Deep Copy Lists Lab

Lab # 5

Generated by Doxygen 1.8.6

Fri Oct 2 2015 12:40:16

Contents

1 Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

2 Class Documentation

2.1 FunctionalList< T > Class Reference

Public Member Functions

- FunctionalList ()
- FunctionalList (FunctionalList< T > lst)
- boolean isEmpty ()
- FunctionalList< T > add (T element)
- FunctionalList< T > append (FunctionalList< T > elements)
- FunctionalList< T > remove (T element)
- FunctionalList< T > reverse ()
- int size ()
- int positionOf (T element)
- T nth (int index) throws IllegalAccessException
- Object[] toArray ()
- FunctionalList< T > subst (T key, T value)
- String toString ()

2.1.1 Detailed Description

Introduction/Instructions

A generic, singly-linked list implementation that ensures *referential transparency*, which is to say that clients operate on *copies* of the structure. In other words, this is a side-effect free implementation. Instead of modifying the underlying list structure in-place, which is the standard approach used by imperative languages, such as Java, this implementation ensures that clients are always working on *copies* of the underlying structure so that changes made by one client are transparent to others. This means, among other things, that the result of adding, removing, reversing, etc., a list are *not reflected* in the structure of the original list, instead, a copy is made (usually recursively) that captures the desired changes. Thus, the client must replace the older (previous) copy of their list with the new copy returned by the method.

Special instructions/restrictions, etc.

Please read and adhere to the following:

- Obviously, your implementation should not use any of Java's `Collections`.
- All methods, with the exception of constructors, and utilities, such as `toString()` and `toArray` should be implemented recursively. In most cases this means that the public method will call a `private` method that recursively implements the specified behavior.
- In addition to the documentation that appears in the preamble of your methods, you must document the internal logic of the methods that actually perform the computation. In the case of recursive methods, your documentation must identify the base case and describe how the "reduction" will solve the problem by moving the computation towards the base case.

Author

UMD CS Department

Parameters

<T>	any subclass of `Object`

2.1.2 Constructor & Destructor Documentation

2.1.2.1 FunctionalList ( )

Creates an empty linked-list.

2.1.2.2 FunctionalList (FunctionalList< T > *lst*)

The copy constructor for this class must provide a deep copy.

Parameters

lst	(any valid `FunctionalList`)

2.1.3 Member Function Documentation

2.1.3.1 FunctionalList<T> add (T *element*)

Reconstruct list by appending `element` onto its end.

Parameters

element	

Returns

(newly created) Functional List

2.1.3.2 FunctionalList<T> append (FunctionalList< T > *elements*)

Creates and return a new list whose elements are the original list with the elements of the FunctionalList parameter appended in their original order.

Parameters

elements	`Node`

Returns

copied `FunctionalList` but with `elements` at the end. All lists should retain the original ordering of their elements.

2.1.3.3 boolean isEmpty ( )

The preferred way of determining whether a FunctionalList object is empty.

Returns

2.1.3.4 T nth (int *index*) throws IllegalAccessException

Returns the nth element of the list, assuming that index is an int >= 0.

- This method throws an `IllegalAccessError` if it is called on an empty list, regardless of the value of `index`.
- This method throws an `IllegalArgumentException` error if it is called with an `index` greater than (or equal to) the number of elements in the underlying list.

Exception Handling Required

The use of a `IllegalAccessException` requires that this method declare that it "throws" the exception and that any caller of this method explicitly "catch" the exception if it is appropriate to do so, or to likewise declare that it, too, "throws" the exception.

Your student tests will need to be written with this in mind.

Parameters

index	an integer greater than or equal to 0

Returns

an object of type T located at index.

Exceptions

IllegalAccessException	

2.1.3.5 int positionOf (T *element*)

Returns -1 iff element is not found in list; returns the 0-based index of element, otherwise.

Parameters

element	any appropriate Object type

Returns

index corresponding to the location of the element, or -1 if not found.

2.1.3.6 FunctionalList$<$T$>$ remove (T *element*)

Returns a copy of List eliminating all occurrences of the `element`.

Parameters

element	any Object of the appropriate type.

Returns

a copy of the original list with `element` removed.

2.1.3.7 FunctionalList$<$T$>$ reverse ( )

Recursively constructs a reversed image of the original list.

Returns

a copy of the original list with its elements reversed.

2.1.3.8 int size ( )

Returns the number of values stored in list.

Returns

an integer greater than or equal to 0.

2.1.3.9 FunctionalList$<$T$>$ subst (T *key,* T *value*)

Returns a new Functional List where each item that was equal to the `key` has been replaced with the `value`.

For example: if list = ["a", "b", "c", "b"], then calling

```
subst( "b", "z", list )
```

returns a new list = ["a", "z", "c", "z"]

Parameters

key	
value	

Returns

2.1.3.10 Object [ ] toArray ( )

A utility method: returns an array whose elements are the elements of the list, in their list-order.

Implementation option

Given the nature of this method, an iterative implementation is likely the most natural.

Returns

an array of objects as they appeared in the list.

2.1.3.11 String toString ( )

Necessary to print what's going on ...

Implementation option

Given the nature of this method, an iterative implementation is likely the most natural way to go.

Parameters

@return	

The documentation for this class was generated from the following file:

- FunctionalList.java

Index