

Functional and object-oriented programming concepts

Exercise sheet 10



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Questions about this exercise sheet preferably in the moodle forum for this sheet!

Winter semester 21/22

Subjects:

Relevant slide sets:

Submission of the homework:

v1.0

Pointed List Structures

07

01/28/2022 until 11:50 p.m

H Homework 10

branched structures

Total 36 points

Mandatory requirements:

- In this homework we also require documentation using Javadoc. Observe all notes on the documentation of Java methods, which you can find on exercise sheet 3, among others.
- All tasks (except JUnit tests) are in Package `h10` in the register `src/main/java/` implement. Make sure you create all files in this exact package. You write the JUnit tests in the directory `src/test/java/`. The packages correspond to the respective packages of the classes to be tested.
- For file names, identifiers and strings, make sure that these **exactly** as requested - that is, do not change the spelling (including capitalization), add punctuation marks, and do not translate into another language unless it will **explicitly** required in the task.
- It may **none** Packages are imported or constructs from the Java standard library are used that are not explicitly allowed. The packages `java.lang` and `org.junit.jupiter.api` are allowed in this homework.

In the package `h10` find a generic one `public-Class MyLinkedList` and a generic one `public-Class ListItem` as in chapter 07, slides 116 ff.

With `objects`, `integer`, `doubles`, `Number`, `thong` and `exception` are the classes from Package `java.lang` meant; With `Predicate`, `BiPredicate` and `function` the generic classes from Package `java.util.function`.

annotation: For a class `MyLinkedList<T>` Of course, one would not write such methods as you have to write in the following, but rather the methods of `java.util.LinkedList<T>`. The following methods in 1 and 2 are for practice purposes only.

Mandatory requirements: The ones in classes `MyLinkedList` and `ListItem` Given attributes and methods (including constructors) may only be changed according to the task.

H1: Exception class for `MyLinkedList`

2 points

Write a `public`-Class `MyLinkedListException`, directly from `Exception` is derived.

the `public`-Constructor of `MyLinkedListException` has a parameter of formal type `Integer` and a formal type parameter `U`. Without checking, it assumes that both current parameter values are not `zero` are, and initializes the message of the exception as `"(i,x)"`, whereby `i` the one encapsulated by the first actual parameter `internal`-value and `x` the return of `toString` applied to the second parameter.

H2: list methods

28 points

H2.1: Decompose lists

14 points

Expand the class `MyLinkedList<T>` by two generic `public`-Object methods `extractIteratively` and `extractRecursively`.

Both methods each have a generic type parameter `U`, a parameter `predT` of the formal type `Predicate<? great T>`, a second parameter `fCT` of the formal type `Function<? great T, ? extends U>` and a third parameter `predU` of the formal type `Predicate<? great U>`. Return type of both methods is `MyLinkedList<U>`. Both methods assume, without verification, that none of the three current parameter values are equal `zero` is. Both methods potentially throw a formal-type exception `MyLinkedListException`.

Expand the class `MyLinkedList<T>` also a generic one `private`-Object method `extractRecursivelyHelper`. The method has, in addition to the parameters of `extractRecursively` one more parameter `src` of the formal type `ListItem<T>` and a parameter `index` of the type `internal`. Return type of the method is also `MyLinkedList<U>` and it also potentially becomes a formal-type exception `MyLinkedListException` thrown. This helper method is used to test the actual functionality of `extractRecursively` to implement.

The following is the list to which `this.head` references as the *source list*, and the list returned as the *target list* designated. Both lists can also be empty, that is, the cases `otherList.head == zero` and `this.head == zero` are not excluded¹, but it won't `zero` returned. Either method removes all `ListItem<T>`-Objects from the source list for whose key value the functional method of `predT` the value `true` delivers. For each of these so distant `ListItem<T>`-Objects becomes a `ListItem<U>`-Object added to the target list. The key value of the latter object is the result of the functional method of `fCT` applied to the key value of the former object. The method does not make any further changes to the source list, and the target list contains no additional elements for either method.

The order of `ListItem<U>`-Objects in the target list is exactly the order in which their associated `ListItem<T>`- Objects in the original source list (i.e. in the source list at the beginning of the call to `extract*`) was.

An exception is thrown if the functional method of `predU` for a key value to be inserted into the target list, the value `false` returns. The second actual parameter of the constructor call for this exception is this key value. The first actual parameter is the index of the associated `ListItem<T>`-object in the source list - but not the index in the current source list, from which potentially already `ListItem<T>`-objects were extracted, but the index at which it was at the beginning of the call to `extract*` was. (As usual, is 0 the index of the first list element.)

¹It will follow implicitly from the further text: If the source list is empty, the target list is also empty in any case.

Mandatory Requirements:

1. To implement the methods `extract*` methods that are not already implemented may not be used or implemented unless an exception is explicitly provided for in the task.
2. The method `extractIteratively` contains exactly one loop, so no nested loops, and recursion is not allowed.

notice: As soon as the target list is no longer empty, the method `haltextractIteratively` a reference to the currently last one `ListItem<U>`-object in the target list, and with the help of which the next one is created `ListItem<U>`-Object appended to the target list.

3. In the methods `extractRecursively` and `extractRecursivelyHelper` loops are not allowed; in particular, recursion is necessary.

Question of comprehension on the edge (0 points): What is the benefit of specifying that `extractIteratively` contains no more than one loop? *notice:* Consider the case where the target list is very long, eg in the millions, and that you `extractIteratively` a few million times in an application program.

H2.2: Merge lists

14 points

Expand the class `MyLinkedList<T>` order two returnless generic `public`-Object methods `mixinIteratively` and `mixinRecursively`.

Both methods each have a generic type parameter `U`, a parameter `otherList` of the formal type `MyLinkedList<U>`, a second parameter `biPred` of the formal type `BiPredicate<? great T, ? great U>`, a third parameter `fct` of the formal type `Function<? great U, ? extends T>` and a fourth parameter `predU` of the formal type `Predicate<? great U>`. Both methods assume, without verification, that none of the four current parameter values are the same `zero` is. Both methods potentially throw a formal-type exception `MyLinkedListException`.

Expand the class `MyLinkedList<T>` also a generic one `private`-Object method `mixinRecursivelyHelper`. The method has, in addition to the parameters of `mixinRecursively` nor the parameters `src` of the formal type `ListItem<U>`, `pDest` of the formal type `ListItem<T>` and `index` of the type `internal`. The method is returnless and it also potentially raises a formal-type exception `MyLinkedListException` thrown. This helper method is used to test the actual functionality of `mixinRecursively` to implement.

The following is the list to which `otherList` references as the *source list*, and the list on which `this.head` references as the *target list* designated. Both lists can also be empty, that is, the cases `otherList.head == zero` and `this.head == zero` are not excluded. Any of the two methods `mixin*` inserts a new element into the target list for each element in the source list. The method does not make any further changes to the source list

Specifically, this means: For each element `listItem` in `otherList` becomes a new one `ListItem<T>`-Object created and inserted into the target list. The key value of the new element is the result of the functional method of `fct` applied on `listItem.key`.

With Z_0 be the state of the target list at the start of the run from `mixin*` designated. for $i \in \{0, 1, 2, \dots\}$ may be Z_i the state of the target list after exactly for that `ListItem<U>`-Objects at the positions $0, \dots, i-1$ of the source list `ListItem<T>`-Object has been inserted into the target list (specifically contains Z_{i-1} agree i items additionally opposite Z_0). For the `ListItem<U>`-Object in position $i \in \{0, 1, 2, \dots\}$ the source list becomes the new one `ListItem<T>`-Object at the smallest position j in Z_{i-1} inserted so that both following conditions for j be valid:

1. For the elements in the positions $0, \dots, i-1$ the source list is the associated new item in one of the positions $0, \dots, j-1$ in Z_{i-1} , that is, the new elements are inserted into the target list in exactly the same order in which the corresponding elements appear in the source list.

2. Either is j no position of Z_{i-1} (That means, Z_{i-1} has length j); or if Z_{i-1} the position j but contains, the functional method of `returns biPred` the value `true` when applied to the key value in Z_{i-1} at position j and the key value in the source list at position $i-1$. If `s biPred` nowhere the value `true` returns, the to be inserted becomes `ListItem` appended to the end of the target list.

An exception is thrown if the functional method of `predUfor` a key value in the source list, the value `false` returns. The second actual parameter of the constructor call for this exception is this key value. The first actual parameter value is the index of the associated one `ListItem<U>`-object in the source list. (As usual, `is0` the index of the first list element.)

Mandatory Requirements:

1. To implement the method `s extract*` methods that are not already implemented may not be used or implemented unless an exception is explicitly provided for in the task.
2. The method `mix in Iteratively` contains exactly one loop, and recursion is not allowed.
notice: Step through all elements of the source list with a reference. In this loop, manage a second reference: Once `0` is no longer a candidate for the next position to insert, this second reference always points to that `ListItem<T>`-Object immediately before the next candidate.
3. In the methods `mix in Recursively` and `mix in Recursively Helper` loops are not allowed; in particular, recursion is necessary.

clarification: "x will be in position j inserted" means that x into position immediately after this paste operation j is located, compare chapter 07, slide 51.

notice: The above, formal definition of the positions at which the new elements are to be inserted in the target list can be read as a 1:1 instruction on how to implement the `mix in` iteratively or recursively.

Question of comprehension on the edge (0 points): It obviously easily leads to misunderstandings if you describe processes on objects and structures that change over time and simply use the identifier under which you address this object or this structure in the source text, because it is not always clear when and with it which state is meant exactly. This is obviously a problem even in the simplest of cases, such as with variables of primitive data types (what is "the value of `internal i`", which point in time in the course of the program is meant?). In the task text of 2 above you can see a common way of describing such processes unambiguously. How can this possibility be formulated in general for any concrete object and structure type, i.e. abstracted from the concrete situation in 2? Follow-up question: Why wasn't that necessary for 1?

H3: Testing using JUnit

6 points

Write in the package `h10one` `public-Class TestMyLinkedList` with two non-generic `public-Object` methods `testExtract` and `testMix in`. Both methods have no parameters and no return, and neither throws an exception.

reminder: The JUnit tests should be in the directory `src/test/java/` to be written.

To make it easy for you to create test examples, see class `MyLinkedList` a method `add` analogous to the method `addBy` interfaces `java.util.Collection`, only without `throws-Clause`. Look again at what this method is supposed to guarantee for classes that `java.util.List` to implement.

Question of comprehension on the edge(0 points): Why has method `addFrom` from `java.util.Collection` one `throws` clause with various exception classes if it's at method `addFrom` `MyLinkedList` but also works without it?

H3.1: test `extract*`

3 points

`IntestExtract` tests the two methods `extract*` from 1 with `T=array of integer` and `u=integer`.

method `testExtract` tests the two methods `extract*` from 1 to three lists of arrays that you create deterministically (that is, without a random number generator). Each of these three lists has six elements, and each array in any of these lists has length 3.

The functional method of the predicate `predT` delivers exactly then `true` returns if at least one of the three array components is evenly divisible by at least one of the other two array components. In concrete terms, of course, this means that the modulo operation must result in at least one pair of array components 0 has.

The functional method of the function `FT` returns the sum of the array components.

The functional method of the predicate `predU` delivers exactly then `true` returns if its current parameter value is nonnegative. ²

For the first two of these three lists, choose all six items such that `extract*` doesn't throw an exception. Specifically, design the first list in such a way that exactly the elements *even* indices through `extract*` should be extracted, and the second list so that exactly those items *odd* indexes should be extracted.

You design the third of these three lists in such a way that `extract*` an exception with position 4 should throw.

Of course, your JUnit test consists of your method `testExtract` tests whether, for each of the first two lists, the target list and the modification of the source list are used by both methods `extract*` are created correctly, and for the third list, whether an exception with correct message from both methods `extract*` is thrown.

Mandatory Requirement: For the three parameters of `extract*` write an appropriate class that implements each formal parameter type, and use references to objects of those three classes as actual parameter values. In particular, don't use lambda expressions.

H3.2: test `mix in*`

3 points

`IntestMixIn` tests the two methods `mix in*` from 2 with `T=Number` and `u=thong`.

The functional method of the parameter `predU` delivers exactly then `true` Returns if the current parameter value is a string representation of a number in the form of `doubles` is (*non-binding notice*: class method `valueOf` of class `doubles`).

The functional method of the parameter `biPred` assumes without verification that the second current parameter value is the string representation of a number in the form of `doubles` is, and returns exactly then `true` returns if that number is less than the number encapsulated in the first current parameter value.

The functional method of the parameter `FT` assumes without verification that the second current parameter value is the string representation of a number in the form of `doubles` is, and returns a reference to a `doubles` object with this number value.

method `testMixIn` tests the two methods `mix in*` from 2 to three pairs of source and target lists. Each source and each target list has four elements.

For the first two pairs of lists, select all elements such that `mix in*` doesn't throw an exception. Specifically, you design the first of these three pairs of lists so that the elements in the source list are exactly the elements in the resulting one

²reminder: "nonnegative" in mathematics and computer science means that the value is greater than or equal to 0 is; analogously, "non-positive" means that the value smaller or equal 0 is. Thus "non-negative" is the logical complement of "negative" and "non-positive" is the logical complement of "positive".

target list *even* Positions are, and the second pair of lists such that the elements in the source list exactly match the elements in the resulting target list *odd* positions are.

Design the third pair of lists in such a way that an exception is thrown.

Mandatory Requirement: The current parameter values for the parameters `biPred`, `FCT` and `predU` are for both methods `mix in*` each formulated as a lambda expression. However, these lambda expressions are not used directly in the method call, but are each held by a constant of a suitable type, and this constant is the current parameter when both methods are called `mix in*`.