Objectives: constructors; data structures: Maps;

Up next: MP5 (Monday, 8PM)

final exam date set

Tu 12/13/2016 1:30-4:30PM

conflict: We 12/14/2016 7:00-10:00PM

1. **What is a constructor? Why do we use them? How are they implemented?**

2. Rewrite the following code to **use constructors** instead of the `set___` and `copy` methods.

```
Ghost g1 = new Ghost();
g1.setX(10); g1.setY(20);
g1.setEdible(true);

Ghost g2 = g1.copy();
```

… and write the two new Ghost constructors for the Ghost class:

i) a constructor that takes 3 parameters

ii) a copy-constructor that takes a reference to another ghost.

```
class Ghost {
   private int x,y;
   private boolean edible;
```

3. **Make a list of U.S. States:**

```
   StateList list = new StateList();
   State ptr = new State("Michigan",0.52, 0.45);
   list.add(ptr);
.
.
.

public class StateList {
     private State[] array = new State[0]; // empty array of pointers.
          // Note Each time add is called we'll make a larger array.

     public State getState(int i) { return array[i];}
     public int getSize() { return array.length; }

     public void add(State s) {
        State[] temp = new State[ this.array.length + 1];
       for (int i=0;i<state.length;i++) temp[i] =  array[i];   ???

       temp[ temp.length - 1 ] =s;                              ???
       this.array = temp; // array pointer now looks at new array
     }

     public void addAll(StateList other) {//Spot the error :-)  ???
       for(int x=0; x < other.length;x++)
            add(other.getState(x));
     }
     // returns states where state.repub > 0.5
     public StateList getRepublicanStates() {
          StateList result = new StateList();
          for(int x=0; x< array.length; x++) {
               State state = getState(x);
               if(state.getRepub() > 0.5)
                    result.add( state );
          }
          return result;
     }
     // ---- CONSTRUCTORS ----
     public StateList() { // do nothin'
     }
     public StateList( StateList other) {
          array = new State[ other.getSize() ];
          for(int x=0; x< array.length; x++) {
               array[x] = other.getState(x); // SHALLOW COPY or

               array[x] = _____// DEEP
           }
     }
}
```

4. **MAPS** (or **dictionaries**): *'collection'* of associations between key-value pairs.

Examples: dictionaries, phonebooks, color tables, …

5. **Implement Caller ID:** MAPS (Store and retrieve a value for a particular key)

```
public class CallerIdPair {
    public int _______; // the extension (a unique key)
    public  String  _______; // the value (can be anything)
}

public class CallerIdMap {

// use an array of pairs
  private


  public ______ add(int extn, String name) {
    // for now, assume that the extension (the key)
    // has not already been added to this map.
    // better implementations would prevent or
    // remove/replace an existing match.









   }

   public String get(int extn) {
      // return "?" if we do not know this extension's name









      }
}
```

6. **Complete** `.equals` **and write the two Ghost constructors so we can make ghosts such as :**

```
new Ghost(); // creates ghost at (1, random Y position)
new Ghost( new int[] {15,20} );  //ghost at (15,20)

public class Ghost {
  private static int count=0;
  private static int nextId() {
    count ++;  // first ghost will have an id of 1
    return count;
  }

  // each ghost has an x,y and unique id
  private int x=1 ,y=2+ (int)(Math.random()*10);
  private int id;// Your c'tor sets id to a unique value

  public String toString() {
       return "Ghost #"+id+": "+x+","+y;
  }

  public boolean equals(Object other) {
     if(other instanceOf Ghost) {
        Ghost g = (Ghost)other; // zombie
        return _______
    } else  return false;
  }
```