Ice Breaker

If you could create any class to be taught at Lehigh, what would it be?

Data Structure Implementation:
Lists, Stack, Queue, Priority Queue

CSE 017 | Prof Ziltz | Spring 23

Weeks 1–8 Review: Muddiest Point?

What did we discuss before break?

What needs to be clarified from ALA6, HW3 and/or ZyBooks?

Exam average: 82% (but wide distribution)

This Week's Plan

- **Implementation of Lists**
  - **ArrayList**
  - LinkedList
- **Implementation Stack**
- Implementation of Queue
- **Implementation PriorityQueue**

Student Learning Outcomes

By the end of this week, you should be able to:

- Implement the List using an array
- Implement the List using a LinkedList (nodes)
- Implement the Stack using an arraylist
- Implement the Queue using a LinkedList
- Implement the Priority Queue using an arraylist
- Analyze the complexity of the operations on all the data structures

Why care about Data Structure Implementation?

- **Available Data Structures in Java API**
  - List, Stack, Queue (using list), PriorityQueue
- **How are they implemented?**
- **How to create new data structures?**
- **Become a data structure designer rather than a data structure user**

Data Structure: List

- **Implemented in Java.util.Collection**
- **Store data in some order**
- **Common operations on List**
  - Retrieve an element from the list
  - Add a new element into the list
  - Remove an element from the list
  - Get the number of elements in the list
- **Implementation Options:**
  - Array Based List - `ArrayList<E>`
    - Fixed array size when the list is constructed
    - New larger array created when the current array is full
  - Linked List - `LinkedList<E>`
    - Size not fixed
    - Nodes are created when an item is added
    - Nodes are linked together to form the list

ArrayList

Size = 0
Capacity = 10

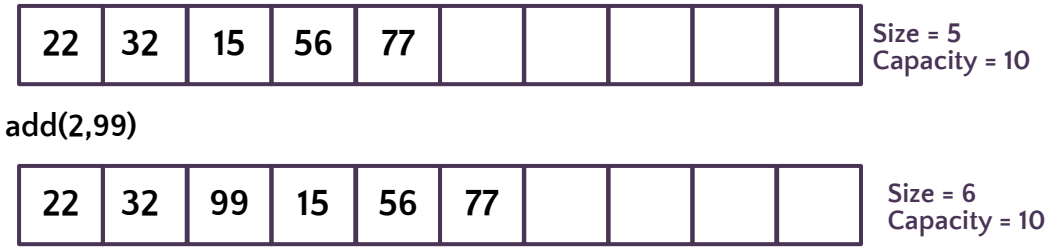
Size = 5
Capacity = 10

Size = 10
Capacity = 10

Size = 11
Capacity = 15

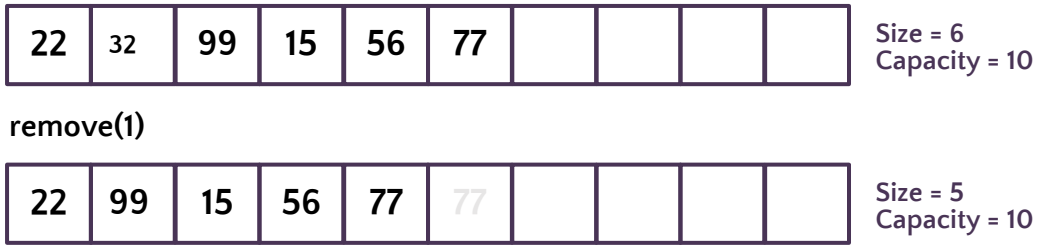
ArrayList Functions: `add`

- **Inserting an element at a specific index**
  - If (size == capacity), create a new array with new size = (1.5 * size)
    - copy all the elements from the current array to the new array
    - The new array becomes the new list (changing the reference)
  - Shift all the elements after the index, modify element at index and increase the size by 1



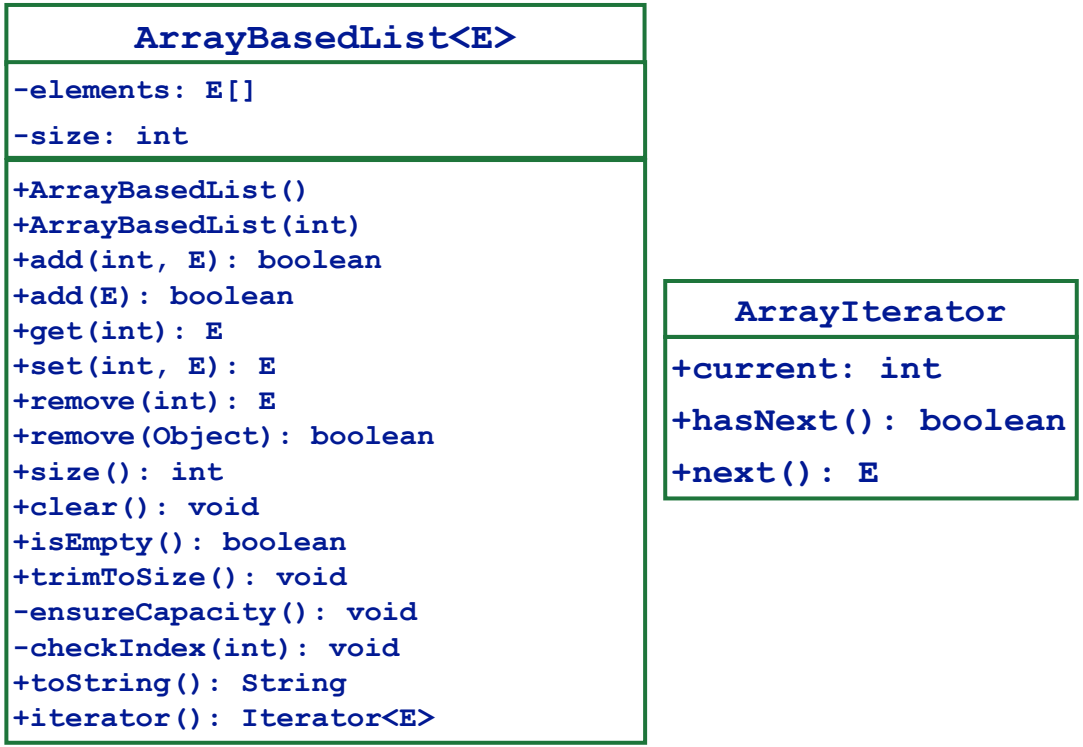
ArrayList Functions: `remove`

- **Removing an element at a specific index**
  - Shift all the elements after the index and decrease the size by 1



ArrayBasedList

Implementing our own Array-based List (instead of using that from Java Collection)



Implementing ArrayBasedList<E>

```
public class ArrayBasedList<E> {
   // data members
   private E[] elements;
   private int size;

   // Constructors
   public ArrayBasedList() {
       elements = (E[]) new Object[10];
       size = 0;
   }

   public ArrayBasedList(int capacity) {
       elements = (E[]) new Object[capacity];
       size = 0;
   }
```

Implementing ArrayBasedList<E>

```
// Adding an item to the list (2 methods)
public boolean add(E item) {
    return add(size, item);
}

public boolean add(int index, E item) {
    if (index > size || index < 0)
        throw new ArrayIndexOutOfBoundsException();
    ensureCapacity();
    for (int i = size - 1; i > index; i--)
        elements[i + 1] = elements[i];
    elements[index] = item;
    size++;
    return true;
}
```

Implementing ArrayBasedList<E>

```
// Getter and Setter
   public E get(int index) {
        checkIndex(index);
        return elements[index];
   }
   public E set(int index, E item) {
        checkIndex(index);
        E oldItem = elements[index];
        elements[index] = item;
        return oldItem;
   }
   // Size of the list
   public int size() {  return size; }
   // Clear the list
   public void clear() { size = 0; }
   // Check if the list is empty
   public boolean isEmpty() { return (size == 0); }
```

Implementing ArrayBasedList<E>

```
// Removing an object from the list
public boolean remove(Object o) {
    E item = (E) o;
    for (int i = 0; i < size; i++)
        if (elements[i].equals(item)) {
            remove(i);
            return true;
        }
    return false; }
// Removing the item at index from the list
public E remove(int index) {
  checkIndex(index);
  E item = elements[index];
  for(int i=index; i<size-1; i++)
         elements[i] = elements[i+1];
  size--;
  return item;}
```

Implementing ArrayBasedList<E>

```
// Shrink the list to size
public void trimToSize() {
    if (size != elements.length) {
        E[] newElements = (E[]) new Object[size];
        for (int i = 0; i < size; i++)
            newElements[i] = elements[i];
        elements = newElements;
    }
}
// Grow the list if needed
private void ensureCapacity() {
    if (size >= elements.length) {
        int newCap = (int) (elements.length * 1.5);
        E[] newElements = (E[]) new Object[newCap];
        for (int i = 0; i < size; i++)
            newElements[i] = elements[i];
        elements = newElements;
    }
}
```

Implementing ArrayBasedList<E>

```
// Check if the index is valid
   private void checkIndex(int index) {
       if (index < 0 || index >= size)
           throw new ArrayIndexOutOfBoundsException(
                   "Index out of bounds. Must be between 0 and " +
                           (size - 1));
   }

   // toString() method
   public String toString() {
       String output = "[";
       for (int i = 0; i < size - 1; i++)
           output += elements[i] + " ";
       output += elements[size - 1] + "]";
       return output;
   }
```

Implementing ArrayBasedList<E>

```
// Iterator for the list
public Iterator<E> iterator() {
    return new ArrayIterator();
}
// Inner class that implements Iterator<E>
private class ArrayIterator implements Iterator<E> {
    private int current = -1;

    public boolean hasNext() {
        return current < size - 1;
    }

    public E next() {
        return elements[++current];
    }
}
```

Testing ArrayBasedList<E>

```
import java.util.Iterator;
public class Test {
   public static void main(String[] args) {
       ArrayBasedList<String> cities = new ArrayBasedList<>();
       cities.add("New York");
       cities.add("San Diego");
       cities.add("Atlanta");
       cities.add("Baltimore");
       cities.add("Pittsburg");
       // toString() to display the content of the list
       System.out.println(cities.toString());
       // iterator to visit and display the elements of the list
       Iterator<String> cityIterator = cities.iterator();
       while (cityIterator.hasNext()) {
           System.out.print(cityIterator.next() + " ");
       }
       System.out.println();
       // get(index) to visit and display the elements of the list
       for (int i = 0; i < cities.size(); i++) {
           System.out.print(cities.get(i) + " ");
       }
   }
}
```

Analyzing the ArrayBasedList

What is the complexity of the operations in the ArrayBasedList?

Method	Complexity	Method	Complexity
`ArrayList()`	**O(1)**	`iterator()`	**O(1)**
`ArrayList(int)`	**O(1)**	`trimToSize`	**O(n)**
`size()`	**O(1)**	`ensureCapacity`	**O(n)**
`checkIndex()`	**O(1)**	`add(int, E)`	**O(n)**
`get(int)`	**O(1)**	`remove(int)`	**O(n)**
`set(int, E)`	**O(1)**	`toString()`	**O(n)**
`isEmpty()`	**O(1)**	`add(E)`	**O(1) - O(n)**
`clear()`	**O(1)**		

Stack and Queue

- **Stack is implemented using an array based list with access only at the end of the list (next available space)**
- **Queue (covered next week) is implemented using a linked list with access at the head and the tail**
  - Remove from head
  - Add to tail
- **Priority Queue is implemented using an array list with access at the first element and last.**
  - List stays in sorted order

Stack Implementation (using ArrayList)

```java
//either import java.util.ArrayList;
//OR use own ArrayList/ArrayBasedList class
public class Stack<E> {
    private ArrayList<E> elements;
    public Stack() {
        elements = new ArrayList<>();
    }
    public Stack(int capacity) {
        elements = new ArrayList<>(capacity);
    }
    public int size() {
        return elements.size();
    }
    public boolean isEmpty() {
        return elements.isEmpty();
    }
```

```java
public void push(E item) {
        elements.add(item);
    }
    public E peek() {
        if (isEmpty())
            throw new EmptyStackException();
        return elements.get(size() - 1);
    }
    public E pop() {
        if (isEmpty())
            throw new EmptyStackException();
        E value = peek();
        elements.remove(size() - 1);
        return value;
    }
    public String toString() {
        return "Stack: " + elements.toString();
    }
}
```

Stack Implementation (using ArrayList)

```
import java.util.Iterator;

public class Test3 {
   public static void main(String[] args) {
       Stack<String> cityStack = new Stack<>();
       cityStack.push("New York");
       cityStack.push("San Diego");
       cityStack.push("Atlanta");
       cityStack.push("Baltimore");
       cityStack.push("Pittsburg");
       System.out.println("City Stack (toString): " +
               cityStack.toString());
       System.out.print("City Stack (pop): ");
       while (!cityStack.isEmpty())
           System.out.print(cityStack.pop() + " ");
   }
}
```

Analyzing the Stack

What is the complexity of the operations in a Stack?

Method	Complexity
`Stack<>()`	O(1)
`peek()`	O(1)
`pop()`	O(1)
`push()`	O(1)/O(n)
`size()`	O(1)
`isEmpty()`	O(1)
`toString()`	O(n)

Priority Queue implementation (using ArrayList)

PriorityQueue<E>
-list: ArrayList<E> -comparator: Comparator<E>
+PriorityQueue() +PriorityQueue(Comparator<E>) +offer(E): void +poll(): E +peek(): E +size(): int +clear(): void +isEmpty(): boolean +toString(): String

Priority Queue implementation (using ArrayList)

```
public class PriorityQueue<E> {
   private ArrayList<E> list;
   private Comparator<E> comparator;
   public PriorityQueue() {
       list = new ArrayList<>();
       comparator = null;
   }
   public PriorityQueue(Comparator<E> c) {
       list = new ArrayList<>();
       comparator = c;
   }
   public E poll() {
       E value = list.get(0);
       list.remove(0);
       return value;
   }
```

Priority Queue implementation (using ArrayList)

```
public void offer(E item) {
    int i, c;
    for (i = 0; i < list.size(); i++) {
        if (comparator == null)
            c = ((Comparable<E>) item).compareTo(list.get(i));
        else
            c = comparator.compare(item, list.get(i));
        if (c < 0) // will be placed after something that is equal
            break;
    }
    list.add(i, item);
}
```

Priority Queue implementation (using ArrayList)

```
    public E peek() { return list.get(0); }
    public String toString() {
        return "Priority Queue: " + list.toString();
    }

    public int size() {
        return list.size();
    }

    public void clear() {
        list.clear();
    }

    public boolean isEmpty() {
        return list.size() == 0;
    }
}
```

Priority Queue implementation (using ArrayList)

```
public class Test5 {
   public static void main(String[] args) {
       PriorityQueue<String> cityPriorityQueue = new PriorityQueue<>();
       cityPriorityQueue.offer("New York");
       cityPriorityQueue.offer("San Diego");
       cityPriorityQueue.offer("Atlanta");
       cityPriorityQueue.offer("Baltimore");
       cityPriorityQueue.offer("Pittsburg");
       System.out.println("\nCity Priority Queue: " +
               cityPriorityQueue.toString());
       System.out.print("City Priority Queue (poll): ");
       while (!cityPriorityQueue.isEmpty()) {
           System.out.print(cityPriorityQueue.poll() + " ");
       }
   }
}
```

Analyzing a PriorityQueue

What is the complexity of the operations in a PriorityQueue?

Method	Complexity
`Queue<>()`	**O(1)**
`offer(E)`	**O(n)**
`poll()`	**O(n)**
`peek()`	**O(1)**
`size()`	**O(1)**
`clear()`	**O(1)**
`isEmpty()`	**O(1)**
`toString()`	**O(n)**

Sorting: Bubble Sort

- Simple sorting algorithm
- Make several passes through the array. On each pass, successive neighboring pairs are compared. If the pair is not in order, its values are swapped; otherwise, the remain unchanged.
- Large values “bubble” to the end of the array (if ascending order)

```
int [] numbers = {9,1,6,5,2,3};
```

```
int [] numbers = {9,1,6,5,2,3};
```

```
{ 9 , 1 , 6 , 5 , 2 , 3 } //start
```

First Pass

```
{ 1, 9, 6, 5, 2, 3}
{ 1, 6, 9, 5, 2, 3}
{ 1, 6, 5, 9, 2, 3}
{ 1, 6, 5, 2, 9, 3}
{ 1, 6, 5, 2, 3, 9}
```

Second Pass

```
{ 1, 6, 5, 2, 3, 9}*
{ 1, 5, 6, 2, 3, 9}
{ 1, 5, 2, 6, 3, 9}
{ 1, 5, 2, 3, 6, 9}
{ 1, 5, 2, 3, 6, 9}*
```

Third Pass

```
{ 1, 5, 2, 3, 6, 9}*
{ 1, 2, 5, 3, 6, 9}
{ 1, 2, 3, 5, 6, 9}
{ 1, 2, 3, 5, 6, 9}*
{ 1, 2, 3, 5, 6, 9}*
```

Fourth Pass

```
{ 1, 2, 3, 5, 6, 9}*
...
```

Fifth Pass

```
{ 1, 2, 3, 5, 6, 9}*
...
```

Sixth Pass

```
{ 1, 2, 3, 5, 6, 9}*
...
```

Sorting: Bubble Sort

LISTING 23.2 Bubble Sort Algorithm

```
1  for (int k = 1; k < list.length; k++) {
2    // Perform the kth pass
3    for (int i = 0; i < list.length - k; i++) {
4      if (list[i] > list[i + 1])
5        swap list[i] with list[i + 1];
6    }
7  }
```

LISTING 23.3 Improved Bubble Sort Algorithm

```
1  boolean needNextPass = true;
2  for (int k = 1; k < list.length && needNextPass; k++) {
3    // Array may be sorted and next pass not needed
4    needNextPass = false;
5    // Perform the kth pass
6    for (int i = 0; i < list.length - k; i++) {
7      if (list[i] > list[i + 1]) {
8        swap list[i] with list[i + 1];
9        needNextPass = true; // Next pass still needed
10     }
11   }
12 }
```

Quick Summary

- **Implementing Data Structures**
- **Lists: ordered set of data**
  - Array based list
  - linked List
- **Stack**
  - implemented using ArrayList
- **Queues**
  - Queue and PriorityQueue using LinkedList and ArrayList
- **Complexity of data structure operations**