BACS2063 Data Structures and Algorithms

Sorted Lists

Chapter 7

PYQ Oct 2022

Question 3

```
public SortedArrayList(int initialCapacity) {
    numberOfEntries = 0;
    array = (T[]) new Comparable[initialCapacity];
  }

  public boolean add(T newEntry) {
    int i = 0;
    while (i < numberOfEntries && newEntry.compareTo(array[i]) > 0)
{
        i++;
    }
    makeRoom(i + 1);
    array[i] = newEntry;
    numberOfEntries++;
    return true;
  }

  public boolean contains(T anEntry) {
    boolean found = false;
    ...

    return found;
  }

  public boolean remove(T anEntry) {
      if (isEmpty()){
          return false;
      }else{
          int i = 0;
          ...
      }
      return false;
  }

  private void makeRoom(int newPosition) {
    int newIndex = newPosition - 1;
    int lastIndex = numberOfEntries - 1;

    for (int index = lastIndex; index >= newIndex; index--) {
      array[index + 1] = array[index];
    }
  }

 private void removeGap(int givenPosition) {
    int removedIndex = givenPosition - 1;
    int lastIndex = numberOfEntries - 1;


    for (int index = removedIndex; index < lastIndex; index++) {
      array[index] = array[index + 1];
    }
  }
```

FIGURE 1: Snippet of methods in sorted ArrayList.

PYQ Oct 2022 (Continued)

Question 3 (Continued)

a) The code segment given in **FIGURE 1** is the implementation of a sorted ArrayList. Based on the above **FIGURE 1**, construct the code for **Contains method** of the SortedArrayList. (8 marks)

b) Removing an item from a list can be done using remove method. Construct the code for **remove method** of the SortedArrayList. (7 marks)

Answers Q3a

```
public boolean contains(T anEntry) {
        boolean found = false;
        for (int index = 0; !found && (index <
numberOfEntries); index++) {

            if (anEntry.equals(array[index])) {
                found = true;
            }
        }
        return found;
    }
```

Answers Q3b

```
public boolean remove(T anEntry) {
      if (isEmpty()){
          return false;

        }else{
          int i = 0;
          while(i < numberOfEntries &&
array[i].compareTo(anEntry)<0){
               i++;
          }
          if (array[i].equals(anEntry)){
             removeGap(i+1);
             numberOfEntries--;
             return true ;
          }
      }

      return false;

  }
```

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 number of entries in the list
  - Check if list is empty

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

Comparing values vs objects

- Which one is bigger?

```
public class Employee{
   private int ID;
   private String name;
   private int salary;
   … … …
}
```

- int x = 10, y = 55;
- Employee emp1 =
  new Employee(111,"Zendaya", 3000);
- Employee emp2 =
  new Employee(222, "Amy",2000);

Comparing Objects

- To compare 2 objects to find out which is bigger/smaller, the entity class for the object needs to implement the **compareTo()** method.

compareTo() method

Suppose we have this code statement:

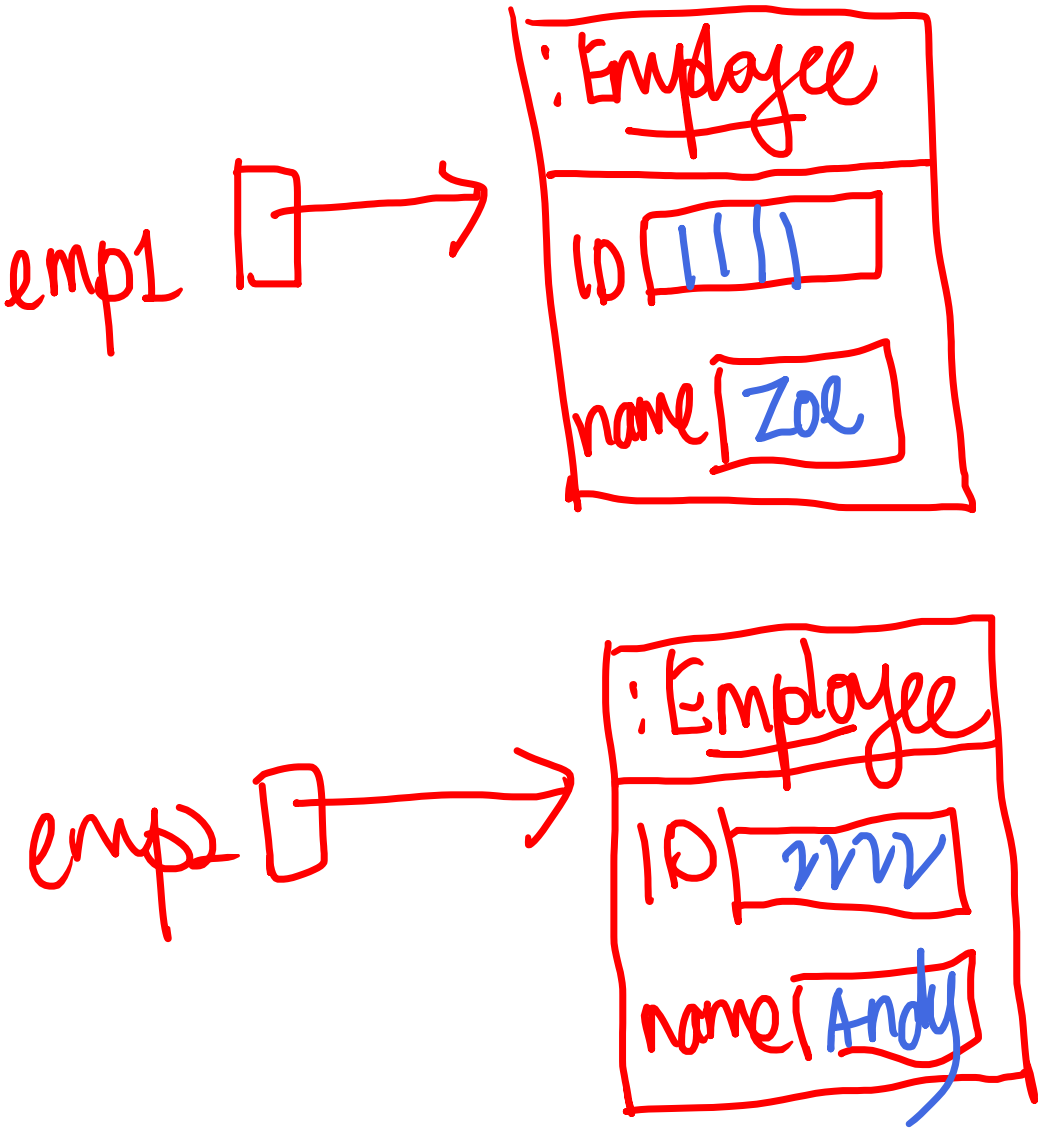
obj1.compareTo(**obj2**)

where *obj1* and *obj2* are of the same type.

The compareTo() method should:

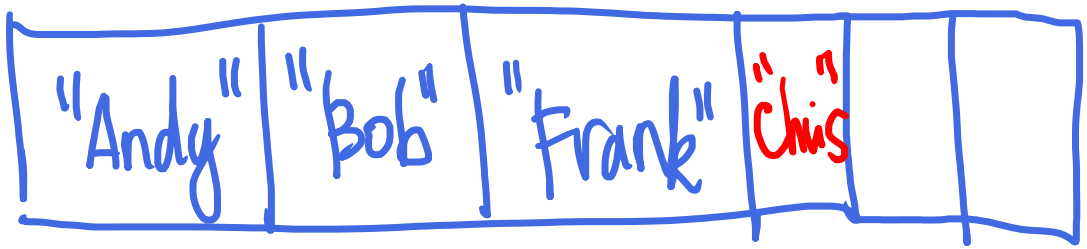
- Return 0 → if obj1 and obj2 have equal value
- Return +ve integer value → if obj1 is bigger than obj2
- Return -ve integer value → if obj1 is smaller than obj2

How to add a compareTo() method to an Entity Class?

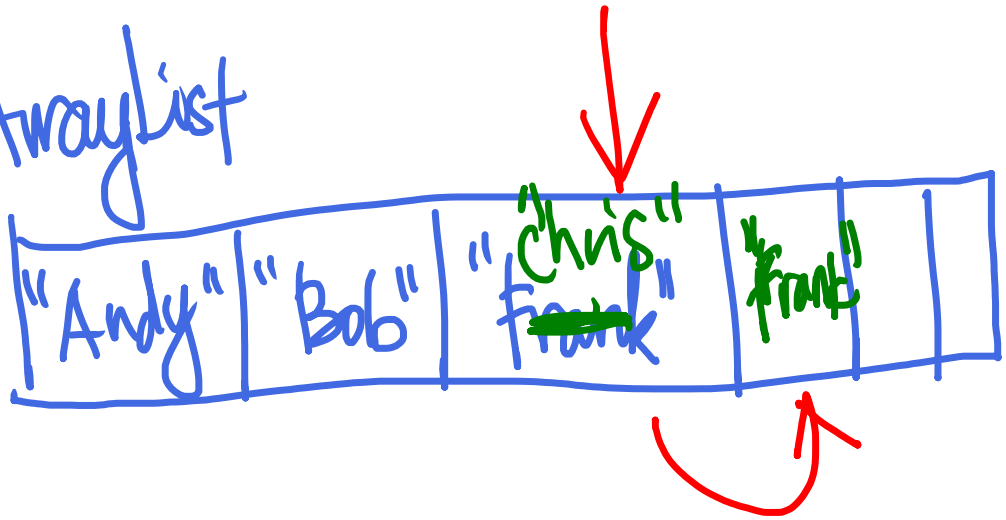

emp1
:Employee
ID
1111
name
Zoe
emp2
:Employee
ID
2222
name
Andy

ArrayList

"Chris"



Sorted ArrayList



Example of CompareTo()

```
1   package entity;
2
3   public class Employee implements Comparable<Employee>{
4       private int id;
5       private String name;
6       private double salary;
7
8       public Employee() {
9           this(0,"",0.0);
10      }
11
12      public Employee(int id, String name, double salary) {
13          this.id = id;
14          this.name = name;
15          this.salary = salary;
16      }
17
18      @Override
19      public int compareTo(Employee o) {
20          //return this.id - o.id;
21          return this.name.compareTo(o.name);
22      }
23
```

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\adt`:
  - `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>>
      ```

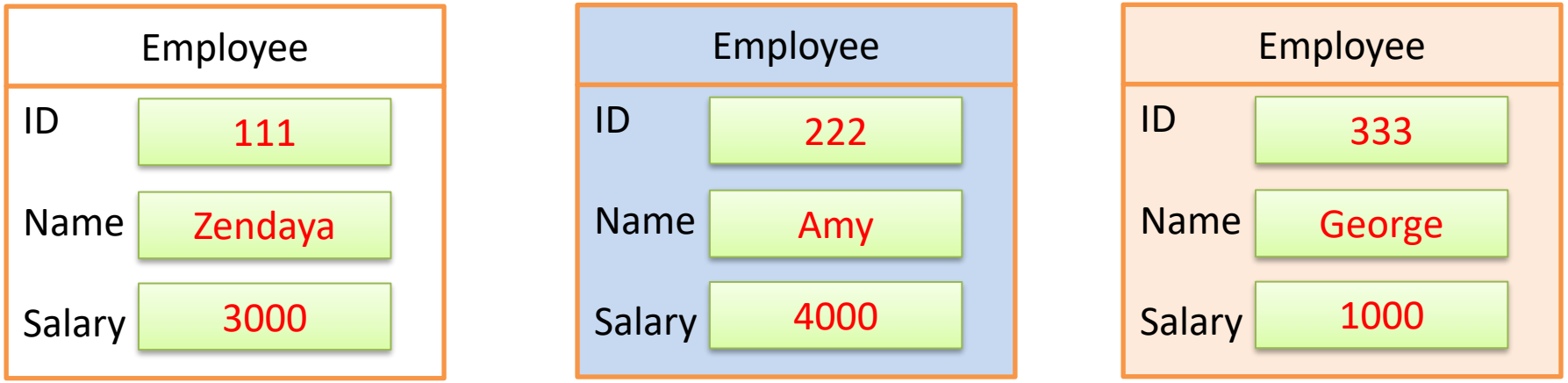
    - Note the `new` statement to construct the array in the constructor:

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

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

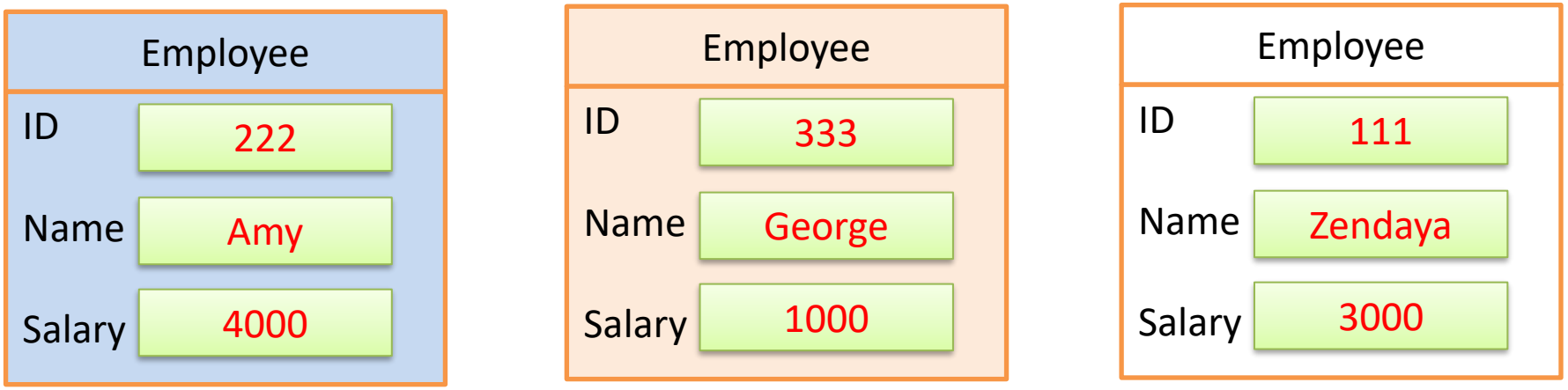
Case 1: CompareTo() Comparing based on ID

Position: 1 2 3



Case 2: CompareTo() Comparing based on Name

Position: 1 2 3



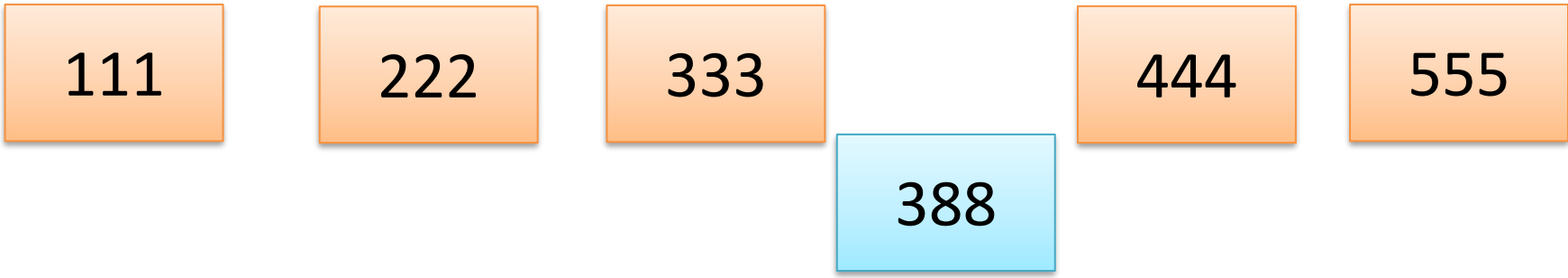
Case 3: CompareTo() Comparing based on Salary

Position: 1 2 3

```
public boolean add(T newEntry) {
  int i = 0;
  while (i < numberOfEntries && newEntry.compareTo(array[i]) > 0) {
    i++;
  }
  makeRoom(i + 1);
  array[i] = newEntry;
  numberOfEntries++;
  return true;
}
```

i 3

numberOfEntries 5



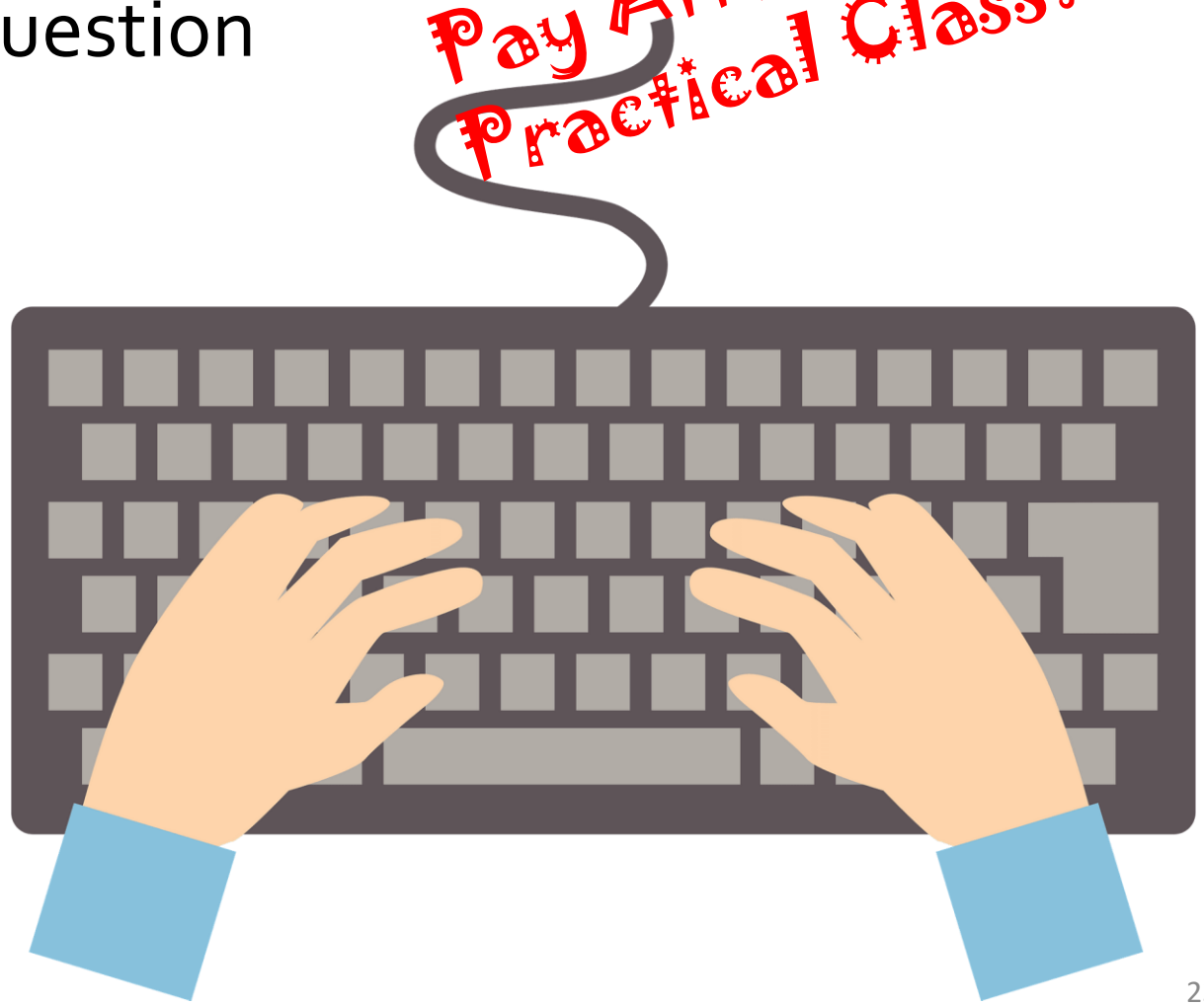
```
public boolean contains(T anEntry) {
  boolean found = false;
  for (int index = 0; !found && (index < numberOfEntries); index++) {
    if (anEntry.equals(array[index])) {
      found = true;
    }
  }
  return found;
}
```

index 3

numberOfEntries 5

Remove() method

- Practical Question (P7 Q1)



```
public boolean remove(T anEntry) {
  if (numberOfEntries == 0) {
    return false;
  } else {
    int index = 0;
    while (index < numberOfEntries && array[index].compareTo(anEntry) < 0) {
      index++;
    }

    if (array[index].equals(anEntry)) { // target found
      removeGap(index + 1);
      numberOfEntries--;
      return true;
    }
  }
  return false;
}
```

Sorted Linked List Implementation

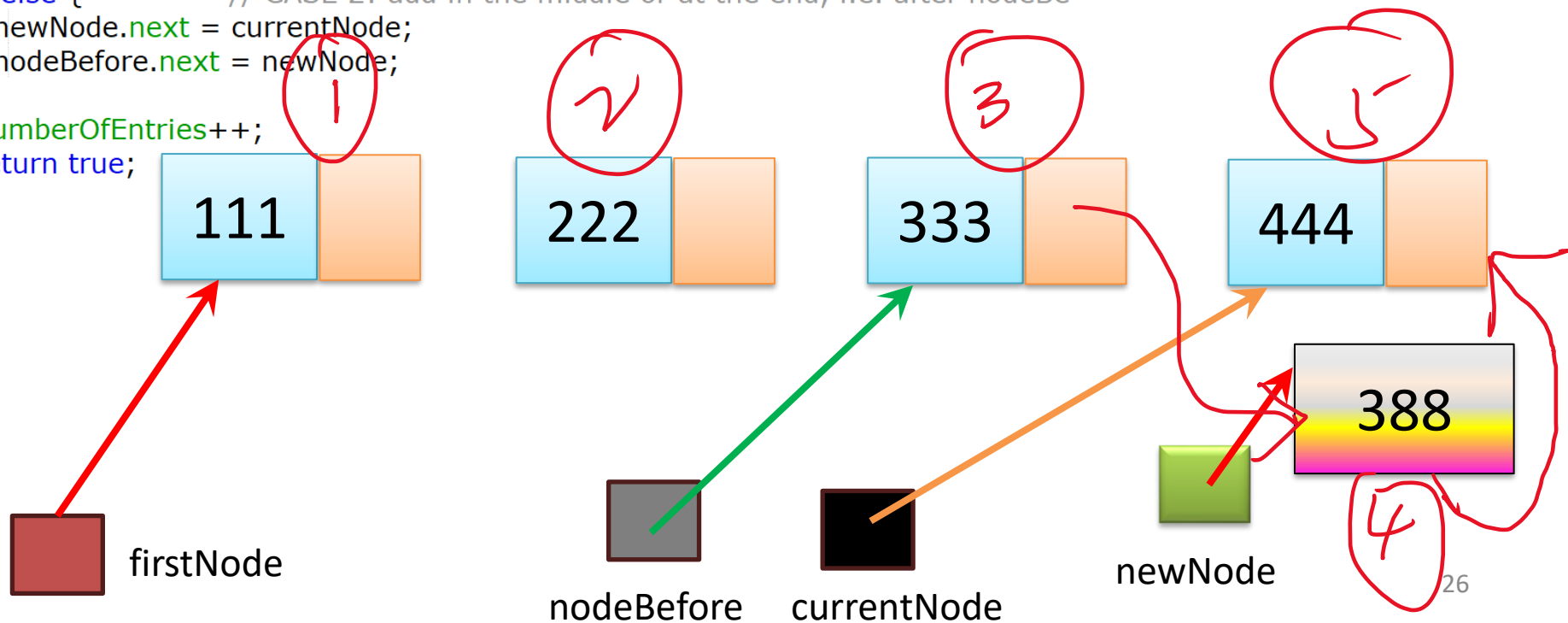
- Sample code in folder `\Chapter7\adt:`
  - `SortedLinkedList.java`
    - **`add()`** method
      - If list is in ascending order, insert new entry just before first entry not smaller than new entry
  - `SortedListInterface.java`

Adding to the middle of the sorted linked list

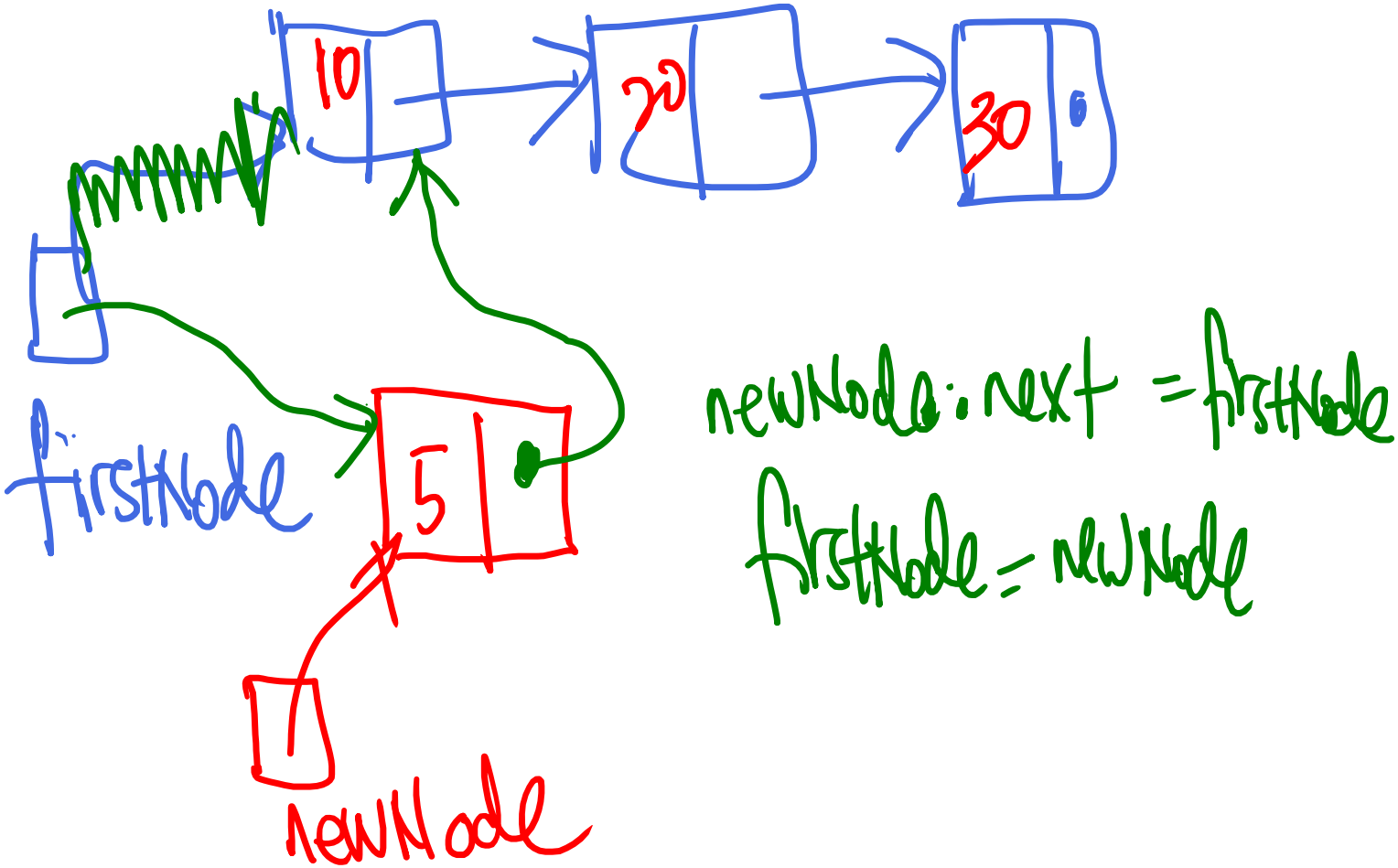
```
public boolean add(T newEntry) {
  Node newNode = new Node(newEntry);

  Node nodeBefore = null;
  Node currentNode = firstNode;
  while (currentNode != null && newEntry.compareTo(currentNode.data) > 0) {
    nodeBefore = currentNode;
    currentNode = currentNode.next;
  }

  if (isEmpty() || (nodeBefore == null)) { // CASE 1: add at beginning
    newNode.next = firstNode;
    firstNode = newNode;
  } else {              // CASE 2: add in the middle or at the end, i.e. after nodeBe
    newNode.next = currentNode;
    nodeBefore.next = newNode;
  }
  numberOfEntries++;
  return true;
}
```

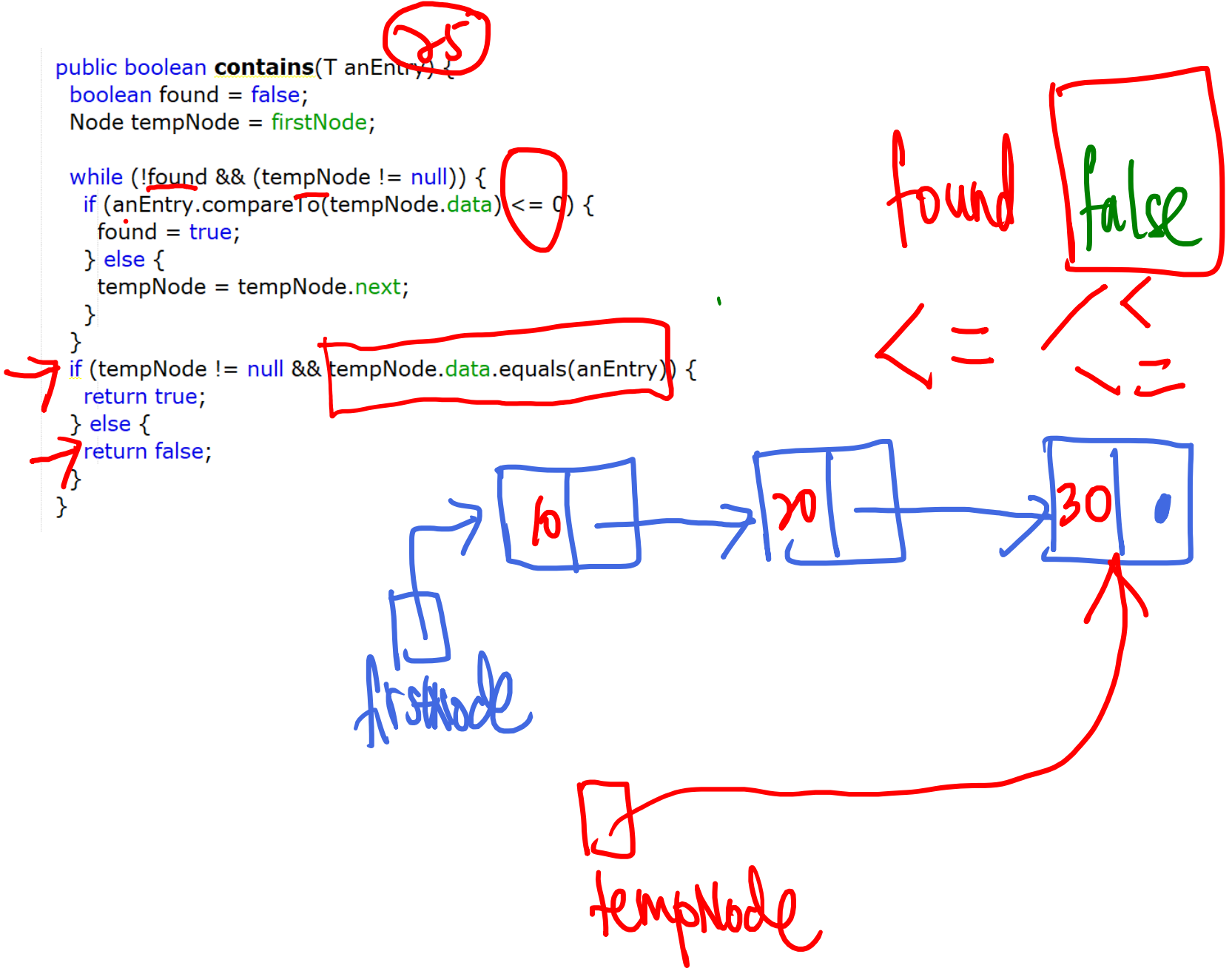


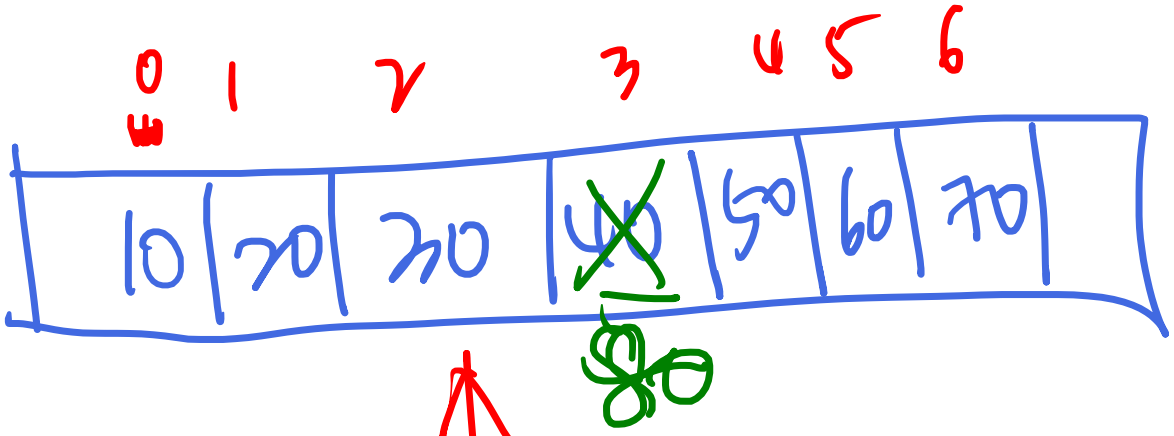
Adding to the front of the sorted linked list



```
public boolean contains(T anEntry) {
  boolean found = false;
  Node tempNode = firstNode;

  while (!found && (tempNode != null)) {
    if (anEntry.compareTo(tempNode.data) <= 0) {
      found = true;
    } else {
      tempNode = tempNode.next;
    }
  }
  if (tempNode != null && tempNode.data.equals(anEntry)) {
    return true;
  } else {
    return false;
  }
}
```





search 35 → 7 comparison

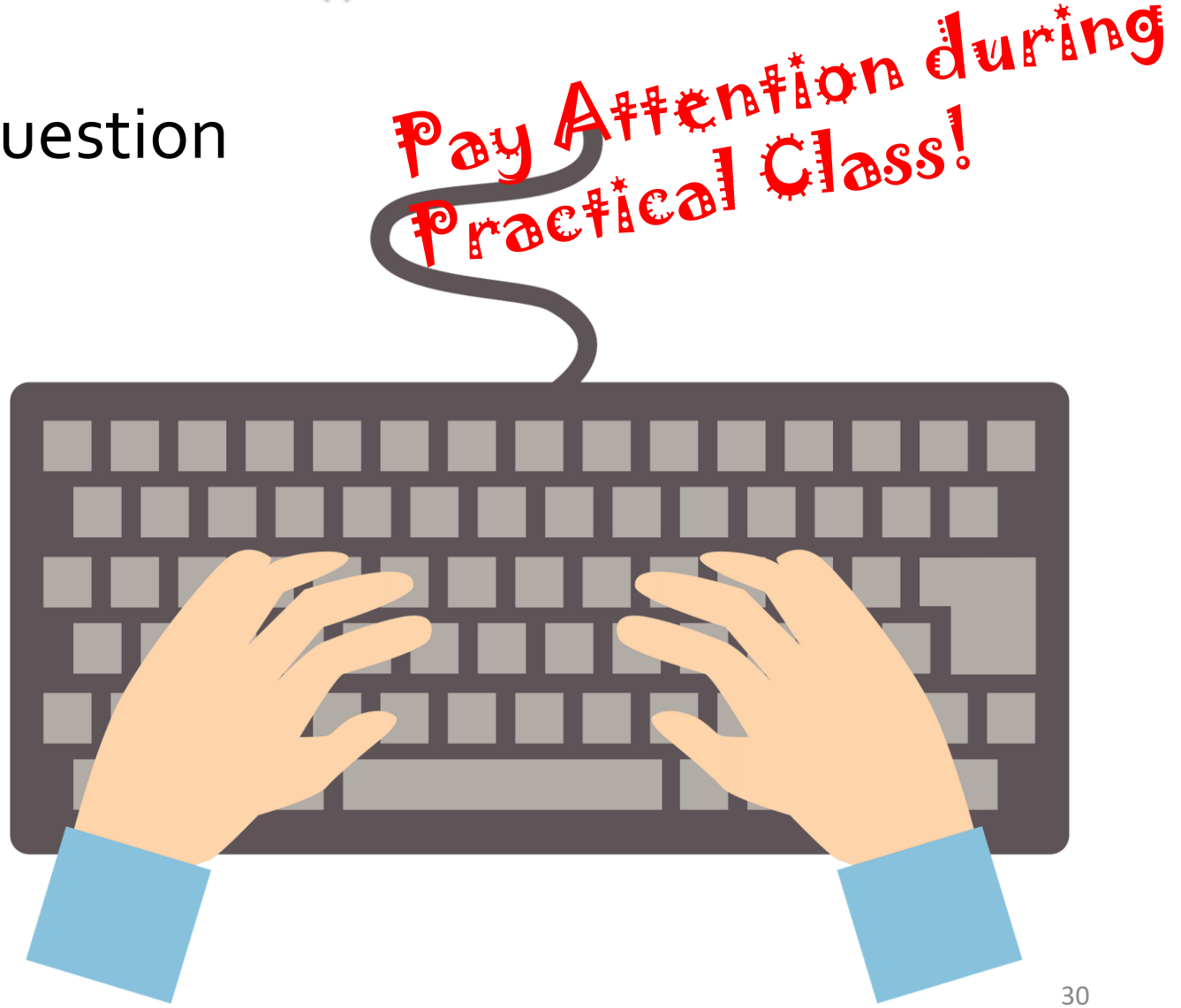
replace(3, 80)

3 comparison
+1

Remove() method

- Practical Question

  P7Q3



The Method **add**

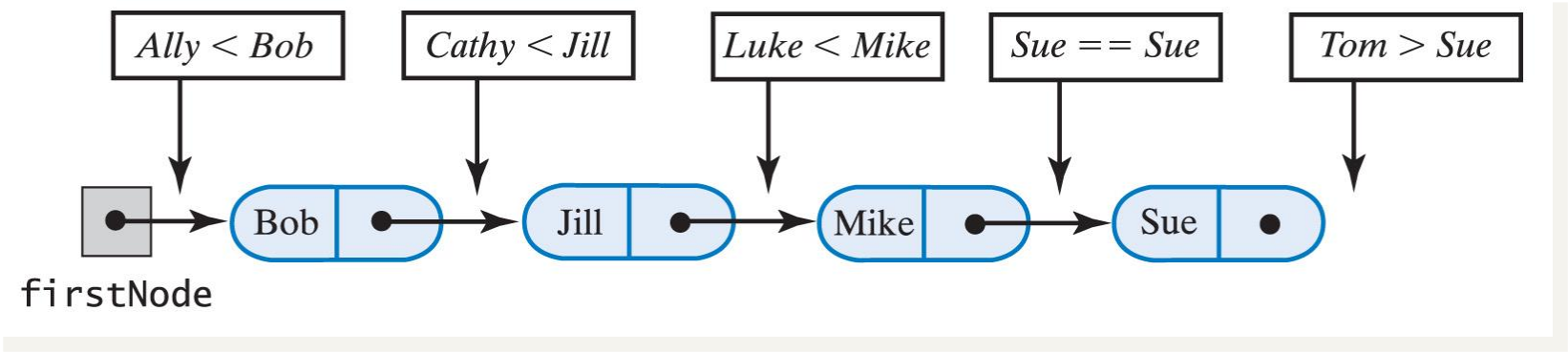


Fig. 13.1: Insertion points of names into a sorted chain of linked nodes.

The Method **add**

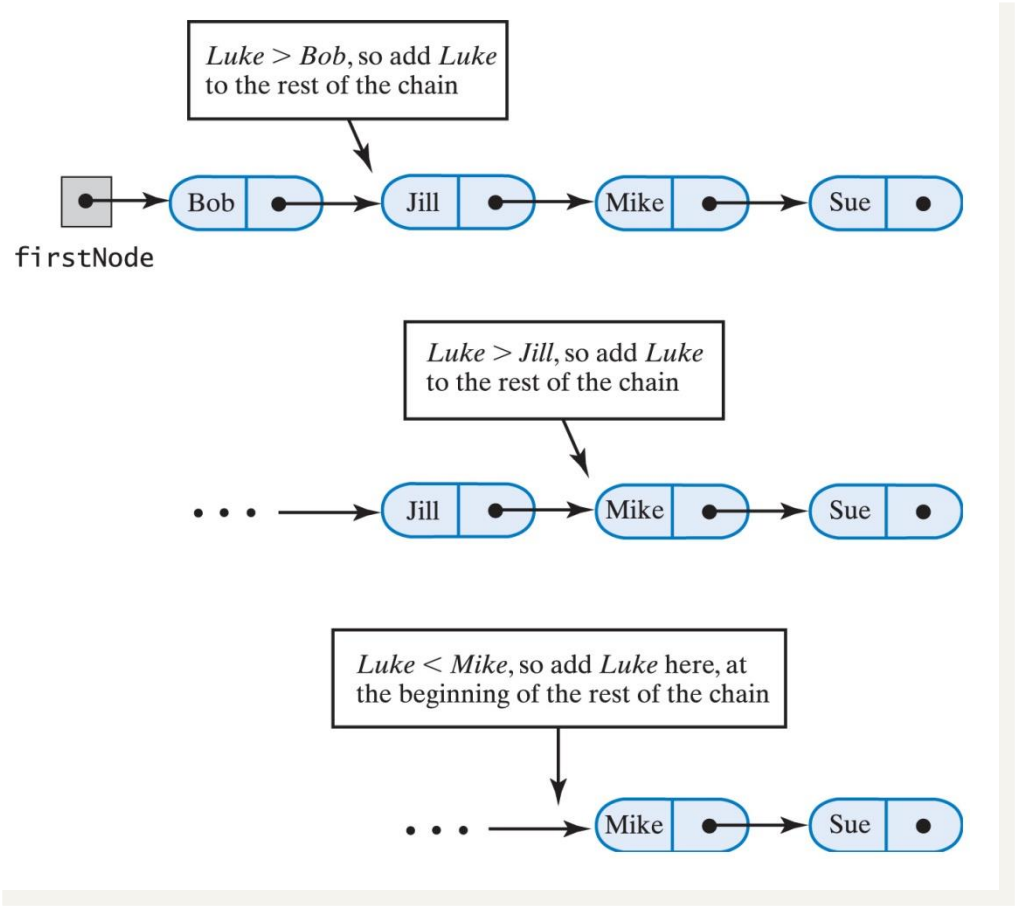


Fig. 13.2: Recursively adding *Luke* to a sorted chain of names

The Method **add**

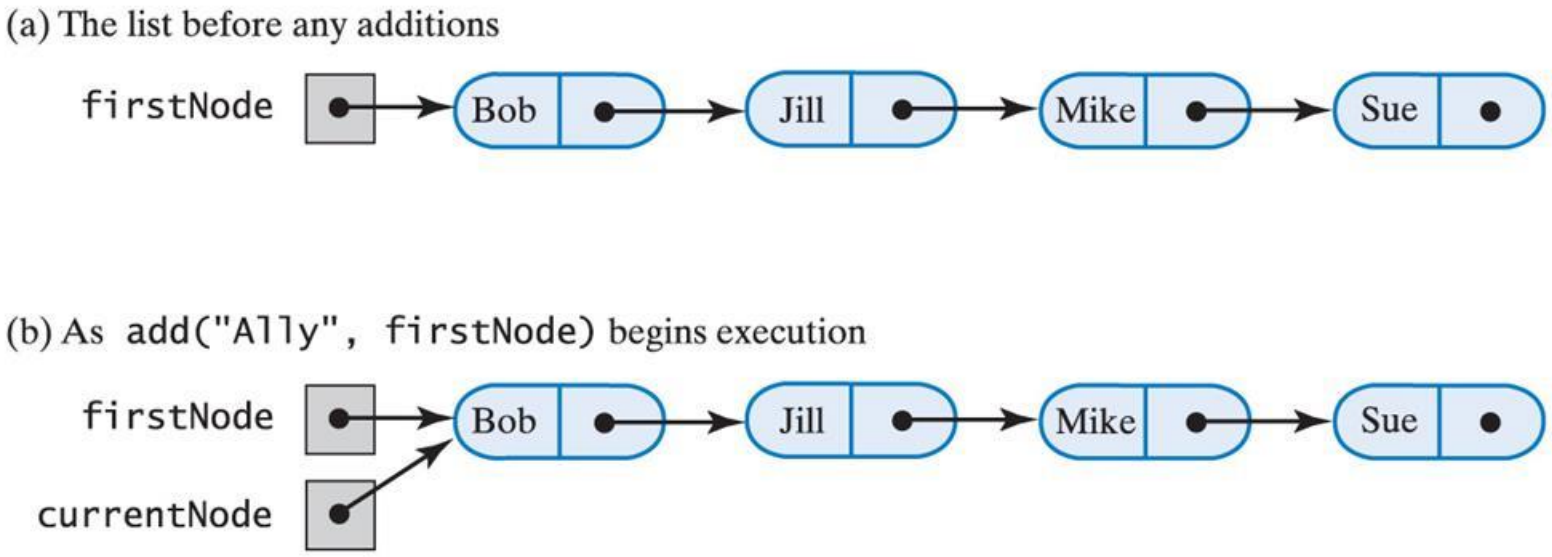


Fig. 13.3: Recursively adding *a* node at the beginning of the chain (continued →)

The Method **add**

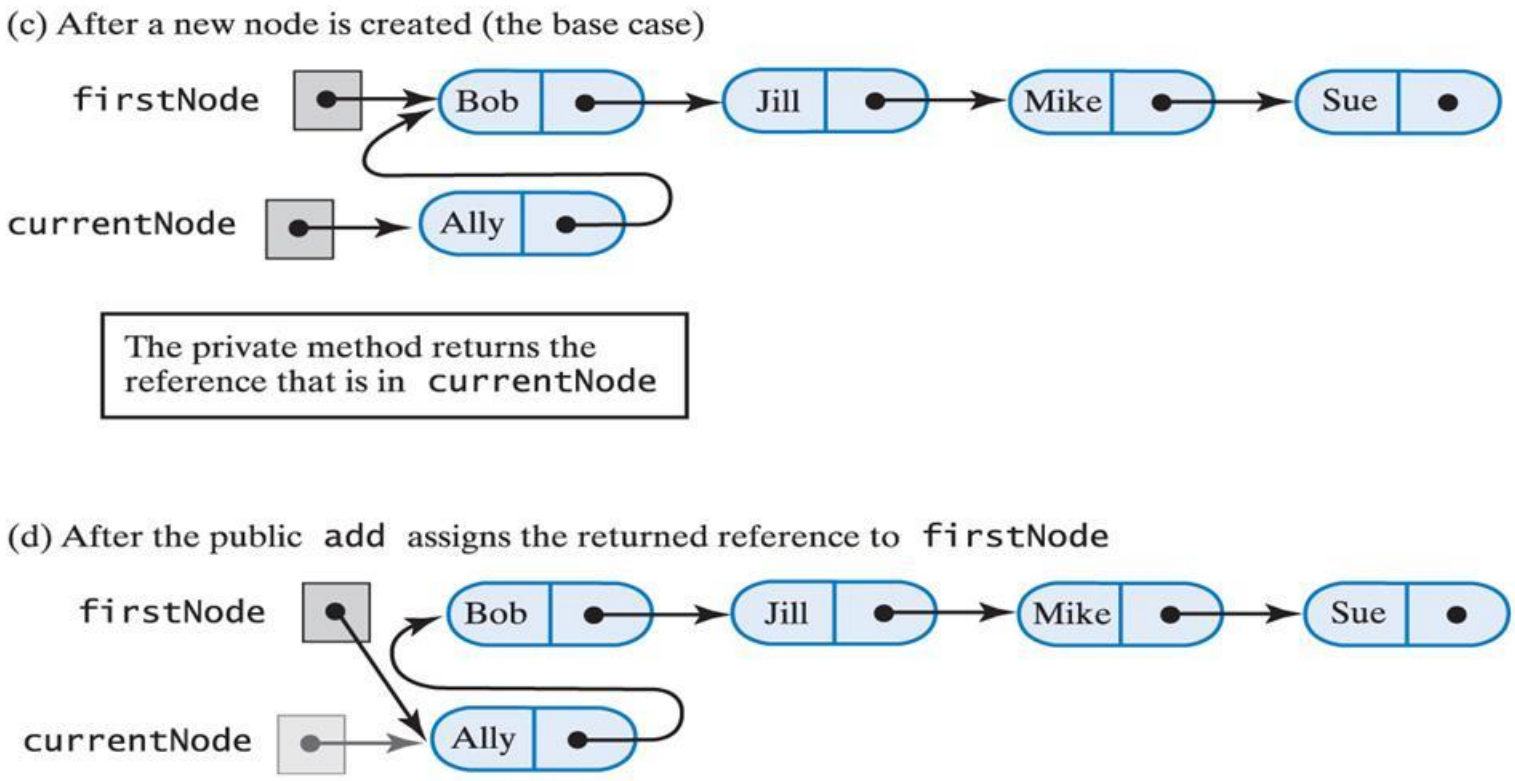


Fig. 13.3: (ctd) Recursively adding *a* node at the beginning of the chain.

The Method **add**

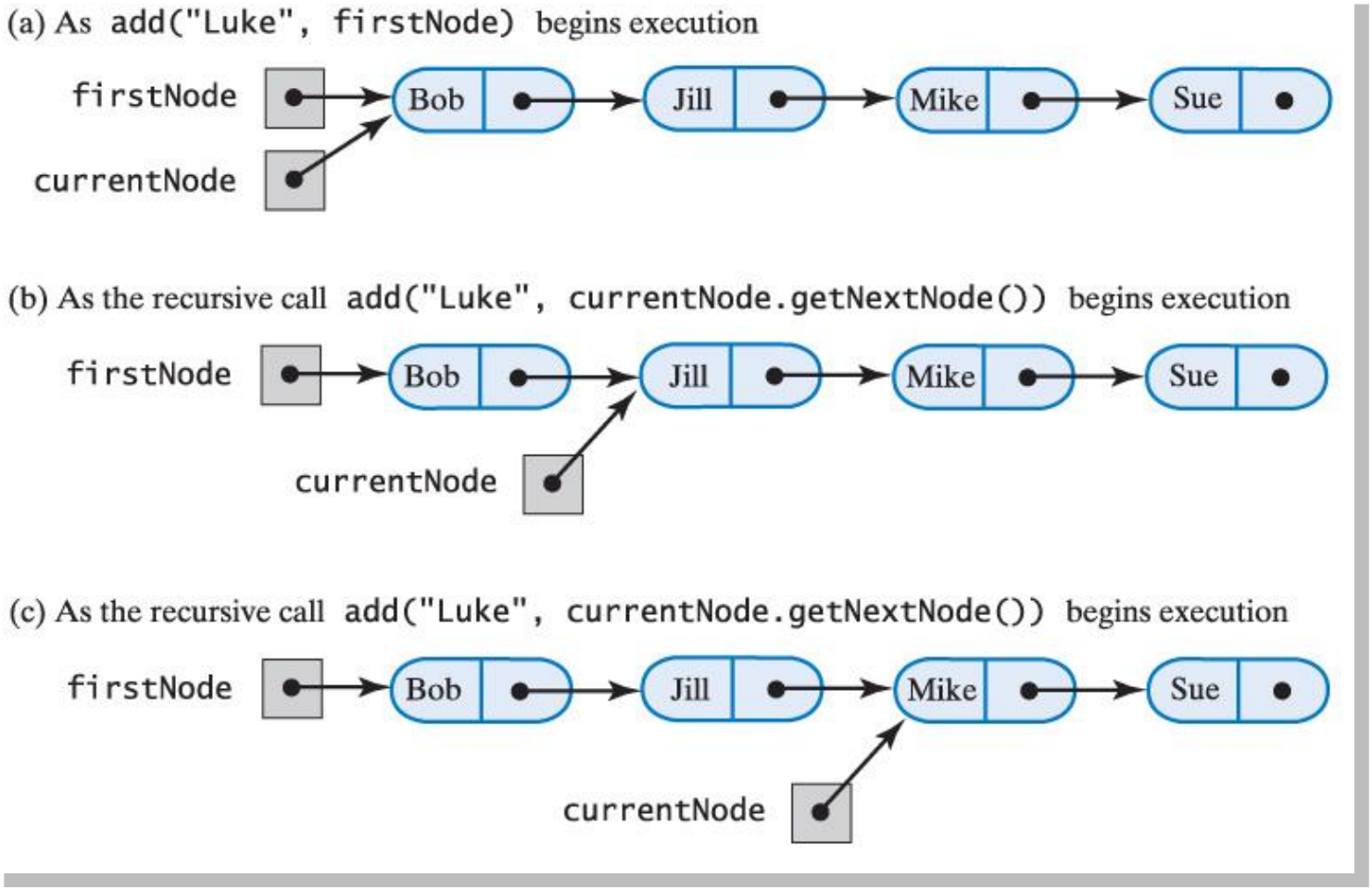


Fig. 13.4: Recursively adding a node between existing nodes in a chain (continued →)

The Method **add**

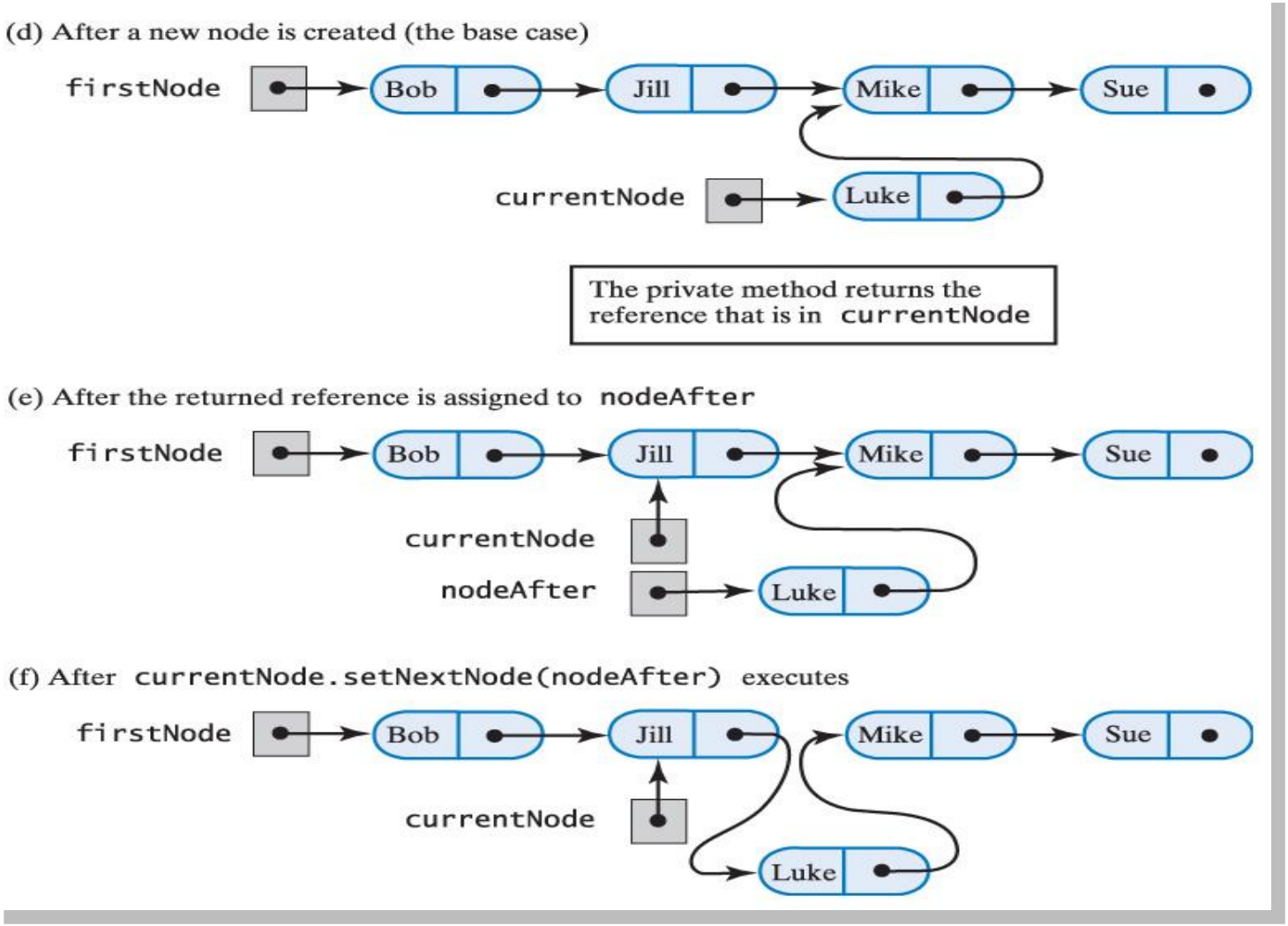


Fig. 13.4: (ctd) Recursively adding a node between existing nodes in a chain.

Efficiency of the Linked Implementation

ADT Sorted List Operation	Array	Linked
`add(newEntry)`	O(*n*)	O(*n*)
`remove(anEntry)`	O(*n*)	O(*n*)
`contains(anEntry)`	O(*n*)	O(*n*)
`clear()`	O(1)	O(1)
`getNumberOfEntries()`	O(1)	O(1)
`isEmpty()`	O(1)	O(1)

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

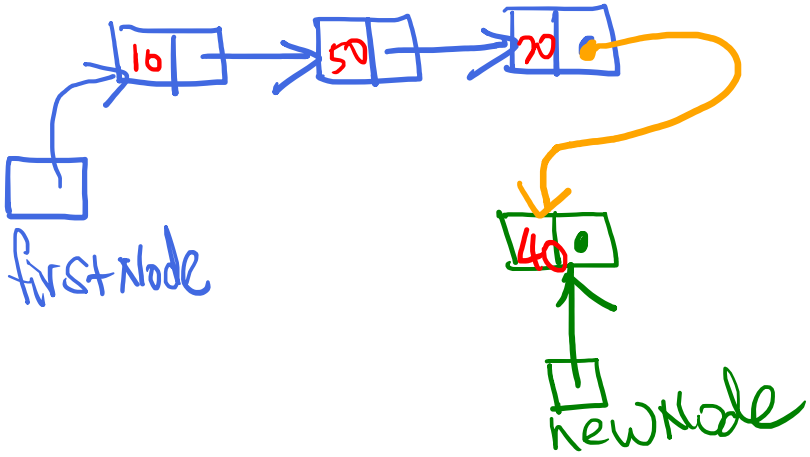
Exercise 7.1

Compare and contrast how each of the following methods would be implemented in an *unsorted list* and a *sorted list* using linked implementation. Provide detailed explanations. Your answers may also include appropriate diagrams for illustration.

(i) `void add(T newEntry)` – adds the given entry to the list. (6 marks)

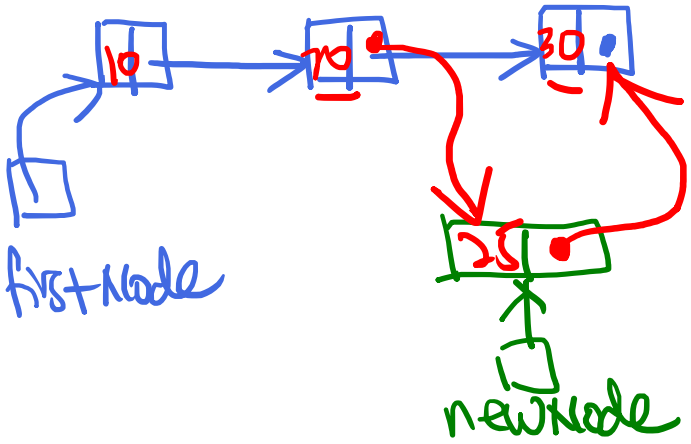
Unsorted Linked List

The new entry will be added to the end of the linked list.



Sorted Linked List

The right position to enter the new entry will be identified and then the new entry will be inserted into the correct position such that the list remains sorted.



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