Programming in Java

Coursework Two — Clarifications

2015–16

Following several questions from students, the following points clarify several aspects that were undefined or unclear in the assignment's description.

1 Constructors

Some students have asked about the constructors of the classes that will implement the interfaces provided. Several different possibilities are available.

In order to clarify this point, to provide a common learning experience, and to facilitate automatic evaluation of the code, this document specifies the constructor(s) required for each class.

Most classes must have only the default constructor, with no arguments. The exceptions are described below. Classes may not have any constructor not described in this document — be careful about this because using the wrong constructors may result in your code not compiling with the automatic tests used for grading.

ReturnObjectImpl: This class must have two constructors, each of them with only one parameter. The constructor used for successful operations must receive an Object[1] and the constructor used for failed operations must receive an `ErrorMessage` as its only parameter.

StackImpl: This class must have only one constructor with only one parameter of type `List` (i.e. a stack can be created using either an `ArrayList` or a `LinkedList` as the underlying data structure).

ImprovedStackImpl: This class must have only one constructor with only one parameter of type `List`.

2 Exceptions

The code must work without throwing any exceptions in any circumstance (e.g. including `NullPointerException`, etc).

[1]java.lang.Object

As we have not covered exception handling yet, the code must not use the keywords `throw`, `throws`, `try`, `catch`, or `finally`. (If you do not know what these keywords are, that is perfectly normal; we will learn about them in due time).

3 A couple of clarifications regarding lists

- Method `add(Object,int)` adds an element to the list, inserting it at the given position. If the index is negative or greater or equal than the size of the list, an error must be returned. Note that this means that a new element cannot be added at the end of the list with this method (instead, `add(Object)` can be used for that).

- Implementations of interface `List` do not have to sort the elements added to them based on any natural ordering (e.g. $1 < 2$, 'a' $<$ 'b', etc). Elements of a list are sorted by index, and for this assignment they *must be sorted by index only*. In other words, if strings "Mark", "Luke", and "John" are added to a list in that order, then `get(2)` must return a `ReturnObject` such that `getReturnValue()` will return the string "John" regardless of the fact that 'J' comes before 'M'.