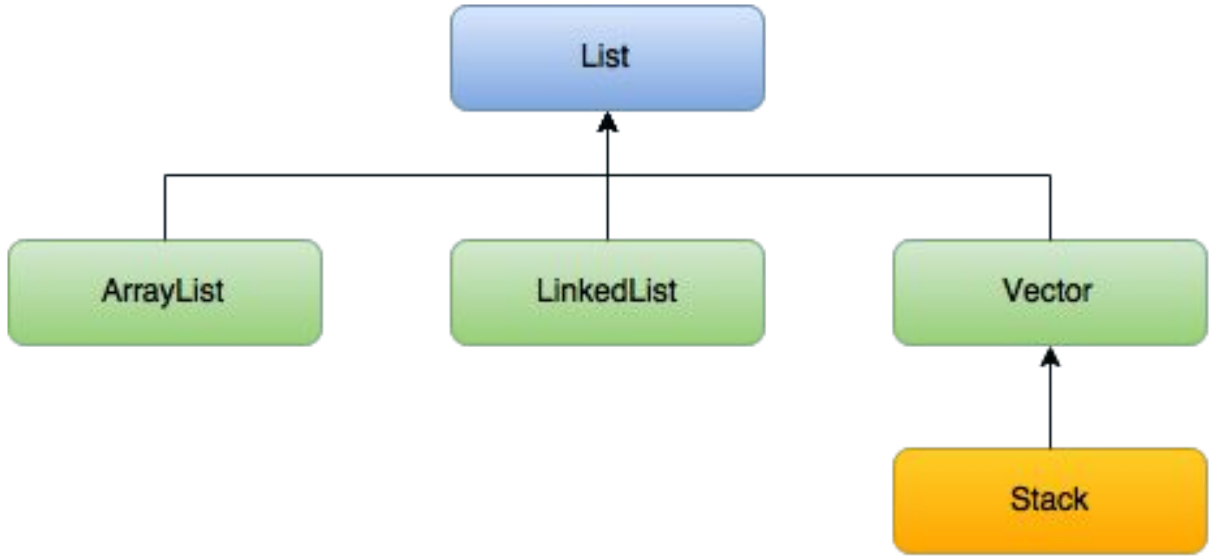
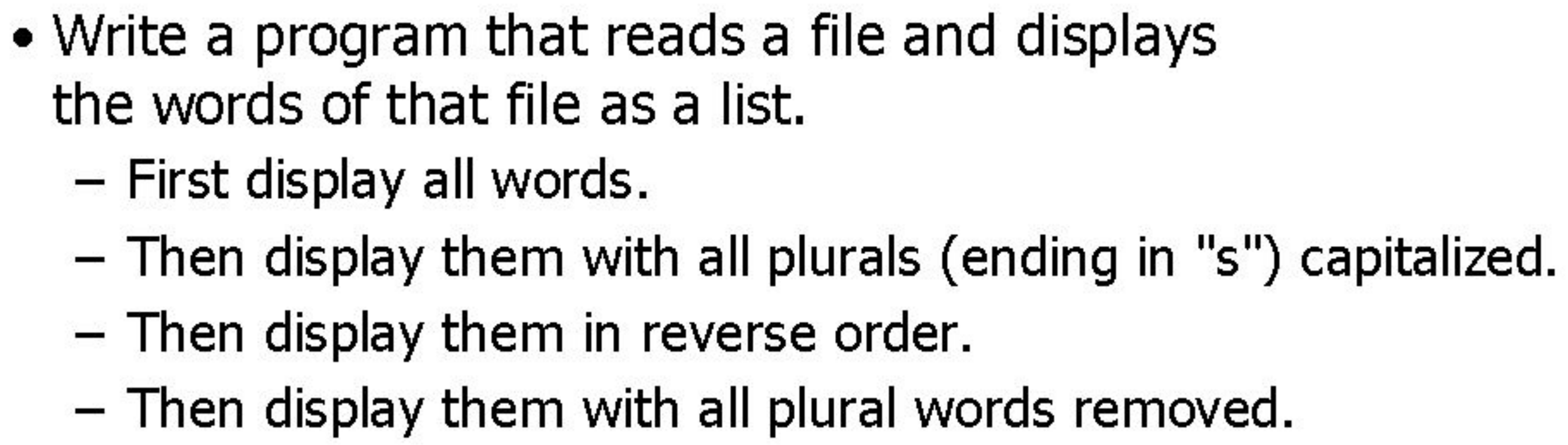


List Interface

Let Think About A Problem That Needs to Be Solved



- Write a program that reads a file and displays the words of that file as a list.
  - First display all words.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them in reverse order.
  - Then display them with all plural words removed.

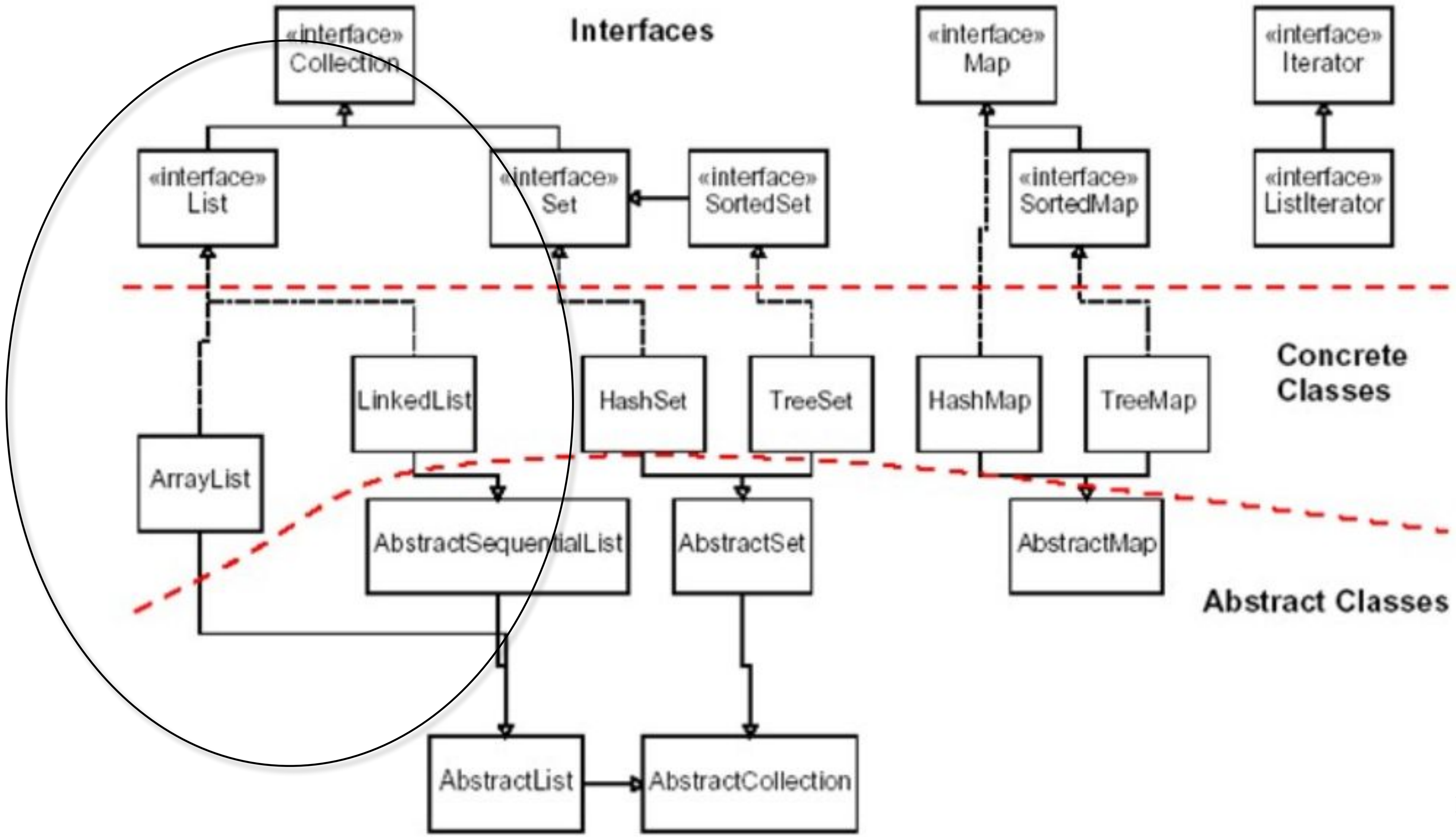
Naïve Solution

```
String[] allWords = new String[1000];
int wordCount = 0;

Scanner input = new Scanner(new File("data.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords[wordCount] = word;
    wordCount++;
}
```

collection : an object that stores data; a.k.a. "data structure"

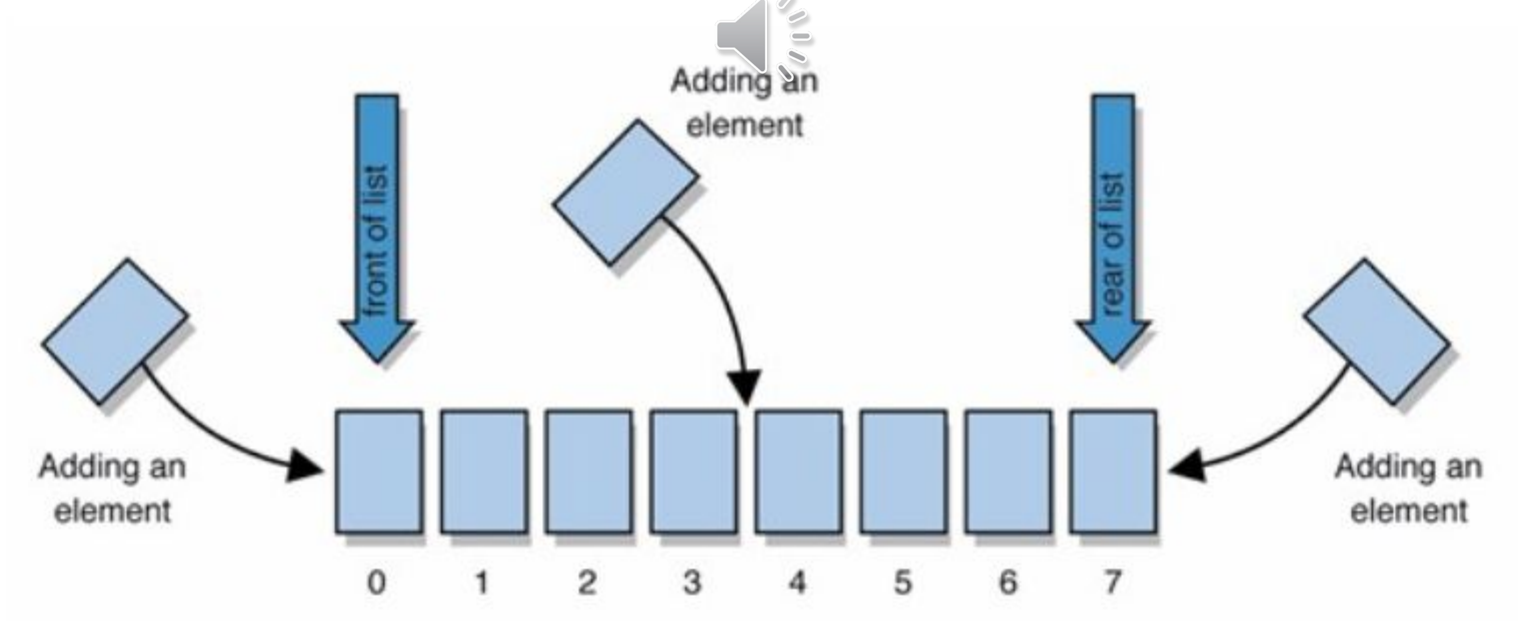
- the objects stored are called **elements**
- some collections maintain an ordering; some allow duplicates
- typical operations: *add*, *remove*, *clear*, *contains* (search), *size*
- examples found in the Java class libraries:
  - `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`
- all collections are in the `java.util` package

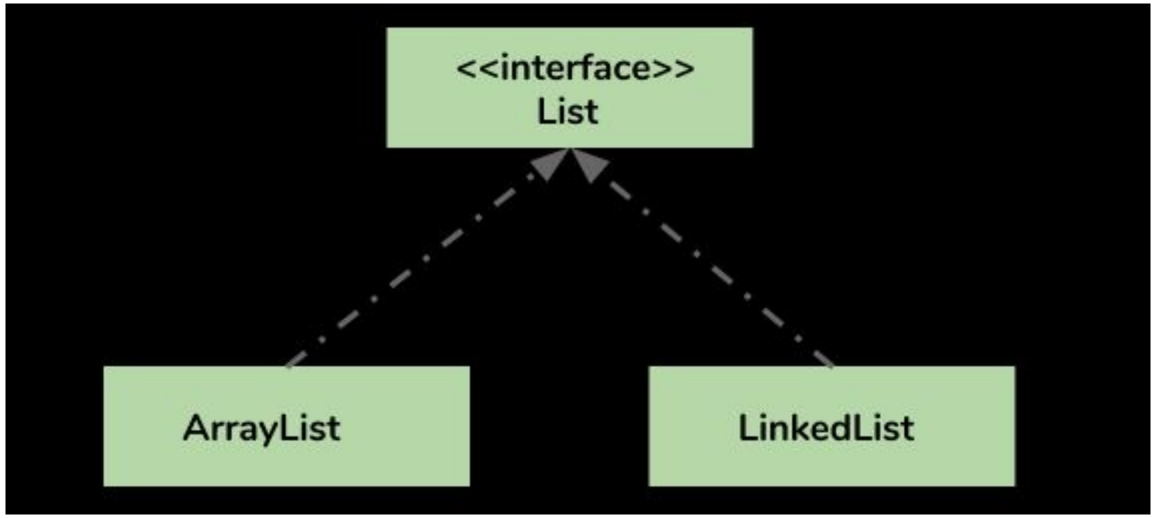
```
import java.util.*;
```


«interface» Collection
Interfaces
«interface» Map
«interface» Iterator
«interface» List
«interface» Set
«interface» SortedSet
«interface» SortedMap
«interface» ListIterator
Concrete Classes
LinkedList
HashSet
TreeSet
HashMap
TreeMap
ArrayList
AbstractSequentialList
AbstractSet
AbstractMap
Abstract Classes
AbstractList
AbstractCollection

list : a collection storing an ordered sequence of elements

- each element is accessible by a 0-based **index**
- a list has a **size** (number of elements that have been added)
- elements can be added to the front, back, or elsewhere
- in Java, a list can be represented as an `ArrayList` object

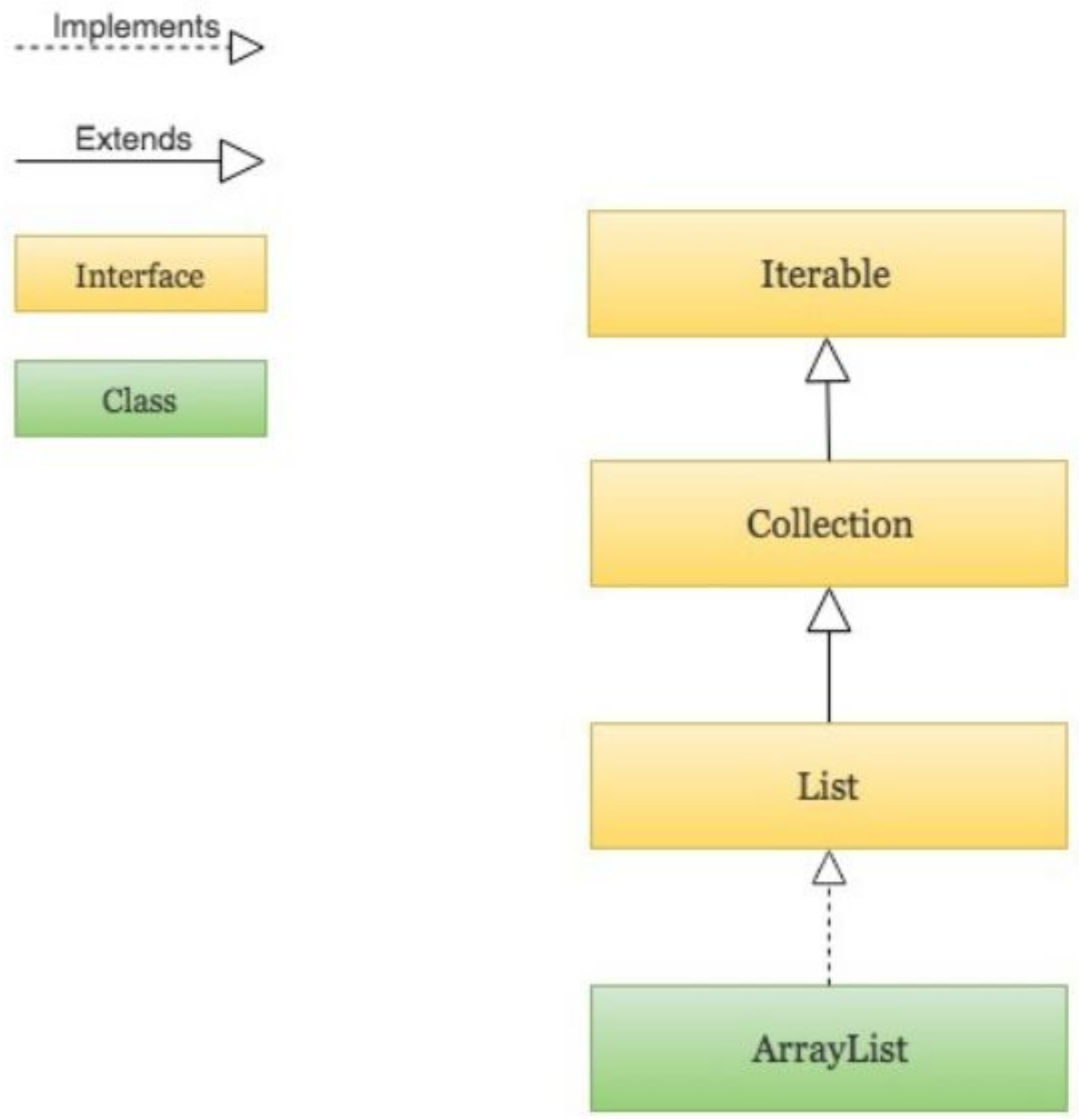




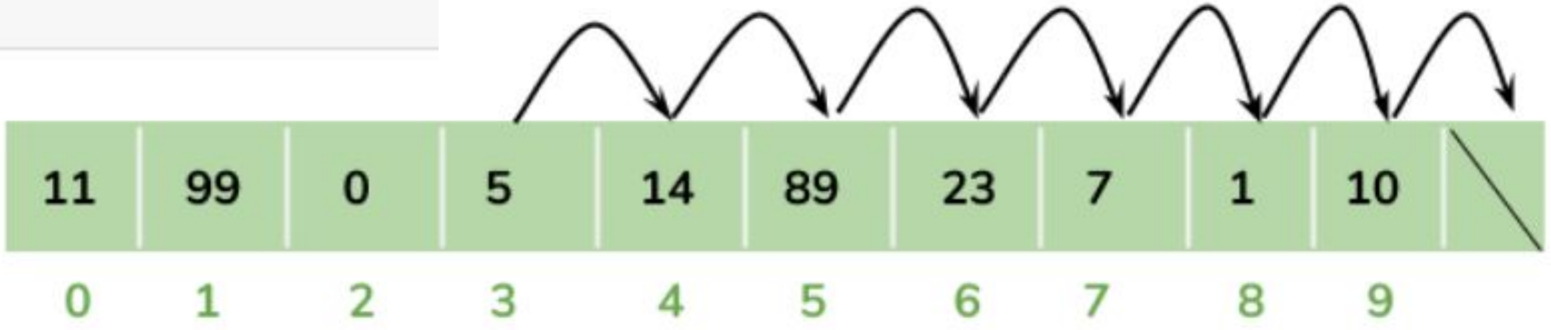
```
List<String> listStrings = new
ArrayList<String>();
listStrings.add("One");
listStrings.add("Two");
listStrings.add("Three");
listStrings.add("Four");
System.out.println(listStrings);
```

```
List<String> listStrings = new
LinkedList<String>();
listStrings.add("Five");
listStrings.add("Six");
listStrings.add("Seven");
listStrings.add("Eight");
System.out.println(listStrings);
```

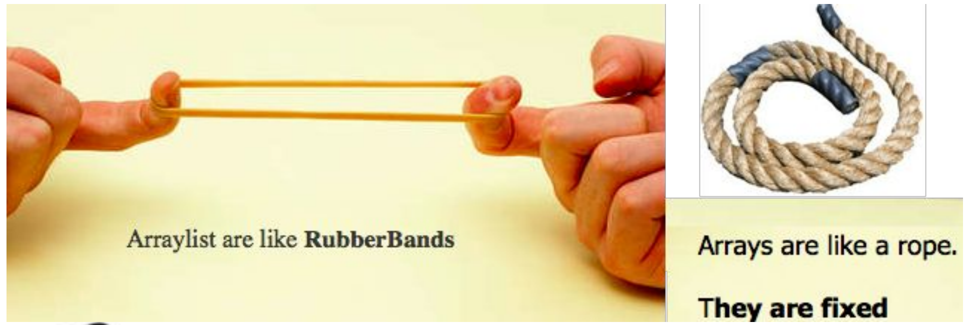
Java ArrayList Class Hierarchy



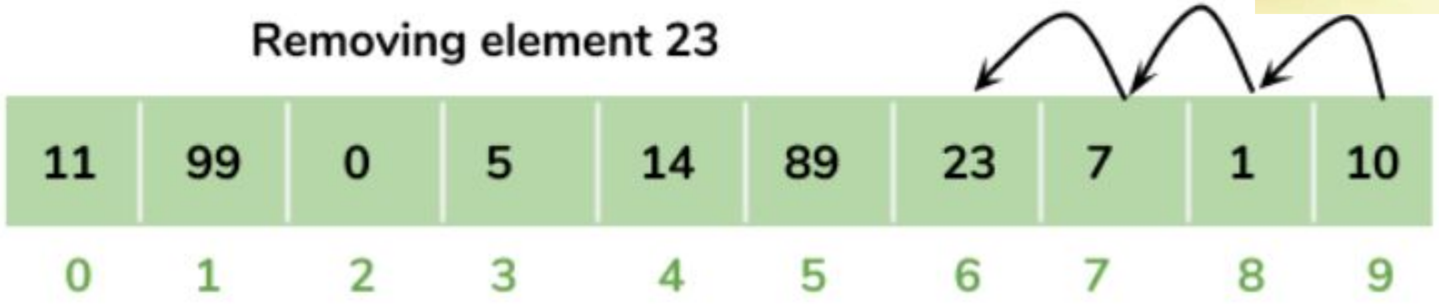
```
add(Object o);
```

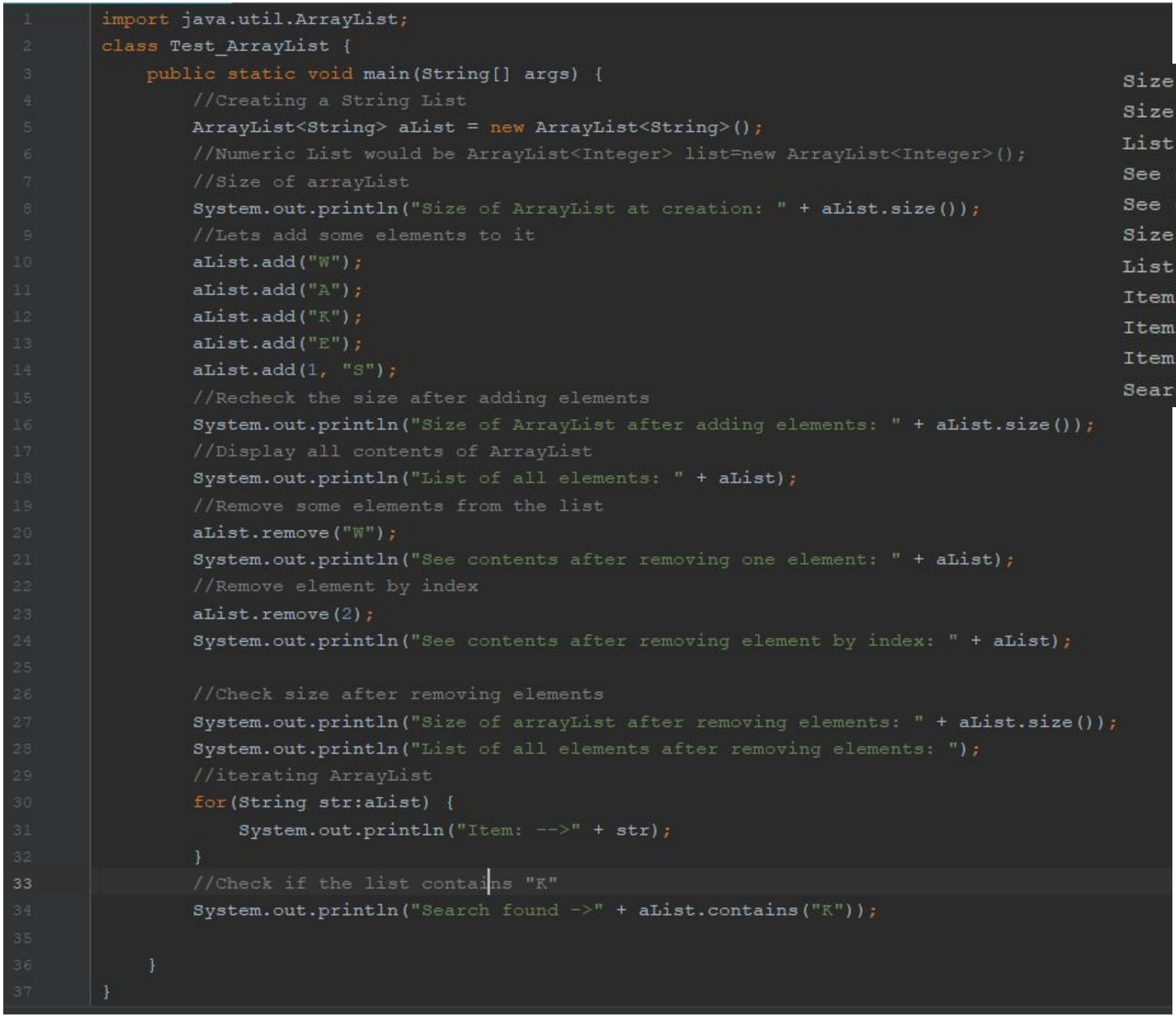


```
remove(Object o);
```

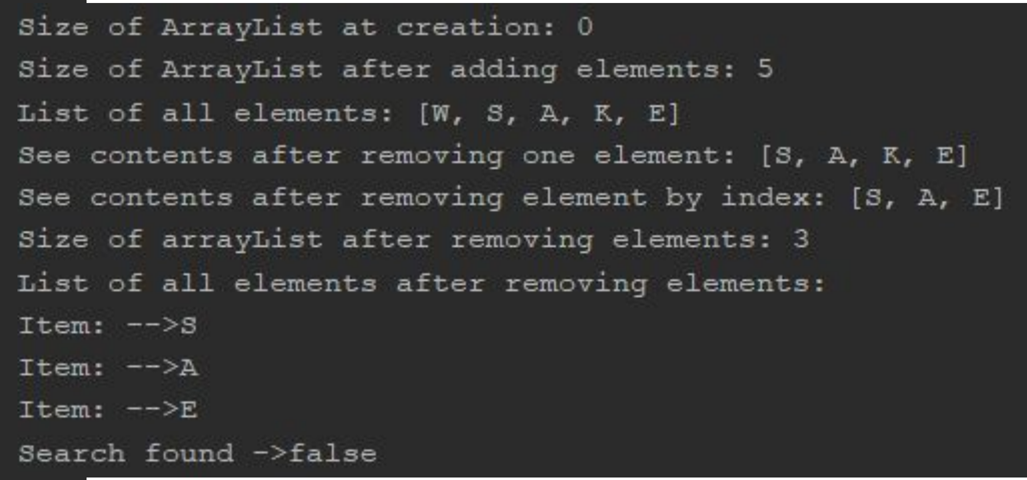


Removing element 23





```java
import java.util.ArrayList;
class Test_ArrayList {
    public static void main(String[] args) {
        //Creating a String List
        ArrayList<String> aList = new ArrayList<String>();
        //Numeric List would be ArrayList<Integer> list=new ArrayList<Integer>();
        //Size of arrayList
        System.out.println("Size of ArrayList at creation: " + aList.size());
        //Lets add some elements to it
        aList.add("W");
        aList.add("A");
        aList.add("K");
        aList.add("E");
        aList.add(1, "S");
        //Recheck the size after adding elements
        System.out.println("Size of ArrayList after adding elements: " + aList.size());
        //Display all contents of ArrayList
        System.out.println("List of all elements: " + aList);
        //Remove some elements from the list
        aList.remove("W");
        System.out.println("See contents after removing one element: " + aList);
        //Remove element by index
        aList.remove(2);
        System.out.println("See contents after removing element by index: " + aList);

        //Check size after removing elements
        System.out.println("Size of arrayList after removing elements: " + aList.size());
        System.out.println("List of all elements after removing elements: ");
        //iterating ArrayList
        for(String str:aList) {
            System.out.println("Item: -->" + str);
        }
        //Check if the list contains "K"
        System.out.println("Search found ->" + aList.contains("K"));

    }
}
```



```
Size of ArrayList at creation: 0
Size of ArrayList after adding elements: 5
List of all elements: [W, S, A, K, E]
See contents after removing one element: [S, A, K, E]
See contents after removing element by index: [S, A, E]
Size of arrayList after removing elements: 3
List of all elements after removing elements:
Item: -->S
Item: -->A
Item: -->E
Search found ->false
```

```
List listSource = new ArrayList();

listSource.add("123");
listSource.add("456");

List listDest   = new ArrayList();

listDest.addAll(listSource);
```

```
List listA = new ArrayList();

listA.add("element 0");
listA.add("element 1");
listA.add("element 2");

//access via index
```
String element0 = (String) listA.get(0);
String element1 = (String) listA.get(1);
String element3 = (String) listA.get(2);

```
List list = new ArrayList();

String element1 = "element 1";
String element2 = "element 2";

list.add(element1);
list.add(element2);
```

int index1 = list.indexOf(element1);
int index2 = list.indexOf(element2);

```
List list = new ArrayList();

String element1 = "element 1";

list.add(element1);
```

boolean containsElement = list.contains("element 1");

```
ArrayList<Integer> arraylist = new ArrayList<Integer>();
 arraylist.add(11);
 arraylist.add(2);
 arraylist.add(7);
 arraylist.add(3);


 Collections.sort(arraylist);


         Collections.sort(arraylist, Collection.reverseOrder();)


      OR
          Collections.reverse(arraylist);
```

```
ArrayList<Integer> arraylist = new
ArrayList<Integer>();
 arraylist.add(11);
 arraylist.add(2);
 arraylist.add(7);
 arraylist.add(3);
 arraylist.sort(Comparator.naturalOrder())

 arraylist.sort(Comparator.reversOrder())
arraylist.add(1);
   arraylist.add(2);
   arraylist.add(3);
   arraylist.add(4);
   arraylist.add(5);
   arraylist.add(6);
   arraylist.add(7);

   //Updating 1st element
   arraylist.set(0, 11);
   //Updating 2nd element
   arraylist.set(1, 22);
   //Updating 3rd element
   arraylist.set(2, 33);
   //Updating 4th element
   arraylist.set(3, 44);
   //Updating 5th element
   arraylist.set(4, 55);
```

`addAll(`**list**`)`
`addAll(`**index** `,` **list**`)`
	adds all elements from the given list to this list
	(at the end of the list, or inserts them at the given index)

`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements

`iterator()`
`listIterator()`
	returns an object used to examine the contents of the list
	(seen later)

`lastIndexOf(`**value**`)`	returns last index value is found in list (-1 if not found)
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from** `,` **to**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns the elements in this list as an array

Problem Revisited

- Write a program that reads a file and displays the words of that file as a list.
  - First display all words.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them in reverse order.
  - Then display them with all plural words removed.

```
ArrayList<String> allWords = new ArrayList<String>();
Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}
System.out.println(allWords);

// remove all plural words
for (int i = 0; i < allWords.size(); i++) {
    String word = allWords.get(i);
    if (word.endsWith("s")) {
        allWords.remove(i);
        i--;
    }
}
```

Questions????

Which of the following correctly inserts the integer value 247 into the third element of the ArrayList pCode?

-
  ```
  pCode.add(247);
  ```

-
  ```
  pCode.add(247, 2);
  ```

-
  ```
  pCode.add(2, 247);
  ```

-
  ```
  pCode.add(3, 247);
  ```

What happens when you add an item to an existing ArrayList, without specifying an index?

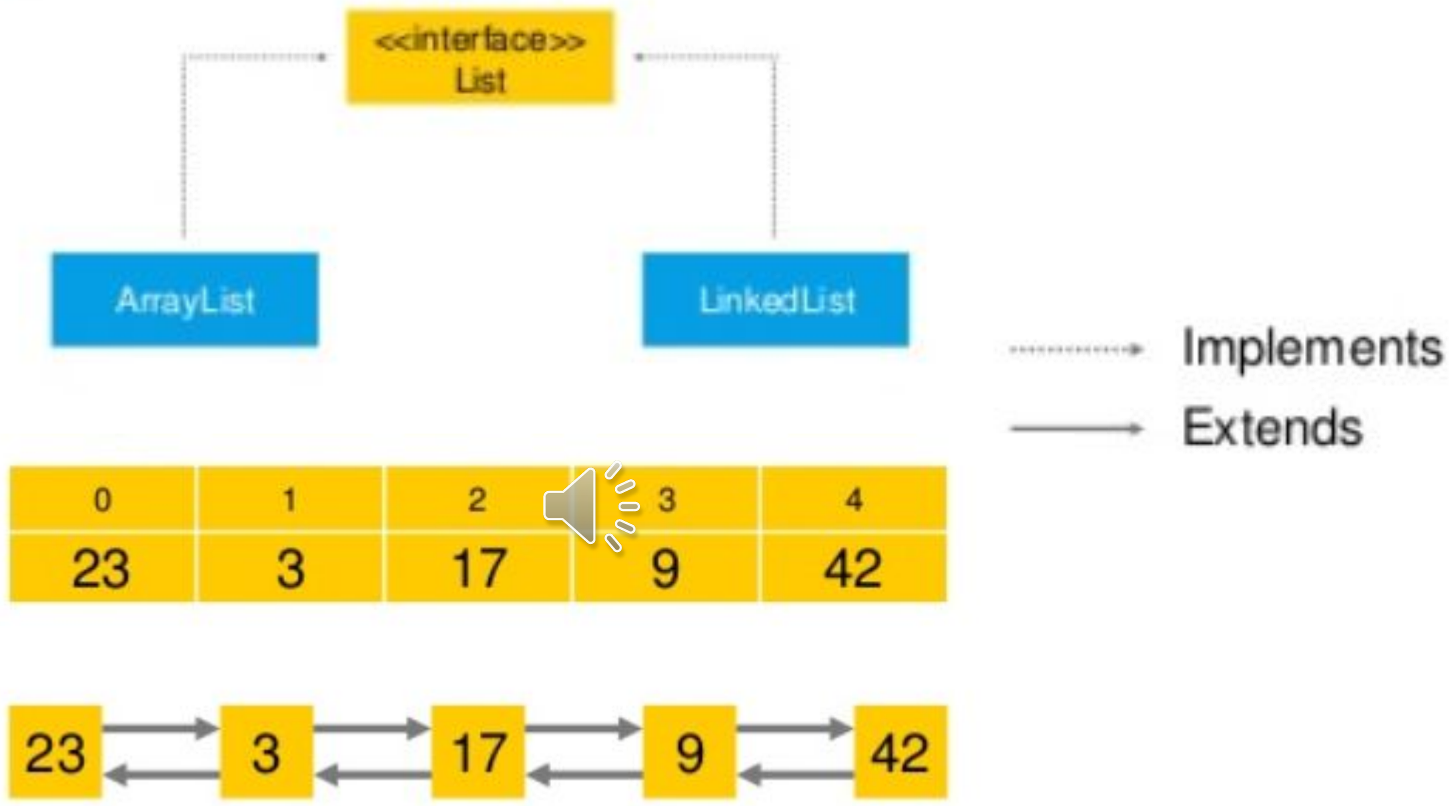
○ Java will insert the item at the end of the list

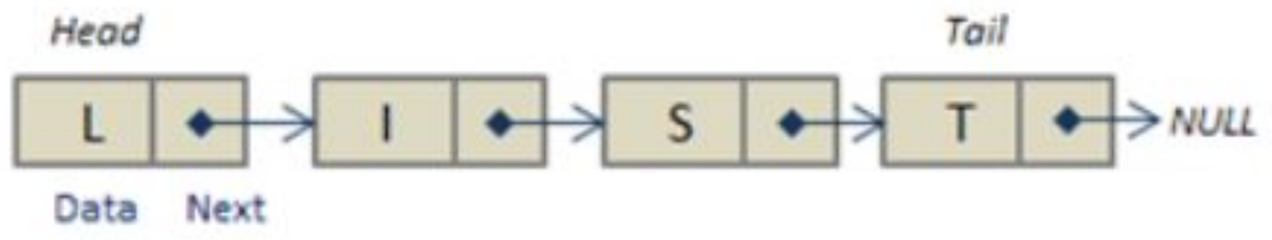
○ Java will return a compiler error

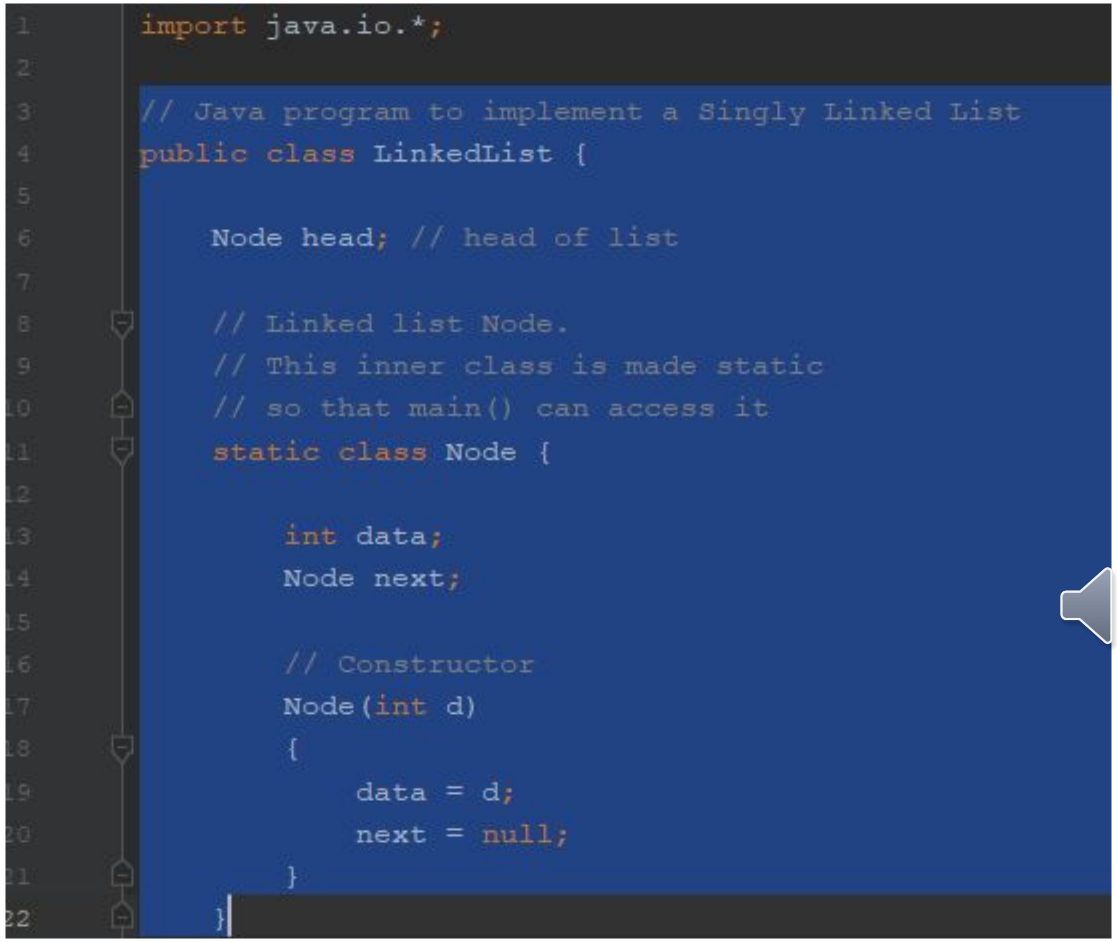
○ Java will insert the item at the beginning of the list

○ It will replace the first item

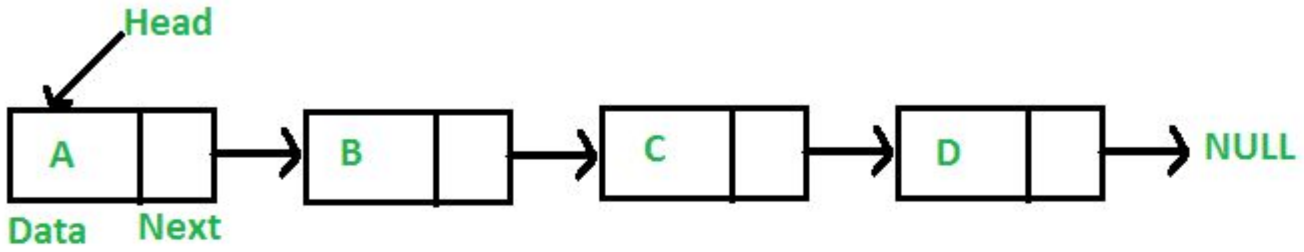
ArrayList vs. LinkedList

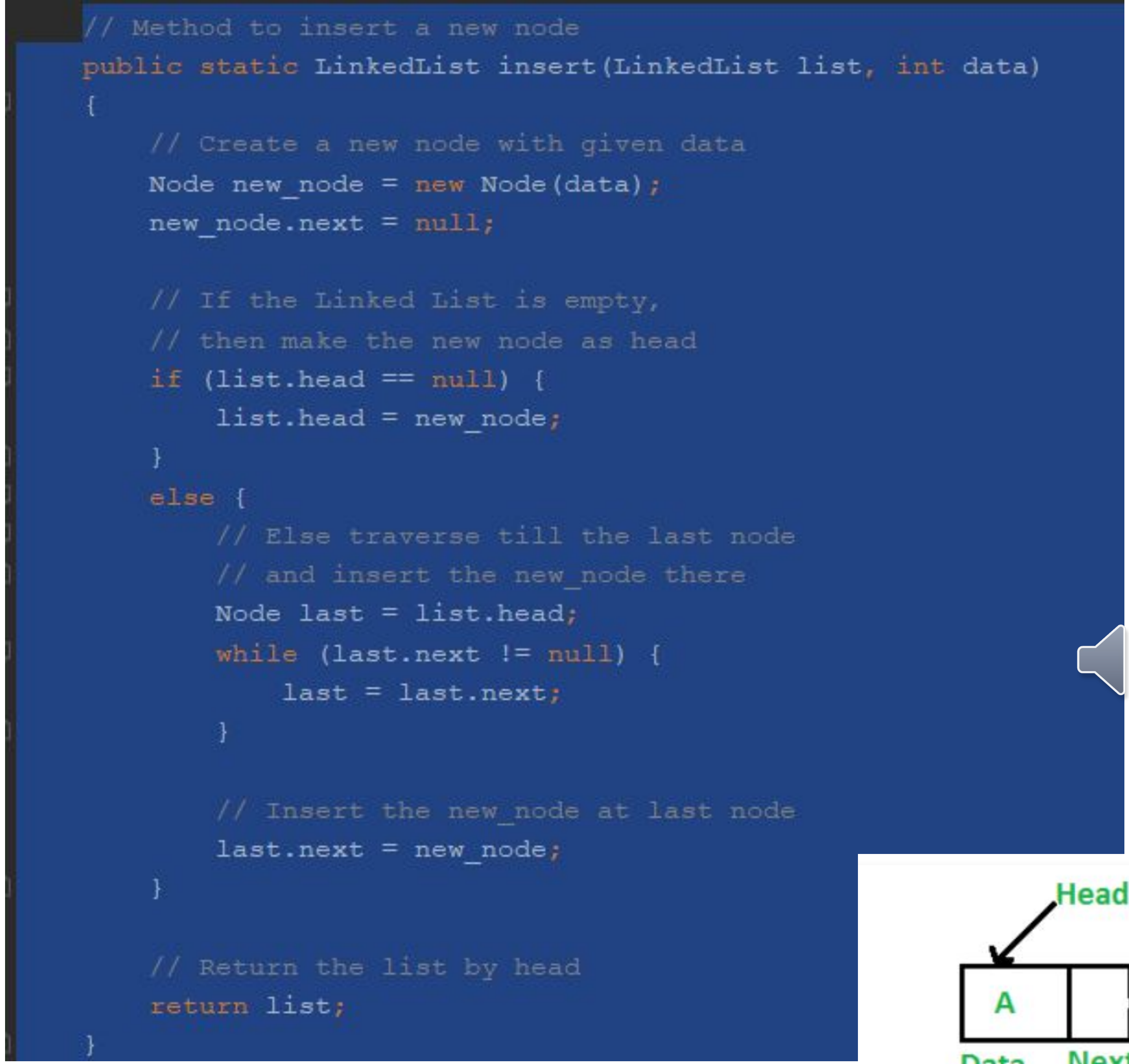



Head
Tail
L
I
S
T
NULL
Data
Next

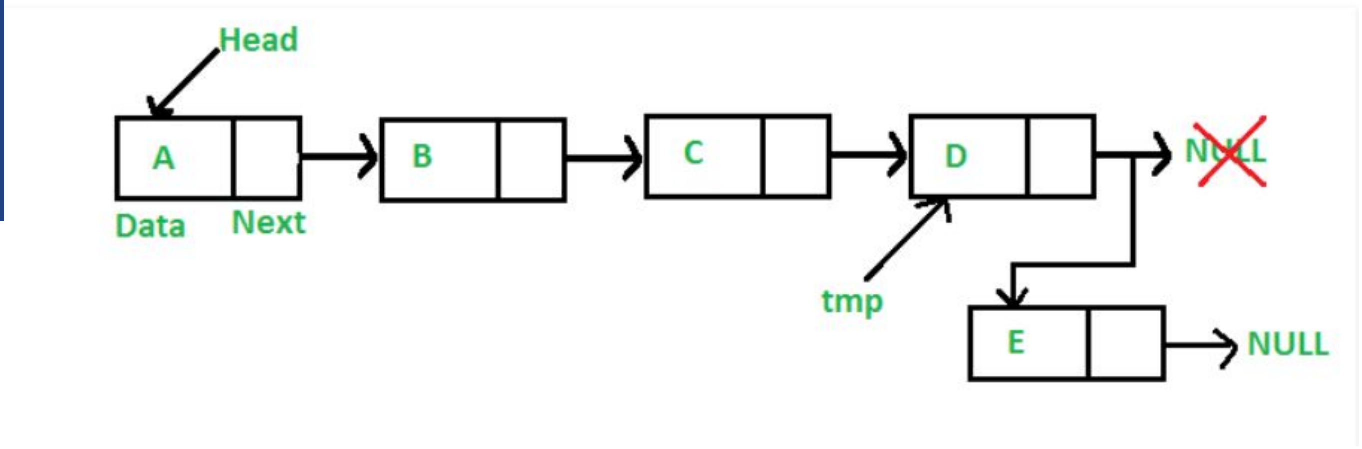


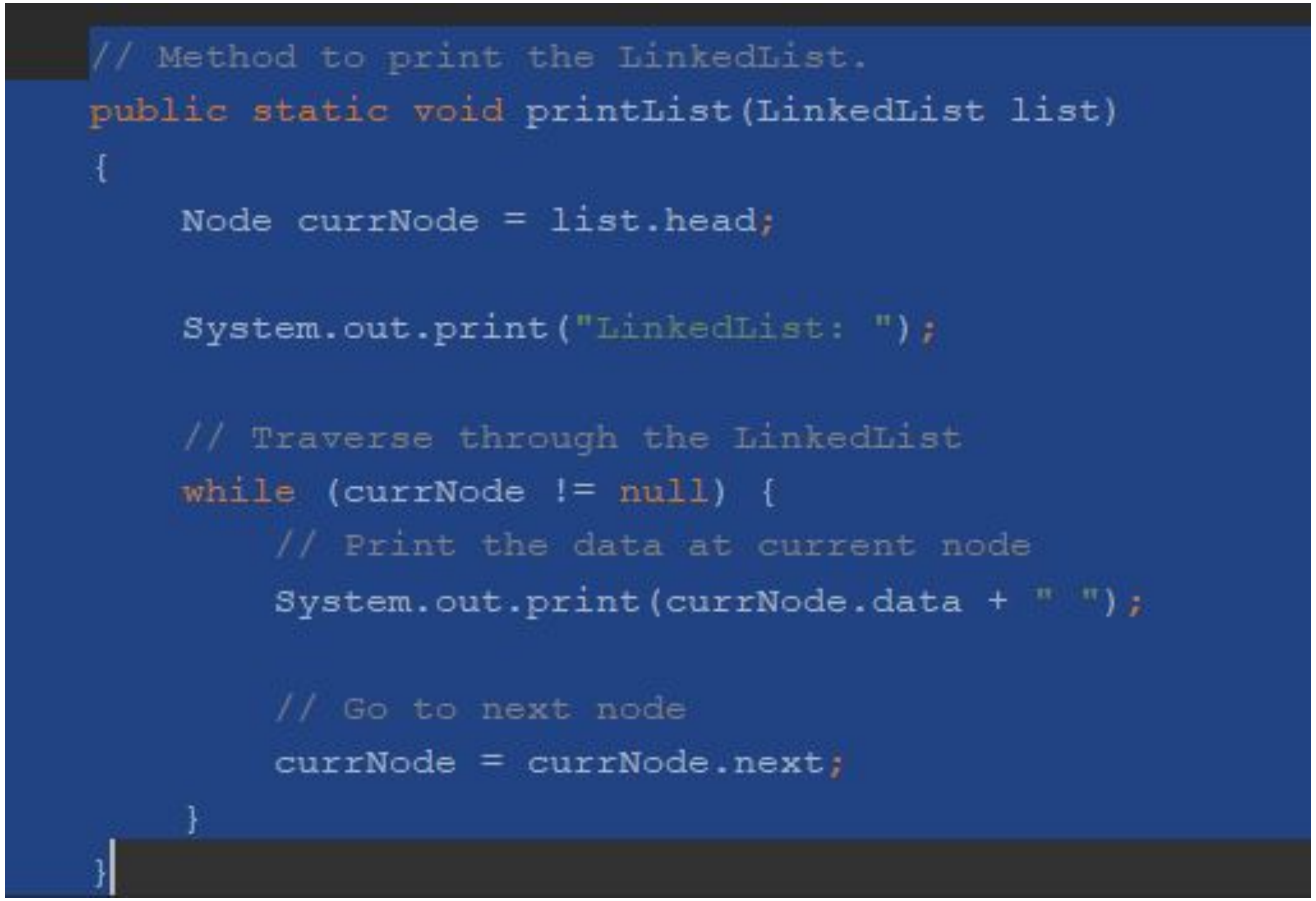
```java
import java.io.*;

// Java program to implement a Singly Linked List
public class LinkedList {

    Node head; // head of list

    // Linked list Node.
    // This inner class is made static
    // so that main() can access it
    static class Node {

        int data;
        Node next;

        // Constructor
        Node(int d)
        {
            data = d;
            next = null;
        }
    }
```



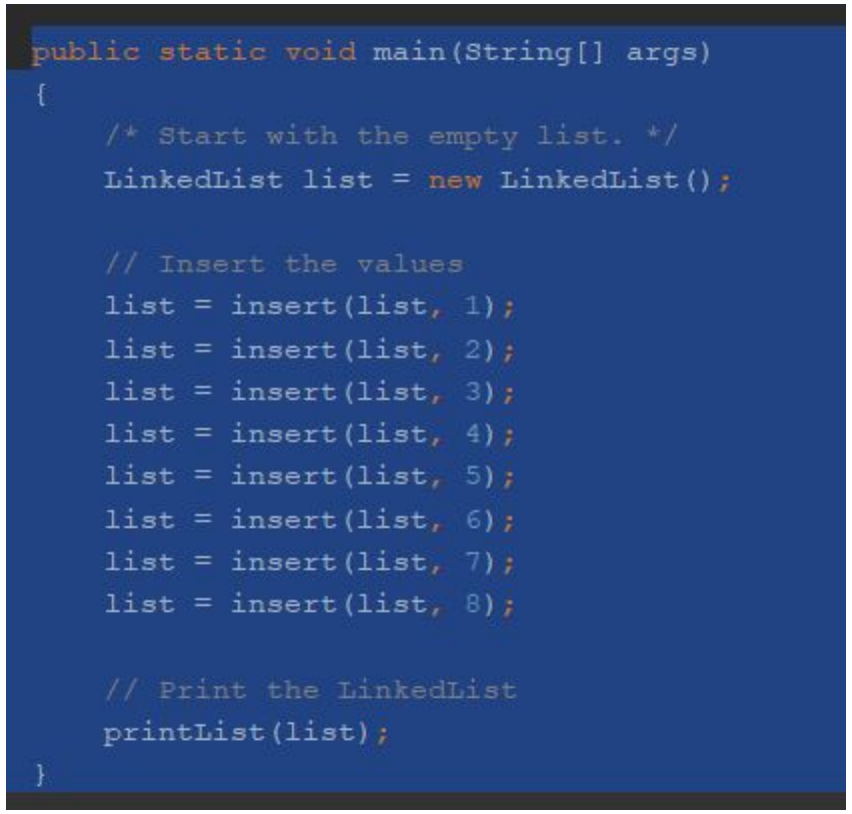


```
// Method to insert a new node
public static LinkedList insert(LinkedList list, int data)
{
    // Create a new node with given data
    Node new_node = new Node(data);
    new_node.next = null;

    // If the Linked List is empty,
    // then make the new node as head
    if (list.head == null) {
        list.head = new_node;
    }
    else {
        // Else traverse till the last node
        // and insert the new_node there
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }

        // Insert the new_node at last node
        last.next = new_node;
    }

    // Return the list by head
    return list;
}
```





```
// Method to print the LinkedList.
public static void printList(LinkedList list)
{
    Node currNode = list.head;

    System.out.print("LinkedList: ");

    // Traverse through the LinkedList
    while (currNode != null) {
        // Print the data at current node
        System.out.print(currNode.data + " ");

        // Go to next node
        currNode = currNode.next;
    }
}
```



```
public static void main(String[] args)
{
    /* Start with the empty list. */
    LinkedList list = new LinkedList();

    // Insert the values
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);

    // Print the LinkedList
    printList(list);
}
```

Note to Joe

Questions????

In the following linked list (named *m* in your code), you need to change the Z to an S. Which code example accomplishes this?

```
[M, A, R, Z];
```

- `m.add("S");`
- `m.set(3, "S");`
- `m.addFirst("S");`
- `m.addLast("S");`

The following linked list, called *j*, exists in your program:

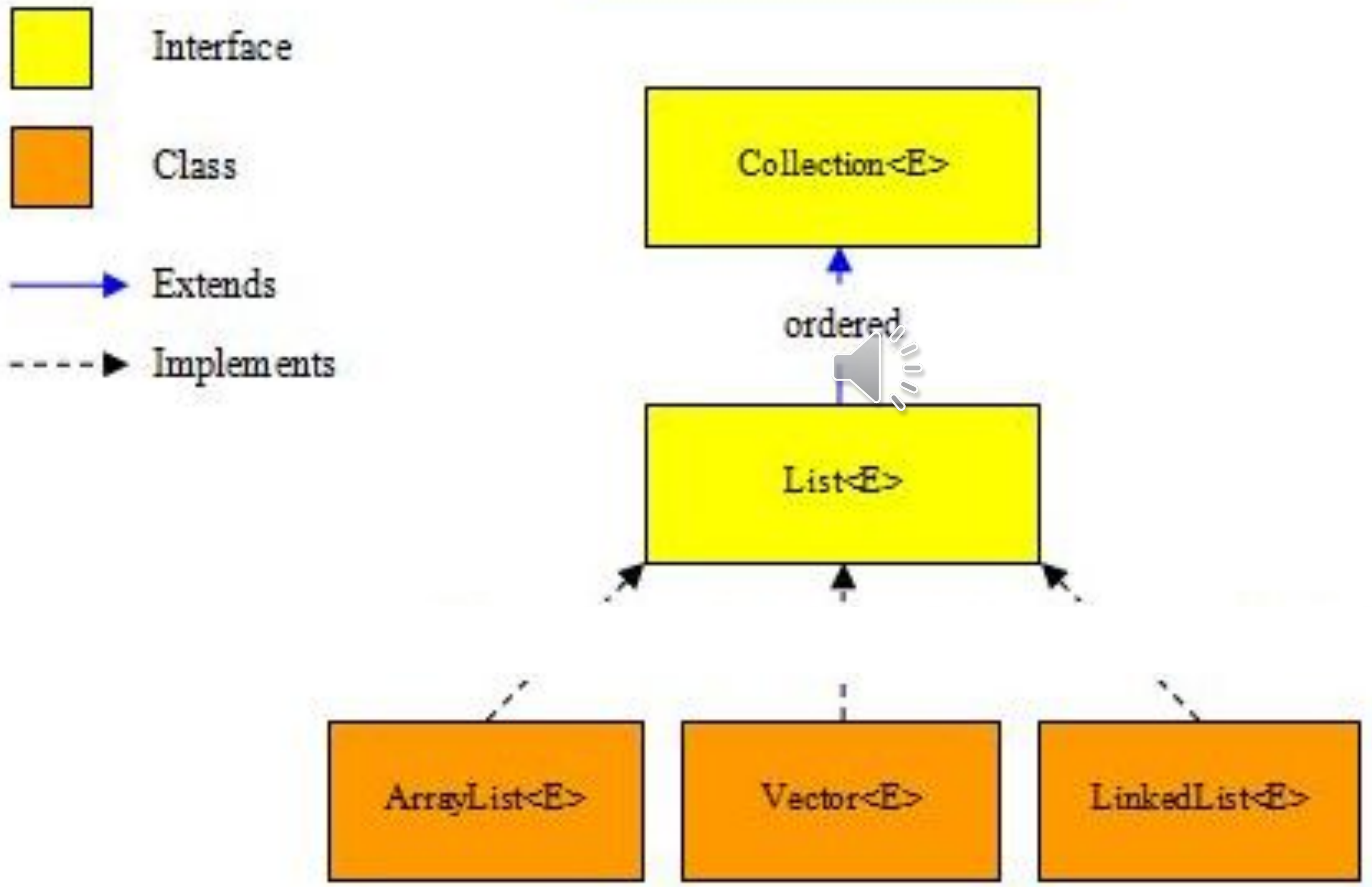
```
[J, U, P, I, T, E, R]
```

If you execute the following code, what will be the final value?

```
j.add(2, "CERES"
```

- ○ `[J, U, P, I, T, E, R, CERES]`
- ○ This will result in a compiler error.
- ○ `[J, CERES, U, P, I, T, E, R]`
- ○ `[J, U, CERES, P, I, T, E, R]`

List Hierarchy


Interface
Class
Extends
Implements
Collection<E>
ordered
List<E>
ArrayList<E>
Vector<E>
LinkedList<E>