Java - elements of generic programming (II)

Working environment setup

1. Download and unzip `lab05` source code
    1. Download `lab05.zip` from the course site (moodle)
    2. Unzip it (you get `lab05` directory)
    3. Move `lab05` to `programming-in-java` directory, i.e.,
        - `programming-in-java`
            - `lab00`
            - ...
            - `lab05 <--`
            - `gradle`
            - ...
2. [`IntelliJ`] Add `lab05` module to the `programming-in-java` project
    1. In the *Project* window click `settings.gradle` file to open it
    2. Modify its content to the following:

```
rootProject.name = 'programming-in-java'
include 'lab00'
...
include 'lab04'
include 'lab05'
```

    3. Save the file
    4. Click `Load Gradle Changes` (a small box in the top right corner)

1) Java Collections Framework - selected interfaces

Iterator, *ListIterator*, *Iterable*, *Collection*, *List*, *Set*/*SortedSet*, *Queue*/*Deque*, *Map*/*SortedMap*

Exercises

1. Familiarise yourself with The Java Tutorials > Collections
2. Familiarise yourself with the following interfaces:
    - Iterator
    - ListIterator
    - Iterable
    - Collection
    - List
    - Set
    - SortedSet
    - Queue
    - Deque
    - Map
    - SortedMap
3. Look briefly at the *Java Collections Framework* class diagram (e.g., here)

2) `List<E>` and its two implementations: `ArrayList<E>` , `LinkedList<E>`

Analyse the source code in package `lst05_01`

Exercises

1. Suppose `lList` is a LinkedList that contains a million int values. Which of the following two loops will run faster?

```
sum = 0;
for (int i = 0; i < lList.size(); i++)
    sum += lList.get(i);
```

```
sum = 0;
```

```
for (int e: lList)
    sum += e;
```

2. Explain the printout of the following code:

```
List<Integer> l1 = new ArrayList<>(List.of(1,2,3,4,5));
l1.remove(2);
System.out.println(l1);
```

3. Explain the printout of the following code:

```
List<Integer> l2 = new ArrayList<>(List.of(1,2,3,4));
for (int i = 0; i < l2.size(); i++)
    System.out.print(l2.remove(i));
```

4. Explain the difference between the following pieces of code:

```
Collection<Integer> l1 = new ArrayList<>(List.of(0, 1, 2));
for (int i = 0; i < 3; i++) {
    System.out.println(l1.remove(i));
}
System.out.println(l1);
```

```
List<Integer> l2 = new ArrayList<>(List.of(0, 1, 2));
for (int i = 0; i < 3; i++) {
    System.out.println(l2.remove(i));
}
System.out.println(l2);
```

3) Iterators, the *for-each* loop (aka. *enhanced for* loop), and *forEach* method

Analyse the source code in package `lst05_02`

Exercises

1. Familiarise yourself with the Iterator (design) pattern

2. Rewrite the following piece of code

```
List<Integer> lst = List.of(1,2,3,4,5);
for (int i = 0; i < lst.size(); i++) {
    System.out.println(lst.get(i));
}
```

using subsequently:

- iterator and the while-loop
- iterator and the for-loop
- enhnaced for-loop
- forEach method

4) `Queue<E>` , `Deque<E>` and their implementations: `PriorityQueue<E>` , and `ArrayDeque<E>`

Analyse the source code in package `lst05_03`

Exercises

1. Familiarise yourself with:

   - PriorityQueue
   - ArrayDeque

2. Explain the execution result of the following method

```
 private static void m() {
     Queue<Integer> pq = new PriorityQueue<>(List.of(6, 1, 5, 3,
4, 2));
     for (int e : pq)
         System.out.print(e + " ");

     System.out.println();

     while (!pq.isEmpty())
         System.out.print(pq.poll() + " ");
```

```
}
```

5) `java.lang.Comparable` and `java.util.Comparator`

Analyse the source code in package `lst05_04`

Exercises

1. Familiarise yourself with the following interfaces:
   - Comparable
   - Comparator
2. Check if the following code is correct:

```
static Object max(Object o1, Object o2) {
    if ((Comparable) o1.compareTo(o2) >= 0) {
        return o1;
    } else {
        return o2;
    }
}
```

   If not, fix the error.

3. Fill in the following code to create a Comparator for two strings in decreasing order of their length

```
private static void m() {
    String[] cities = {"Copenhagen", "Warsaw", "Budapest"};
    Arrays.sort(cities, ___);
    System.out.println(Arrays.toString(cities));
}
```

6) `Set<E>` and its implementations: `HashSet<E>` , `LinkedHashSet<E>` , `TreeSet<E>` , and `EnumSet<E extends Enum<E>>`

Analyse the source code in package `lst05_05`

Exercises

1. Explain the key differences between the four implementations of the Set interface
2. Explain how to compute the union, intersection, and difference of two sets, using just the methods of the Set interface and without using loops
3. Write a function that takes a TreeSet of strings and returns a new TreeSet with each string being transformed to uppercase
4. [optional] Compare the performance of methods `add` , `remove` , and `contains` for the four implementations of Set interface

7) `Map<K,V>` and its implementations: `HashMap<K,V>` , `LinkedHashMap<K,V>` , `TreeMap<K,V>` , and `EnumMap<K extends Enum<K>,V>`

Analyse the source code in package `lst05_06`

Exercises

1. Explain the key differences between the four implementations of the Map interface
2. Write a program that reads all words in a file and prints out how often each word occurred. Use a `TreeMap<String, Integer>`
3. [optional] Write a program that reads all words in a file and prints out on which line(s) each of them occurred. Use a map from strings to sets

8) Selected algorithms from

`java.util.Collections` and `java.util.Arrays`

Analyse the source code in package `lst05_07`

Exercises

1. Familiarize yourself with the methods of `Collections` and `Arrays` utility classes
2. Explain why `binarySearch` algorithm requires sorted data
3. Explain the difference between *linear search* and *binary search* in terms of the time complexity
4. [optional] Compare experimentally the performance of these algorithms

7) Push the commits to the remote repository