CS 125 - Lecture 27

Objectives: constructors, accessors, lists and MAPS

Up next: MP5 due tonight; MP6 is out tomorrow. Quiz this week.

1. Complete the US State class below so that we can create states in the following way :

```
State s1 = new State();

// 52% are democ. voters, 45% are repub. votes. 3% other-
State ptr = new State("Michigan",0.52, 0.45);

State copyOfMichigan = new State(ptr);

class State {
   private String name
   private double dem; // likelihood of democratic result 0..1
   private double repub; // likelihood of republican result 0..1
   private double other; //likelihood of independent results 0..1
   double getDem()   {                                  }
   double getRepub() {                                  }
   double getOther() {                                  }
   String getName()  {                                  }
   String toString() { return getName() + ": " + getDem()
                    + ", " + getRepub() + ", " + getOther(); }
```

2. **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
           }
     }
}
```

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

Examples: dictionaries, phonebooks, color tables, …

6. **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 phoneNumber, 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 phoneNumber) {
      // return "?" if we do not know this extension's name




      }
}
```

7. **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 constructor 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;
  }
```

CS 125 - Lecture 27

Objectives: inheritance; super

MP5 due tonight; MP6 is out

1. **Discuss one awesome and one not-so-awesome trait you have likely inherited from your parents …**

2. **Class hierarchy:**

3. **Subclasses:**

- The inherited fields/variables can be used directly, just like any other fields.
- You can declare a field in the subclass with the same name as the one in the superclass, thus ***hiding*** it (not recommended).
- You can declare new fields in the subclass that are not in the superclass, thus ***extending*** the subclass.
- The inherited methods can be used directly as they are.
- You can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus ***overriding*** it.
- You can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.
- You can declare new methods in the subclass that are not in the superclass, thus ***extending*** the subclass.
- Constructors are not inherited. You can (and often will) write a subclass constructor that invokes the constructor of the superclass.

2a. **Inheritance menagerie in Java**

```
public class Animal {
   public boolean isMultiCellular;
   public Animal[] parents;
   public mateWith(Animal other) { … }
}

public class Mammal extends Animal {
   public int averageFurLength;
   public int gestationInMonths;
}

public class Reptile extends Animal {
   public boolean hasLegs;
}

public class Dog extends Mammal {
   public String breedType;
}
```

2b. **Inheritance menagerie in Java:**

Think: " … is a … "

```
Animal a = new Animal();
Mammal m = new Mammal();
Dog d = new Dog();

System.out.println(m instanceof Animal);
System.out.println(d instanceof Mammal);
System.out.println(d instanceof Animal);

Dog fido = new Dog();
Animal a = fido;
Object o = fido;
```

Polymorphism: the ability of an object to take on many forms. Any Java object that can pass more than one *IS-A* test is considered to be polymorphic. Polymorphism means different objects can respond to the same message in different ways.

Are polymorphic objects rare in java?

4. **Inheritance in-class demo:**

5. **Let's build a game...**

```
public class Sprite {
    private int x,y,dir,shape;
    public void setX(int newX) { if(newX>0) this.x = newX; }
    public int getX() { return x; }
    public int getShape() {return shape;}
    public void move() { if(dir==1) x++; …}

    public void draw() {
      if(shape==1) Zen.drawImage("InkyGhost.png", x, y);
      if(shape==2) Zen.drawImage("Pacman.png", x, y);
      //    ... there has to be a better way...
   }

// Create a constructor to initialize the sprite using a string
whose format is xvalue,yvalue (i.e. comma-separated values)




}
```

6. **Write code to create sprites based on the position data in a file.**

```
public class Game {
   public static void main(String[] ) {









   }
}
```

7. **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 constructor 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;
  }
```