AP Computer Science Name: _____
Chapter 4 Notes (2)

MORE POLYMORPHISM, CHOREOGRAPHERS, and OBJECT ORIENTED DESIGN

Choreographers:

There is an even more interesting way to carry out some complex tasks if we let one robot coordinate the actions of some others. For this plan to work we need at least two different kinds of robots. One kind of robot will be called a `Choreographer`, because it directs the others, which can be ordinary, standard issue `UrRobot` robots. The trick here is that the `Choreographer` will set up the others and then will guarantee that they mimic the actions of the `Choreographer`.

For this to work, the `Choreographer` needs to know the names of the other robots and have complete control over them, so we will make these names *private* names of the `Choreographer` itself. We have not seen this feature of the robot programming language previously. Rather than declare robots in the main task block, we can define robot names within a new class. Robots declared like this will be available as helpers to robots of the class being declared, but may not be used by other robots or in the main task block. This is because the names of the helper robots will be private to the robot of the new class.

This also brings up the second major feature of objects. Objects can **do** things. For example, robots can move. But objects can also **remember** things. So a `Choreographer` can remember the names of its helpers. The things that robots, and objects in general, can do are represented by its methods, like `turnLeft()`. The things that objects remember are called its **instance variables** or **fields**. When one robot remembers the name of another, it sets up a (one way) association between the robots. A `Choreographer` will know who its helpers are, but the helper may not know the `Choreographer`. Usually human associations are two way, of course, but it is not the same with robots.

Our `Choreographer` will also need to override all of the Robot methods so that, for example, if we tell the `Choreographer` to move, that it can direct the other to move as well. Below we show just the interface of this class so that we can see it all at once. Note that within the class definition we define two robots. These two robots will be helpers for our `Choreographer` robot.

```
public class Choreographer extends UrRobot
{
      private UrRobot lisa = new UrRobot(4, 2, East, 0);
      private UrRobot tony = new UrRobot(6, 2, East, 0);

      public void harvest(){...}
      public void pickARow(){...}
      public void pickTwoRows(){...}
      public void turnRight(){...}
      public void turnAroundLeft(){...}
      public void turnAroundRight(){...}
      public void move() {...}
      public void pickBeeper(){...}
      public void turnLeft(){...}
      public void turnOff(){...}

      public Choreographer(...){...}
}
```

Here is the main block for our program.

```
public static void main(String[] args)
{
     Choreographer karel = new Choreographer(2, 2, East, 0);
     karel.harvest();
     karel.turnOff();
}
```

Instance Variables: Robots or other variables defined within a class. They are associated with each instance (robot) that is created. These are normally private. They are therefore only available within the class.