Deep Copy Lists Lab

Lab # 6

Generated by Doxygen 1.8.8

Thu Mar 5 2015 15:22:46

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

Classes

- class **Node**

Public Member Functions

- FunctionalList ()
- FunctionalList (FunctionalList< T > lst)
- 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)
- Object[] toArray ()
- T[] toArray (T[] arrayIn)
- String toString ()

Protected Member Functions

- FunctionalList (Node nodes)

2.1.1 Detailed Description

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 situ*, 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.

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

any	valid `FunctionalList`

2.1.2.3 FunctionalList (Node *nodes*) [protected]

This is a protected constructor because it's used only by internal methods, or, perhaps by any subclasses.

Parameters

a	chain (possibly empty) of `Node` objects.

2.1.3 Member Function Documentation

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

Reconstruct list by appending `element` onto its end.

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

Create 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<T>` but with `elements` at the end. All lists should retain the original ordering of their elements.

2.1.3.3 T nth (int *index*)

Make sure that this method throws an `IllegalAccessError` if it is called on an empty list, regardless of the value of `index`.

Make sure that this method throws an `ArrayIndexOutOfBounds` error if it is called with an `index` greater than (or equal to) the number of elements in the underlying list.

Parameters

index	an integer greater than or equal to 0

Returns

an object of type T located at index.

2.1.3.4 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.

2.1.3.5 FunctionalList<T> remove (T *element*)

Reconstruct 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.

2.1.3.6 FunctionalList<T> reverse ( )

Remove the last element in a list. Note, this method handles null and singleton lists appropriately.

Returns

Recursively constructs a reversed image of the original list.
a copy of the original list with its elements reversed.

2.1.3.7 int size ( )

Returns the number of values stored in list.

Returns

an integer greater than or equal to 0.

2.1.3.8 Object [ ] toArray ( )

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

Returns

an array of objects as they appeared in the list.

2.1.3.9 T [ ] toArray (T[ ] *arrayIn*)

Utility method: (Students are not responsible for this method.) Returns a Typed array whose elements are the elements of the list in order.

Parameters

arrayIn	note the type

Returns

a *typed array* of the objects contained in the list in their original order.

2.1.3.10 String toString ( )

Necessary to print what's going on ...

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

- FunctionalList.java