AbstractSequentialList<E> class

This class provides skeleton implementation for [List<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-5list-interface.html) interface. It reduces effort of class that supports sequential access like LinkedList<E> class. For random access example [ArrayList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-5list-interface.html) we use [AbstractList<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-11-abstractlist.html) class.

**Code for this class is referred and taken from Javadocs.**

Let us understand all methods that are placed under AbstractSequentialList<E> class.

**AbstractSequentialList<E> class definition**

This class is abstract and it extends AbstractList<E>. In turn, AbstractList<E> extends [AbstractCollection<E>](http://data-structure-learning.blogspot.com/2015/05/java-collection-part-10.html) and implement List<E> interface.

**public** **abstract** **class** AbstractSequentialList<E> **extends** AbstractList<E> {

**Constructor for AbstractList<E> class**

Constructor of this class is protected by definition. And why is that? The reason is that the invocation can be done by subclass constructors only.

**protected** AbstractSequentialList() {

}

**get(int index)**

This method returns the element at the specified index passed as parameter. What is does is it calls the listIterator(index) method which has underlying implementation of [ListInterface<E>](http://data-structure-learning.blogspot.com/2015/05/java-collections-part-9-listiterator.html). And it also has an overloaded constructor which takes index as input. next() method calls the next element and returns it.

It throws IndexOutOfBoundsException(str) if iteration has no next element.

**public** E get(**int** index) {

**try** {

**return** listIterator(index).next();

} **catch** (NoSuchElementException exc) {

**throw** **new** IndexOutOfBoundsException("Index: "+index);

}

}

**set(int index, E element)**

This method replaces value at index with element passed as parameter. It uses listIterator(index ) and returns the old value.

**public** E set(**int** index, E element) {

**try** {

ListIterator<E> e = listIterator(index);

E oldVal = e.next();

e.set(element);

**return** oldVal;

} **catch** (NoSuchElementException exc) {

**throw** **new** IndexOutOfBoundsException("Index: "+index);

}

}

**add(int index, E element)**

Inserts the element E passed as parameter into index. Uses listiterator(index) and then adds the element at that index. It shifts other elements to right and adds one to their indices.

**public** **void** add(**int** index, E element) {

**try** {

listIterator(index).add(element);

} **catch** (NoSuchElementException exc) {

**throw** **new** IndexOutOfBoundsException("Index: "+index);

}

}

**remove(int index)**

This method is used to remove the element at specified index. Element that is removed is returned.

**public** E remove(**int** index) {

**try** {

ListIterator<E> e = listIterator(index);

E outCast = e.next();

e.remove();

**return** outCast;

} **catch** (NoSuchElementException exc) {

**throw** **new** IndexOutOfBoundsException("Index: "+index);

}

}

**addAll(int index, Collection<? extends E> c)**

addAll method adds entire collection c at specified index. All other elements after index are shifted right.

Collection<? extends E> is used because we use <? extends E> when we need to get values from collection.

**public** **boolean** addAll(**int** index, Collection<? **extends** E> c) {

**try** {

**boolean** modified = **false**;

ListIterator<E> e1 = listIterator(index);

Iterator<? **extends** E> e2 = c.iterator();

**while** (e2.hasNext()) {

e1.add(e2.next());

modified = **true**;

}

**return** modified;

} **catch** (NoSuchElementException exc) {

**throw** **new** IndexOutOfBoundsException("Index: "+index);

}

}

**iterator()**

returns the listIterator()

**public** Iterator<E> iterator() {

**return** listIterator();

}

**listIterator(int index)**

Returns the listIterator over the elements in the list.

**public** **abstract** ListIterator<E> listIterator(**int** index);