BACS2063 Data Structures and Algorithms

Sorted Lists

Chapter 7

Introduction

- For the ADT list, entries are ordered simply by their *positions*.
- However, some applications require sorted data.
  - Hence, an ADT that maintains data in sorted order would be convenient.

Learning Outcomes

At the end of this lecture, you should be able to

- Use a sorted list in a program
- Describe the differences between the ADT list and the ADT sorted list
- Implement the ADT sorted list by using an array
- Implement the ADT sorted list by using a chain of linked nodes

Specifications for the ADT Sorted List

Refer to Appendix 7.1 for ADT Sorted List

- Data
  - A collection of objects in sorted order, same data type
  - The number of objects in the collection
- Operations
  - Add a new entry
  - Remove an entry
  - Check if a certain value is contained in the list
  - Clear the list
  - Return the length of the list
  - Check if list is empty

Note: a sorted list will not let you add or replace an entry by position

Comparing Entries

- We need to be able to *compare entries* in order to determine the correct location to insert a new entry.
  - Thus, the objects in the sorted list must be ***`Comparable`***, i.e. must implement the method **`compareTo`**.
  - To enforce this requirement, we write

    ```
    <T extends Comparable<T>>
    ```

Sorted Array List Implementation

- Sample code in **\Chapter7\array**:
  - **SortedListInterface.java**
    - Note the generic type declaration in the interface header:

      ```
      <T extends Comparable<T>>
      ```

  - **SortedArrayList.java**
    - Note the generic type declaration in the class header:

      ```
      <T extends Comparable<T>>
      ```

    - Note the **new** statement to construct the array in the constructor:

      ```
      list = (T[]) new Comparable[initialCapacity];
      ```

      because the generic type enforces the requirement that the entries are *Comparable*.
  - **SortedArrayListDriver.java**

Sorted Linked List Implementation

- Sample code in folder **`\Chapter7\linked`**:
  - **`SortedLinkedList.java`**
    - **`add`** method
      - If list is in ascending order, insert new entry just before first entry not smaller than new entry
  - **`SortedListInterface.java`**

The Method `add`

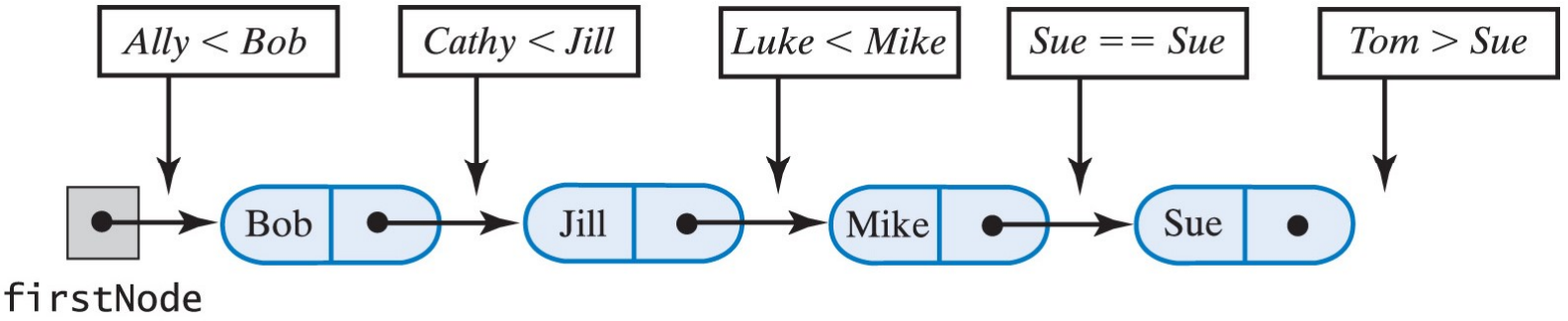

Fig. 13.1: Insertion points of names into a sorted chain of linked nodes.

Efficiency of the Linked Implementation

ADT Sorted List Operation	Array	Linked
`add(newEntry)`	O(*n*)	O(*n*)
`remove(anEntry)`	O(*n*)	O(*n*)
`getPosition(anEntry)`	O(*n*)	O(*n*)
`getEntry(givenPosition)`	O(1)	O(*n*)
`contains(anEntry)`	O(*n*)	O(*n*)
`remove(givenPosition)`	O(*n*)	O(*n*)
`display()`	O(*n*)	O(*n*)
`clear()`, `getLength()`, `isEmpty()`, `isFull()`	O(1)	O(1)

Fig. 13.5: The worst-case efficiencies of the operations on the ADT sorted list for two implementations

Exercise

An ordered list may be implemented using a *linked list* or an *array*.
Given an ordered list with the following values:

1234, 2587, 3422, 4656, 5321, 6931, 7574, 8888, 9325

Which implementation would be more efficient when performing the following operations: the *linked list* implementation or the *array* implementation?

Justify your answers in time efficiency for Big O notations.

a) Remove the value **1234** from the ordered list.

b) Add the value **5555** to the ordered list.

The Method `add`

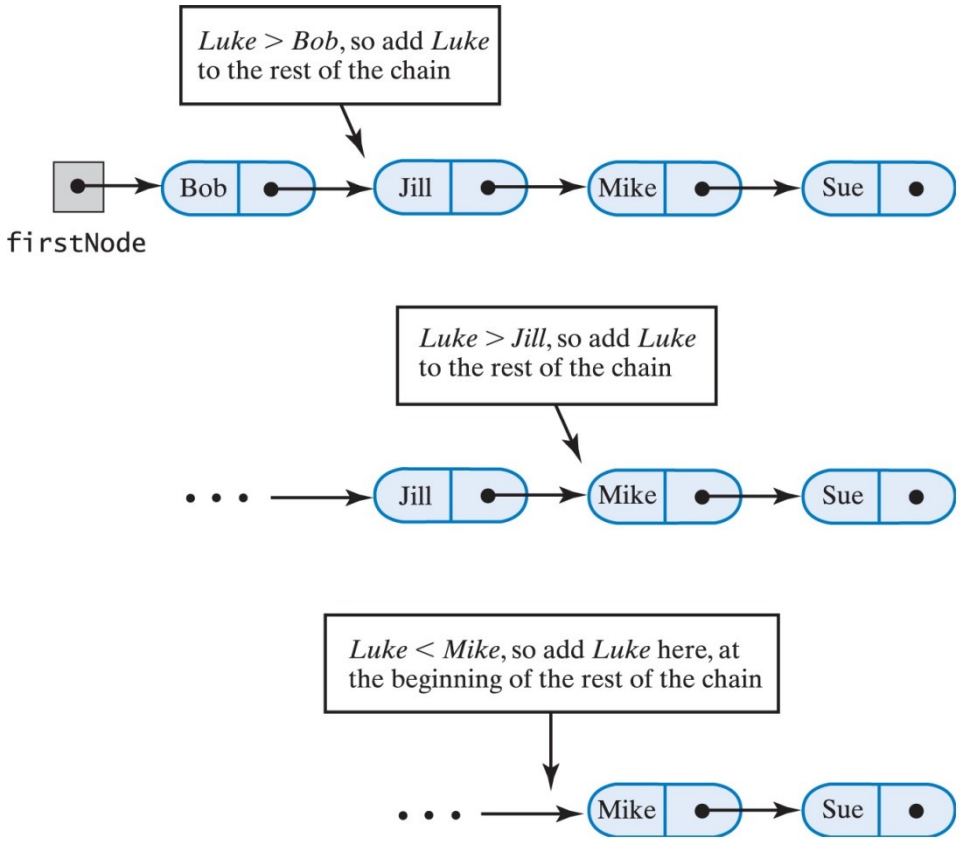

Fig. 13.2: Recursively adding *Luke* to a sorted chain of names

The Method `add`

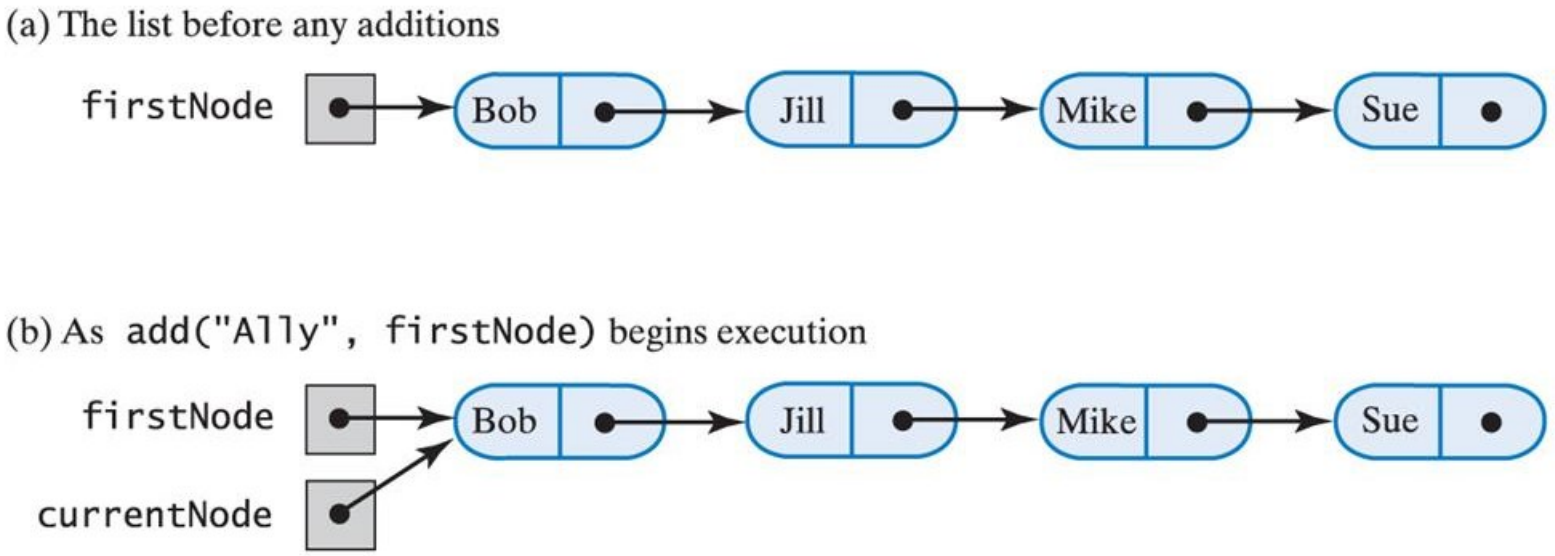

Fig. 13.3: Recursively adding *a* node at the beginning of the chain (continued →)

The Method `add`

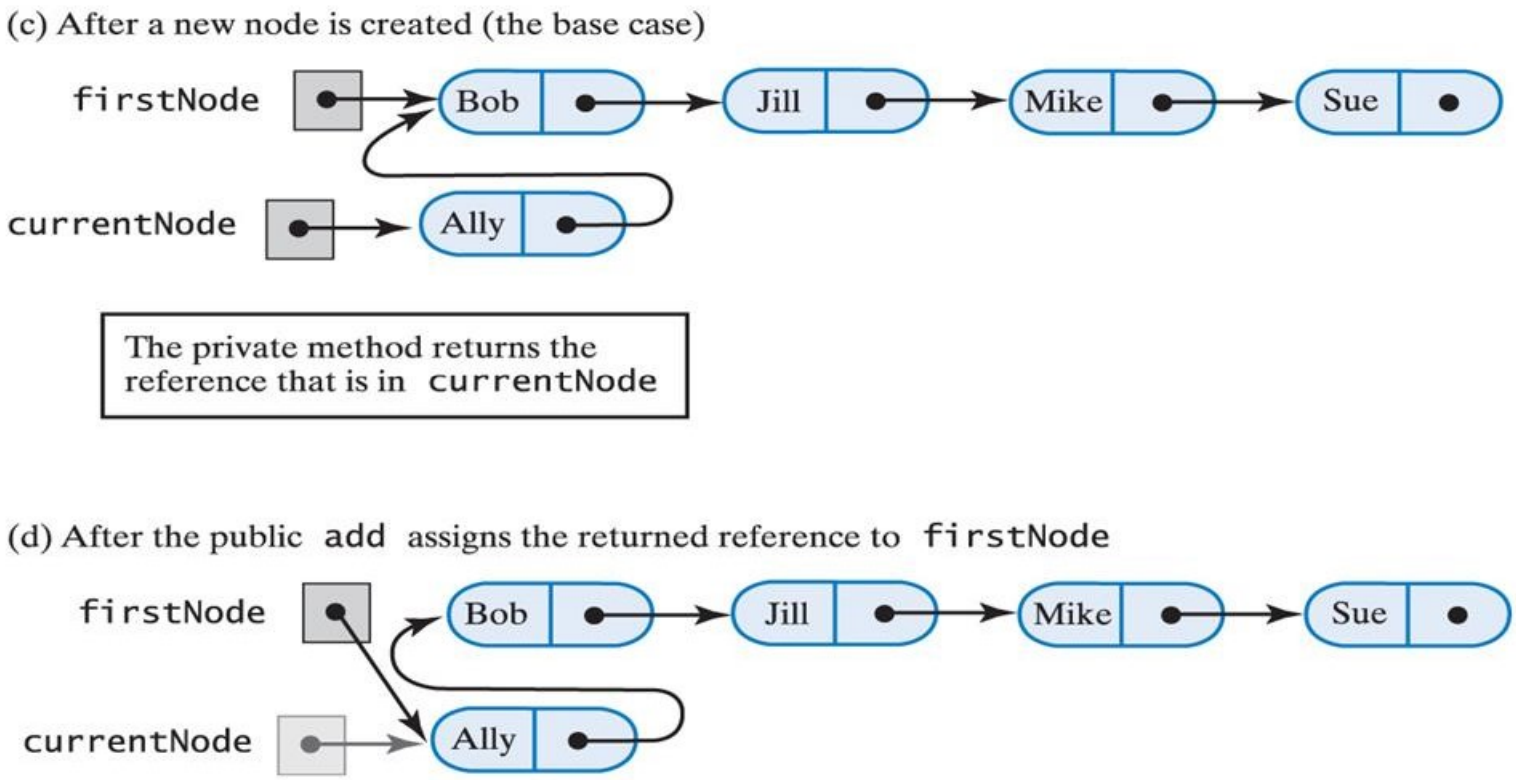

Fig. 13.3: (ctd) Recursively adding *a* node at the beginning of the chain.

The Method `add`

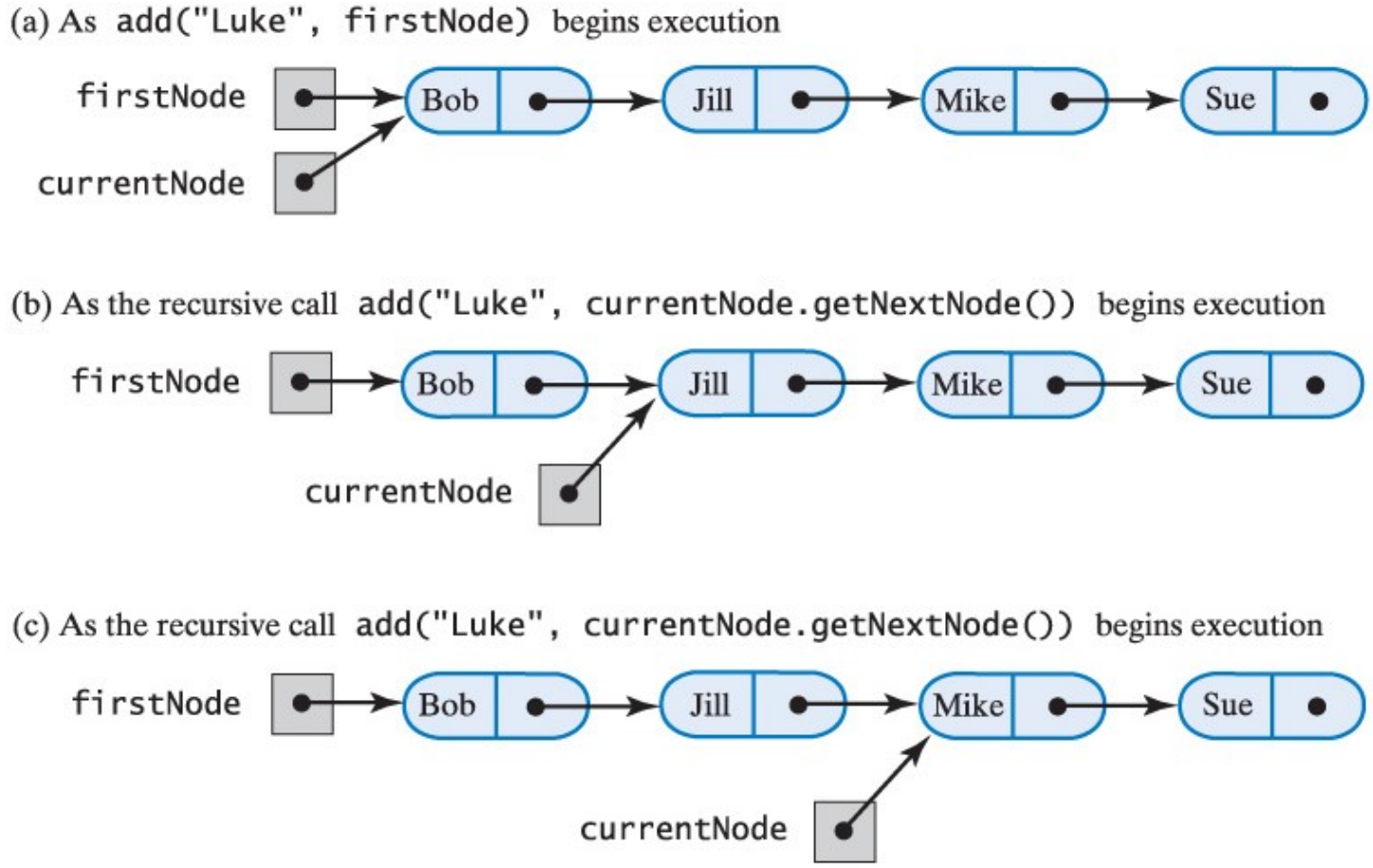

Fig. 13.4: Recursively adding a node between existing nodes in a chain (continued →)

The Method `add`

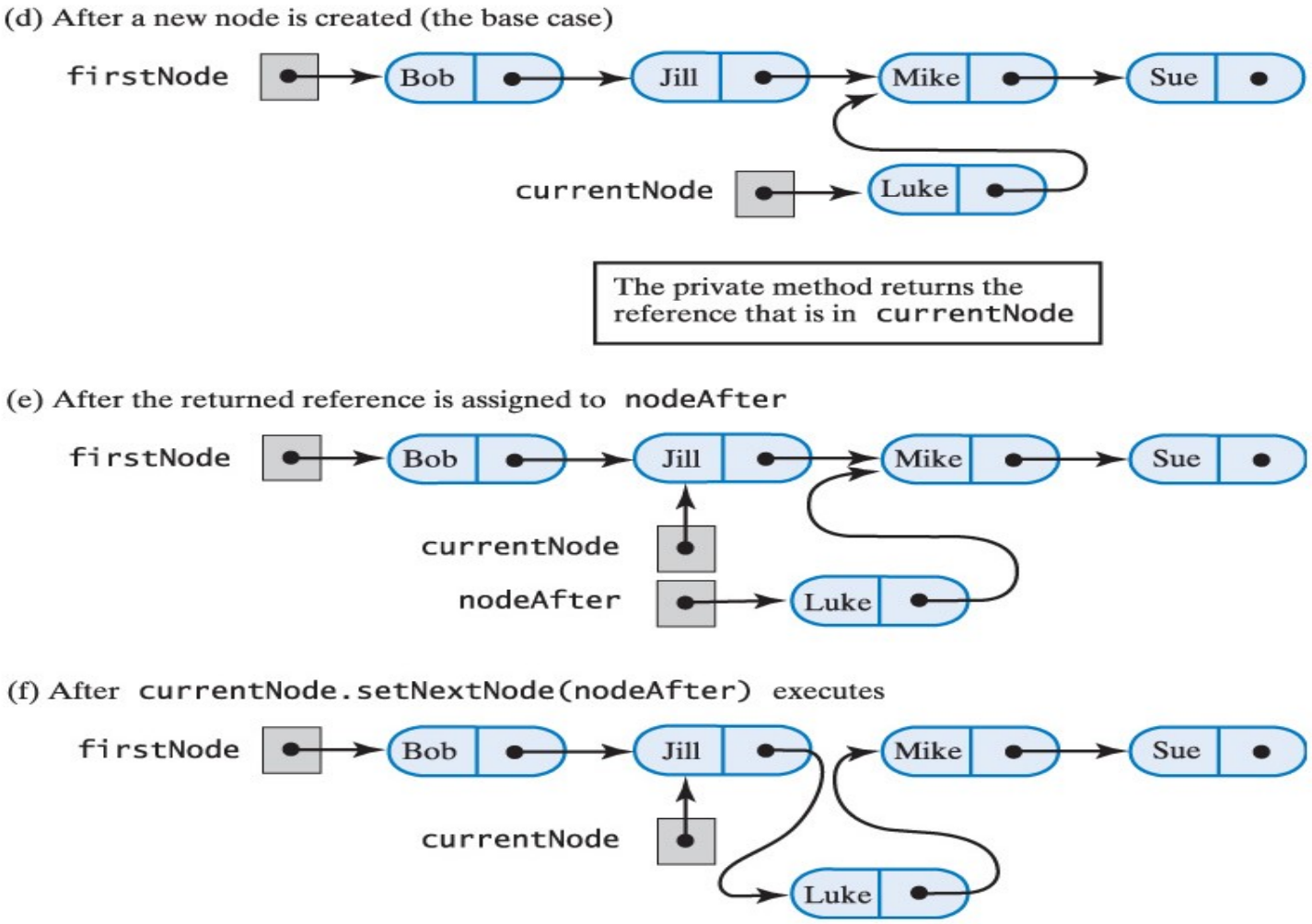

Fig. 13.4: (ctd) Recursively adding a node between existing nodes in a chain.

Comparing Entries within a Class Hierarchy

- If a sorted list consists of entries that share the same superclass, we need to compare entries within the class hierarchy.
- The generic type declaration to enforce this requirement:

```
<T extends Comparable<? super T> >
```

  - The notation `? super T` means any superclass of the generic type `T`.
  - Hence, `T` is *comparable* as it is a subclass of `Comparable<? super T>`.
  - Because of `Comparable<? super T>`, `T` can be compared with other objects of itself and its superclass(es).

Review of Learning Outcomes

You should now be able to

- Use a sorted list in a program
- Describe the differences between the ADT list and the ADT sorted list
- Implement the ADT sorted list by using an array
- Implement the ADT sorted list by using a chain of linked nodes

References

- Carrano, F. M., 2019, Data Structures and Abstractions with Java, 5th edn, Pearson
- Liang, Y.D., 2018. Introduction to Java Programming and Data Structures.11th ed.United Kingdom:Pearson