Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática – CEEI

Departamento de Sistemas e Computação – DSC

Professores: Gustavo (turma1), Kyller (turma 2), Livia (turma3), Raquel (Turma 4)

Disciplina: Laboratório de Programação 2

Período: 2013.2

Laboratório 07

Neste laboratório iremos praticar o uso de coleções do tipo lista e introduzir o conceito de itens comparáveis para usar recursos de ordenação de itens de uma coleção. Além disso, estimular a realização de testes de unidade e confecção de documentação e continuar desenvolvendo habilidades na definição da lógica de controle de programas Java.

Instruções

Este laboratório terá duração de 1 semana, sendo que a primeira parte não deve ser entregue e a segunda parte será entregue seguindo as datas abaixo:

- Data de entrega para todas as turmas: 17/12/2013 até 23:59h;
- Data de entrega com penalidade (50% da nota): 18/12/2013 até 23:59h;
- Crie o projeto lab07 no eclipse e programe todos os exercícios dentro do pacote lp2.lab07;
- Salve o pacote criado, com as classes implementadas, em um arquivo zip chamado lab07-<seuNome>-<suaTurma>.zip (ou .tgz) e o submeta através deste formulário;
- Certifique-se de que seus programas não têm erros de compilação, caso contrário eles não poderão ser corrigidos;
- Antes da definição de cada classe escreva o seu nome em comentário (/*Aluno: <seu nome>*/);
- Lembre de escrever o javadoc das classes criadas (não precisa para os testes).
    - Desenvolver seus códigos usando TDD pois aumenta a qualidade e a confiabilidade do mesmo, portanto o uso dessa técnica, assim como *os testes gerados serão considerados na nota do Laboratório*. Da mesma forma, *a documentação também* é essencial para a manutenção e entendimento do código, para tanto, escreva o javadoc das classes desenvolvidas.

Leia antes de fazer o lab

De uma forma bem coloquial, uma coleção, no contexto de orientação a objetos, é um "saco" de objetos, ou seja, um objeto que contém outros objetos. Mais formalmente, uma coleção é uma estrutura de dados que pode armazenar ou agrupar referências a outros objetos (um *container*). Essencialmente, uma coleção deve oferecer métodos para adicionar e remover objetos, recuperar um objeto específico e varrer todos os seus objetos. Existem diversos tipos de coleções, mas, nesse momento queremos nos concentrar em uma lista implementada como um array.

Uma lista é uma coleção indexada de objetos. Índices de uma lista iniciam em zero, isto é, o índice do primeiro elemento é zero. As principais características de uma lista são: (i) o usuário da lista tem total controle sobre onde os elementos devem ser inseridos ou de onde eles devem ser removidos e (ii) é possível acessar elementos da lista usando seu índice numérico, que indica a posição do elemento na lista. Uma implementação de lista muita usada em Java é representada pela classe java.util.ArrayList (uma implementação de array dinâmico; sem limitação de tamanho).

A classe ArrayList (de fato, ArrayList<E>) fornece diversos métodos (você vai ver mais olhando a documentação da API de java), entre os quais:

`boolean add(E elemento)`: Adiciona o elemento especificado no final da lista;

`E get(int index)`: Retorna o elemento da lista no index especificado;

`int size()`: Retorna o número de elementos da lista.

Observe o uso do tipo <E> na definição dos métodos acima. <E> refere-se a um tipo genérico que será especificado no momento da instanciação da classe. Por exemplo, se quisermos criar uma agenda de contatos bem simples, podemos fazer isso usando um ArrayList de strings. Precisaremos primeiro declarar o objeto agenda:

```
List<String> agenda = new ArrayList();
```

e em seguida adicionar nossos contatos.

```
agenda.add("David Gilmour;11 1111-1111");

agenda.add("Mick Jagger;22 2222-2222");

agenda.add("Eric Clapton;33 3333-3333");
```

Depois podemos listar todos os nossos contatos.

```
for (int i = 0; i < agenda.size(); i++) {

    System.out.printf("%d - %s\n", i, agenda.get(i));

}
```

Note que ao declarar um objeto você vai indicar que ele é do tipo List. Mas ao fazer new você vai escolher uma das implementações da interface List, neste lab, você escolherá ArrayList. Fazemos isso para escrever código menos acoplado às implementações Leia mais em: Explorando a Classe ArrayList Java e no material da disciplina teórica, P2.

Parte I

A classe CD representa um CD de músicas de um artista (ou banda). Cada CD possui como atributos o artista (pode ser cantor, banda, conjunto, trio pé de serra, etc.), título, trilha principal (principal música do CD) e uma coleção de M músicas. Título e autor não podem ser vazios. O valor de M (M > 0) é definido no momento da criação de um objeto do tipo CD. Se necessário novos atributos podem ser acrescentados à sua classe para melhorar a manipulação dos dados. Escreva a classe especificada, incluindo:

- 2 construtores: o primeiro recebe título, autor e M, enquanto o segundo recebe título, autor, M e música principal. Se não for informado a trilha principal de um CD o seu valor deve ser vazio. Considere que, inicialmente, as faixas do CD possuem valor vazio;
- Métodos acessadores para os atributos da classe, e um método modificador para a trilha principal;
- Método que retorna a i-ésima faixa do CD, onde i é passado como parâmetro. Se o valor de i não for válido, retorna null;
- Cadastro das faixas de músicas do CD. O método de cadastro de faixas de músicas recebe como parâmetro o nome da faixa que deve ser adicionado no conjunto de M faixas na próxima posição livre (ex. na primeira vez que o método for chamado, a faixa será adicionada na posição 0 da lista; na segunda vez, na posição 1 e assim por diante). Se o usuário tentar cadastrar mais faixas do que a quantidade máxima de faixas permitida (M), o método deve retornar false, sem adicionar a nova faixa; caso contrário, a faixa será adicionada e o método deve retornar true ;
- *toString*: para retornar uma String que apresenta as informações do CD. Esta String deve conter todas as informações de cada CD na seguinte ordem: autor, título, trilha principal e todas as músicas do CD seguindo o formato “número da música – nome da música”. As músicas devem ser separadas por “/”;

- *equals*: para comparar dois CDs. CDs são considerados iguais se tiverem o mesmo autor e título;

Não esqueça de implementar os testes da classe CD e escrever o javadoc. Use exceções sempre que necessário para lidar com situações de erro no seu sistema.

Parte II

Escreva uma classe chamada MinhaCDteca que representa a sua coleção de CDs. Esta classe deve conter uma lista de CDs como atributo (objetos da classe CD). As informações referentes aos CDs devem ser armazenadas através de um objeto do tipo List<CD>. Assim, esta nova classe de coleção de CDs não tem restrição de tamanho. Esta classe deve encapsular uma coleção de CDs, tornando a manipulação da coleção simples, correta e útil. Os seguintes métodos devem ser implementados:

- *adicionaCD*: para adicionar um objeto CD dentro da coleção de CDs. Um CD pode ser inserido independente de já existir na coleção ou não. Como o CD sempre será inserido não é necessário nenhum retorno;
- *adicionaCDs*: para adicionar uma coleção de CDs dentro da coleção de CDs. Mais uma vez deve ser permitido inserir CDs duplicados na coleção. Como CDs sempre serão inserido não é necessário nenhum retorno;
- *removeCD*: para remover um objeto da coleção a partir de uma chave, que neste caso é o título do CD. Se o CD a ser removido existir na coleção de CDs, então o método retornará o CD removido. Caso contrário deve retornar null. Se existir mais de um CD em questão, um deles deve ser removido;
- *removeCDs*: para remover uma coleção de CDs da coleção original. Se pelo menos um CD for removido da coleção de CDs original, então o método retornará true. Caso contrário deve retornar false;
- *pesquisaCD*: para pesquisar (achar a referência a) um CD particular da coleção, dada uma chave (que neste caso é o título do CD). Quando o CD pesquisado não existir na coleção, retorna null;
- *numeroDeCDs*: para retornar o número de CDs existentes na coleção;
- *toString*: para retornar uma String que representa a coleção de CDs. Deve iterar em todos os CDs da coleção para compor a String a ser retornada. Esta String deve conter informações sobre cada CD da coleção, um por linha);
- *equals*: para comparar duas coleções. Duas coleções são iguais se os CDs que elas armazenam são os mesmos, independentemente da ordem em que eles estão armazenados na coleção;

Não esqueça de escrever o javadoc desta classe e de implementar os testes da classe MinhaCDteca. Use exceções sempre que possível para lidar com situações de erro no seu sistema.

Parte III – Extra opcional[1]

A ideia agora é usar facilidades de java no que diz respeito a ordenação de elementos de uma coleção. Se você estudar qualquer algoritmo de ordenação vai perceber que é imprescindível identificar se um item é menor, igual ou maior que outro item. Em java, por convenção, este método se chama compareTo. Ele faz parte de uma interface chamada Comparable (você vai entender muito melhor isso lá na frente, por enquanto o importante é aprender a usar o recurso). Suponha que você quer comparar dois objetos do tipo Pessoa. Então na classe pessoa você tem que acrescentar o seguinte

```
public classe Pessoa implements Comparable<Pessoa>
```

Dentro da classe Pessoa você deve implementar o método compareTo. O método tem um parâmetro de entrada; nesse caso ele será do tipo Pessoa. O método será chamado da seguinte forma:

```
Pessoa p1 = new Pessoa("Raquel");
```

[1] Se você não fizer a parte III, ainda assim pode tirar 10,0 no lab (se tudo o mais estiver correto). Se você fizer a parte III é como se seu lab fosse corrigido valendo mais de 10,0 pontos, então a chance de tirar 10,0 é maior. Mesmo que você erre algo, ainda tem chance de tirar 10,0.

```
Pessoa p2 = new Pessoa(“Livia Maria”);

if(p1.compareTo(p2) == 0) syso(“Pessoas tem nomes iguais”);

if(p1.compareTo(p2) < 0) syso(“p1 tem nome menor que p2”);

if(p1.compareTo(p2) > 0) syso(“p1 tem nome maior que p2”);
```

Este método retorna um número inteiro. Esse número é igual a zero se as duas pessoas tem nomes iguais; um número menor que zero se p1 tem nome menor que p2 e um número maior que zero se p1 tem nome maior que p2. Note que nesse caso de string, um nome ser menor indica que ele vem antes na ordem alfabética. Nesse exemplo se o método estiver implementado corretamente, a execução deve entrar no terceiro if, já que Raquel vem depois de Livia Maria (de acordo com o alfabeto).

A boa notícia é que a grande maioria das classes em Java já implementam essa interface Comparable e consequentemente já tem o método compareTo.

Agora volte à implementação da classe CD e implemente o método compareTo da interface Comparable:

- *compareTo*: para comparar dois CDs usando o artista associado ao CD de forma ascendente no alfabeto. Por exemplo, se o CD cujo artista é “Academia da Berlinda” for comparado ao CD cujo artista é “Nação Zumbi” – academia.compareTo(nação) – então o valor retornado deve ser menor que zero.

Lembre-se de escrever o javadoc e testar o novo método criado de forma apropriada.

Agora volte à implementação da classe MinhaCDTeca e acrescente o seguinte método:

- *cdTecaOrdenadaPorArtista()*: este método deve retornar uma String em que cada linha contém “Titulo do CD” – “artista do CD”, de forma que os artistas estão ordenados em ordem ascendente (A □ Z). Dica: olhe a classe Collections.

Lembre-se de escrever o javadoc e testar o novo método criado de forma apropriada.