PS Programming Methodology

University of Innsbruck - Department of Computer Science

Blaas A., Ernst M., Frankford E., Huaman Quispe E., Hupfauf B., Kaltenbrunner L., Moosleitner M., Priller S., Simsek U.

universität innsbruck
Department of Computer Science

June 27th, 2022

Sheet 10 – Wrap-Up

Exercise 1 (Generics) [2 Points]

Take a look at the following lines of code:

```java
List<Integer> l1 = new LinkedList<>();
List l2 = new LinkedList();
LinkedList<Integer> l3 = new LinkedList<Integer>();
List<Number> l4 = new LinkedList<Integer>();
```

Decide for each of the four statements whether it compiles. Explain what happens and discuss the implications. What are the (dis-)advantages? Should one use this statement in practice?

Submit

`at/ac/uibk/pm/gXX/zidUsername/s10/e01/Generics.txt`

Exercise 2 (Debugging) [3 Points]

Take a look at the class in the file `ListSet.java` OLAT, which implements a generic set based on a list. The class does not implement the Java interface `Set`, as this would require additional methods and make the example more complex. However, the methods provided *should* behave as specified in the java docs of the interface `Set`.
Create a simple example programm and play around with the methods of the class `ListSet`. You'll find that there are number of serious problems in this class and the code does not work as expected. Implement a series of JUnit tests to cover the most important requirements of a `Set`. Try to think about edge-cases and cover as many different possibilities as you can. There should be at least a handful of tests for each method. Since the implementation is currently flawed, the tests will obviously fail at first.
Next, try to fix all issues in the code with **minimal changes**. For instance, do not remove the field `s` from the class, but fix it. Keep the implementation of methods like `contains` – obviously, you could easily delegate the functionality to the inner list (`return elements.contains(element)`), but this is not the point of the exercise. Add a comment to each line that you change and explain the problem as well as your solution.
Even if it is not strictly necessary, use the debugger of your IDE to analyze the code. You should be able to explain the basic terminology of debugging and use the tools accordingly.

Hint

In a real-world example, the collections already provided by Java are generally the best option. Take note, that this exercise is not about software design or refactoring, but reading code and finding bugs. Try to keep changes to a minimum. Your goal is not to find a better solution, but to fix the code you got.

Submit

- `at/ac/uibk/pm/gXX/zidUsername/s10/e02/ListSet.java`
- `at/ac/uibk/pm/gXX/zidUsername/s10/e02/ListSetTest.java`

Exercise 3 (The Packing List) [3 Points]

In the course of this exercise, you'll develop a small application to handle a packing list. A packing list consists of simple items (e.g. a swim suit) as well as list items. A list item is a sub-list of items that are frequently packed together. If you play tennis, your tennis gear could be a list item, which consists of a tennis racket and tennis balls. This way, you can just re-use the sub-list for your tennis gear and don't have to add all items on it individually to your packing list. And now comes the best part: list items can not only contain simple items, but other list items as well. This is somewhat similar to a file system, where folders can contain files and folders, which can contain files and folders, and so on.

Although simple items and list items have different properties, we can find some common ground to work with. We'll keep it simple and limit the functionality to two methods: (1) a method `isPacked` that returns true when the item is packed and (2) a method `print` that prints an item. Use a class or an interface `Item` to reference simple items (`SimpleItem`) and list items (`ListItem`). Read through the remainder of the exercise first and then decide whether `Item` should be an interface, a concrete class or an abstract class.

Continue by implementing the class `ListItem`. As described above, a list item is a container for other items (simple items or list items). Find an appropriate data structure to store items and implement a method to add items. You'll also have to implement the methods `isPacked` and `print`. For a list item, `isPacked` is true, if all items it contains are packed. Similarly, the method `print` prints all items of the list item to standard out. It suffices if all items are aligned to the left, you do not have to indent sub-lists etc.

List items are a great concept, but at some point, we need to have simple items. These are the items you actually pack, while list items are just a way of grouping items together. Implement the class `SimpleItem` as follows. Simple items consist of a name and a boolean for the status (packed / not packed). Add a method to change the status of a simple item from not packed to packed. Again, you'll have to implement the methods `isPacked` and `print`. For a simple item, `isPacked` returns true, if the item is already packed. The method `print` prints a line of the form `[ ] swim suit` or `[X] tennis racket` on standard out, where `[X]` denotes the status "packed".

Create a demo application and generate a packing list with multiple items (simple items and at least two list items). Make sure not all of the items are packed and print the packing list. Now pack the remaining items and print the list again.

Submit

- at/ac/uibk/pm/gXX/zidUsername/s10/e03/Item.java
- at/ac/uibk/pm/gXX/zidUsername/s10/e03/ListItem.java
- at/ac/uibk/pm/gXX/zidUsername/s10/e03/SimpleItem.java
- at/ac/uibk/pm/gXX/zidUsername/s10/e03/Exercise3Application.java

Exercise 4 (Custom Collector) [2 Points]

In the course of this exercise, you'll implement a custom collector that can be used as the final step in a stream. The `LeaveOutCollector` takes the elements of a stream and returns them as a list; however, the `n` smallest and `n` largest elements will be left out. As you might have guessed, `n` should be a variable. For instance, if `n = 1`, all elements, but the smallest and the largest, are returned. Use a `Comparator` to define an order on the elements. Consider the following example:

- Let's assume we have a stream of 5 strings: `java, pm, intellij, c, programming`
- We'll use a `LeaveOutCollector` with `n = 2` and a comparator, that compares the strings by length.
- The two smallest elements (in this case shortest strings), `c` and `pm`, are removed. The two largest elements, `intellij` and `programming`, are removed too.
- The result is a list with a single element, namely `java`

It can happen, that there are no elements left after leaving out the smallest and largest ones; in this case, simply return an empty list. In the example above, this would be the case for `n = 3`. Finally, implement a small example program and demonstrate how to use your collector. Show how your collector can be used on different classes and can use different orders.

Submit

- at/ac/uibk/pm/gXX/zidUsername/s10/e04/LeaveOutCollector.java
- at/ac/uibk/pm/gXX/zidUsername/s10/e04/Exercise4Application.java

Important: Submit your solution to OLAT and mark your solved exercises with the provided check-boxes. The deadline ends at 6:00 pm (18:00) on the day before the discussion.