Pratical Exercices N° 10 - Debriefing

Package, Data Structure, Implementation Relation

Exercice 1 - Linked List

1. Create a `Link` record.
    - The visibility of the `Link` record should be `default`.

```
record Link(int value, Link next) {
   public static void main(String[] args) {
       Link firstLink = new Link(13, null);
       Link second = new Link(144, firstLink);
   }
}
```

2. Terminal command to execute the `main` method of the record is : `java fr.umlv.data.Link`
3. Create a `LinkedList` class to manipulate the links.

```
public class LinkedLink {
    private Link headList;
    private int length;
}
```

    - Create `add(value)` method to add a link at the front of the list.

```
public class LinkedLink {
  ...
  public void add(int value) {
      Link newLink = new Link(value, headList);
      headList = newLink;
      length++;
  }
}
```

    - Create `get(index)` method to get the link at the given index.

```
public class LinkedLink {
  ...
  public int get(int index) {
      Objects.checkIndex(index, length);

      Link<T> headPointer = headList;
      for (int current_index = 0; current_index < index; current_index++) {
          headPointer = headPointer.next();
      }
      return headPointer.value();
  }
}
```

- Create `forEach` method to iterate over the list.

```
public class LinkedLink {
   ...
   public void forEach(IntConsumer consumer) {
       Link current = headList;
       while (current != null) {
           consumer.accept(current.value());
           current = current.next;
       }
   }
}
```

  - Create `toString` method to print the list.

```
public class LinkedLink {
   ...
   public String toString() {
       /*Stream.iterate(null, null, null) TODO: Ask the professor about how
to use iterate */
       var listString = new StringJoiner(" --> ");

       for (int i = 0; i < length; i++) {
           listString.add(get(i).value()+"");
       }

       return listString.toString();
   }
}
```

Exercice 2 - Linked List (Continuing)

1. Change `Link` record and `LinkedList` class for a generic usage based on `Object`.

```
record Link(Object value, Link next) {
    public static void main(String[] args) {
        Link firstLink = new Link(13, null);
        Link second = new Link(144, firstLink);
    }
}
```

```
public class LinkedLink {
    ...
    public void add(Object value) {
        Link newLink = new Link(value, headList);
        headList = newLink;
        length++;
    }
}
```

2. To make the code works, we had to :

```
var l = new LinkedLink();
l.add("hello");
l.add("world");
l.forEach( s -> {
    var string = (String)s.value();
    System.out.println("string " + string + " length " + string.length());
});
```

Exercice 3 - Generic Linked List

1. The interest of using a generic type is to be able to use the same method for different types.
2. Change `LinkedLink` to be generic

```java
public class LinkedLink<T> {
    private Link<T> headList;
    private int length;

    public void add(T value) {
        Link<T> newLink = new Link<T>(value, headList);
        headList = newLink;
        length++;
    }

    public T get(int index) {
        Objects.checkIndex(index, length);

        Link<T> headPointer = headList;
        for (int current_index = 0; current_index < index; current_index++) {
            headPointer = headPointer.next();
        }
        return headPointer.value();
    }

    public void forEach(Consumer<T> consumer) {
        Link<T> pointer = headList;
        while (pointer != null) {
            consumer.accept(pointer.value());
            pointer = pointer.next();
        }
    }

    public void removeIf(Predicate<T> predicate) {

    }

    @Override
    public String toString() {
        /*Stream.iterate(null, null, null) TODO: Ask the professor about how to
use iterate */
        var listString = new StringJoiner(" --> ");
        for (int i = 0; i < length; i++) {
            listString.add(get(i)+"");
        }
        return listString.toString();
    }
}
```

3. Change `Main` in consequence Done