Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática – CEEI
Unidade Acadêmica de Sistemas e Computação – UASC

Disciplina: Laboratório de Programação 2

Laboratório 04

Como usar esse guia:

- Leia atentamente cada etapa
- Quadros com dicas tem leitura opcional, use-os conforme achar necessário
- Preste atenção nos trechos marcados como importante (ou com uma exclamação)!

Sumário

Acompanhe o seu aprendizado

Conteúdo sendo exercitado

- Coleções como objetos que armazenam outros objetos
- Operações básicas sobre coleções: adicionar, remover, pesquisar e iterar
- Coleções em Java: listas, conjuntos, mapas
- O conceito de generics aplicado a coleções Java
- Implementações de coleções baseadas em tabelas hash
- O “contrato” hashcode+equals

Objetivos de aprendizagem

Ao final desse lab você deve conseguir:

- Entender coleções como uma estrutura de dados representada como um objeto de objetos;
- Conhecer o funcionamento e a implementação das operações básicas de estruturas de dados (adicionar, remover, pesquisar e iterar) no contexto de objetos;
- Usar a API de Java de coleções com *generics*, especialmente para as estruturas ArrayList, HashSet e HashMap.

Perguntas que você deveria saber responder após este lab

- Como se diferenciam as listas, conjuntos e mapas oferecidos em Java? Explique com exemplos.
- Qual a semântica de funcionamento de um ArrayList? Por exemplo, você entende o que acontece internamente quando chamamos a operação add() sobre um objeto do tipo ArrayList?
- Qual a semântica de funcionamento de um HashSet?
- Qual a semântica de funcionamento de um HashMap?
- Considerando as implementações de coleções de Java, em quais situações cada coleção é mais apropriada para ser usada?
- O que é *generics*? E qual a sua vantagem no contexto de coleções?
- Explique o “contrato” entre equals e hashcode e porque é importante.

Para se aprofundar mais...

- Referências bibliográficas incluem:
  - material de referência desenvolvido por professores de p2/lp2 em semestres anteriores (ONLINE)
  - o livro Use a cabeça, Java (LIVRO-UseCabecaJava)
  - o livro Java para Iniciantes (Livro-JavaIniciantes)
- Tudo que você precisava saber sobre o framework de coleções Java (https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html)
- Entenda a relação entre hashcode e equals (http://blog.algaworks.com/entendendo-o-equals-e-hashcode/)

Introdução

No laboratório anterior, a maioria dos alunos controlavam o número de itens a serem inseridos a partir de um array. Um array é uma estrutura de dados rápida e eficiente, fácil de ser trabalhar mas apresenta suas limitações.

Um problema do array é que, para o mesmo ser eficiente, ele não é capaz de ser expandido ou reduzido. Um array de 10 elementos sempre terá 10 elementos. Existem duas alternativas para essa situação, simplesmente não adicionar o elemento ou expandir o array. Essas duas soluções são descritas no código abaixo:

```
// Alternativa 1: Não adicionar o elemento
private Object[] elementos = new Object[10];

public boolean adicionarElemento(Object elemento) {
    for (int i = 0; i < this.elementos.length; i++) {
        if (this.elementos[i] == null) {
            this.elementos[i] = elemento;
            return true;
         }
    }
    return false;
}
```

```
// Alternativa 2: Expandir o array
private Object[] elementos = new Object[10];

public boolean adicionarElemento(Object elemento) {
    for (int i = 0; i < this.elementos.length; i++) {
        if (this.elementos[i] == null) {
            this.elementos[i] = elemento;
            return true;
        }
    }
    Object[] novoElementos = new Object[this.elementos.length * 2];
    novoElementos[this.elementos.length] = elemento;
    System.arraycopy(elementos, 0, novoElementos, 0, this.elementos.length);
    this.elementos = novoElementos;
}
```

A necessidade de ter estruturas de dados automaticamente expansíveis é tão comum que Java (e demais linguagens) oferece classes básicas para tais entidades. Estas entidades fazem parte da API (interface de programação) de coleções de Java, que fazem parte do pacote de utilitários de java (java.util).

Dizemos assim que tais coleções encapsulam os dados que as compõem e as operações a serem realizadas sobre essa estrutura. Em OO, o conceito de encapsulamento rege todo comportamento e estrutura de código.

Das diferentes estruturas de dados descritas nas coleções de Java, três são de maior importância:

- **Lista:** representa uma coleção de elementos (com possibilidade de repetição) e com ordem
- **Conjunto:** representa uma coleção de elementos (sem repetição) e sem ordem
- **Mapa:** representa uma associação entre dois conjuntos (chave e valor)

Abaixo, descreveremos uma classe de cada tipo de coleções.

1. Lista

Enquanto um array é também um agrupamento sequencial, as listas das coleções de Java oferecem métodos que facilitam a manipulação de tais estruturas. Um exemplo é o java.util.ArrayList que apresenta os métodos abaixo:

```
private ArrayList listaElementos = new ArrayList();

public boolean adicionarElemento(Object elemento) {
    return listaElementos.add(elemento);
}

public void removerElemento(Object elemento) {
    listaElementos.remove(elemento);
}

public Object pegarElemento(int posicao) {
    return listaElementos.get(posicao);
}

public boolean pertence(Object elemento) {
    return listaElementos.contains(elemento);
}

public int pegaPosicao(Object elemento) {
    return (Integer) listaElementos.indexOf(elemento);
}

public int tamanho() {
    return listaElementos.size();
}
```

Para detectar se um objeto pertence ou não a um ArrayList, java simplesmente faz uso da comparação de igualdade já existente em cada objeto, como mostra o código-fonte da classe ArrayList:

```
public int indexOf(Object o) {
    if (o == null) {
        for (int i = 0; i < size; i++)
            if (elementData[i]==null)
                return i;
    } else {
        for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
                return i;
```

```
    }
    return -1;
}
```

Listas devem ser usadas quando a ordem dos elementos é importante. Existem três formas distintas de iterar sobre (processar) tais coleções:

```
// Alternativa 1: For-each
private ArrayList elementos = new ArrayList();

public int somarElementos() {
    int soma = 0;
    for (Object o : elementos) {
        Integer i = (Integer) o;
        soma += i;
    }
    return soma;
}
```

```
// Alternativa 2: Utilizando um objeto Iterator
private ArrayList elementos = new ArrayList();

public int somarElementos() {
    int soma = 0;
    Iterator itr = elementos.iterator();
    while (itr.hasNext()) {
        Integer i = (Integer) itr.next();
        soma += i;
    }
    return soma
}
```

```
// Alternativa 3: Utilizando o índice
private ArrayList elementos = new ArrayList();

public int somarElementos() {
    int soma = 0;
    for (int i = 0; i < elementos.size(); i++) {
        Integer i = (Integer) elementos.get(i);
        soma += i;
    }
    return soma;
}
```

O iterável, apesar de ser mais complexo de ser usado, é útil especialmente nas situações em que é preciso remover um elemento da lista (pois o método `itr.remove()` não afetará o elemento a ser processado da lista).

Importante! GENERICS

Nos exemplos acima, o ArrayList manipulou a classe Object (ou seja, a classe mais básica de Java). Como todos os objetos de Java são do tipo Object, é possível inserir

qualquer objeto nessa lista e recuperar tais objetos realizando uma operação de conversão (cast).

Entretanto, para facilitar a manipulação de objetos compostos (como as listas, conjuntos e mapas), Java criou uma estrutura chamada GENERICS.

Generics é um artifício da linguagem Java em que o código feito para classes de composição são feitos de forma genérica, mas especializadas durante a sua codificação. Por exemplo, para criar um ArrayList que apenas opere com Integer, devemos usar a notação descrita a seguir:

```
private ArrayList<Integer> elementos = new ArrayList<>();

public int somarElementos() {
    int soma = 0;
    for (Integer i : elementos) {
        soma += i;
    }
    return soma;
}
```

A importância do uso de *generics* é que ele permite evitar erros de tipagem. Uma classe com *generics* só vai permitir a inserção, captura e teste de objetos do tipo indicado no *generics*.

2. Conjunto

Um conjunto representa uma composição de elementos sem ordem e que evita a repetição. Ao contrário de listas, um conjunto é extremamente rápido para checar a relação de pertencimento. Vamos ver abaixo um exemplo de uso do conjunto básico de Java, o java.util.HashSet e já aplicando um exemplo de *generics*.

```
private HashSet<String> palavras = new HashSet<>();

public boolean adicionarElemento(String elemento) {
    return palavras.add(elemento);
}

public boolean pertence(String elemento) {
    return palavras.contains(elemento);
}

public boolean remove(String elemento) {
    return palavras.remove(elemento);
}

public int tamanho() {
    return palavras.size();
}
```

O conjunto é chamado de **HASH**SET pois faz uso da informação de hashcode para escolher onde e como armazenar um elemento dentro do conjunto. Por esse motivo, é extremamente importante que ao implementar um equals próprio para uma classe, o hashcode também seja implementado. Via de regra, é importante garantir que:

Se o1.equals(o2), então o1.hashcode() tem o mesmo valor de o2.hashcode()

Observe que dois objetos podem ter o mesmo hashcode e ser diferentes, no entanto, todo objeto igual deve ter o mesmo hashcode.

Dicas - Como funciona um hashset por baixo dos panos?

Imagine que você tem N posições na memória e quer armazenar um número X entre 0 e N. Uma maneira bem fácil de armazenar esse número, seria colocá-lo na posição X da memória. Assim, para checar se esse número está ou não na memória, basta olhar se a memória na posição X está ocupada.

Ou seja:

1. Pegue o número X, coloque o bit 1 na posição X da memória
2. Se a posição X da memória está ocupada com o bit 1, este número foi armazenado.

Agora imagine que você não quer armazenar números, mas objetos. O mesmo princípio pode ser usado para armazenar tal objeto:

1. Gere um número X entre 0 e N que representa o objeto O
2. Coloque uma referência ao objeto O na posição X da memória

Basicamente, o que o hashcode faz é gerar esse número que representa o objeto O. Entretanto, vários objetos diferentes podem ter o mesmo hashcode. Para resolver o problema de conflito, além do hashcode, o java usa também o equals para verificar se o objeto O que está na memória é igual ao objeto que está sendo passado como parâmetro do *contains*.

Os conjuntos podem ser processados da mesma forma que listas através do for-each ou de objetos Iterators. Um HashSet deve ser usado quando deseja-se armazenar elementos, ignorando a ordem que são inseridos e descartando repetições.

Existem duas formas de processar conjuntos. A primeira alternativa é através de um for-each e a segunda através de um iterator. Veja os exemplos de uso abaixo.

```
// Alternativa 1: For-each
private HashSet<String> palavras = new HashSet<>();

public int contarPalavrasComecandoEmVogais() {
    int conta = 0;
    for (String palavra : palavras) {
        if (palavra.startswith(“a”) || palavra.startswith(“e”) ||
palavra.startswith(“i”) || palavra.startswith(“o”) || palavra.startswith(“u”)) {
            conta += 1;
        }
    }
    return conta;
}
```

```
// Alternativa 2: Utilizando um objeto Iterable
private HashSet<String> palavras = new HashSet<>();

public int contarPalavrasComecandoEmVogais() {
    int conta = 0;
    Iterator<String> itr = palavras.iterator();
    while (itr.hasNext()) {
```

```
        String palavra = itr.next();
        if (palavra.startswith(“a”) || palavra.startswith(“e”) ||
palavra.startswith(“i”) || palavra.startswith(“o”) || palavra.startswith(“u”)) {
            conta += 1;
        }
    }
    return conta;
}
```

3. Mapa

Um mapa é uma estrutura de dados que associa elementos (chaves) a outros elementos (valores). Por exemplo, é possível associar uma string (como matrícula) a um objeto (como aluno). Veja o código abaixo que faz uso da classe mapa básica, o java.util.HashMap:

```
private HashMap<String, Aluno> mapaMatriculaAlunos = new HashMap<>();

public Aluno adicionaAluno(String matricula, Aluno aluno) {
    return this.mapaMatriculaAlunos.put(matricula, aluno);
    // retorna o aluno anteriormente associado a essa matricula, ou nulo se não existia
tal aluno
}

public boolean existeAluno(String matricula) {
    return this.mapaMatriculaAlunos.containsKey(matricula);
}

public boolean existeAluno(Aluno aluno) {
    return this.mapaMatriculaAlunos.containsValue(aluno);
}

public Aluno recuperaAluno(String matricula) {
    return this.mapaMatriculaAlunos.get(matricula);
}

public Aluno remove(String matricula) {
    return this.mapaMatriculaAlunos.remove(matricula);
}

public int numeroDeAlunos() {
    return this.mapaMatriculaAlunos.size();
}
```

Comumente, para processar um mapa existem três formas diferentes de fazer isto, dependendo do seu objetivo.

```
// Alternativa 1: Pelas chaves
private HashMap<String, Aluno> mapaMatriculaAlunos = new HashMap<>();

public void alterarTurma(String turma) {
    for (String matricula : this.mapaMatriculaAlunos.keySet()) {
        Aluno aluno = this.mapaMatriculaAlunos.get(matricula);
```

```
        aluno.setTurma(turma);
    }
}
```

```
// Alternativa 2: Pelos valores
private HashMap<String, Aluno> mapaMatriculaAlunos = new HashMap<>();

public void alterarTurma(String turma) {
    for (Aluno aluno : this.mapaMatriculaAlunos.values()) {
        aluno.setTurma(turma);
    }
}
```

```
// Alternativa 3: Pelas chaves e valores
private HashMap<String, Aluno> mapaMatriculaAlunos = new HashMap<>();

public void alterarTurma(String turma) {
    for (Entry<String, Aluno> entry : this.mapaMatriculaAlunos.entrySet()) {
        Aluno aluno = entry.getValue();
        aluno.setTurma(turma);
    }
}
```

Para o objetivo acima (alterar a turma de todos os alunos) o uso do processamento dos valores (alternativa 2) teria sido a melhor alternativa.

Construindo o Sistema - Mr. Bet

Você e outros colegas que estão cursando LP2 resolveram montar uma startup para produzir uma *engine* para a realização de apostas. A ideia é criar a base do sistema e vendê-lo para uma outra empresa maior que vai especializá-lo, posteriormente. Nesse laboratório, iremos construir essa base do Mr.Bet.

Como passo inicial, considere que será necessário cadastrar e consultar times no Mr.Bet. Cada time tem um código identificador, nome e seu mascote. O identificador do time é único e do tipo String. Ele é formado por: *3 digitos_estado-do-time,* exemplo - "250_PB". Esse código será útil para realizar consultas pelos times. Não deve ser possível alterar os dados dos times depois que eles são cadastrados.

Além disso, temos os campeonatos que são os confrontos entre os times. Eles são realizados com um conjunto predefinido de N participantes. Os campeonatos são identificados unicamente por seu nome, ignorando a diferença entre letras maiúsculas e minúsculas, (String) exemplo - "Campeonato Paraibano 2023". Certamente, um time bom pode participar de mais de um campeonato ao longo de um ano. Deve ser possível adicionar times aos campeonatos, respeitando o seu limite de participantes.

As apostas do Mr.Bet são simplificadas. Os palpites dos usuários envolvem apenas a colocação que um time terá em determinado campeonato e, obviamente, o valor depositado para aquela aposta. O sistema emite um alerta caso a colocação informada no palpite seja maior que a quantidade de times participantes naquele campeonato.

Para exercitar o Mr.Bet, deve ser criada uma aplicação Java que exibe um Menu como aparece a seguir:

```
(M)Minha inclusão de times
(R)Recuperar time
(.)Adicionar campeonato
(B)Bora incluir time em campeonato e Verificar se time está em
campeonato
(E)Exibir campeonatos que o time participa
(T)Tentar a sorte e status
(!)Já pode fechar o programa!

Opção>
```

Se ao longo da execução do sistema o usuário passar uma entrada inválida (nula ou vazia), deve ser lançada uma exceção e encerrado. Após cada ação de menu concluída pelo sistema, o menu deve ser novamente impresso para que uma opção possa ser novamente selecionada.

Leia com atenção e realize o que se pede em cada uma das etapas desse laboratório que foram detalhadas a seguir.

1. Incluir Times

Para incluir times, deve-se selecionar a opção "M". O sistema deve pedir informações como: código do time, nome e mascote. Se a inclusão for bem sucedida, deve ser exibida uma mensagem de sucesso, como mostramos abaixo:

```
Código: 250_PB
Nome: Nacional de Patos
Mascote: Canário
INCLUSÃO REALIZADA!
```

Não é possível incluir um time com um código já existente no cadastro. Em um caso como esse, em que o código do time que está se tentando cadastrar já exista, deve ser exibido uma mensagem como a mostrada a seguir.

```
Código: 250_PB
Nome: Sport Lagoa Seca
Mascote: Carneiro
TIME JÁ EXISTE!
```

2. Recuperar Times

Para recuperar times do cadastro, deve-se selecionar a opção "R". O sistema vai recuperar os times a partir do código do time digitado pelo usuário. Caso o time exista, uma saída como a mostrada a seguir deve ser apresentada. Caso contrário, deve ser informado que o time não existe no cadastro.

```
Código: 250_PB
[250_PB] Nacional de Patos / Canário
```

Exemplo de saída do sistema, no fluxo em que o time não existe no cadastro:

```
Código: 152_SC
TIME NÃO EXISTE!
```

3. Adicionar Campeonato

É nos campeonatos onde ocorrem os confrontos entre os times. Para adicionar um campeonato ao sistema é preciso informar o nome do campeonato e o número de participantes. O nome do campeonato é o seu identificador, ou seja, não deve ser admitido o cadastro de campeonatos com o mesmo nome. Observe que as maiúsculas e minúsculas são indistintas para esta verificação. Veja os exemplos que seguem: (1) de fluxo em que a adição é bem sucedida e (2) de fluxo em que a adição não dá certo.

(1)

```
Campeonato: Brasileirão Série A 2023
Participantes: 20
CAMPEONATO ADICIONADO!
```

(2)

```
Campeonato: brasileirão série A 2023
Participantes: 30
CAMPEONATO JÁ EXISTE!
```

4. Bora Incluir Time em Campeonato

A seleção da opção "B" leva para um submenu com duas opções: (I) Incluir time em campeonato e (V) Verificar se time faz parte de um campeonato. Veja o como é feita essa interação com o usuário a seguir:

```
Opção> B
(I) Incluir time em campeonato ou (V) Verificar se time está em
campeonato? I
```

Caso o usuário selecione "I", que se refere à opção de incluir time em um dado campeonato, o sistema segue solicitando as informações necessárias para realizar essa ação. É solicitado o nome do campeonato e o código do time que se deseja incluir. Deve ser observado se a quantidade de participantes do campeonato já atingiu o seu limite e se os times e campeonatos informados estão cadastrados no sistema.

Observe a seguir, os fluxos de sucesso - quando a inclusão é realizada com sucesso e os fluxos alternativos quando, por algum motivo, não se consegue realizar a inclusão.

```
Código: 250_PB
Campeonato: Campeonato Paraibano 2023
TIME INCLUÍDO NO CAMPEONATO!
```

(1) Time não cadastrado no sistema

```
Código: 152_SC
```

```
Campeonato: Basileirão série A 2023
TIME NÃO EXISTE!
```

(2) Campeonato não cadastrado no sistema

```
Código: 250_PB
Campeonato: Copa do Nordeste de Futebol de 2023
CAMPEONATO NÃO EXISTE!
```

(3) Número de participantes de um campeonato excedido

```
Código: 252_PB
Campeonato: Campeonato Paraibano 2023
TODOS OS TIMES DESSE CAMPEONATO JÁ FORAM INCLUÍDOS!
```

5. Verificar se o time está no Campeonato

Se o usuário selecionar no submenu a opção (V) "Verificar se time faz parte de um campeonato", vejamos o que vai acontecer:

```
Opção> B
(I) Incluir time em campeonato ou (V) Verificar se time está em
campeonato? V
```

O sistema vai solicitar as informações referentes ao identificador do time e ao nome do campeonato para checar se o referido time está no campeonato. Veja os exemplos de fluxos de sucesso e alternativos a seguir.

```
Código: 250_PB
Campeonato: Campeonato Paraibano 2023
O TIME ESTÁ NO CAMPEONATO!
```

ou

```
Código: 250_PB
Campeonato: Basileirão série A 2023
O TIME NÃO ESTÁ NO CAMPEONATO!
```

(1) Time não está cadastrado no sistema

```
Código: 152_SC
Campeonato: Campeonato Paraibano 2023
O TIME NÃO EXISTE!
```

(2) Campeonato não foi cadastrado no sistema

```
Código: 250_PB
Campeonato: Campeonato Catarinense 2023
O CAMPEONATO NÃO EXISTE!
```

6. Exibir campeonatos que o time participa

Para exibir os campeonatos dos quais um time participa, o usuário deve selecionar a opção "E". O sistema informa o nome do campeonato, bem como, a quantidade de times cadastrados e a quantidade de participantes neste campeonato exemplo: "1/20". Significa que apenas um time, dos 20 participantes possíveis, foi cadastrado.
A listagem dos campeonatos que o time participa deve ser exibida da seguinte forma:

```
Time: 250_PB

Campeonatos do Nacional de Patos:
* Campeonato Paraibano 2023 - 10/14
* Brasileirão Série A 2023 - 1/20
```

Para o fluxo alternativo, "Time não cadastrado no sistema", use a mesma saída especificada nas seções anteriores.

Importante: Observe que o nome do campeonato que será exibido deve ser o mesmo informado quando do cadastro do campeonato. Inclusive considerando as maiúsculas e minúsculas.

7. Tentar a sorte e status

Ao selecionar a opção "T", o usuário é levado novamente para um submenu com duas opções: (A) Apostar em time e (S) Status das apostas. A interação com o usuário neste submenu, ocorre de forma análoga a que já vimos anteriormente:

```
Opção> T
(A)Apostar ou (S)Status das Apostas? A
```

Ao selecionar a opção "A", o usuário deve inserir os dados para a criação de uma aposta Mr.Bet. Aposta-se, basicamente, qual será a colocação de um time em determinado campeonato. Para ser criada uma aposta, é necessário o código identificador do time, o nome do campeonato, a colocação (int) e o valor da aposta (double). Veja exemplos de interação como usuário:

```
Código: 250_PB
Campeonato: Campeonato Paraibano 2023
Colocação: 2
APOSTA REGISTRADA!
```

(1) Colocação excede número de participantes do campeonato

```
Código: 250_PB
Campeonato: Basileirão série A 2023
Colocação: 22
APOSTA NÃO REGISTRADA!
```

Trate os fluxos alternativos (1) em que o time não é cadastrado no sistema e (2) em que o campeonato não está cadastrado no sistema como sugerido nas seções anteriores.

8. Status das Apostas

Caso o usuário faça a opção pela alternativa "S" - Status das apostas, o sistema vai apresentar uma lista das apostas realizadas na ordem em que elas foram criadas. A interação com o usuário neste caso, ocorre da seguinte forma:

```
Opção> T
(A)Apostar ou (S)Status das Apostas? S
```

A apresentação das apostas, segue a ordem em que elas foram incluídas no sistema e a seguinte formato:

```
#numero da aposta
<time>
<campeonato>
<colocacao>/<participantes>
<valor>
```

Veja um exemplo de saída:

```
Apostas:

1. [250_PB] Nacional de Patos / Canário
Campeonato Paraibano 2023
2/14
R$ 50.00

2. [252_PB] Sport Lagoa Seca / Carneiro
Nordestão 2023
1/20
R$ 250.00
```

9. Já pode fechar o programa

Ao selecionar a opção "!", o usuário informa que quer fechar o programa. Veja esta interação com o usuário no exemplo:

```
Opção> !

Por hoje é só pessoal!
```

Bônus. Histórico

Adicione ao menu uma opção de "(H) Histórico". Esta opção vai apresentar dados sobre o histórico de interação dos usuários com o sistema. Ela serve para apresentar um registro estatístico das informações acumuladas pelo sistema, até o momento.
Apresente as seguintes informações históricas:

- O time que foi incluído em mais campeonatos até o momento (em caso de empate, todos os times empatados devem ser apresentados) e a quantidade de campeonatos que eles foram incluídos;
- O time ou os times que não participaram em nenhum campeonato até o momento.
- A quantidade de vezes em que os times apareceram na colocação de primeiro lugar nas apostas registradas no sistema. Apresente um time por linha, exemplo:

```
Participação mais frequente em campeonatos
[002_RJ] Clube de Regatas do Flamengo / Urubu

Ainda não participou de campeonato
[105_PB] Sociedade Recreativa de Monteiro (SOCREMO) / Gavião

Popularidade em apostas
Sport Lagoa Seca / 1
Clube de Regatas do Flamengo / 1250
```

Observação: Lembre-se de fazer testes para o bônus!

Testando o sistema Mr.Bet

Já sabemos que testar o software é uma forma de garantir a sua corretude, ou seja, que ele funciona de acordo com a especificação dada. Além disso, um bom teste é aquele que:

- É capaz de encontrar erros no programa;
- É simples;
- Não é redundante.

O sistema Mr.Bet oferece as seguintes funcionalidades: incluir times, exibir times, adicionar campeonato, incluir time em campeonato, verificar se um time está em um campeonato, realizar apostas, exibir o status das apostas no sistema e mostrar os campeonatos que um time faz parte. **Para testar esse sistema precisamos de um plano de testes que envolve casos de testes, as entradas a serem usadas em cada caso e as saídas esperadas.** A seguir vamos discutir um **plano de testes** para o sistema envolvendo as **funcionalidades sobre campeonato**; esse plano de testes deve ser implementado no JUnit.

Para prosseguirmos, vamos considerar que existe uma **classe SistemaMrBet que gerencia todos os times e campeonatos cadastrados e as operações sobre os mesmos**. Então, as coleções de times e campeonatos serão atributos dessa classe e você pode pensar que existirá uma outra classe *Main* que irá conter o método *main* e operar sobre um objeto do tipo SistemaMrBet para realizar as funcionalidades esperadas para esse sistema a partir da interação com o usuário.

Considere a existência de 4 times no sistema:

- `[250_PB] Nacional de Patos / Canário`
- `[252_PB] Sport Lagoa Seca / Carneiro`
- `[002_RJ] Clube de Regatas do Flamengo / Urubu`
- `[105_PB] Sociedade Recreativa de Monteiro (SOCREMO) / Gavião`

Caso 1: Cadastrar um campeonato sem restrição

- Cadastrar o campeonato com nome "Brasileirão série A 2023"
    - Saída esperada: `INCLUSÃO REALIZADA!`

Caso 2: Cadastrar um campeonato com nome já existente

- Cadastrar o campeonato com nome "Brasileirão série A 2023"
    - Saída esperada: `TIME JÁ EXISTE!`

Caso 3: Incluir time em um campeonato

- Cadastrar o campeonato com nome "Brasileirão série A 2023"
- Incluir time com o código `250_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`
- Incluir o time `252_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`

Caso 4: Incluir time em um campeonato em que ele já foi incluído

- Cadastrar o campeonato com nome "Brasileirão série A 2023"
- Incluir time com o código `250_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`
- Incluir o time `252_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`
- Incluir o time `252_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`
    - O sistema informa que o time foi incluído no campeonato, mas não insere ele novamente.

Caso 5: Incluir time em um campeonato quando o time não foi cadastrado.

- Cadastrar o campeonato com nome "Brasileirão série A 2023"
- Incluir time com o código `005_PB` no campeonato "Basileirão série A 2023"
    - Saída esperada: `O TIME NÃO EXISTE!`

Caso 6: Incluir um time em um campeonato quando o campeonato não existe.

- Incluir o time `252_PB` no campeonato "Brasileirão série D 2023"
    - Saída esperada: `O CAMPEONATO NÃO EXISTE!`

Caso 7: Incluir time em um campeonato excedendo a quantidade de participantes.

- Cadastrar o campeonato com nome "Brasileirão série A 2023" informando número de participantes (1 time)
- Incluir o time `252_PB` no campeonato "Brasileirão série A 2023"
    - Saída esperada: `TIME INCLUÍDO NO CAMPEONATO!`
- Incluir o time `250_PB` no campeonato "Brasileirão série A 2023"
    - Saída esperada: `TODOS OS TIMES DESSE CAMPEONATO JÁ FORAM INCLUÍDOS!`

Caso 8: Verificar se um time pertence a um campeonato

- Cadastrar um campeonato com nome "Copa do Nordeste 2023"
- Incluir o time `250_PB` no campeonato "Copa do Nordeste 2023"
- Verificar se o time `250_PB` está incluído no campeonato "Copa do Nordeste 2023"
    - É verdade que o time está incluído no campeonato.
- Verificar se o time `252_PB` está incluído no campeonato "Copa do Nordeste 2023"
    - É falso que o time está incluído no campeonato.

Caso 9: Verificar se um time pertence a um campeonato que não foi cadastrado

- Verificar se o time `252_PB` está incluído no campeonato "Brasileirão série D 2023"
    - Saída esperada: `O CAMPEONATO NÃO EXISTE!`

Caso 10: Verificar se um time não cadastrado pertence a um campeonato.

- Cadastrar um campeonato com nome "Copa do Nordeste 2023".
- Incluir time com o código `005_PB` no campeonato "Copa do Nordeste 2023".
    - Saída esperada: `O TIME NÃO EXISTE!`

ATENÇÃO: Os casos de teste apresentados acima serão convertidos em testes JUnit para a classe de sistema sugerida. Além disso, remetem para apenas algumas funcionalidades dessa classe, ou seja, existem outras funcionalidades a serem testadas e você deve gerar esses testes no JUnit, bem como para as demais classes implementadas.

Entrega

Faça o programa **Sistema Mr.Bet** que atenda as funcionalidades descritas anteriormente. Lembre-se de lançar as exceções apropriadas e considerar todos os fluxos alternativos de execução (ex.: o que acontece quando tenta-se listar um campeonato que não existe, etc.).

É importante que o código esteja devidamente documentado (com javadoc).

Ainda, você deve entregar um programa com testes para as classes com lógica testável (isso exclui a classe do main(...)). Tente fazer um programa que possa ser devidamente testado e que explore as condições limite; observe os casos de teste definidos para guiar a implementação dos testes com JUnit.

Para a entrega, faça um **zip** da pasta do seu projeto. Coloque o nome do projeto para: LAB4_SEU_NOME e o nome do zip para LAB4_SEU_NOME.ZIP. Exemplo de projeto: LAB4_ELIANE_ARAUJO.ZIP. Este zip deve ser submetido pelo Canvas.

Seu programa será avaliado pela corretude e, principalmente, pelo DESIGN do sistema. É importante:

- Usar nomes adequados de variáveis, classes, métodos e parâmetros.
- Fazer um design simples, legível e que funciona. É importante saber, apenas olhando o nome das classes e o nome dos métodos existentes, identificar quem faz o que no código.