## LI3 - ÚLTIMAS NOTAS SOBRE O TP DE JAVA

1) Tenho verificado que muitos alunos iteram os Map<K, V>, sejam hashMaps ou treeMaps, usando a construção for(Entry<K, V> entry : map.entrySet()). Esta construção não é muito eficiente comparativamente com o uso de keySet() + get(), ou até de values(), tanto mais que implicará fazer o getKey() e o getValue().

Assim, parece razoável que se apresente a nova forma de em Java 8 iterar um Map<K, V>, e que consiste em usar o novo método forEach(*lambda*), que tem a forma geral:

```
map.forEach( (k,v) -> fazer qualquer coisa com k e/ou v );

// exemplo
map.forEach( (k,v) ->
               out.println("Chave: " + k + ": Valor: " + v));
```

2) As interfaces NavigableSet<E> e NavigableMap<K, V> podem ser muito úteis para a navegação em grandes coleções de dados. Em geral, a coleção é convertida no tipo navegável e, em seguida, criam-se e usam-se iteradores por ordem crescente, decrescente, etc. para navegar sobre os dados. Vejamos dois exemplos:

```
NavigableSet<String> navigableSet =
  new TreeSet<String>(Arrays.asList("X", "B", "A", "Z", "T"));

Iterator<String> iterator = navigableSet.descendingIterator();
out.println("Ordem descendente :");
   while(iterator.hasNext()) {
     out.println(iterator.next());
   }

iterator = navigableSet.iterator();
out.println("Ordem normal :");
while(iterator.hasNext()) {
   out.println(iterator.next());
}
```

Para Map<K, V> teremos algo semelhante mas com diferentes métodos:

```
NavigableMap<String,Integer> navigableMap =
  new TreeMap<String, Integer>();
....

out.printf("Tabela Inversa: %s%n", navigableMap.descendingMap());
out.printf("Descendente: %s%n", navigableMap.descendingKeySet());
out.printf("Primeiro Par: %s%n", navigableMap.firstEntry());
```

Muitos outros métodos estão disponíveis para estas navegações nos dados.

3) O novo "default method" sort(comparator) da interface List<E> pode ser muito útil para com grande facilidade se realizarem ordenações de listas. cf. o exemplo:

```
List<String> nomes =
    Arrays.asList("Pedro", "Ana", "Luisa", "Rita");
nomes.sort( (n1, n2) -> n1.compareTo(n2) );
out.println(nomes);

 List<String> nomes1 = new ArrayList<String>(nomes);
```

```
nomes1.sort( (n1, n2) -> (n1.compareTo(n2)) );
out.println(nomes);
// [Ana, Luisa, Pedro, Rita]
// [Ana, Luisa, Pedro, Rita]
```

F. Mário Martins
25/5/2016