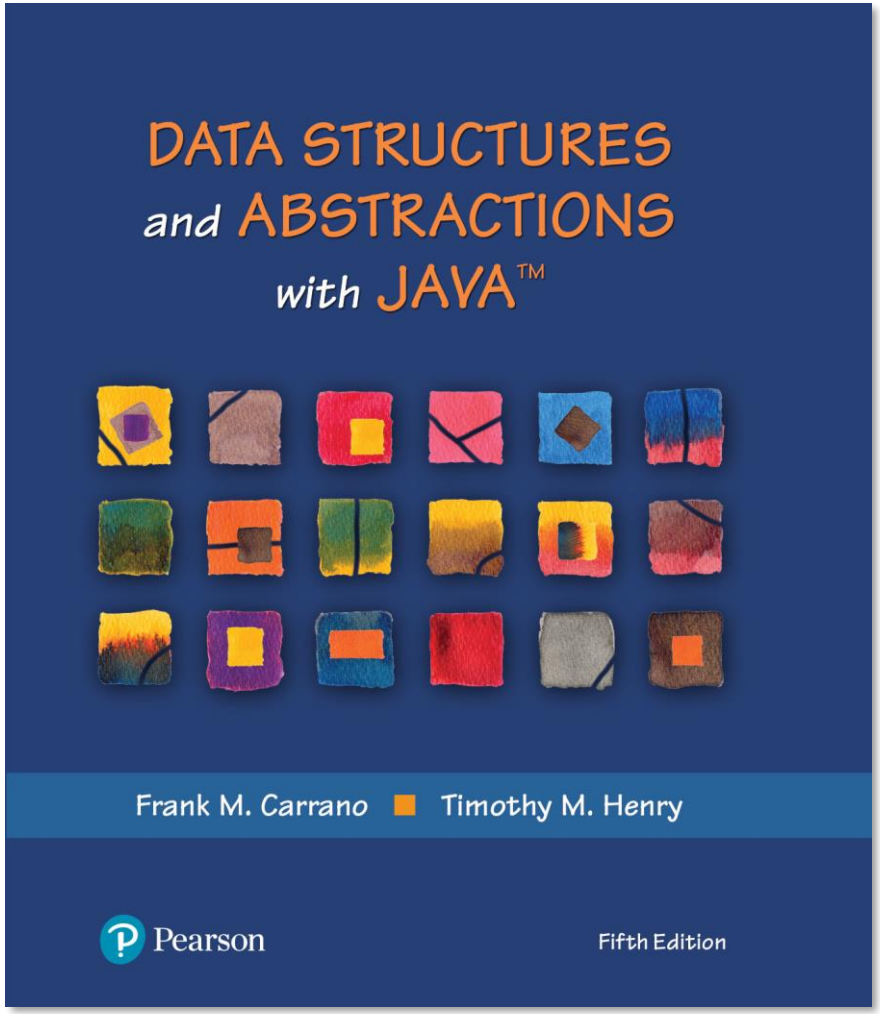
Data Structures and Abstractions with Java™

5th Edition



Chapter 10

Lists

Lists

- A way to organize data
- Examples
  - To-do list
  - Gift lists
  - Grocery Lists
- Items in list have position
  - May or may not be important
- Items may be added anywhere

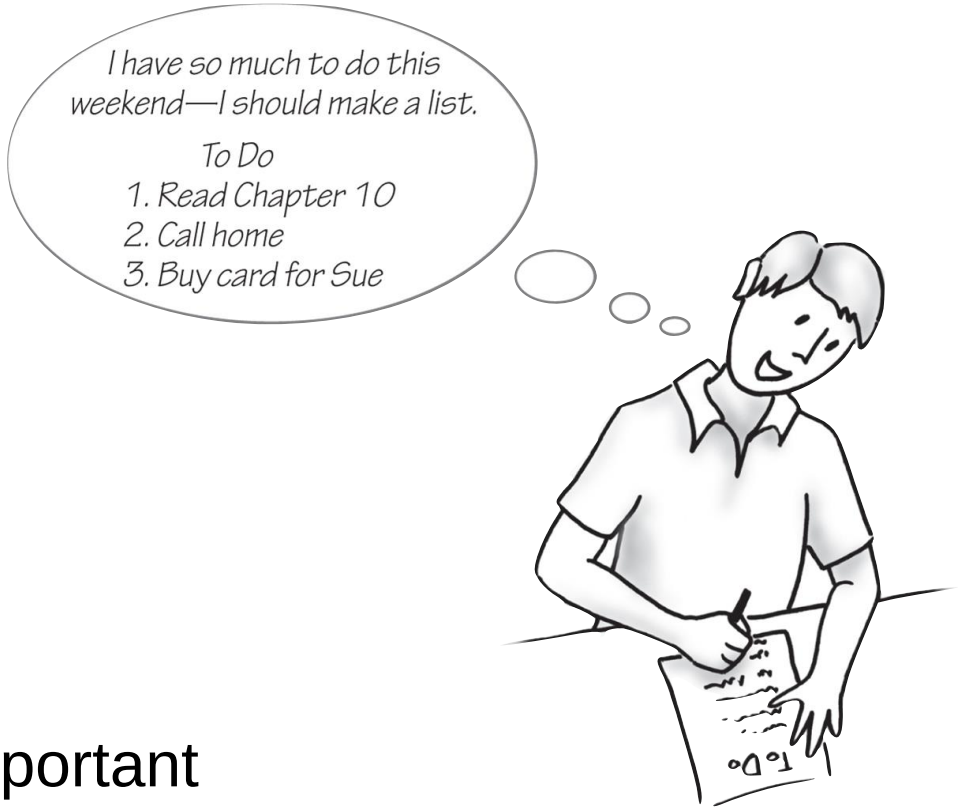


FIGURE 10-1
A to-do list

Specifications for the ADT List

- `add(newEntry)`
- `add(newPosition, newEntry)`
- `remove(givenPosition)`
- `clear()`
- `replace(givenPosition, newEntry)`
- `getEntry(givenPosition)`
- `toArray()`
- `contains(anEntry)`
- `getLength()`
- `isEmpty()`

Specifications for the ADT List

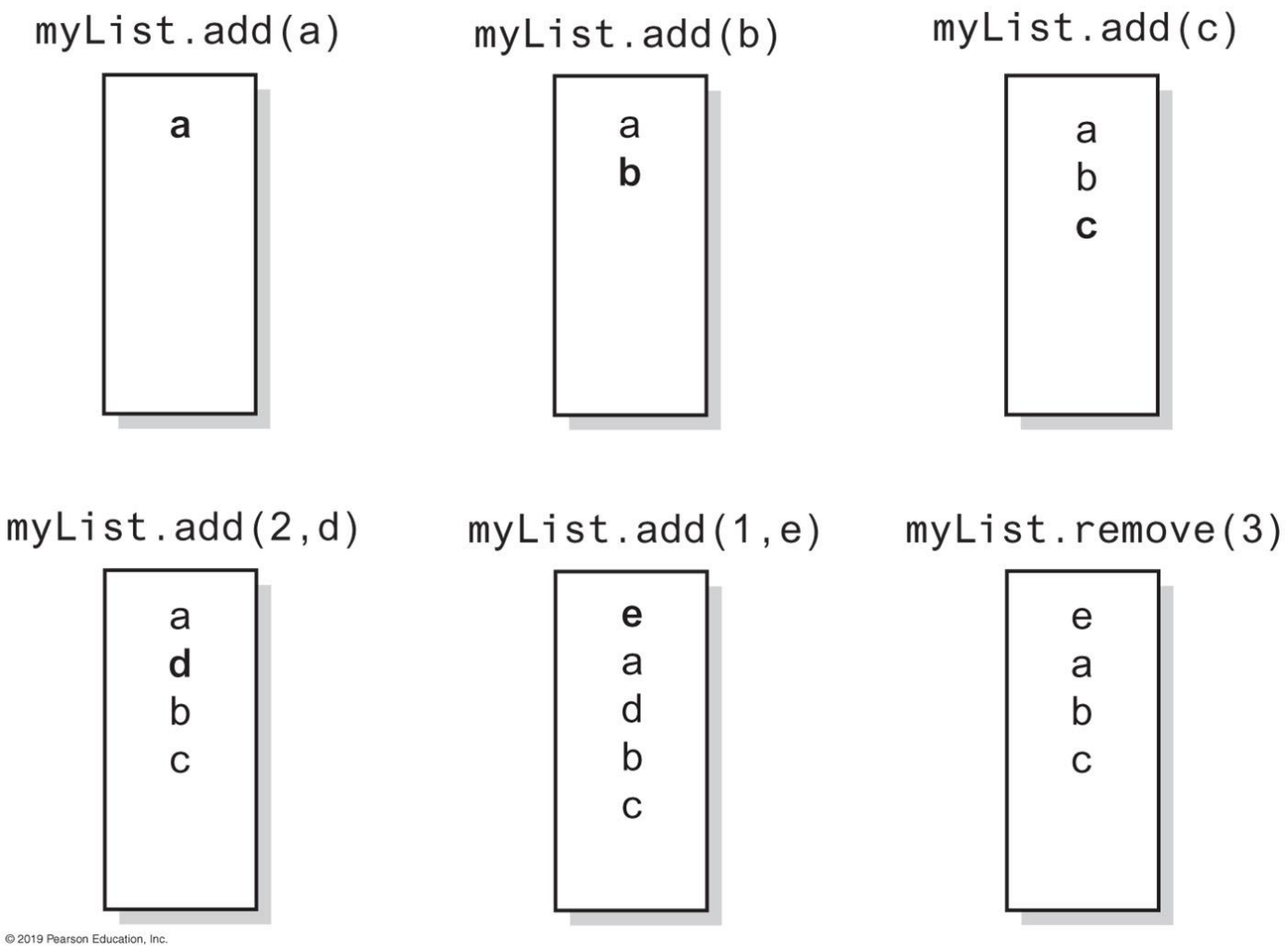


FIGURE 10-2 The effect of ADT list operations on an initially empty list

Specifications for the ADT List

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

Pseudocode	UML	Description
add(newEntry)	**+add(newEntry: T): void**	Task: Adds newEntry to the end of the list. Input: newEntry is an object. Output: None.
add(newPosition, anEntry)	**+add(newPosition: integer, anEntry: T): void**	Task: Adds newEntry at position newPosition within the list. Position 1 indicates the first entry in the list. Input: newPosition is an integer, newEntry is an object. Output: Throws an exception if newPosition is invalid for this list before the operation.
remove(givenPosition)	**+remove(givenPosition: integer): T**	Task: Removes and returns the entry at position givenPosition. Input: givenPosition is an integer. Output: Either returns the removed entry or throws an exception if givenPosition is invalid for this list. Note that any value of givenPosition is invalid if the list is empty before the operation.
clear()	**+clear(): void**	Task: Removes all entries from the list. Input: None. Output: None.
replace(givenPosition, anEntry)	**+replace(givenPosition: integer, anEntry: T): T**	Task: Replaces the entry at position givenPosition with newEntry. Input: givenPosition is an integer, newEntry is an object. Output: Either returns the replaced entry or throws an exception if givenPosition is invalid for this list. Note that any value of givenPosition is invalid if the list is empty before the operation.

Specifications for the ADT List

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

Pseudocode	UML	Description
getEntry(givenPosition)	**+getEntry(givenPosition: integer): T**	Task: Retrieves the entry at position givenPosition. Input: givenPosition is an integer. Output: Either returns the entry at position givenPosition or throws an exception if givenPosition is invalid for this list. Note that any value of givenPosition is invalid if the list is empty before the operation.
toArray()	**+toArray: T[ ]**	Task: Retrieves all entries that are in the list in the order in which they occur. Input: None. Output: Returns a new array of the entries currently in the list.
contains(anEntry)	**+contains(anEntry: T): boolean**	Task: Sees whether the list contains anEntry. Input: anEntry is an object. Output: Returns true if anEntry is in the list, or false if not.
getLength()	**+getLength(): integer**	Task: Gets the number of entries currently in the list. Input: None. Output: Returns the number of entries currently in the list.
isEmpty()	**+isEmpty(): boolean**	Task: Sees whether the list is empty. Input: None. Output: Returns true if the list is empty, or false if not.

LISTING 10-1 The interface `ListInterface` (Part 1)

```
** An interface ADT list. Entries in a list have positions that begin with 1.  */
public interface ListInterface<T> {
 /** Adds a new entry to the end of this list.
     Entries currently in the list are unaffected.
     The list's size is increased by 1.
     @param newEntry  The object to be added as a new entry. */
  public void add(T newEntry);

  /** Adds a new entry at a specified position within this list.
     Entries originally at and above the specified position
     are at the next higher position within the list.
     The list's size is increased by 1.
     @param newPosition  An integer that specifies the desired
                         position of the new entry.
     @param newEntry     The object to be added as a new entry.
     @throws  IndexOutOfBoundsException if either
           newPosition < 1 or newPosition > getLength() + 1. */
  public void add(int newPosition, T newEntry);

  /** Removes the entry at a given position from this list.
     Entries originally at positions higher than the given
     position are at the next lower position within the list,
     and the list's size is decreased by 1.
     @param givenPosition  An integer that indicates the position of
                           the entry to be removed.
     @return  A reference to the removed entry.
     @throws  IndexOutOfBoundsException if either
           givenPosition < 1 or givenPosition > getLength(). */
  public T remove(int givenPosition);
```

LISTING 10-1 The interface `ListInterface` (Part 2)

```
/** Removes all entries from this list. */
public void clear();

/** Replaces the entry at a given position in this list.
   @param givenPosition  An integer that indicates the position of
                         the entry to be replaced.
   @param newEntry  The object that will replace the entry at the
                    position givenPosition.
   @return  The original entry that was replaced.
   @throws  IndexOutOfBoundsException if either
            givenPosition < 1 or givenPosition > getLength(). */
public T replace(int givenPosition, T newEntry);

/** Retrieves the entry at a given position in this list.
   @param givenPosition  An integer that indicates the position of
                         the desired entry.
   @return  A reference to the indicated entry.
   @throws  IndexOutOfBoundsException if either
            givenPosition < 1 or givenPosition > getLength(). */
public T getEntry(int givenPosition);

/** Retrieves all entries that are in this list in the order in which
   they occur in the list.
   @return  A newly allocated array of all the entries in the list.
            If the list is empty, the returned array is empty. */
public T[] toArray();

/** Sees whether this list contains a given entry.
   @param anEntry  The object that is the desired entry.
   @return  True if the list contains anEntry, or false if not. */
public boolean contains(T anEntry);
```

LISTING 10-1 The interface `ListInterface` (Part 3)

```
  /** Gets the length of this list.
      @return  The integer number of entries currently in the list. */
  public int getLength();

  /** Sees whether this list is empty.
      @return  True if the list is empty, or false if not. */
  public boolean isEmpty();
} // end ListInterface
```

Using the ADT List

FIGURE 10-3 A list of numbers that identify runners in the order in which they finished

Using the ADT List

```
/**   A driver that uses a list to track the
   runners in a race as they cross the finish line. */
public class RoadRace {
       public static void main(String[] args)  {
       recordWinners();
       }  // end main

       public static void recordWinners() {
     ListInterface<String> runnerList = new AList<>();
  //  runnerList has only methods in ListInterface

     runnerList.add("16"); // Winner
     runnerList.add(" 4"); // Second place
     runnerList.add("33"); // Third place
     runnerList.add("27"); // Fourth place
     displayList(runnerList);
   } // end recordWinners

   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);
     System.out.println();
   } // end displayList
} // end RoadRace
```

Program Output

```
The list contains 4 entries,
as follows:
16 is entry 1
 4 is entry 2
33 is entry 3
27 is entry 4
```

LISTING 10-2 A client of a class that implements `ListInterface`

Using the ADT List

```
// Make an alphabetical list of names as students enter a room
ListInterface<String> alphaList = new AList<>();

alphaList.add(1, "Amy");    // Amy
alphaList.add(2, "Elias");  // Amy Elias
alphaList.add(2, "Bob");    // Amy Bob Elias
alphaList.add(3, "Drew");   // Amy Bob Drew Elias
alphaList.add(1, "Aaron");  // Aaron Amy Bob Drew Elias
alphaList.add(4, "Carol");  // Aaron Amy Bob Carol Drew Elias
```

Example

Using the ADT List

```
// Make a list of names as you think of them
ListInterface<Name> nameList = new AList<>();
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);
```

A list of Name objects, rather than String

Java Class Library: The Interface `List`

```
public void add(int index, T newEntry)
public T remove(int index)
public void clear()
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
public boolean isEmpty()
```

Method headers from the interface List

Java Class Library: The Class `ArrayList`

- Available constructors
  - **`public ArrayList()`**
  - **`public ArrayList(int initialCapacity)`**
- Similar to **`java.util.vector`**
  - Can use either **`ArrayList`** or **`Vector`** as an implementation of the interface **`List`**.

End

Chapter 10