Lists

Chapter 12
Contents

• Specifications for the ADT List
• Using the ADT List
• Java Class Library: The Interface List
• Java Class Library: The Class ArrayList
Objectives

- Describe the ADT list
- Use the ADT list in a Java program
Lists

• A collection
  ▪ Has order … which may or may not matter
  ▪ Additions may come anywhere in list

Figure 12-1 A to-do list
Lists

• Have we seen lists before?
  ▪ What functions were available for lists in Scheme?

• Typical actions with lists
  ▪ Add item at end (although can add anywhere)
  ▪ Remove an item (or all items)
  ▪ Replace an item
  ▪ Look at an item (or all items)
  ▪ Search for an entry
  ▪ Count how many items in the list
  ▪ Check if list is empty
ADT List

• Data
  ▪ A collection of objects in a **specific order** and having the same data type
  ▪ The number of objects in the collection

• Operations
  ▪ add(newEntry)
  ▪ add(newPosition, newEntry)
  ▪ remove(givenPosition)
  …

• Q? How are lists different than Sets?
ADT List

• Operations (ctd.)
  ▪ clear()
  ▪ replace(givenPosition, newEntry)
  ▪ getEntry(givenPosition)
  ▪ contains(anEntry)
  ▪ getLength()
  ▪ isEmpty()
  ▪ toArray()

• Q? Name a few more possible operations on Lists.
  ▪ For example, analogs of set intersection, union, ...

• Q? Compare with Scheme functions on lists. Which of the above methods are available as primitive functions in Scheme? Which ones are not?

• Q? Identify ListInterface methods corresponding to following Scheme functions:
  • (a.) car  (b.) cdr  (c.) cons  (d.) list  (e.) null  (f.) length  (g.) reverse,  (h.) append
  • (i.) list-ref  (j.) member  (k.) delete  (l.) map  (m.) set-car!  (n.) set-cdr!

Fig. 12-2 Effect of ADT list operations on an initially empty list

Note: Java List interface may count position starting with 0.
List

• View list interface, Listing 12-1
• Using the ADT List
  ▪ Don’t need to know *how* of implementation
  ▪ Only need to know *what* it does
• Consider keeping list of finishers of a running race
  ▪ View client code, Listing 12-2
  ▪ Output

Note: Code on last few slides. Code files must be in same folder as PowerPoint files for links to work.
public interface ListInterface <T> {
    public void add(T newEntry);
    public void add(int newPosition, T newEntry);
    public T remove(int givenPosition);
    public void clear();
    public boolean replace(int givenPosition, T newEntry);
    public T getEntry(int givenPosition);
    public boolean contains(T anEntry);
    public int getLength();
    public boolean isEmpty();
    public T[] toArray();
}

// Q? Compare with java.util.List interface using java-doc, e.g.,
• Positions inside a list start with a 0
• Method get(int index) instead of getEntry(int givenPosition), size() instead of getLength()
• Many more methods, e.g., equals, indexOf, iterator(), listIterator(),
    • Remove(Object O), removeAll(Object O), retainAll(Collection c), subList(int fromIndex, toIndex)
Old Java version uses Object class instead of generic type

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Generic Data Types (Appendix B, pp. B-26-B-29)

• Generic Data Type
  • A placeholder instead of an actual type in definition of a class or interface
  • Examples: public class MyClass<T>; // definition header
    • MyClass<String> item = new MyClass<String>() ; // use with a reference type
    • MyClass<int> item = new MyClass<int>() ; // Illegal with a primitive type!
    • MyClass<Integer> item = new MyClass<Integer>;

• Restriction on <>
  • <T> follows identifier name in the class header
  • <T> does not follow constructor names in their definitions
  • T (not <T>) can be used to type variables, parameters or method return value

• Pros
  • Generics are better than “Object” by providing restrictions and type-checking
  • See Question 5 in self-test exercises on later slide

• Caution
  • Generics are hard to understand for beginners
  • It is useful to test those with a few specific types, e.g., Integer, String, etc.
  • Try “Find and replace” strategy to review code

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Figure 12-3 A list of numbers that identify runners in the order in which they finished a race.
public class ListClient {

// Assume class AList implements interface ListInterface by providing method definitions
public static void main(String[] args) { testList(); }

public static void testList() {
    ListInterface<String> runnerList = new AList<String>();
    runnerList.add("16"); runnerList.add("4"); runnerList.add("33"); runnerList.add("27");
    displayList(runnerList);
}

public static void displayList(ListInterface<String> list) {
    int numberOfEntries = list.getLength();
    System.out.println("The list contains "+ numberOfEntries + " entries, as follows: ");
    for (int position = 1; position <= numberOfEntries; position++)
        System.out.println( list.getEntry(position) + " is entry "+ position);
}

} // Output on page 314 or Textbook.

// Q? Is runnerList.add(16) a valid call to add() method ? Why?
// Q? displayList() assumes a list of Strings. Generalize it for lists of any object type/class.
// Q? runnerList object is constrained to call methods in ListInterface.
// Generalize testList() to allow calls to other methods defined in AList class.

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Java Class Library: The Interface List

• Method headers
  ▪ public T remove(int index)
  ▪ public void clear()
  ▪ public boolean isEmpty()
  ▪ public boolean add(T newEntry)
  ▪ public void add(int index, T newEntry)
  ...
Java Class Library: The Interface List

- Method headers (ctd.)
  - `public T set(int index, T anEntry)` // like replace
  - `public T get(int index)` // like getEntry
  - `public boolean contains (Object anEntry)`
  - `public int size()` // like getLength
Java Class Library: The Interface ArrayList

• Implementation of ADT list with resizable array
  ▪ Implements java.util.list

• Constructors available
  ▪ public ArrayList()
  ▪ public ArrayList (int initialCapacity)
Self-Test Questions

Q2. Write psuedo-code to swap 3\textsuperscript{rd} and 7\textsuperscript{th} entries in a list of 10 objects.

Q4. Write statements to swap Ellen and Bob, and then swap Ellen and Drew for sorting!

// Assume class AList implements interface ListInterface by providing method definitions
ListInterface<String> runnerList = new AList<String>();
runnerList.add("Amy"); runnerList.add("Ellen");
runnerList.add("Bob"); runnerList.add("Drew");
// Statements to swap Ellen and Bob, and then swap Ellen and Drew for sorting!

Q4b. Propose a new method for ListInterface to simplify code needed for Q4.

Q5. Suppose return type of getEntry() was Object instead of generic type. Would this change affect how you use the method? Will the last statement in following code be correct?

ListInterface<Name> nameList = new AList<Name>();
Name amy = new Name("Amy", "smith");
nameList.add(amy);
nameList.add(new Name("Tina", "Drexel") );
nameList.add(new Name("Robert", "Jones");
Name secondName = nameList.getEntry(2);
End

Chapter 12