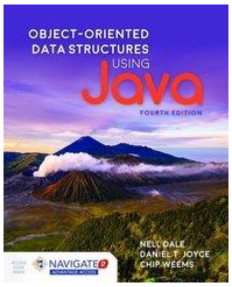
Chapter 6

# The List ADT



Section 1

## INTRODUCTION TO LISTS

## Objectives

- To study lists as an abstract data type

3

## Everyday Lists

- Dean's List
- What-to-do list
- Groceries to be purchased
- List of assignments for a course

4

## Lists

- A **list** is a collection of elements of the *same type* that exhibits a *linear relationship* among its elements
- Properties:
  – *Homogeneous*
  – *Each element except the first has a unique predecessor, and each element except the last has a unique success*or

5

## List Size

- The list **size** is the number of elements in a list
- The size can vary over time
  – Can have <u>no</u> elements
  – Can have a single element

6

## Varieties of Lists

- Unsorted list
  - A list in which elements are placed in no particular order; the only relationship between data elements is the list predecessor and successor relationships
- Sorted list
  - A list that is sorted by some property of its elements; there is an ordered relationship among the elements in the list, reflected by their relative positions

7

## Varieties of Lists

- Indexed list
  - A list in which each element has an index value associated with it
- There can be lists of lists

8

## Assumptions for Our Lists

- Our lists are unbounded
- We allow duplicate elements on our lists
- We do not support null elements
- Other than prohibiting null elements, we have *minimal preconditions* on our operations

9

## Assumptions for Our Lists

- Our sorted lists are sorted in increasing order, as defined by the `compareTo` operation applied to list objects
- The `equals` and `compareTo` methods of our sorted list elements are consistent
- In our indexed lists, the indices in use at any given time are *contiguous*, starting at 0

10

## List Formal Specification

- We define a small, but useful, set of operations for use with our lists
- We capture the formal specifications of our List ADT using the Java interface construct
- We pull common list method descriptions together into a single interface, called `ListInterface`
- We extend the `ListInterface` with a separate interface for indexed lists, since indexing a list allows additional operations

11

## List Iteration

- Because a list has a linear relationship among its elements, we can support *iteration* through a list
- **Iteration** means that we provide a mechanism to process the entire list, element by element, from the first element to the last element
- Each of our list variations provides the operations `reset` and `getNext` to support this activity

12

## `ListInterface` Operations

- **size**
  - Returns the number of elements in the list
- **add**
  - Adds an element to the list
- **contains**
  - Passed an argument and returns a `boolean` indicating whether the list contains an equivalent element

13

## `ListInterface` Operations

- **remove**
  - Passed an argument
  - If an equivalent element exists on the list, removes one instance of that element
  - Returns a `boolean` value indicating whether an element was actually removed
- **isEmpty**
  - Returns a `boolean` value indicating whether the list is empty or not

14

## `ListInterface` Operations

- **toString**
  - Returns a nicely formatted string representing the list
- **reset**
  - R-initializes the list
  - Sets the number of elements in the list to zero

15

## `IndexedListInterface` Operations

- **add**
  - Passed an element and an integer index
  - It adds the element to the list at the position index
- **set**
  - Passed an element and an integer index
  - It replaces the current element at the position index with the argument element

16

## `IndexedListInterface` Operations

- **get**
  - Passed an integer index
  - It returns the element from the list at that position
- **indexOf**
  - Passed an element
  - It returns the index of the first such matching element
  - It returns -1 if there is no matching element

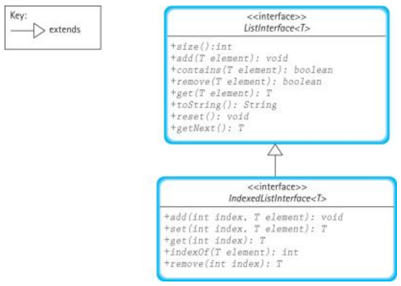
17

## `IndexedListInterface` Operations

- **remove**
  - Passed an integer index
  - It removes the element at that index
- **isFull**
  - Returns a `boolean` value indicating whether the list has reached its maximum capacity or not
- **toString**
  - Returns a nicely formatted string representing the list

18

## UML Diagram for Our List Interfaces



19