Data Structures and Algorithms

BSc. In Computing

Semester 5 Lecture 1

Lecturer: Dr. Simon McLoughlin

Data Structures & Algorithms Course books

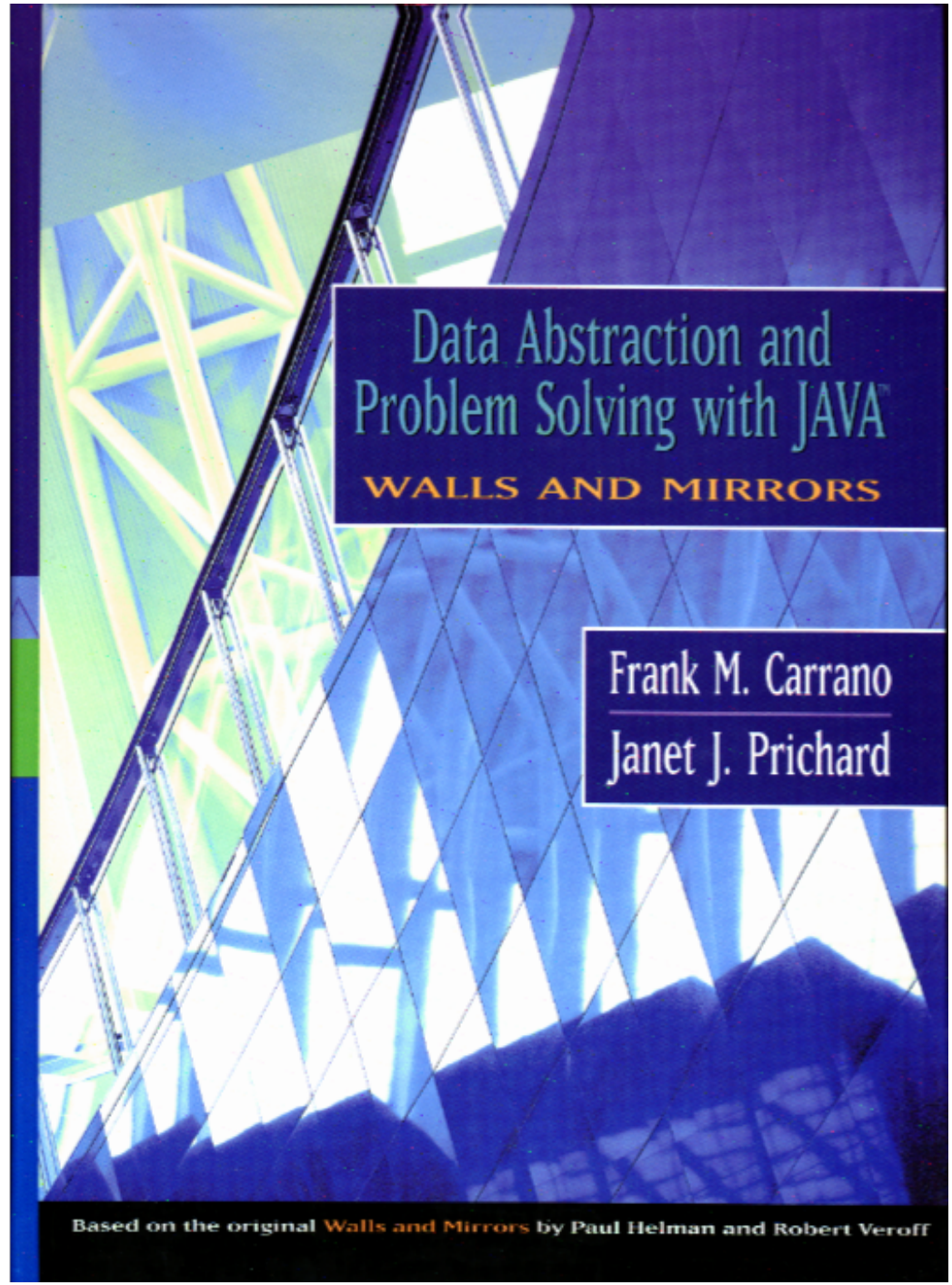
Book 1

Published in 2001 by Addison Wesley

Cost about £40

Order via Hodges Figgis

Library has copies



Data Structures & Algorithms Course books

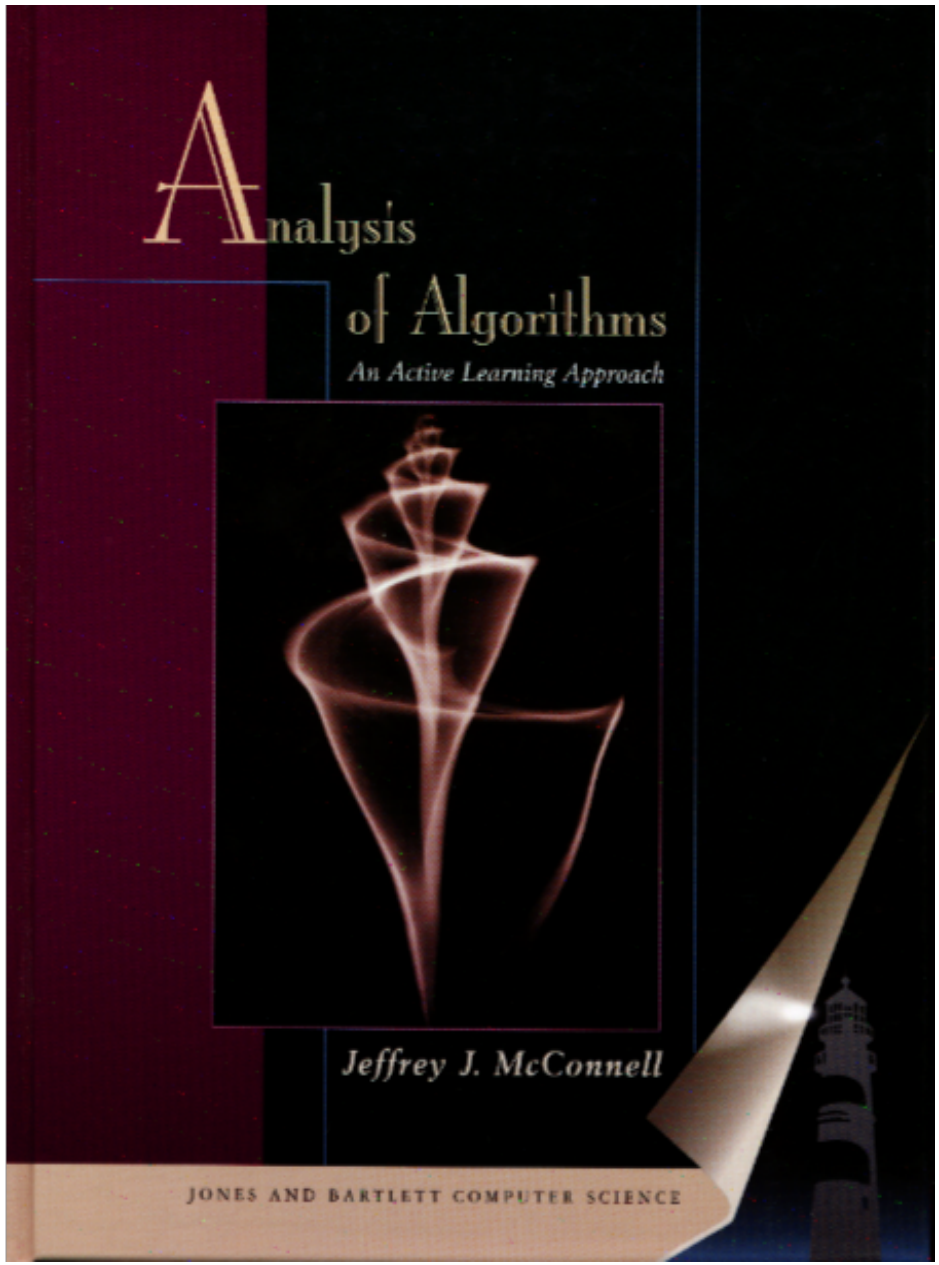
Book 2

Published in 2001 by
Jones & Bartlett
Computer Science Series

Cost about £35

Order via Hodges Figgis

Library has copies



Subject Matter - What is this course about!

Abstract Data Types

Lists, Stacks and Queues

Trees of different colours, shapes and sizes

Analysis of Algorithms

Algorithmic Techniques

Searching and Sorting Algorithms

Matching Algorithms

Graph Algorithms

And of course, Software Development through Java ☺

The plan:

- Lectures
- Lab Exams (2) **15% each** based on work done in labs
- One assignment (**15**% of award marks)
- Lab attendance/Participation (**5**% of award marks)
- Exam at end of semester (**50**% of award marks)

Data Abstraction

Abstract Data Types

Specifying ADTs
- The ADT List
- The ADT Sorted List
- Designing an ADT
- Axioms

Implementing ADTs
- Java Classes
- Java Interfaces
- Java Exceptions
- An Array-based implementation of the ADT List

Problem solving and Software Engineering

What is problem solving?

The term **problem solving** refers to the entire process of

- taking the **statement of a problem** (**The Problem Definition**) and
- **developing a computer program** that provides a solution.

This process requires that you pass through many phases:

- gaining an **understanding** of the problem to be solved,
- **designing** a conceptual solution,
- **implementing** the solution with a computer program.

In our situation, the implementation language will, of course, be Java.

What is a solution?

Typically, a solution consists of two components:

- Algorithms
- Ways to store the data.

An algorithm is a step-by-step specification of a method to solve a problem within a finite amount of time.

During the design of a solution, you will need to support several operations on the data and therefore need to define abstract data types (ADTs).

A collection of data together with a set of operations on that data is called an Abstract Data Type.

We will introduce some simple abstract data types.

- Only after you have clearly specified the operations of an abstract data type should you consider data structures for implementing it.

We explore implementation issues and introduce Java classes to hide the implementation of an ADT from its users.

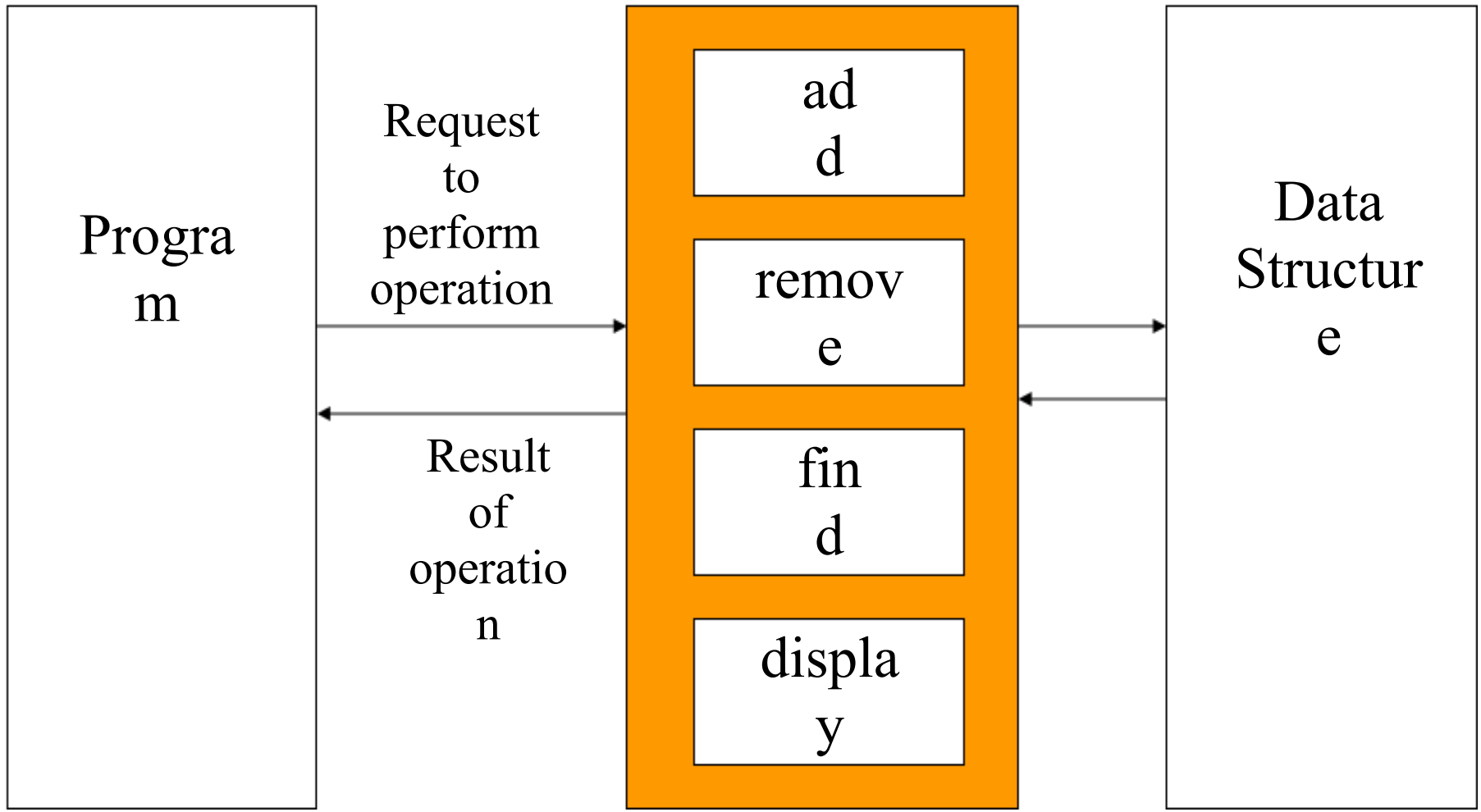
ADTs vs. Data Structures

- An abstract data type is a collection of data and a set of operations on that data.

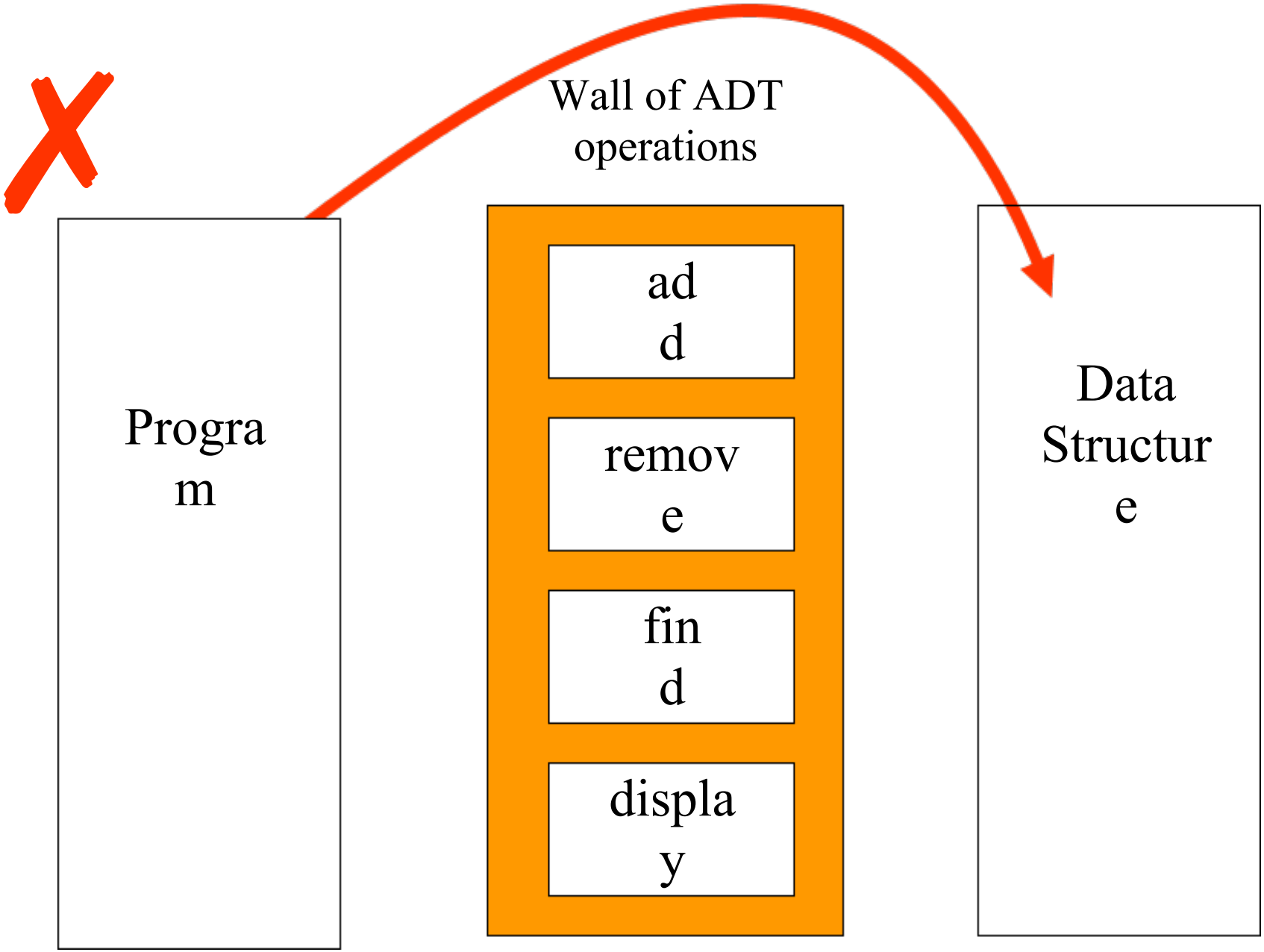
- A data structure is a construct within a programming language that stores a collection of data.

Data abstraction results in a wall of A DT operations between data structures and the program that accesses the data within these data structures.

If you are on the program side of the wall, **you will see the interfaces that enable you to communicate** with the data structures, that is, you request the ADT operations to manipulate the data in the data structures, and they **pass the result of these manipulations back** to you.



A wall of ADT operations isolates a data structure from the program that uses it



Violating the wall of ADT operations

Specifying ADTs

To elaborate on the notion of an abstract data type, we can consider a typical list that you might use:

- Shopping list,
- things to do list,
- list of addresses

Assuming that you write in one column, most of the time we **add** new items to the **end** of the list.

We could, however, **add** new items to the **beginning** of the list.

We could also **sort** the list in alphabetical order, as with a list of names and addresses.

Some Common List Features

The list always appears in **sequence**.

The list has one **first** item and one **last** item.

The first item, the **head** or front of the list, does not have a **predecessor**.

The last item, the **tail** or end of the list, does not have a **successor**.

Lists contain items of the **same type**, i.e. grocery items

What can you do with the items on a list?

You can:

- **Count** the items to determine the length of the list.
- **Add** an item to the list
- **Remove** (**Delete**) an item from the list
- Look at (**Retrieve**) an item from the list

The items on the list, together with operatio ns that you can perform on the items, form an abstract data type.

You must specify the behaviours of the A DT operations on its data, that is, the list items.

Focus only on specifying the operations and not on how you will implement them.

ADT List Operations

1. Create an empty list
2. Determine whether a list is empty
3. Determine the number of items on a list
4. Add an item at a given position in the list
5. Remove the item at a given position in the list.
6. Remove all the items from the list
7. Retrieve (GET) the item at a given position in the list

Pseudocode for the ADT list

createList()
//creates an empty list

isEmpty()
//determines whether a list is empty

size()
//returns the number of items that are in a list

add(index, item)
//inserts item at position index of a list, if
// 1 <= index <= sixe()+1
// if index <= size(), items are renumbered as
// follows: the item at index becomes the item at
// index+1, the item at index+1 becomes the item at index+2, etc.
// throws an exception when index is out of range or
// if the item cannot be placed on the list, i.e. the list is full.

remove(index)
// removes the item at position index of a list,
// if 1 <= index <= size(). If index < size(), items are
// renumbered as follows: The item at index+1
// becomes the item at index, the item at index+2 becomes
// the item at index+1, etc.
// Throws an exception when index is out of range or
// if the list is empty

removeAll()
// remove all the items in the list

get(index)
// returns the item at position index of a list
// if 1<= index <= size(). The list is
// left unchanges by this operation.
// Throws an exception if index is out of range.

To get a better idea of how the operations work, apply them to the list:

milk, eggs, butter, apples, bread

where milk is the first item on the list, and bread is the last item.

How can we construct this list by using the operations of the ADT list?

How can we construct this list by using the operations of the ADT list?

First create an empty list **aList** followed by a series of operations to append successively the items to the list:

```
aList.createList()
aList.add(1, milk)
aList.add(2, eggs)
aList.add(3, butter)
aList.add(4, apple)
aList.add(5, bread)
```

The notation **aList.op** indicates that an operation **op** applies to the list **aList**.

The list's insertion operation can insert new items into any position of the list, not just at the front or end.

According to **add**'s specification, if a new item is inserted into position k, the position of each item that was at a position of k or greater is increased by 1.

Consider: aList.add(4, nuts)

The list now contains:

milk, eggs, butter, nuts, apples, bread

These examples illustrate that an ADT can specify the effects of its operations without having to indicate how to store the data or how to implement the operations

The specification of the seven operations are the sole terms of the contract for the ADT list.

If you request that these op erations be performed, this is what they will do for you!

Once you have **specified the behaviour of an ADT** , you can design applications that access and manipulate the ADTs data solely **in terms of its operations** and **without regard for its implementation.**

For example, suppose that you need to **display the items** on a list.

- The wall between the implementation of the ADT and the rest of the program prevents you from seeing how the data is stored.
- You can still write a Java method **displayList** in terms of the operations that define the ADT list.

Pseudocode

```
displayList(aList)
//displays the items on the list aList

for (index = 1 through aList.size() )
{     dataItem = aList.get(index)
      display dataItem
}//end for
```

As long as the ADT list is implemented correctly, the **displayList** method will perform its task because it does not depend on how you implement the list.

As another application of the ADT operations, suppose you need a method **replace** that replaces the item in position **k** with a new item.

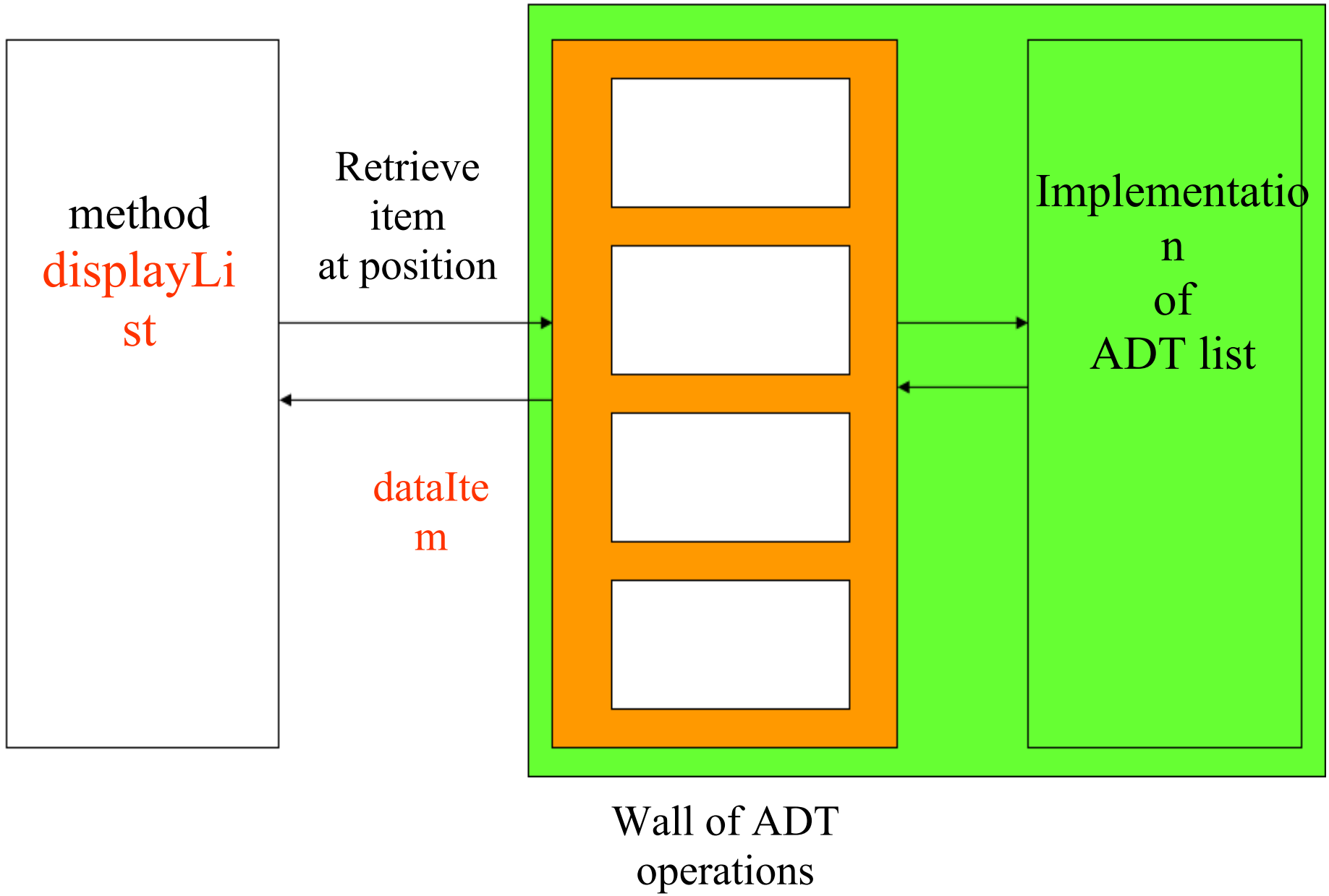
If the $\mathbf{k}^{th}$ item exists, replace deletes the item and inserts the new item at position **k**

Pseudocode

```
Replace(aList, k, newItem)
//replaces the k^th item on the list aList with newItem

if (k >=1 and k <= aListsize() )
{
      aList.remove(k)
      aList.add(k, newItem)
} //end if
```

We simply focus on the task at hand without any concern for the implementation details of ADT List.



The wall between displayList and the implementation that uses it

Pseudocode for the ADT Sorted List Operations

createSortedList()
//Create an empty list

sortedIsEmpty()
//determined wheter a sorted list is empty

sortedSize()
// returns the number of items that are in a sorted list

sortedAdd(item)
// Inserts item into its proper sorted position in a
// sorted list. Throws an exception if the item
// cannot be placed on the list (list full)

sortedRemove(item)
// delete item from a sorted list
// Throws an exception if the item is not found

sorted Get(index)
// returns the item at position index of a
// sorted list, if 1 <= index <= sortedSize().
// The list is left unchanged by this operation.
// Throws an exception if the index is out of the range

locateIndex(item)
//Returns the position where item belongs or
// exists in a sorted list; item and list are unchanged.

The ADT sorted list differs from the ADT list in that a sorted list inserts and deletes items in a sorted order defined by their values.

Designing an ADT

The design of an abstract data type should evolve naturally during the problem-solving process.

- As an **example** of how this process might occur, suppose that you want to create a **software based appointment book** that spans a one-year period.

- **Assume** that appointments can only be made on the hour and half-hour between 8AM and 5PM

- **Assume** that the appointments are 30 minutes in duration

We want the system to store some brief notation about the nature of each appointment along with data and time.

We define an ADT appointment book.

The **data items** in this ADT are the **appointments**, where an appointment consists of a **date**, **time**, and **purpose**.

What are the operations?

- **Make an appointment** for a certain date, time and purpose. (make sure that you do not make an appointment for an already occupied time)
- **Cancel the appointment** for a certain date and time.
- **Ask whether you have an appointment** at a given time.
- **Determine the nature of your appointment** at a given time.
- We will need the **usual initialisation operation** for an ADT

Pseudocode for the ADT appointment book

CreateAppointmentBook()
// create an empty appointment book

isAppointment(date, time)
// returns true is an appointment exists for the date
// and time specified; otherwise returns false

makeAppointment(date, time, purpose)
// inserts the appointment for the date, time, and purpose
// specified as long as it does not conflict with an existing
// appointment. Returns true if successful, false otherwise

cancelAppointment(date, time)
// returns the purpose of the appointment at
// the given date/time, if one exists. Otherwise, returns null.

CheckAppointment(date, time)
// returns the purpose of the appointment at
// the given date/time, if one exists. Otherwise, returns null.

An Array-based Implementation of the ADT List

We will implement the ADT list as a class in Java.

The ADT List operations are:

createList()
isEmpty()
size()
add(newPosition, newItem)
remove(index)
removeAll()
get(index, dataItem)

At first glance we might think that an array is a natural "fit" for a list.
This is not quite true!

The ADT list has operations such as **removeAll**() that an array does not.

- Later, we will look at a different type of implementation that does not use an array.

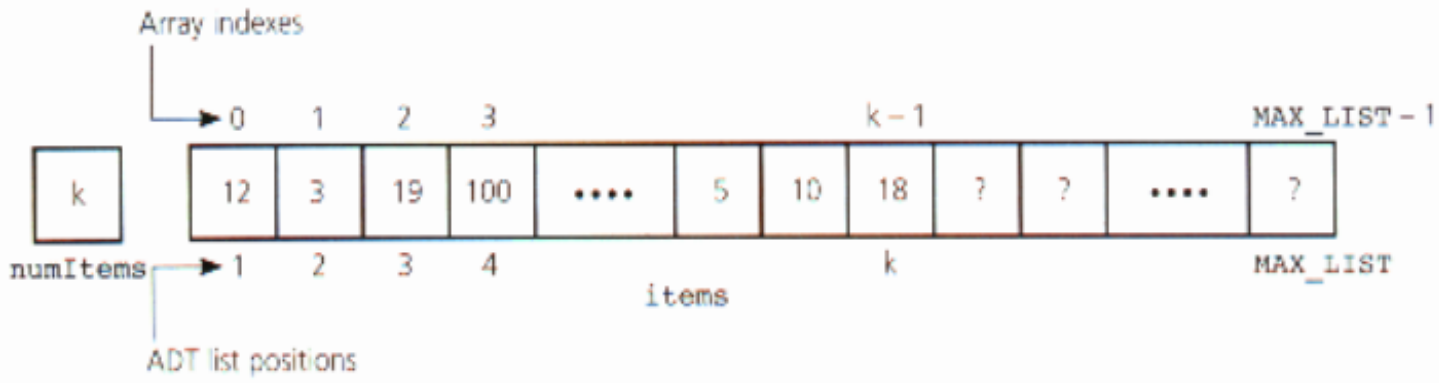
In this array, we can store a list's $\mathbf{k}^{\mathbf{th}}$ item in **items**[k-1]

- We may occupy all of the array, or we may not!

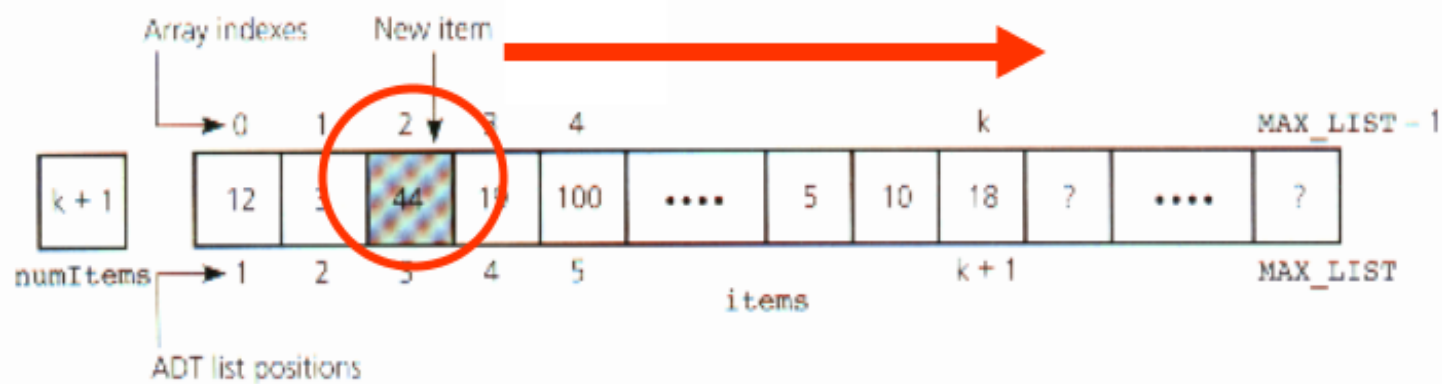
- We need to keep track of the number of items assigned to the array. We will refer to this as its **logical size**

How will this work?

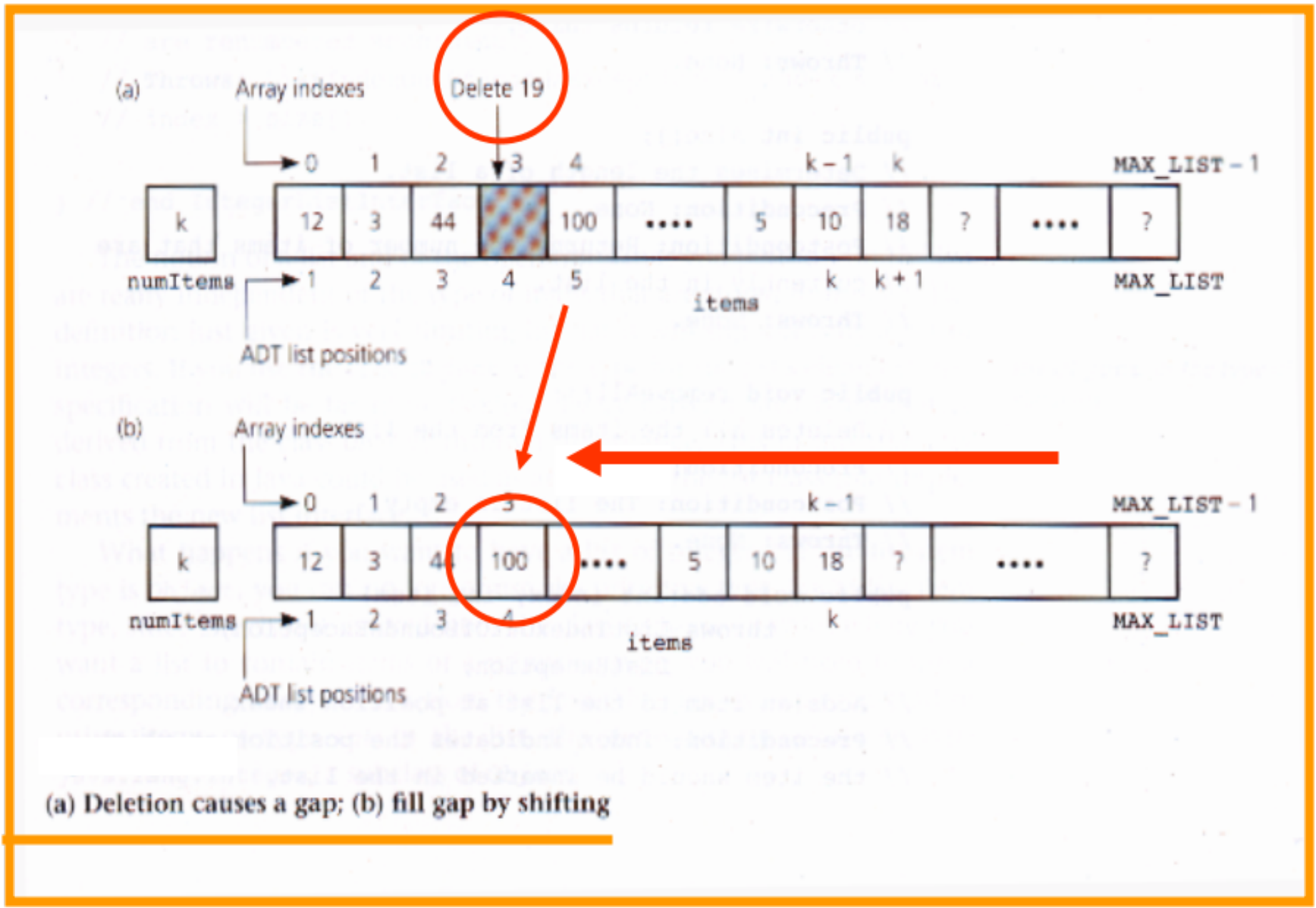
Look at the diagrams:



An array-based implementation of the ADT list



Shifting items for insertion at position 3

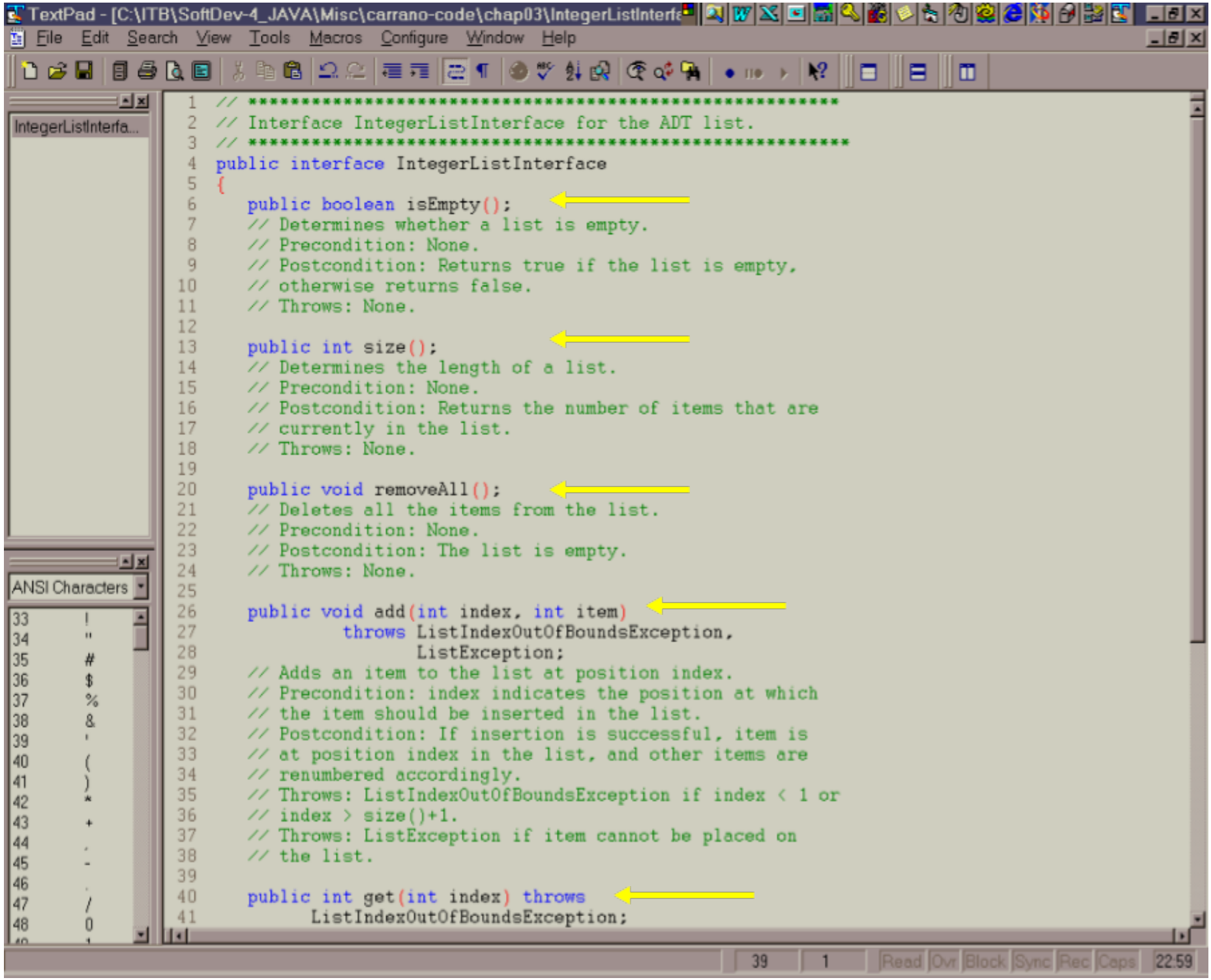


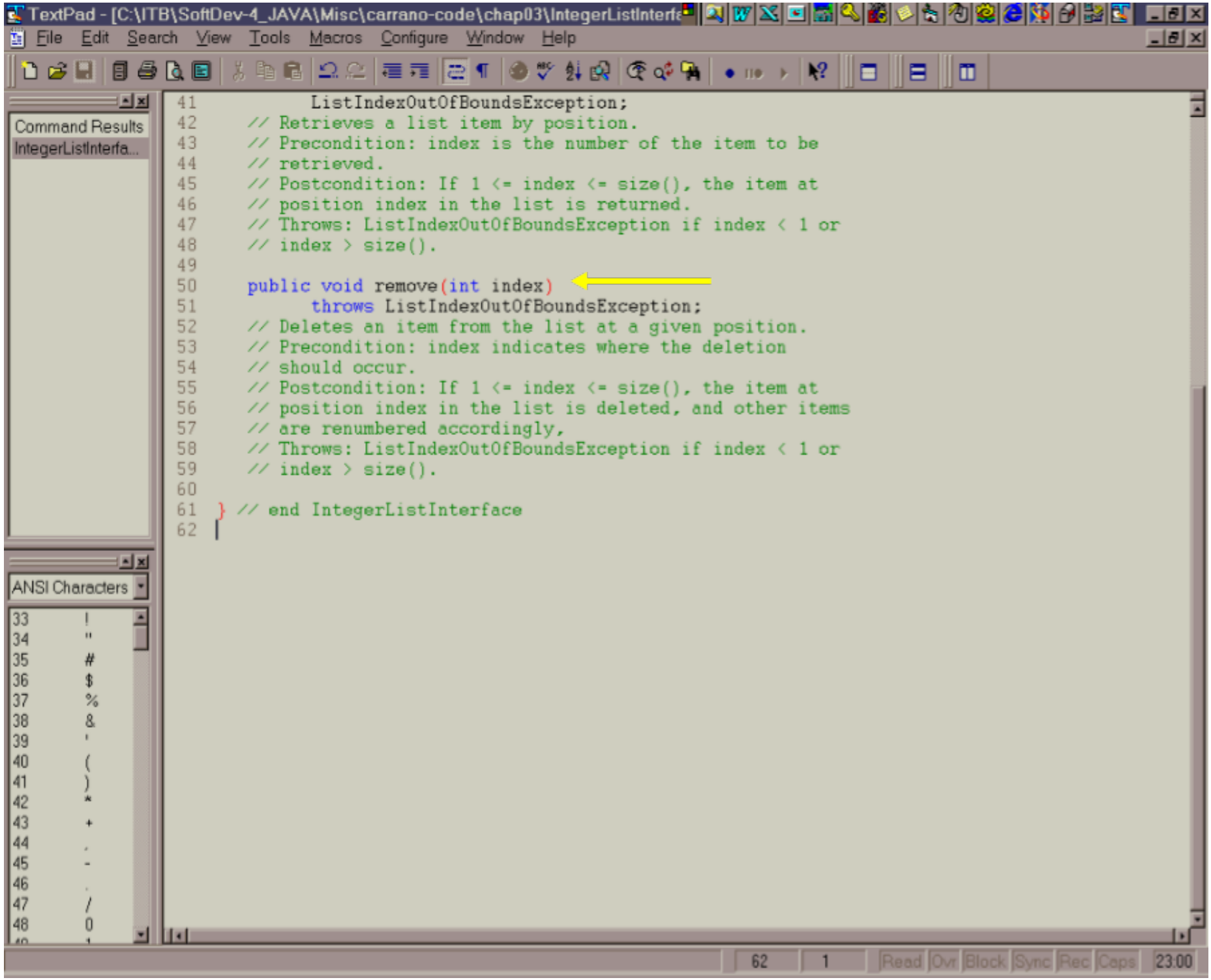
(a) Deletion causes a gap; (b) fill gap by shifting

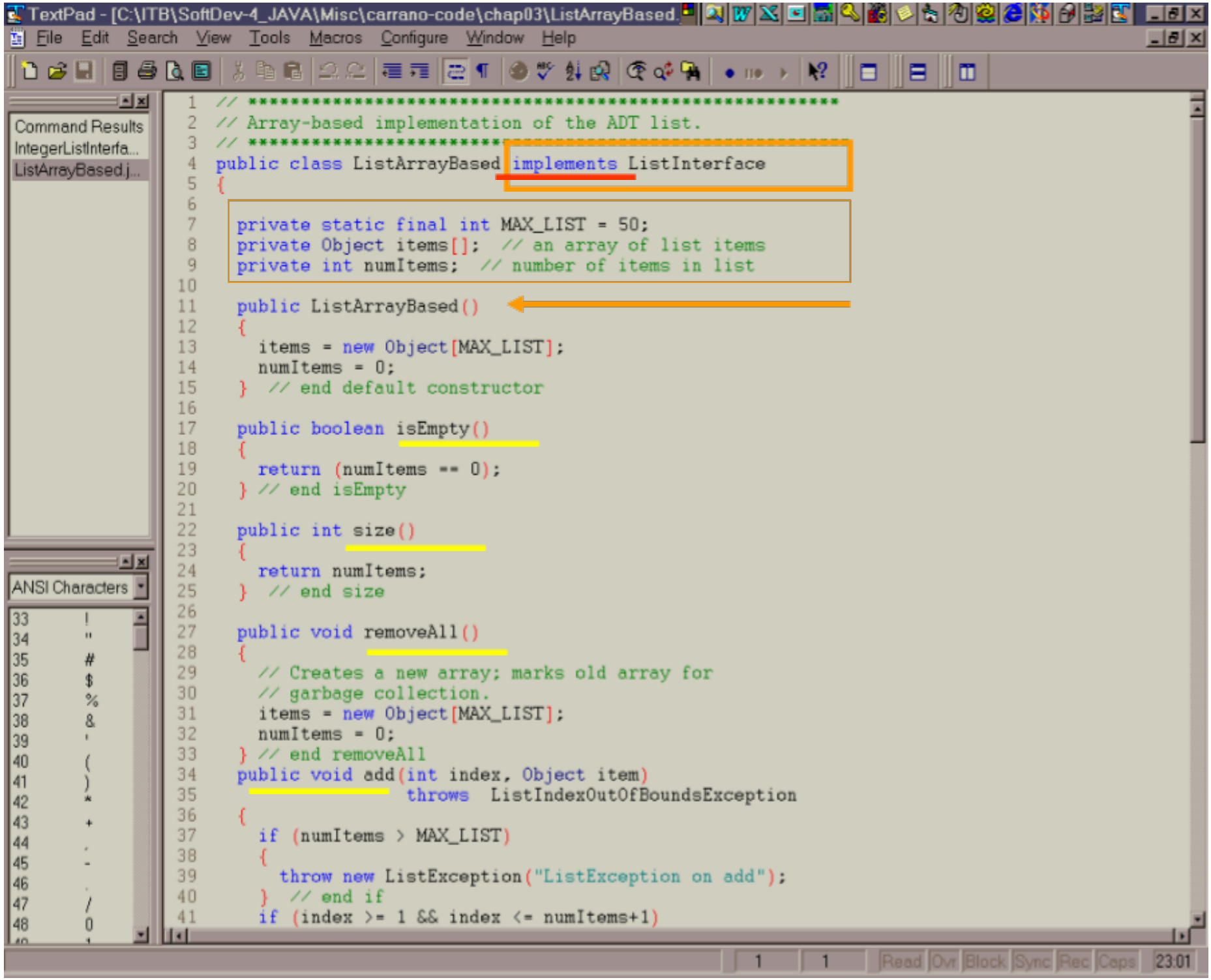
- Implement each ADT operation as a **method of a class**.
- Each operation will require access to both the array items and the list's length **numItems**, so these need to be **data fields of the class**.
- These **data fields of the class** should be **hidden** so make them **private**.
- Access to these is via the operations (= **public methods**)
- Define a translate (position) operation (= method) which returns the **index of the array element** that contains the list item at position index. That is, **translate(position)** returns the index value position - 1.
- **Throw exceptions as appropriate**, i.e. if an index is out of range.

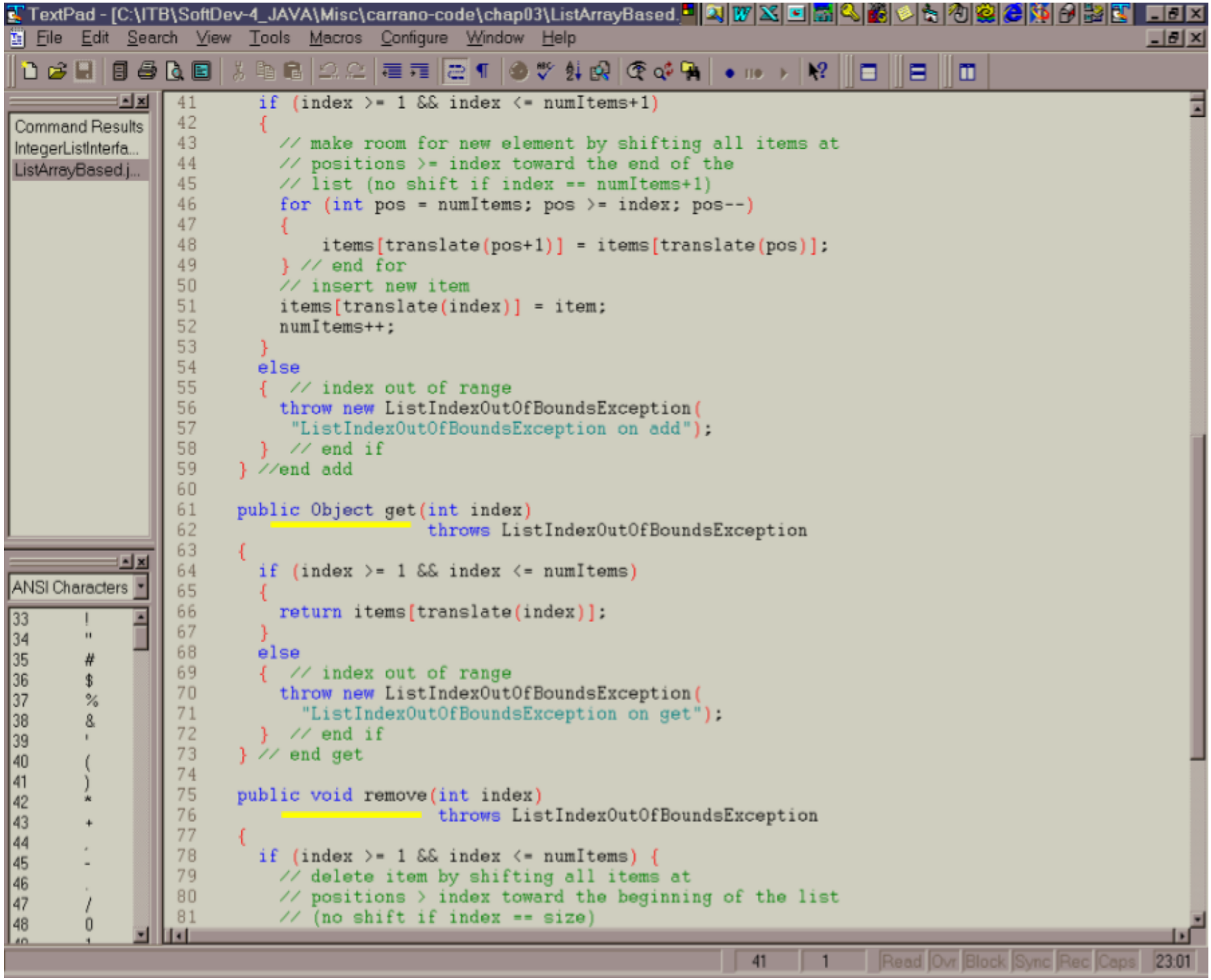
The following interface provides the specification for the list operation.

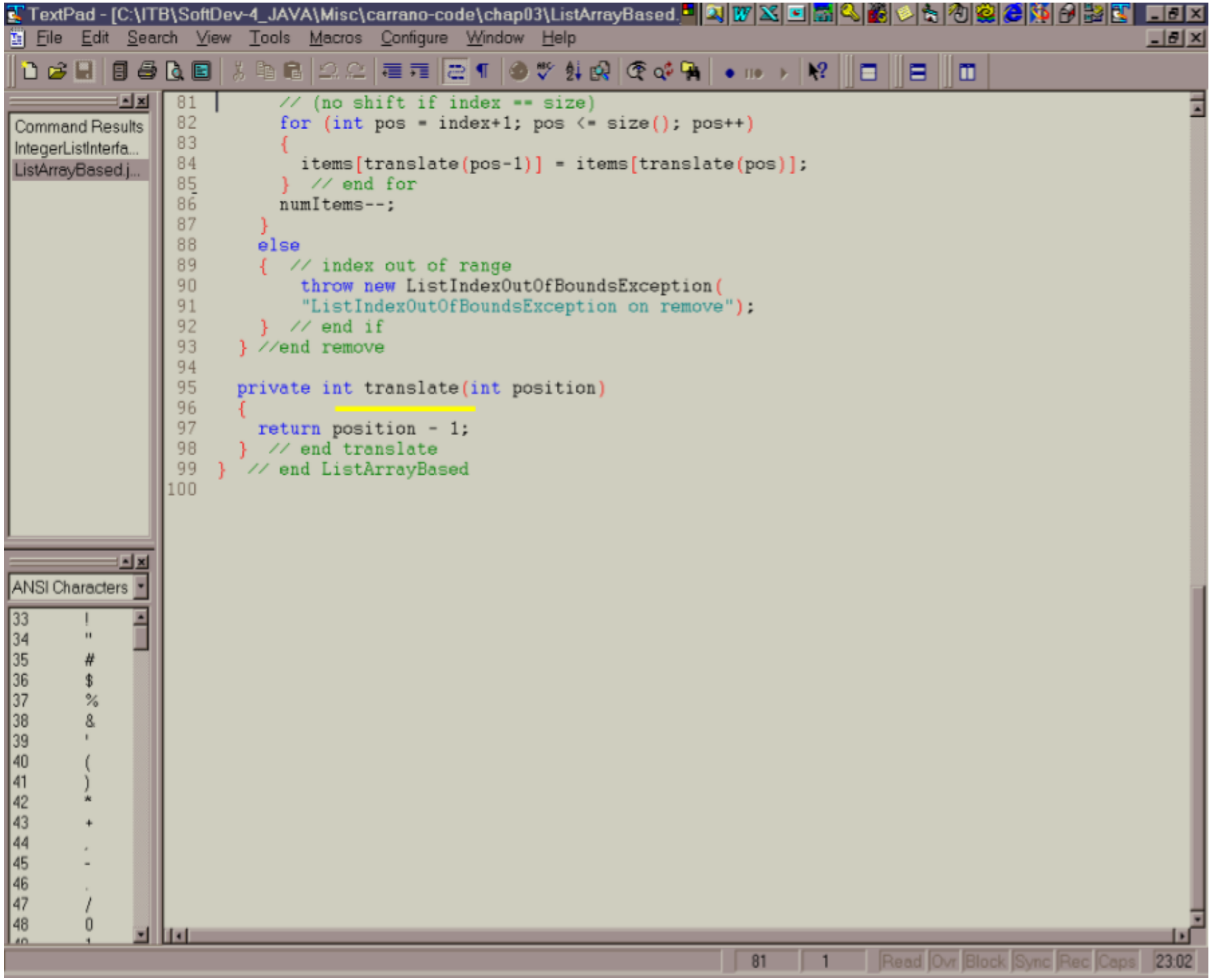
The ADT operation **createList** does not appear in the interface because we expect it to be implemented as a constructor.


TextPad - [C:\ITB\SoftDev-4_JAVA\Misc\carrano-code\chap03\IntegerListInterfa
File Edit Search View Tools Macros Configure Window Help
IntegerListInterfa...
ANSI Characters
// **
// Interface IntegerListInterface for the ADT list.
// **
public interface IntegerListInterface
{
public boolean isEmpty();
// Determines whether a list is empty.
// Precondition: None.
// Postcondition: Returns true if the list is empty,
// otherwise returns false.
// Throws: None.
public int size();
// Determines the length of a list.
// Precondition: None.
// Postcondition: Returns the number of items that are
// currently in the list.
// Throws: None.
public void removeAll();
// Deletes all the items from the list.
// Precondition: None.
// Postcondition: The list is empty.
// Throws: None.
public void add(int index, int item)
throws ListIndexOutOfBoundsException,
ListException;
// Adds an item to the list at position index.
// Precondition: index indicates the position at which
// the item should be inserted in the list.
// Postcondition: If insertion is successful, item is
// at position index in the list, and other items are
// renumbered accordingly.
// Throws: ListIndexOutOfBoundsException if index < 1 or
// index > size()+1.
// Throws: ListException if item cannot be placed on
// the list.
public int get(int index) throws
ListIndexOutOfBoundsException;
39 1 Read Ovr Block Sync Rec Caps 22:59


TextPad - [C:\ITB\SoftDev-4_JAVA\Misc\carrano-code\chap03\IntegerListInterfa
File Edit Search View Tools Macros Configure Window Help
Command Results
IntegerListInterfa...
ANSI Characters
33 !
34 "
35 #
36 $
37 %
38 &
39 '
40 (
41)
42 *
43 +
44 ,
45 -
46 .
47 /
48 0
41 ListIndexOutOfBoundsException;
42 // Retrieves a list item by position.
43 // Precondition: index is the number of the item to be
44 // retrieved.
45 // Postcondition: If 1 <= index <= size(), the item at
46 // position index in the list is returned.
47 // Throws: ListIndexOutOfBoundsException if index < 1 or
48 // index > size().
49
50 public void remove(int index)
51 throws ListIndexOutOfBoundsException;
52 // Deletes an item from the list at a given position.
53 // Precondition: index indicates where the deletion
54 // should occur.
55 // Postcondition: If 1 <= index <= size(), the item at
56 // position index in the list is deleted, and other items
57 // are renumbered accordingly,
58 // Throws: ListIndexOutOfBoundsException if index < 1 or
59 // index > size().
60
61 } // end IntegerListInterface
62
62 1 Read Ovr Block Sync Rec Caps 23:00


TextPad - [C:\ITB\SoftDev-4_JAVA\Misc\carrano-code\chap03\ListArrayBased.
File Edit Search View Tools Macros Configure Window Help
Command Results
IntegerListInterfa...
ListArrayBased.j...
ANSI Characters
// **
// Array-based implementation of the ADT list.
// **
public class ListArrayBased implements ListInterface
{
private static final int MAX_LIST = 50;
private Object items[]; // an array of list items
private int numItems; // number of items in list
public ListArrayBased()
{
items = new Object[MAX_LIST];
numItems = 0;
} // end default constructor
public boolean isEmpty()
{
return (numItems == 0);
} // end isEmpty
public int size()
{
return numItems;
} // end size
public void removeAll()
{
// Creates a new array; marks old array for
// garbage collection.
items = new Object[MAX_LIST];
numItems = 0;
} // end removeAll
public void add(int index, Object item)
throws ListIndexOutOfBoundsException
{
if (numItems > MAX_LIST)
{
throw new ListException("ListException on add");
} // end if
if (index >= 1 && index <= numItems+1)
Read Ovr Block Sync Rec Caps
23:01


TextPad - [C:\ITB\SoftDev-4_JAVA\Misc\carrano-code\chap03\ListArrayBased.
File Edit Search View Tools Macros Configure Window Help
Command Results
IntegerListInterfa...
ListArrayBased.j...
ANSI Characters
33 !
34 "
35 #
36 $
37 %
38 &
39 '
40 (
41)
42 *
43 +
44 ,
45 -
46 .
47 /
48 0
41 if (index >= 1 && index <= numItems+1)
42 {
43 // make room for new element by shifting all items at
44 // positions >= index toward the end of the
45 // list (no shift if index == numItems+1)
46 for (int pos = numItems; pos >= index; pos--)
47 {
48 items[translate(pos+1)] = items[translate(pos)];
49 } // end for
50 // insert new item
51 items[translate(index)] = item;
52 numItems++;
53 }
54 else
55 { // index out of range
56 throw new ListIndexOutOfBoundsException(
57 "ListIndexOutOfBoundsException on add");
58 } // end if
59 } //end add
60
61 public Object get(int index)
62 throws ListIndexOutOfBoundsException
63 {
64 if (index >= 1 && index <= numItems)
65 {
66 return items[translate(index)];
67 }
68 else
69 { // index out of range
70 throw new ListIndexOutOfBoundsException(
71 "ListIndexOutOfBoundsException on get");
72 } // end if
73 } // end get
74
75 public void remove(int index)
76 throws ListIndexOutOfBoundsException
77 {
78 if (index >= 1 && index <= numItems) {
79 // delete item by shifting all items at
80 // positions > index toward the beginning of the list
81 // (no shift if index == size)
41 1 Read Ovr Block Sync Rec Caps 23:01


TextPad - [C:\ITB\SoftDev-4_JAVA\Misc\carrano-code\chap03\ListArrayBased.
File Edit Search View Tools Macros Configure Window Help
Command Results
IntegerListInterfa...
ListArrayBased.j...
ANSI Characters
33 !
34 "
35 #
36 $
37 %
38 &
39 '
40 (
41)
42 *
43 +
44 ,
45 -
46 .
47 /
48 0
// (no shift if index == size)
for (int pos = index+1; pos <= size(); pos++)
{
items[translate(pos-1)] = items[translate(pos)];
} // end for
numItems--;
}
else
{ // index out of range
throw new ListIndexOutOfBoundsException(
"ListIndexOutOfBoundsException on remove");
} // end if
} //end remove
private int translate(int position)
{
return position - 1;
} // end translate
} // end ListArrayBased
81
1
Read Ovr Block Sync Rec Caps
23:02

Summary

To implement an ADT, given implementation-independent specifications of the ADT operations…

- Choose a data structure to contain the data.
- Define and implement a class within a Java source file.
- The ADT operations are public methods within the class.
- The ADT data is represented as data fields that are typically **private**.
- Implement the class's methods within an implementation file.
- The program that uses the class will be able to access the data only by using the ADT operations.

Programs to do this week:

- These will be demonstrated by you in class
- Be prepared to answer questions on your work
- Your answers will determine the marks awarded

1 Create an ADT List implemented over an array

- Write out the specifications for the ADT List (in the notes)
- Implement the ADT List as a Java class over an array such that the "walls" are reinforced (in the notes)
- This class must implement the ADT operatio ns as public methods of the class (in the notes)
- Write a small driver program that demonstrates the use of the ADT class and therefore the ADT List (not in the notes)
- This program should print results to the screen in some simple manner