CS206

Software/OO design
ArrayList

Software design: Already discussed

- Good variable names
- Comments
- In Java
  - Avoid statics
  - Minimize main
  - Use inheritance and class design

Software Design Goals

- Robustness
  - software capable of error handling and recovery
  - programs should never crash
    - "falling with style" is not crashing
- Adaptability
  - software able to evolve over time and changing conditions (without huge rewrites)
- Reusability
  - same code is usable as component of different systems in various applications
  - The story of Mel — https://www.cs.utah.edu/~elb/folklore/mel.html

OOP Design Principles

- Modularity
  - programs should be composed of "modules" each of which do their own thing
    - each module is separately testable
  - Large programs are built by assembling modules
  - Objects (Classes) are modules
- Abstraction
  - Get to the core — non-removable essence of a thing
  - Most pencils are yellow, but yellowness does not required
- Encapsulation
  - Nothing outside a class should know about how the class works.
    - For instance, does the Object class have any instance variables. (Of what type?)
  - Allows programmer to totally change internals without external effect

OOP Design

- Responsibilities/Independence: divide the work into different classes, each with a different responsibility and are as independent as possible
- Behaviors: define the behaviors for each class carefully and precisely, so that the consequences of each action performed by a class will be well understood by other classes that interact with it.

Constructors

- Constructors are never inherited
- A class may invoke the constructor of the class it extends via a call to `super` with the appropriate parameters
  - e.g. `super()`
  - `super` must be in the first line of constructor
  - If no explicit call to `super`, then an implicit call to the zero-parameter `super` will be made
- A class make invoke other constructors of their own class using `this()`
  - `this` must be first
  - Cannot explicitly use both `super` and `this` **in single constructor**
    - One or the other would not be first
  - See ArraList (slide 14)

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.
    - Programmer of class that "implements" interface MUST write method bodies.

Java Interfaces

In a file Vehicle.java

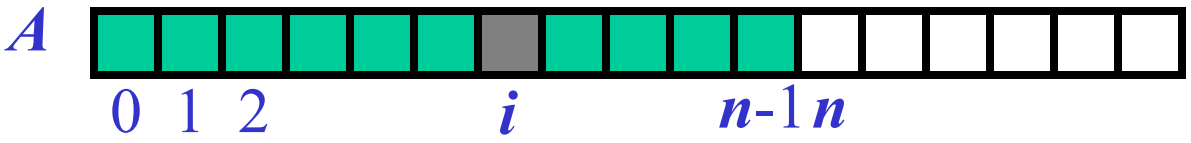
Interfaces are usually EXTENSIVELY documented so programers know what is intended for implementation

```
public interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

Methods defined in interfaces are always public, so public can be omitted. Clashes with class definition in which “” indicates package (Horrific inconsistency!)

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 can not be changed



ArrayList

- 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

Interface for ArraList

```
public interface ArraListInterface {
    boolean add(Object t);
    void add(int index, Object t) throws IndexOutOfBoundsException;
    Object get(int index) throws IndexOutOfBoundsException;
    void remove(int index) throws IndexOutOfBoundsException;
    boolean set(int index, Object t) throws IndexOutOfBoundsException;
    int size();
    int indexOf(Object t);
    void clear();
}
```

show in VSC

ArrayList implementation

- ArrayList is usually implemented with 2 private variables
  - an array to hold information
  - A variable (call it `count`) keeps track of the number of elements in the ArrayList
- Key Operations of Array List
  - 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
  - deletion
    - shift elements to the left and decrement `count`
    - (Optional)If number of elements in AL is much smaller than AL, shrink.

Implementing ArraListInterface

```
public class ArraList implements ArraListInterface {
    private static final int DEF_CAPACITY = 10;
    private static final double GROWTH_RATE = 1.618033; // the gol
mean
    private int count; // number of items currently in ArraList
    private Object[] arra; // the array underlying the ArraList
    public ArraList() {
       this(DEF_CAPACITY);
    }
    public ArraList(int initialCapacity) {
       arra = new Object[capacity];
    }
}
```

Size, Clear

```
/**
 * Returns the number of elements in this list.
 *
 * @return the number of elements in this list.
 */
int size() {
    return count;
}

/**
 * Removes all of the elements from this list.
 * The list will be empty after this
 * call returns.
 */
void clear() {
    count=0;
    // Enough?????
}
```

Get/Set

```
public Object get(int index) throws IndexOutOfBoundsException {
        if (index > count) {
            throw new IndexOutOfBoundsException("Can only get where
there are  already items");
        }
        if (index < 0) {
            throw new IndexOutOfBoundsException("Cannot store to
negative location");
        }
        return arra[index];
    }
```

Add to ArraList

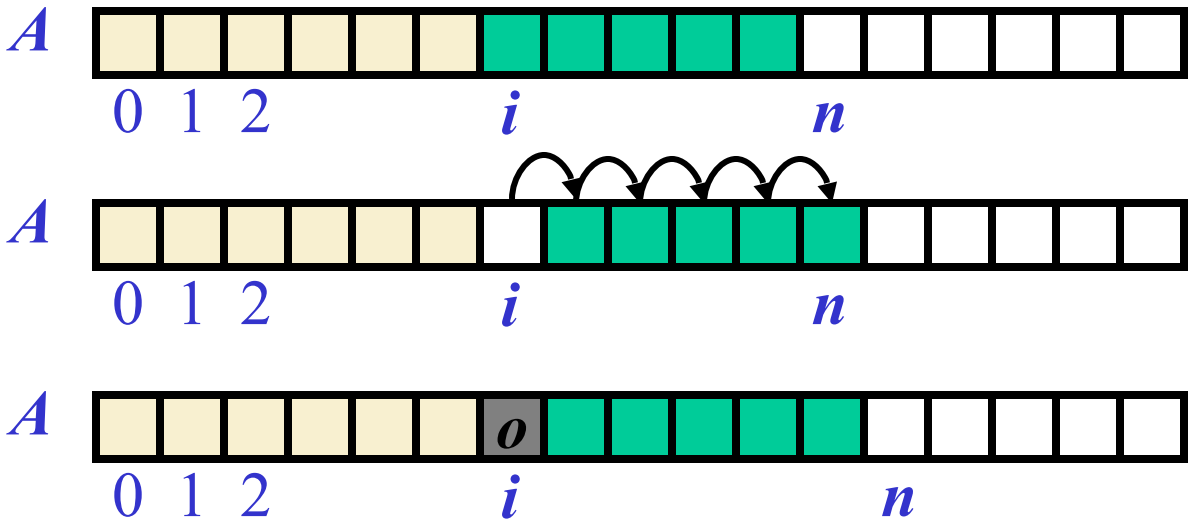
```
/**
 * Add an item to the arraylist
 *
 * @param t the item to be added return true.
 */

 void add(Object t) throws IndexOutOfBoundsException;
```

Simplest – just put the item into the array and increment the counter that holds the number of items

What to do is there is no space for another item – need to grow!

Add At Location

- In an operation `add(i, o)`, we make room for the new element by shifting forward/to the right the elements `A[i]`, …, `A[n - 1]`

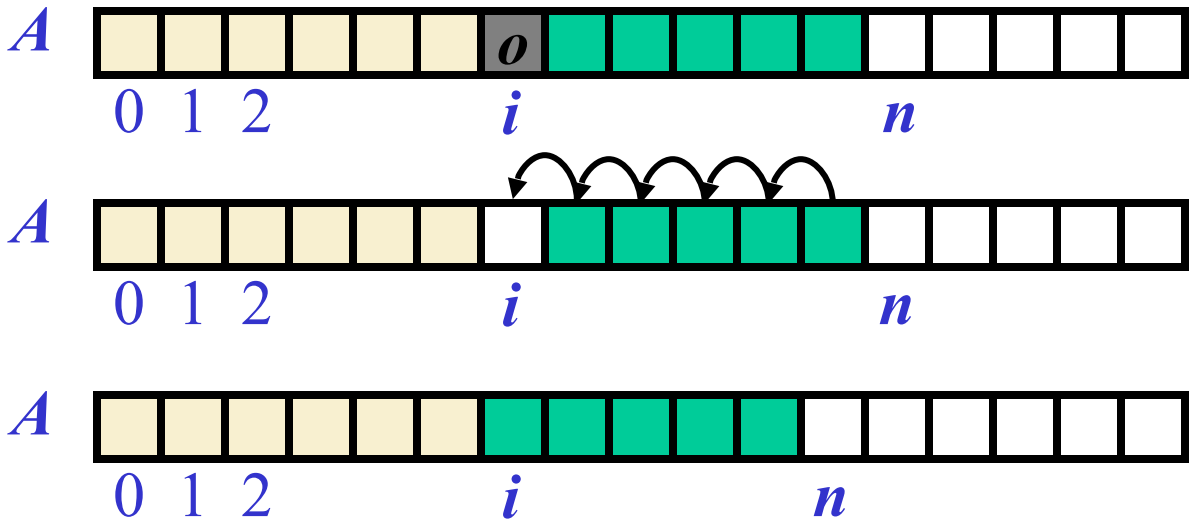


Add at a location

```
/**
 * Add an item to the array list at a particular location. Inserts the specified
 * element at the specified position in this list. Shifts the element currently
 * at that position (if any) and any subsequent elements to the right (adds one
 * to their indices).
 *
 * @param index the location to add the item at
 * @param t  the item to be added
 * @return
 * @throws if the index is out of range (index < 0 || index > size())
 */
boolean add(int index, Object t) throws IndexOutOfBoundsException;
```

Deletion

- In an operation `remove(i)`, we fill the hole by shifting backward/to the left the elements `A[i + 1]`, …, `A[n - 1]`



Lab

Write a method to implement remove for array list

```
/**
 * Removes the element at the specified position in this list.
 * Shifts any subsequent elements to the left
 * (subtracts one from their indices).
 *
 * @param index the index of the element to be removed
 */
void remove(int index) throws IndexOutOfBoundsException;
```

Suggestion: start by drawing a good picture of what you want to do label the picture extensively

Generics

- A way to write classes and methods that can operate on a variety of data types without being locked into specific types at the time of definition
- Write definitions & implementations with "Generic" parameters
- The generics are instantiated (locked down) when objects are created

Generic Methods

```
import java.util.Random;
/************************
 * @author gTowell
 * Created: August 28, 2019
 * Modified: Jan 24, 2019
 * Purpose:
 * Generic Methods
 ************************/
public class GenericMethod {
    public static void main(String[] args) {
        Integer[] jj = { 1, 2, 3, 4, 5, 6, 7, 8, 9 }; // NOTE AUTOBOXING!!!
        new GenericMethod().randomize(jj);
        for (int j : jj)
            System.out.println(j);
        String[] ss = { "A", "B", "c", "d", "E", "F" };
        new GenericMethod().randomize(ss);
        for (String s : ss)
            System.out.println(s);
    }

    public <T> void randomize(T[] data) {
        Random r = new Random();
        for (int i = 0; i < data.length; i++) {
            int tgt = r.nextInt(data.length);
            T temp = data[i];
           data[i]=data[tgt];
           data[tgt]=temp;
        }}}
```

— generic swap method
— use reflection to check class

Generic Class

```
import java.io.BufferedReader;
import java.io.StringReader;
/**
 * Simple generic class example
 * @author gtowell
 *
 * @param <A>
 */
public class GenericClass<A> {
    /** A non-generic value */
    private double amount;
    /** A generic value */
    private A otherValue;
    /**
     * Constructor.
     * @param other the generic value
     * @param amt a double value.
     */
    public GenericClass(A other, double amt) {
        this.otherValue = other;
        this.amount = amt;
    }
    public static void main(String[] args) {
        GenericClass<String> gString = new GenericClass<String>("ASDF", 24.5);
        System.out.println(gString);
        GenericClass<Double> gDouble = new GenericClass<Double>(99.5, 44.5);
        System.out.println(gDouble);
        GenericClass<BufferedReader> gBR = new GenericClass<BufferedReader>(
                new BufferedReader(new StringReader("When in the course")), 99.8);
        System.out.println(gBR);
    }}
```

write a toString function for this class

Generics Restrictions

- No instantiation with primitive types
  - `Genre<Double>` ok, but `Genere<double>` is not
- Can not declare static instance variables of a parameterized type
- Can not create arrays of parameterized types
  - but you can create an array of `Object` then cast `new T[10]`
    - `(T[])new Object[10]`

My implementation of ArraList

```
public void remove(int index)
        throws IndexOutOfBoundsException {

}
```

Creation with Type Parameters

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

```
ArraList<String> l1 = new ArraList<>();
ArraList<Integer> l2 = new ArraList<>()
```

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 efficient.
- Part of Java collections framework
- import java.util.ArrayList
- Use ArrayList rather than ArraList (ours) for Homework 3 and Lab 2.

Collections