Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática – CEEI

Departamento de Sistemas e Computação – DSC

Disciplina: Laboratório de Programação 2

Laboratório 04 - 30/04/2015 - 11/05/2015

Neste laboratório iremos praticar o **uso de coleções** do tipo lista, mapa e conjunto. Também vamos introduzir o conceito de **objetos comparáveis** para usar recursos de **ordenação** de itens de uma coleção: `Collections.sort(...)`. Além disso, estimular a **realização de testes de unidade** e continuar desenvolvendo habilidades na definição da lógica de controle de programas Java. Em resumo os Objetivos são:

- Praticar o uso das diferentes coleções de Java;
- Utilizar Exception para tratamento de erros;
- Criar e utilizar testes de unidade utilizando o JUnit;
- Criar objetos comparáveis e utilizar Collections.sort(...)

Para ver exemplos de uso e documentação sobre o JUnit, clique aqui.

Descrição da Implementação: sp2fy

O **sp2fy** é um programa para armazenar e organizar as músicas de um **perfil** de usuário.Todas as músicas estão organizadas em álbuns que, além das músicas, possuem um artista, um título e um ano de lançamento. Lembre que, em um álbum, as músicas estão organizadas em faixas, de forma que eu possa tocar a música da faixa 1, 2, 3, e assim por diante.

Tanto os albuns como as músicas serão utilizadas pelo usuário da seguinte forma: Cada perfil de usuário (ou simplesmente **perfil**) possui um agrupamento de albuns. Este agrupamento deve ser ordenável de acordo com o título do álbum. É possível que o usuário possa definir também um conjunto de álbuns favoritos.

Cada perfil possui também playlists, que são várias músicas agrupadas de acordo com o desejo do usuário. Lembrando que cada playlist tem um nome. Por exemplo *"Para relaxar"*, *"Arrumar a Casa"*, *"Para fazer Laboratorios"*, etc. Associado a cada nome temos um agrupamento de músicas. Lembrando que, em cada uma dessas playlists, as músicas são tocadas em uma ordem. No seu código, deve ser possível definir vários nomes de playlists e associar um agrupamento de músicas a cada uma. **Ao criar uma playlist com um nome já existente, a antiga será sobrescrita.**

Agora que você está ficando experiente em Java, realize o **tratamento de erros com Exception**, evitando que sejam criadas músicas/álbuns com títulos, gêneros e artistas vazios e também ano ou duração negativa. Também escreva testes de unidade para verificar se sua implementação apresenta defeitos.

Siga os seguintes passos para a implementação do sp2fy:

Passo 1: Criação das Músicas

Importante: Baixe a classe de teste de música aqui.

Implemente a(s) entidade(s) necessária(s) para representar uma Música. Cada música possui um título, uma duração **(em minutos inteiros)** e gênero. Duas músicas são considerada iguais quando possuem o mesmo título e a mesma duração. Forneça os métodos e atributos para que Musica funcione da forma adequada. Para verificar se sua implementação está correta, utilize a classe **`TestaMusica.java`** que possui alguns casos de teste do JUnit. Se sinta livre de criar mais casos de teste.

Passo 2: Criação dos Álbuns

Implemente a(s) entidade(s) necessária(s) para representar os álbuns do seu perfil do sp2fy. Portanto, o álbum deve armazenar um agrupamento de músicas que permita tocar as músicas de acordo com sua faixa. Para facilitar o uso do sp2fy, **todo álbum começa vazio** e então adicionamos Músicas ao álbum. A ordem em que as músicas são adicionadas correspondem à faixa da música. Em outras palavras, a primeira música adicionada será a faixa 1, a segunda será a faixa 2, e assim por diante. Também é importante saber qual o **tempo total de duração de álbum.** Note que esse tempo muda de acordo com a duração de todas as músicas que estão contidas no álbum.

Além das músicas, um álbum deve possuir um artista, um título e um ano de lançamento. Uma vez que a comparação de todas as músicas de um álbum é custosa, iremos considerar que dois álbuns serão iguais se possuírem o mesmo nome e o mesmo artista.

Escreva a classe `TestaAlbum.java` contendo alguns testes para a criação de álbuns. Dentre eles:

- Título, Artista e Ano devem ser construídos da forma correta.
- Crie algumas músicas e adicione as músicas no álbum.
    - Verifique se as Músicas pertencem realmente ao álbum.
    - Verifique se as faixas estão correspondentes à ordem de inserção no álbum.
- Remova algumas músicas do seu álbum.
    - Verifique se as músicas foram realmente removidas do álbum.

Passo 3: Criação dos Perfis de Usuários

Implemente a(s) entidade(s) necessária(s) para representar um perfil de usuário. Para esse passo, faça apenas com que cada perfil, **ao ser criado**, possua um nome, **um agrupamento de álbuns vazio** e um agrupamento vazio de álbuns favoritos. Em outras palavras, um perfil começa vazio e o usuário adiciona álbuns ao seu perfil, durante o seu uso. **Note que apenas álbuns que pertencem ao usuário podem ser adicionados à coleção de favoritos.** Para isso, use dois métodos distintos: `adicionaAlbum(Album album)` e `adicionaAosFavoritos(Album album)`.

Escreva a classe `TestaPerfilSimples.java` que vai:

1. Criar músicas, e a partir delas, criar álbuns.
2. Após a criação de músicas e álbuns, crie um perfil;
3. No perfil criado comece a adicionar os álbuns criados em '1'.
4. Comece também a adicionar os álbuns favoritos ao perfil.

Passo 4: Criação das Playlists

Utilize o Perfil criado para implementar as playlists. Lembre que são armazenadas várias playlists, cada uma com seu respectivo nome (nome da playlist) que permite recuperar quais as músicas associadas a essa playlist. Crie o método adicionaPlaylist(String nomePlaylist, String nomeAlbum, int faixa). Esse método realiza as seguintes operações:

1. Verifica se a playlist com esse nome existe.
    a. Se não existir, cria um agrupamento vazio de músicas e associa esse agrupamento vazio ao nome especificado.
    b. Se existir, recupere o agrupamento (playlist) existente.
2. Recupera o álbum com o nome especificado no método (nomeAlbum)
    a. Se não existir, use Exception para enviar mensagem dizendo que "Álbum não pertence ao Perfil especificado".
    b. Se existir, pegue a faixa do álbum especificado na assinatura do método e adicione no agrupamento resultante da operação em 1.
3. Guarde a playlist atualizada no seu agrupamento de playlists.

Para esse passo, tenha cuidado com a verificação de faixas, nomes de álbuns e adição da música especificada na playlist. Note que são usados dois agrupamentos distintos: um para armazenar todas as playlists de acordo com seu nome, e outro agrupamento para armazenar as músicas da playlist especificada. Além disso, a playlist é um agrupamento de músicas que não pode ser um Album, pois são entidades diferentes. Ou seja, para esse sistema você não precisa criar a classe Playlist. Crie apenas se achar necessário com o objetivo de melhorar seu design. Para criar seus testes, crie a classe `TestaPerfilPlaylist.java`.

Passo 5 (Extra): Ordenar os Álbuns

Realize as modificações necessárias para que os seus álbuns sejam ordenáveis por ano de lançamento. Para que isso ocorra é necessário determinar que os albuns sejam comparáveis. Isso é realizado usando o Comparable<Album>.

Outras sugestões: Realize **também** outros três métodos para permitir diferentes ordenações. Para isso, leia sobre o uso do Comparator<T>.

1. Extra 1: Ordenação por nome de artista do álbum.

2. Extra 2: Ordenação por quantidade de músicas na lista.
3. Extra 3: Ordenação por duracao total do álbum.

Dicas para a implementação:

Verificação de Instâncias repetidas:

Para verificar se um agrupamento já possui uma instância (método contains(), ou o uso de conjuntos), é necessário ensinar a Java o que são duas Musicas/Albuns iguais. Portanto, não esqueça de implementar .equals e .hashCode para todas as suas classes. É importante que você pense quais são os elementos "chave" das minhas classes. Em outras palavras, que atributos serão utilizados para determinar que duas instâncias são iguais. Para facilitar a implementação do hashCode, use a implementação automática dele fornecida pelo Eclipse que gera automaticamente o hashCode() e o equals(...). **É altamente recomendado que você implemente o seu próprio equals(..).**

Uso do Collections.sort(....) e Comparable<T>:

O Comparable é usado para determinar que uma classe T (uso do generics) é comparável. Ou seja, é possível determinar entre duas instâncias t1 e t2 de T qual delas é maior, igual ou menor que a outra. Em outras palavras, determinar se: t1 < t2; t1 == t2; t1 > t2. Para isso deve ser implementado o método compareTo(T outraInstancia){...} que retorna um inteiro de forma que:

- **Retorna 1** se t1 > t2;
- **Retorna 0** se t1 == t2;
- **Retorna -1** se t1 < t2;

Considere a implementação de uma classe Pessoa abaixo:

```
public class Pessoa implements Comparable<Pessoa> {
	private String nome;
	private int idade;

	public Pessoa(String nome, int idade) {
		this.nome = nome;
		this.idade = idade;
	}
	//imagine que aqui temos os gets e sets de Pessoa

	//o compareTo compara pessoas apenas usando suas idades.
	public int compareTo(Pessoa outraPessoa){
		if(this.getIdade() > outraPessoa.getIdade()){
			return 1;

		}else if(this.getIdade() == outraPessoa.getIdade()){
			return 0;

		}else{//caso restante: this.getIdade < outraPessoa.getIdade()
			return -1;
```

```
            }
        }
}
```

Agora vamos comparar: Pessoa eu = new Pessoa(“Neto”,28) e Pessoa vovo = new Pessoa(“Maria”,70). Os resultados são os seguintes:

- eu.compareTo(vovo) → vai retornar -1, pois eu < vovo
- vovo.compareTo(eu) → vai retornar 1, pois vovo > eu

Após adicionar o Comparable<T>, a coleção é automaticamente ordenada utilizando o método estático: Collections.sort(suaColecao). Note que isso não deve ser feito em conjuntos HashSet, pois eles não devem possuir uma indexação e portanto, não podem ser ordenados.

Uso de Collections

Como vimos na aula teórica, uma coleção, no contexto de orientação a objetos, é um agrupamento 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, cada uma com suas características específicas.

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 - http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), entre os quais:

boolean add(E elemento): Adiciona o elemento especificado na coleção;

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 - http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298 - e no material da disciplina teórica, P2.