

# Utilização de Coleções em Java

## Exercício

**Vinicius Takeo Friedrich Kuwaki**

Universidade do Estado de Santa Catarina

# Seções

Exemplo

Resolução

Métodos úteis

Exercício

## Exemplo

Crie uma função que armazene em um Map a tabuada do número 1 ao número 10 e exiba no console a tabuada do número  $n$  armazenada em um List.

# Seções

Exemplo

Resolução

Métodos úteis

Exercício

- Vamos utilizar um List no nosso exercício para armazenar os valores de 1 a 10 da tabuada de um número  $n$ ;
- Instanciaremos esse List como um ArrayList, pois ele tem melhor desempenho nas consultas aos dados armazenados;
- Vamos criar uma classe chamada **Principal** e importar a interface List e a classe ArrayList:

```
import java.util.List;  
import java.util.ArrayList;
```

- Vamos utilizar também um Map em nosso exercício;
- Como o próprio nome já diz, um Map é um mapa que associa um objeto a uma chave única de acesso/busca;
- Logo, os Maps são compostos de pares ordenados  $\langle K, V \rangle$ , onde  $K$  corresponde a chave (key) e  $V$  corresponde ao valor (value) associado a respectiva chave;
- Aqui, o Map será utilizado para armazenar as tabuadas de 1 a 10 dos números 1 a 10;
- Assim como o List, o Map é uma interface que necessita ser instaciado por uma classe que o implementa;
- Nesse exercício vamos instanciar o Map como um HashMap;

- Assim como para o List e ArrayList, vamos importar a interface Map e a classe HashMap:

```
import java.util.Map;  
import java.util.HashMap;
```

## Resolução

- Importadas as bibliotecas que utilizaremos, vamos construir a função que retorna a List contendo a tabuada de um número  $n$ ; //
- Faremos tudo dentro da classe Principal que também contém o método **main()**;
- Nosso método, que será estática, recebe como parâmetro o número inteiro  $n$  do qual queremos saber a tabuada e retorna um List de inteiros com os valores da tabuada de 1 a 10;
- Coleções utilizam apenas objetos, como inteiros são tipos primitivos, utilizaremos seu Wrapper **Integer**:

```
public static List<Integer> tabuada(int n){  
    ...  
}
```



- Vamos primeiro instanciar um objeto do tipo ArrayList;
- Através de polimorfismo, ele será um List:

```
public static List<Integer> tabuada(int n){  
    List<Integer> tabuada = new ArrayList<Integer>();  
}
```

- Vamos iterar sobre uma variável  $i$  de 1 até 10 e adicionar no List a multiplicação de  $i$  por  $n$ ;
- Para adicionar um valor no List, utilizamos o método **add()**:

```
public static List<Integer> tabuada(int n){  
    List<Integer> tabuada = new ArrayList<Integer>();  
  
    for (int i = 1; i <= 10; i++) {  
        tabuada.add(i * n);  
    }  
  
    return tabuada;  
}
```

## Resolução

- Agora, vamos declarar um método **main()**;
- Dentro do método **main()** vamos instanciar o **HashMap**;
- Novamente, através de polimorfismo ele será um **Map**;
- A chave será um **Integer** e o valor será uma lista de inteiros, contendo a tabuada da respectiva chave;
- Logo o valor será: **List<Integer>**;

```
public static void main(String [] args) {  
    Map<Integer , List<Integer>> todasAsTabuadas = new HashMap<Integer , List<Integer>>();  
}
```

- Diferente do List que possui um método **add()**, o Map possui um método **put(chave, valor)** para inserir os pares  $\langle K, V \rangle$ ;
- Vamos adicionar dentro do nosso Map a tabuada de 1 a 10;
- Para isso, utilizaremos o método **put()** e a nossa função **tabuada()**, iterando um valor  $i$  de 1 a 10:

```
for (int i = 1; i <= 10; i++) {  
    todasAsTabuadas.put(i, tabuada(i));  
}
```

- Agora vamos exibir os valores das tabuadas armazenadas no Map;
- O Map possui um método chamando **forEach()** que recebe um par ordenado (K, V) e um trecho de código;
- Esse trecho de código será executado para cada par ordenado (K, V) do Map;
- Vamos primeiro chamar o método e depois construir o código de manipulação do Map:

```
todasAsTabuadas.forEach(  
    (chave, tabuada) -> {  
        // Código a ser executado para cada par ordenado (chave, tabuada)  
    }  
);
```

- Para cada par ordenado vamos imprimir a chave e os valores da tabuada associados a essa chave e que estão armazenados no List;
- Vamos começar pela chave, que em nosso exemplo chamamos ela de “chave”:

```
todasAsTabudas.forEach(  
    (chave, tabuada) -> {  
        System.out.print("Tabuada de " + chave + ": ");  
    }  
);
```

- Agora vamos percorrer a lista que chamamos de “tabuada”, imprimindo o valor de cada posição da lista:

```
todasAsTabudas.forEach(  
    (chave, tabuada) -> {  
        System.out.print("Tabuada de " + chave + ": ");  
        for (int x : tabuada) {  
            System.out.print(x + " ");  
        }  
    }  
);
```

- Finalizado o **forEach()** vamos executar a **main()**:

```
Tabuada de 1: 1 2 3 4 5 6 7 8 9 10
Tabuada de 2: 2 4 6 8 10 12 14 16 18 20
Tabuada de 3: 3 6 9 12 15 18 21 24 27 30
Tabuada de 4: 4 8 12 16 20 24 28 32 36 40
Tabuada de 5: 5 10 15 20 25 30 35 40 45 50
Tabuada de 6: 6 12 18 24 30 36 42 48 54 60
Tabuada de 7: 7 14 21 28 35 42 49 56 63 70
Tabuada de 8: 8 16 24 32 40 48 56 64 72 80
Tabuada de 9: 9 18 27 36 45 54 63 72 81 90
Tabuada de 10: 10 20 30 40 50 60 70 80 90 100
```



# Seções

Exemplo

Resolução

Métodos úteis

Exercício

## Métodos úteis - List

- **int size():** retorna o numero de objetos na lista;
- **boolean add(T objeto):** adiciona um objeto ao final da lista;
- **boolean add(int posicao, T objeto):** adiciona um objeto em um posição específica da lista;
- **boolean remove(T objeto):** remove um objeto da lista. A classe ao qual pertence o objeto deve implementar o método equals();
- **boolean remove(int posicao):** remove um objeto em uma posição específica da lista;
- **T get(int posicao):** retorna o objeto da posição x da lista;
- **boolean contains(T objeto):** retorna true caso o objeto esteja dentro da lista e false caso contrário. A classe ao qual pertence o objeto deve implementar o método equals();

- **int size():** retorna o numero de objetos na lista;
- **boolean containsKey(K chave):** retorna true caso essa chave exista dentro do map e false caso contrário. A classe ao qual pertence o objeto deve implementar o método equals();
- **boolean remove(T objeto):** remove um objeto (K ou V) do map. A classe ao qual pertence o objeto deve implementar o método equals();
- **V get(K chave):** retorna o valor associado a chave K;
- **boolean put(K chave, V valor ):** adiciona um par ordenado (K,V) no Map;

# Seções

Exemplo

Resolução

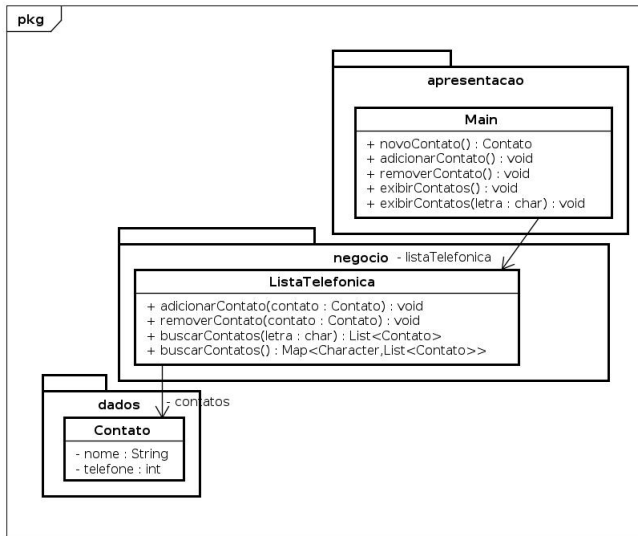
Métodos úteis

Exercício

## Exercício

Utilizando o framework de coleções, implemente o diagrama de classes a seguir:

# Exercício



## Exercício

- A classe ListaTelefonica deve manter uma lista de contatos telefonicos;
- O método **exibirContatos()** da classe Main deve exibir todos os contatos, ordenados de acordo com a primeira letra do nome;

- Por exemplo:

A:

- André: 983748574

- Ana: 97364985

B:

- Bianca: 947346543

C:

D:

- Daniel: 973648374

E:

F:

## Exercício

G:

H:


...

Y:

Z:

- Os contatos não necessariamente precisam ser exibidos em ordem alfabética, apenas devem estar agrupados de acordo com a inicial do seu primeiro nome.
- Utilize HashMap para indexar o contatos pela inicial.
- O método **removerContato()** da classe Main deve requisitar ao usuário a inicial do contato que ele deseja remover, após o usuário entrar com ela, deve ser exibido uma lista contendo todos os contatos que possuem essa inicial. O usuário deverá então escolher um.



 KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <<https://github.com/takeofriedrich/slidesUdescLattex>>. Acesso em: 24 jan. 2020.

Duvidas:  
Vinicius Takeo Friedrich Kuwaki  
[vinicius.kuwaki@edu.udesc.br](mailto:vinicius.kuwaki@edu.udesc.br)  
[github.com/takeofriedrich](https://github.com/takeofriedrich)