrega, além do programa, pacotes, etc. Deve ser deixado um banco de dados (os arquivos com as HashTables) com 5 usuários cadastrados, 5 livros cadastrados, 1 usuário com empréstimo em vigor e um usuário com livro devolvido. O nome dos arquivos devem ser respectivamente *u.dat* e *l.dat*. Com isso, para avaliar o programa, já vamos abrir este banco de dados, verificar os cadastros e avaliar os relatórios. Esse ponto é MUITO importante.

Não temos menus de exemplo. Você vai precisar desenvolver os seus. Mas temos os critérios de peso para os itens avaliados:

1) (4) O programa funciona? [vou logar na conta do usuário e executar o programa.]
2) (3) Arquivos de exemplo com as HashTables de usuário e livros
3) (5) Leitura de arquivos correta. Salvamento de arquivos correto. Permita a especificação de novos nomes para salvamento.
4) (5) Uso de pacote correto
5) (3) Uso de herança e toString adequados
6) (5) Datas com Gregorian Calender corretas, empréstimo e devolução, cálculo de multas, etc.
7) (2) Gerencia Empréstimos corretamente
8) (2) Gerencial Devoluções corretamente
9) (3) Trata as exceções do enunciado
10) (4) Atende aos requisitos
11) (1) Scripts funcionando? (Windows e Linux)
12) (2) Estrutura do código

Plus: permite parametrizar a configuração: atrasos, multas, etc. Usa o arquivo "properties"?