Prática 9

Tópicos

- Java Collections

Exercício 9.1

A classe *java.util.ArrayList<E>* é uma implementação de um vetor dinâmico fornecida pelo JavaSE. Consulte a documentação desta classe e execute o seguinte programa.
(https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html)

```
import java.util.ArrayList;
import java.util.Collections;

public class ALDemo {

   public static void main(String[] args) {
      ArrayList<Integer> c1 = new ArrayList<>();
      for (int i = 10; i <= 100; i+=10)
         c1.add(i);
      System.out.println("Size: " + c1.size());
      for (int i = 0; i < c1.size(); i++)
         System.out.println("Elemento: " + c1.get(i));

      ArrayList<String> c2 = new ArrayList<>();
      c2.add("Vento");
      c2.add("Calor");
      c2.add("Frio");
      c2.add("Chuva");
      System.out.println(c2);
      Collections.sort(c2);
      System.out.println(c2);
      c2.remove("Frio");
      c2.remove(0);
      System.out.println(c2);

   }
}
```

a) Modifique-o livremente para testar algumas das funções que estão disponíveis nesta classe (*add, contains, indexOf, lastIndexOf, set, subList*, ...).

b) Com base neste exemplo, crie uma nova coleção (c3) que use um HashSet, em vez de ArrayList, e que contenha elementos do tipo Pessoa (Exercício 6.1).

```
   Set<Pessoa> c3 = new HashSet<>();
   ...
```

Insira 5 elementos distintos e use um iterador para listar todos os elementos no écran.
Verifique a ordem da listagem relativamente à ordem de inserção.
Teste a inserção de elementos repetidos (que não podem existir num Set).

c) Crie uma nova coleção (c4) que use um TreeSet de datas (classe *Date*, Exercício 7.2).

```
   Set<Date> c4 = new TreeSet<>();
   ...
```

Insira 5 elementos distintos e verifique a ordem da listagem relativamente à ordem de inserção.
Teste com as diferentes representações de data criadas na aula 7.

Exercício 9.2

Usando como base o código seguinte (*CollectionTester.java*) compare o desempenho de algumas das coleções em Java, tais como *ArrayList*, *LinkedList*, *HashSet* e *TreeSet*.

```
public class CollectionTester {

   public static void main(String[] args) {
      int DIM = 5000;

      Collection<Integer> col = new ArrayList<>();
      checkPerformance(col, DIM);
   }

   private static void checkPerformance(Collection<Integer> col, int DIM) {
      double start, stop, delta;
      // Add
      start = System.nanoTime(); // clock snapshot before
      for(int i=0; i<DIM; i++ )
         col.add( i );
      stop = System.nanoTime();  // clock snapshot after
      delta = (stop-start)/1e6; // convert to milliseconds
      System.out.println(col.size()+ ": Add to " +
         col.getClass().getSimpleName() +" took " + delta + "ms");
      // Search
      start = System.nanoTime(); // clock snapshot before
      for(int i=0; i<DIM; i++ ) {
         int n = (int) (Math.random()*DIM);
         if (!col.contains(n))
            System.out.println("Not found???"+n);
      }
      stop = System.nanoTime();  // clock snapshot after
      delta = (stop-start)/1e6; // convert nanoseconds to milliseconds
      System.out.println(col.size()+ ": Search to " +
         col.getClass().getSimpleName() +" took " + delta + "ms");
      // Remove
      start = System.nanoTime(); // clock snapshot before
      Iterator<Integer> iterator = col.iterator();
      while (iterator.hasNext()) {
          iterator.next();
          iterator.remove();
      }
      stop = System.nanoTime();  // clock snapshot after
      delta = (stop-start)/1e6; // convert nanoseconds to milliseconds
      System.out.println(col.size() + ": Remove from "+
         col.getClass().getSimpleName() +" took " + delta + "ms");
   }
}
```

d) Adapte o programa de modo a medir os resultados para várias dimensões da coleção (por exemplo, criando uma tabela semelhante à seguinte). Analise os resultados comparando estruturas e operações.

Collection	1000	5000	10000	20000	40000	100000
ArrayList						
add	0,5	...				
search	11,5					
remove	1,2					
LinkedList						
...						

Sugestões: modifique o método checkPerformance de modo a devolver as 3 medições (add, search e remove) e retire todas as instruções println dentro deste método.

```
private static double[] checkPerformance(Collection<Integer> col, int DIM) {
...
```

Exercício 9.3

Escreva um programa que controle uma frota de aviões. Cada avião da frota é um objeto da classe *Plane*. Para o programa de gestão da frota de aviões faça uso de Java Collection, selecionando as estruturas de dados que lhe pareçam mais indicadas.

Todos os aviões devem incluir os seguintes atributos:

- Identificador único (String)
- Fabricante (String)
- Modelo (String)
- Ano de produção (int)
- Número máximo de passageiros (int)
- Velocidade máxima (int)

O programa deve incluir as seguintes classes:

- Plane: A classe principal que representa um objeto avião. Deverá ter todos os atributos apresentados acima, função toString() e os setters e getters apropriados.
- CommercialPlane: Uma subclasse do Plane que representa um avião comercial. Deverá ter um atributo adicional que identifica o número de tripulantes (int) e os setters e getters necessários. Também deverá fazer override da função toString() e da getPlaneType(), que retornará "Comercial".
- MilitaryPlane: Uma subclasse do Plane que representa aviões militares. Deverá conter um atributo adicional a indicar o número de munições (int), os setters e getters necessários, e fazer override à função toString() e getPlaneType(), que retornará "Militar".
- PlaneManager: Uma classe que faz a gestão da frota de aviões. Deverá conter uma estrutura de dados para guardar os aviões e os seguintes métodos:
    - addPlane(Plane plane): adiciona um novo avião à frota.
    - removePlane(String id): Remove um avião da frota com base no seu identificador único.
    - searchPlane(String id): Procura por um avião dentro da frota com base no identificador do avião. Retorna um objeto plane ou null se o avião não existir.
    - getCommercialPlanes(): Retorna uma lista com todos os aviões comerciais da frota.
    - getMilitaryPlanes(): Retorna uma lista com todos os aviões militares da frota.

- printAllPlanes(): Imprime a informação de todos os aviões da frota.
- getFastestPlane(): Retorna um objeto *Plane* do avião mais rápido da frota.

O programa também deve incluir um método principal chamado PlaneTester que tem um menu para conseguir controlar a informação da frota de aviões usando o PlaneManager. Deve conseguir executar as seguintes tarefas:
Criar uma instância do PlaneManager.

- Adicionar aviões tanto comerciais como militares à frota.
- Remover aviões da frota.
- Procurar um avião em específico.
- Imprimir a informação de todos os aviões da frota.
- Imprimir a lista de todos os aviões comerciais ou a lista de todos os aviões militares.
- Imprimir as informações do avião mais rápido da frota.