

## 2ª Lista de Exercícios PCII - 2015/2

Em todos os exercícios a(o) aluno(a) deve, além de atender aos requisitos enunciados, utilizar os conceitos e características de orientação a objetos do Java:

- Encapsulamento (incluindo modificadores de acesso),
- Herança (de classe e interface) e polimorfismo;
- Sobrecarga / Sobreposição de métodos;
- Geração / Captura e Tratamento de exceções;
- Uso das classes básicas (*Object*, por exemplo);
- Classes / pacotes
- Leitura de dados via fluxo de entrada
- Uso de *generics*: não recomendado, mas será aceito - se o aluno souber o que está fazendo...

(ou seja, boas praticas de programação orientada a objetos devem ser empregadas mesmo se não foram explicitamente solicitadas)

### Exercício 1 – Biblioteca, em 5 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, e acessá-los quando necessário.

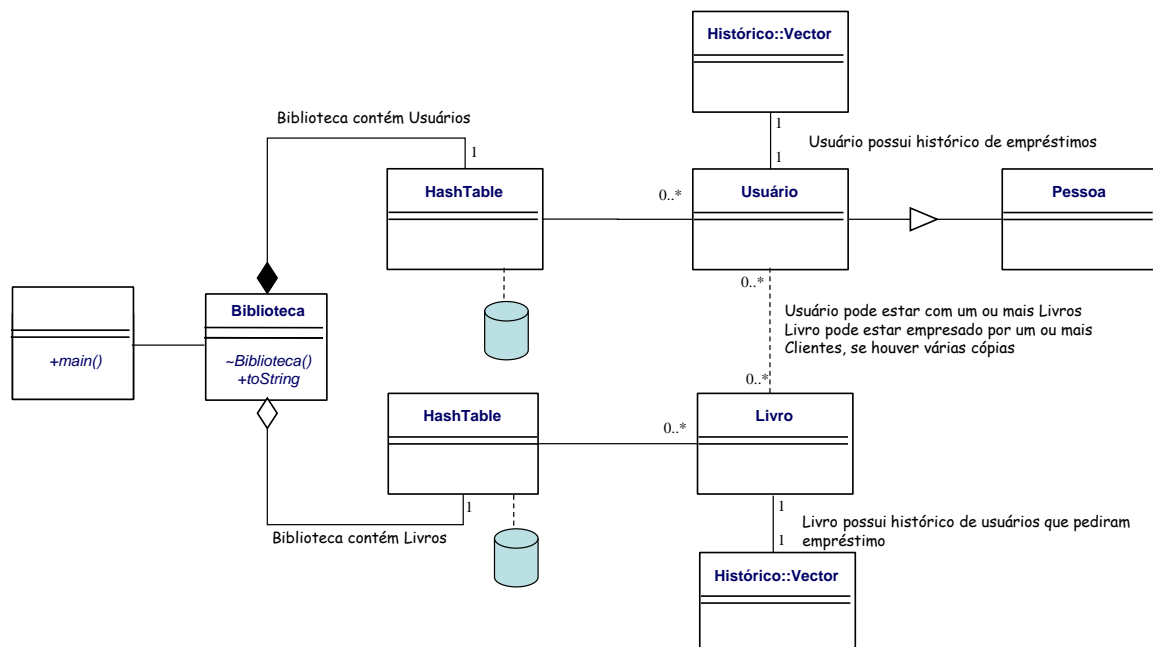


Figura 1: Modelo da Biblioteca

Requisitos:

**As classes dos itens I, II, III e IV devem pertencer a um pacote *chamado* `lp2gXX.biblioteca` (onde XX é o número do usuário).**

I) Crie uma classe *Pessoa* com os campos *nome* e *dataNasc* (da classe [`GregorianCalendar`](#) (`int year, int month, int dayOfMonth`)), que devem ser inicializados pelo construtor.

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

A classe *Usuário* também deve ter um campo *Histórico* da classe *Vector*. Este campo deve armazenar objetos da classe *Empréstimo*.

A classe *Empréstimo* deve conter a data do empréstimo, data de devolução e o código do *Livro* solicitado para empréstimo.

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

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

- Código do Livro (*String*)
- Título do Livro
- Categoria (que pode ser Aventura, Ficção, Romance, etc.)
- Quantidade (inteiro)
- Emprestados (inteiro)

... e os seguintes construtores:

- Um construtor que inicializa todos os seus campos
- Um construtor que recebe como parâmetro um objeto *String* e inicializa o campo do Título do Livro.

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*.

A classe *Livro* também deve ter um campo *Histórico* da classe *Vector*. Este campo deve armazenar objetos da classe *EmprestadoPara*.

A classe *EmprestadoPara* deve conter a data do empréstimo, data de devolução e o código do usuário que pegou o livro emprestado.

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

IV) Implemente a classe *Biblioteca* com:

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

A classe *Biblioteca* classe deve possuir dois construtores:

- um que inicialize os campos
- um construtor que carregue o cadastro de usuários e o cadastro de livros salvos em dois arquivos distintos.

*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 código do usuário deve ser utilizado como chave;
- *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;
- *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;
- *lêArquivo*: de retorno *void*, recebe como parâmetros um objeto da classe *String* contendo o nome do arquivos 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);
- *emprestaLivro*: recebe como parâmetros a referência a um objeto *Usuário* a referencia 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 objeto *Usuário* chamando *adLivroHist*. A data do empréstimo é obtida consultando o relógio/calendário do sistema no momento da operação.
- *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.
- *String imprimeLivros()*: Devolve uma string com a lista de livros cadastrados, ordenados pelo título.
- *String imprimeUsuários()*: Devolve uma string com a lista de usuários cadastrados, ordenados pelo nome.

- *Livro* *getLivro (String cód)*: 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 (int cód)*: Recebe o código 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 *Usuário* e *Livro*, deve exibir inclusive o histórico].

**As classes dos itens I, II, III e IV devem pertencer a um pacote chamado *lp2gXX.biblioteca* (onde XX é o número do usuário).**

V) Desenvolver o programa principal, que deve ter os seguintes módulos:

- *Manutenção*, que cria, abre e salva os arquivos (dois) que contém, cada um, uma HasTable (livros e usuários);
- *Cadastro*, que cadastra usuários e livros. Deve exibir um menu com opções para: cadastrar usuários, cadastrar livros e salvar 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.
- *Empréstimo*: que executa a locação de um livro para um usuário. Um menu com as opções de exibir o cadastro de livros ou fazer um empréstimo 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 registro do livro mantido (e salvo na *Hashtable* – dependendo de como for programado) e o histórico do usuário atualizado.

Obs: 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.

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

VI) 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”).

### **Exercício 2: Applet – Tabela de produtos**

Você vai adaptar a 3ª questão da Lista 1, da lista de produtos com ordenação.

Faça um painel para entrar com produtos, usando campos e botões.

Coloque os produtos em uma tabela gráfica (pode usar o exemplo *SwitgSet2*), permitindo a ordenação por qualquer cambo (como feito para o Ex 3 da Lista 1)

Faça a previsão de mensagens de erro (em janelas pop-up ou campos de texto).

Utilize componentes adicionais ou recursos que você junte necessários.

O *applet* deve ser acessado pela Internet, através da URL: 152.92.236.11/~<seu login>

Para isso você vai precisar criar um diretório *public\_html*, dentro do seu diretório *home* e, lá dentro, um arquivo *index.html* com as permissões e o conteúdo adequado, além das classes em Java.

### **Exercício 3: Threads- Problema do Banheiro Unissex**

Suponha que em um local movimentado haja apenas um banheiro, “Unisex”. O banheiro possui em seu interior cabines individuais, mas homens e mulheres não podem usá-lo ao mesmo tempo.

Vamos utilizar técnicas de programação concorrente para controlar o acesso ao banheiro e simular essa situação.

Vamos, também, criar algumas possibilidades para esta solução, usando herança.

I) Crie as classes de exceção *BanheiroOcupado* e *BanheiroVazio* que herdem de *ArrayIndexOutOfBoundsException* e exibam, respectivamente, as mensagens "Desculpe - O banheiro esta lotado" e "O banheiro esta vazio". Crie as classes *Alerta* e *Espera* que herdem de *Exception* e exibam, respectivamente, as mensagens "Homens e mulheres não podem usar o banheiro ao mesmo tempo" e “Aguarde um momento por favor”.

II) Crie a classe *Humano* com os campos *gênero* (que pode ser masculino ou feminino), *identificador* (que é um valor inteiro) e um campo contendo uma referência para um objeto da classe *Banheiro* (que será definida abaixo). O construtor da classe *Humano* recebe como parâmetros um objeto da classe *Banheiro* e um valor inteiro, e inicializa os campos correspondentes.

III) Crie a classe *Banheiro* com campos inteiros *quantidade* e *capacidade* e o array *cabines*, onde se possa alocar instâncias da classe *Humano*. Seu construtor utiliza o campo *capacidade* para inicializar o tamanho do array, que deve ser 5. A classe *Banheiro* deve ter os seguintes métodos:

- *void entraBanheiro(Humano h)*: Homens e mulheres não podem usar o banheiro ao mesmo tempo, logo este método deve verificar o gênero (feminino ou masculino) de quem tenta entrar no banheiro, e verificar se dentro do banheiro há pessoas do sexo oposto. Caso haja, deve lançar a exceção *Alerta*, senão, o método tenta inserir o objeto no banheiro e em caso de sucesso imprime a mensagem "<Homem ou Mulher> <identificador> Entrou". Se o banheiro estiver lotado, deve lançar a exceção *BanheiroOcupado*.

- *void saiBanheiro()*: Tenta retirar um dos objetos (Homem ou Mulher) que estão usando o banheiro. Caso o banheiro esteja vazio, lança a exceção *BanheiroVazio*.

Lembre-se de utilizar mecanismos que garantam que homens e mulheres tenham as mesmas chances de entrar no banheiro. A exceção *Espera* deve ser lançada nos casos em que as condições deste mecanismo não tenham sido atendidas.

IV) - Crie a classe *BanheiroExt*, que estende *Banheiro* e modifica o número de cabines disponíveis (tamanho do banheiro). Seu construtor recebe como parâmetro o novo tamanho máximo do banheiro. Os métodos *entraBanheiro* e *saiBanheiro* devem manter as mesmas funções, mas serão sobrescritos da seguinte forma:

- Terão o modificador *synchronized* acrescentado à sua assinatura, para que seja resolvido o problema da concorrência.

- Para sincronizar o acesso ao banheiro, deve ser usada a seguinte "receita":

No início do método *entraBanheiro()*:

```
while(banheiro cheio)
{
    wait(); //Espera alguém sair
}
```

No início do método *saiBanheiro()*:

```
while(banheiro vazio)
{
    wait(); //Espera alguém entrar
}
```

No fim de ambos os métodos:

```
notifyAll();
```

V) - Crie as Classes *Homem* e *Mulher* que estendam a classe *Humano*, implementem a interface *Runnable* e tenham o campo de classe *status* (que pode ser utilizado para garantir justiça no acesso ao banheiro). O construtor de cada classe recebe como parâmetros um objeto da classe *Banheiro* e um valor inteiro, inicializa esses campos e

também o campo gênero com as palavras "Masculino" para *Homem* e "Feminino" para *Mulher*. Em cada classe implemente os métodos:

- *private void entraBanheiro()*: Tenta colocar um objeto (um homem para a classe *Homem* e uma mulher para a classe *Mulher*) no banheiro. A cada tentativa deve exibir a mensagem "Tentando entrar <Homem ou Mulher> <identificador>". A cada vez que entrar no banheiro, os homens e mulheres devem permanecer lá por um tempo aleatório (use: `Thread.sleep((long)(Math.random()*100))`). Se a entrada for bem-sucedida imprime a mensagem "<Homem ou Mulher> <identificador> Entrou". Caso ocorra alguma exceção do tipo *BanheiroOcupado* ou *Alerta*, deve "dormir" um tempo aleatório e tentar novamente.

- *private void saiBanheiro()*: Tenta retirar um objeto do banheiro. A cada tentativa imprime a mensagem "Saindo <homem ou mulher> <identificador>". "<Homem ou Mulher> <identificador> Saiu". Caso capture uma exceção do tipo *BanheiroVazio*, "dorme" um tempo aleatório e tenta novamente.

- *public static boolean getStatus()*: Retorna o status das instâncias da classe.

- *public static boolean setStatus()*: Altera o status das instâncias da classe.

- O método *run* deve consistir de um loop infinito, que em cada passo tenta colocar um homem ou mulher no banheiro, dorme alguns segundos, e em seguida tenta retirar alguém do banheiro.

VI) - O programa principal deve perguntar qual tipo de banheiro será criado: *Banheiro* ou *BanheiroExt* e também o número de homens e mulheres que tentarão utilizar o banheiro. O programa inicializa o "banheiro" com o tamanho adequado (fornecido pelo usuário no caso de *BanheiroExt* e default para *Banheiro*) e as instâncias de *Homem* e *Mulher*, que em seguida, terão suas atividades iniciadas, como *threads*.

Exemplo de Execução:

```
1 - Banheiro  2 - Banheiro Unissex
1
```

```
Numero de Mulheres: 8
```

```
Numero de Homens: 5
```

```
Tentando Entrar - Homem 2
```

```
Homem 2 Entrou
```

```
Tentando Entrar - Homem 4
```

```
Tentando Entrar - Homem 3
```

```
Tentando Entrar - Homem 1
```

```
Homem 4 Entrou
```

```
Tentando Entrar - Mulher 1
```

```
Tentando Entrar - Homem 5
```

```
Homem 1 Entrou
```

```
Homem 3 Entrou
```

```
Tentando Entrar - Mulher 5
```

```
Mulher 1: Homens e mulheres nao podem usar o banheiro ao mesmo tempo
```

```
Homem 5 Entrou
```

Mulher 5: Homens e mulheres nao podem usar o banheiro ao mesmo tempo  
Homem 3 Saiu  
Homem 4 Saiu  
Homem 2 Saiu  
Tentando Entrar - Homem 4  
Tentando Entrar - Mulher 8  
Homem 1 Saiu  
Homem 4: Aguarde um Momento  
Mulher 8: Homens e mulheres nao podem usar o banheiro ao mesmo tempo  
Homem 5 Saiu  
Tentando Entrar - Mulher 6  
Tentando Entrar - Mulher 3  
Tentando Entrar - Homem 4  
Tentando Entrar - Mulher 8  
Tentando Entrar - Mulher 5  
Mulher 6 Entrou  
Mulher 3 Entrou  
Tentando Entrar - Mulher 2  
Homem 4: Homens e mulheres nao podem usar o banheiro  
Mulher 8 Entrou  
Mulher 5 Entrou  
Tentando Entrar - Mulher 7  
Tentando Entrar - Mulher 1  
Tentando Entrar - Mulher 4  
Tentando Entrar - Homem 1  
Mulher 2 Entrou  
Mulher 7: Desculpe - O banheiro esta lotado  
Mulher 1: Desculpe - O banheiro esta lotado  
Mulher 4: Desculpe - O banheiro esta lotado  
Homem 1: Homens e mulheres nao podem usar o banheiro ao mesmo tempo