CS206

ArrayList
Array

- An array is a sequenced collection of homogenous variables (elements)
- Each element of an array has an index
- The entire array is contiguous in memory
  - allocated by new (e.g., new int[10])
- The length of an array is fixed and cannot be changed

A

\[
\begin{array}{cccccccc}
A & 0 & 1 & 2 & i & n-1 & n \\
\end{array}
\]
Array/List

- Dynamically-sized array
- Stores an ordered sequence of objects
  - Not sorted, ordered in the sense that arrays are ordered
- Can grow and shrink when items are added/removed
- Standard array features all supported, but with different syntax
Array/List

- ArrayList is implemented with an array
- A variable (call it count) keeps track of the number of elements in the ArrayList
  - deletion
    - shift elements to the left and decrement count
  - addition
    - put new item on end and increment count
    - if not enough space
      - Create new, bigger array
      - Copy elements of old array into new one
Insertion

- In an operation $\text{add}(i, 0)$, we make room for the new element by shifting forward/to the right the elements $A[i], \ldots, A[n - 1]$.
Deletion

- In an operation \( \text{remove}(i) \), we fill the hole by shifting backward/to the left the elements \( A[i+1], \ldots, A[n-1] \)
Java Interfaces

- Java allows only single inheritance.
  - A class can only extend one class
  - As a result, Java does not need any collision resolution.
- BUT a class can “implement” any number of Interfaces
  - Interfaces only define methods
    - they do not provide method bodies so no collision resolution required.
public interface ArraListInterface<T> {
    boolean add(T t);
    void add(int index, T t) throws IndexOutOfBoundsException;
    T get(int index) throws IndexOutOfBoundsException;
    void remove(int index) throws IndexOutOfBoundsException;
    boolean set(int index, T t) throws IndexOutOfBoundsException;
    int size();
    int indexOf(T t);
    void clear();
}
public class ArraList<T> implements ArraListInterface<T> {
    private int capacity = 10;
    private static final double GROWTH_RATE = 1.618033; // the golden mean
    private int count; // number of items currently in ArraList
    private T[] arra; // the array underlying the ArraList
    public ArraList() {
        arra = (T[]) new Object[capacity];
        count = 0;
    }
    public ArraList(int initialCapacity) {
        capacity = initialCapacity;
        arra = (T[]) new Object[capacity];
        count = 0;
    }
}
More implementation of ArrayList

```java
public boolean add(T t) {
}
```

Suggestion: start by drawing a good picture of what you want to do
YM implementation of ArraList

```java
public void remove(int index) throws IndexOutOfBoundsException {
}
```
Creation with Type Parameters

- When constructing an `ArrayList`, you must specify the type of elements via `<>

```java
ArrayList<String> l1 = new ArrayList<>();
ArrayList<Integer> l2 = new ArrayList<>();
```
Example usage

• Write a program to collect then print all unique words in a file

• Problem: you do not know the number of distinct words!

  • Solution
    • allocate a really big array
    • Use ArraList!
WordCounter —
Count the unique words in file!

WordCounter.java
java.util.ArrayList

- Implements much the same interface as ours
  - Their implementation has a few more functions
- Theirs is probably more more more efficient.
- Part of Java collections framework
- import java.util.ArrayList
- Use ArrayList rather than ArraList (ours) for Homework 3 and Lab 2.
Collections

The diagram illustrates the hierarchy and relationships among various collection types in Java. The main classes are:

- **Collection**
- **Iterable**
- **AbstractCollection**
- **AbstractSequentialList**
- **AbstractList**
- **AbstractQueue**
- **AbstractSet**
- **AbstractMap**

Key collections include:

- **Set**
- **List**
- **Queue**

Specific implementations are:

- **SortedSet**
- **NavigableSet**
- **TreeSet**
- **LinkedHashSet**
- **LinkedTreeSet**
- **ArrayList**
- **Vector**
- **PriorityQueue**
- **LinkedBlockingQueue**
- **Stack**

This diagram helps in understanding the structure and implementation of collection classes in Java.