Applying Design by Contract to Java ArrayList Data Structure

Aldo Gabriele Di Rosa, Brenda Ruiz
& Jacopo Fidacaro

December 16, 2018

1 Selection Process

The selected project for implementing Design by Contract was the ArrayList and the Vector classes from Java Collections. We picked these two data structures given that their behaviour can be modelled by implementing Contracts. (...)

2 Contract Implementation

First, we implemented an Interface "ListContract" which defines the ArrayList methods and where the each of the Contracts were implemented. Then, we created a class "ContractedArrayList" which implements our interface each of its methods. The approach followed for the implementation of the Contracts was first reading the class documentation for each method to understand and determine each one of the preconditions and postconditions. One of the challenges we faced during the implementation was understanding how to properly use the jSicko library. In some cases, jSicko would produce errors which were difficult to understand what flaw they were pointing out and sometimes seemed arbitrary. (add an example?)

3 Contracts Overview

For the ArrayList class, a total of 16 Contracts and one Invariant were implemented, shown below:

- Invariant: nonNegativeSize. Ensures the size of the list is always equal or greater than 0.

Method	**Precondition**	**Postcondition**
add(E e)	sizeIncreases	
add(int index, E element)	indexInRange	sizeIncreases
get(int index)	indexInRange	
remove(Object o)		sizeDoesNotIncrease
remove(int index)	indexInRange	sizeDecreases
addAll(Collection c)		containsAll
addAll(int index, Collection c)	indexInRange	
removeAll(Collection c)	collectionNotNull	
retainAll(Collection c)	collectionNotNull	
clear()	listEmpty	
set(int index, E element)	indexInRange	
indexOf(Object o)	equalObjectAtFirstOccurrence	
lastIndexOf(Object o)	equalObjectAtLastPosition	
subList(int fromIndex, int toIndex)	indexesInRange	
toArray(E[] a)	arrayNotNull	

Vector class Contracts

- Invariant: nonNegativeSize. Ensures the size of the vector is always equal or greater than 0.

Method	**Precondition**	**Postcondition**
addElement(E obj)	sizeIncreases	
elementAt(int index)	indexWithinBounds	
removeElementAt(int index)	indexWithinBounds	
indexOf(Object o, int index)	nonNegativeIndex	sizeStaysTheSame
lastElement()	nonEmptyVector	
firstElement()	nonEmptyVector	
removeAllElements()	emptyVector	
removeIf(Predicate¡? super E¿ filter)	filterNotNull	
lastIndexOf(Object o, int index)	indexWithinBounds	sizeStaysTheSame