CS 125 - Lecture 29

**Objectives:** inheritance; super; object-oriented contacts app

**Up next:** Recursion; Programming fork in the road

**Contacts App:** maintain names and phone numbers

**Two important concepts in Java:**

1. **Strongly Typed at compile time: The list of methods you can call depend on the type(class) of variable.**
2. **Polymorphic at runtime: What code will be executed depends on the actual class of the object at runtime.**

Post OOP … Where do we go from here?

Theory: Recursion

Algorithm efficiency

Algorithm development:

Searching and sorting

A fork in the road:

Programming:

Graphical interfaces

Mobile development

iOS vs. Android

```
public class Person {
   protected String name;
   protected int phoneNumber;

   // constructor takes name and number




   // toString method




   // printDescription


}
```

```
import java.util.Date;
public class Friend extends Person {

   // instance variable - friendsSince of type Date


   // constructor that takes a name and phone number
   // sets the date of becoming friends to now




   // constructor that takes a Person




   // toString method




   // printDescription


}
```

```
import java.util.Date;
public class SoulMate extends Friend {









}
```

8. **Part 2: Model the dictionary (MAP) of callerId objects …**

```
// Implements a MAP collection of CallerId key-value pairs
public class CallerIdMap {

     //  use an array of pairs



     //  add method: takes a key and a value and adds
     //  the pair object to the array
     public void add(int newNumber, String newName) {



     }

     //  get method:  returns a name for a given phone number



     //  get method: returns a key-value pair for a given phone number



}
```

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;
  }
```

1. **Refactoring: the art of restructuring your code**

look for places with repeating code, look for places with lots of code, look to build on previous code

We will refactor our Contacts Apps:

- use 'super' in our toString()

- use 'super' in our constructors

- engineer in spouses and jealousy

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;
  }
```