Controller Example ; Command Pattern

From: Thinking in Java, 3rd ed. Revision 4.0

Eckel: http://www.faqs.org/docs/think_java/TIJ310.htm

Event.java

```
/*
 *       Ch.8: Interfaces & Inner Classes
 */
package controller;

abstract public class Event {
        private long evtTime;
        String status;

        public Event(long eventTime, String stat) {
          evtTime = eventTime;
          status  = stat;
        }

        public boolean ready() {
          return System.currentTimeMillis() >= evtTime;
        }

          public String toString() {
              return status;
            }

        abstract public void action();
        // abstract public String description();
      }
```

Controller.java

```
/**
  */
public class Controller {
        private List<Event> eventList = new ArrayList<Event>();

        public void addEvent(Event c) { eventList.add(c); }

        public void run() {
          while(eventList.size() > 0) {
            for( Event e : eventList ) {
            //System.out.println("Event = " + e);
              if(e.ready()) {
                System.out.println(e);
                e.action();
                eventList.remove(i);
              }
            }
          }
        }
      }
```

GreenHouseController.java

```
public class GreenhouseController {
        public static void main(String[] args) {
          GreenhouseControls gc = new GreenhouseControls();
          // Instead of hard-wiring, you could parse
          // configuration information from a text file here:
//        gc.addEvent(gc.new Bell(900));
```

```
          Event[] eventList = {
          //gc.new ThermostatNight(0),
            gc.new LightOn(200),
            gc.new LightOff(400),
            gc.new WaterOn(600),
            gc.new WaterOff(800),
            gc.new ThermostatDay(1400)
          };

          gc.addEvent(gc.new Restart(2000, eventList));
          if(args.length == 1)
            gc.addEvent(
              gc.new Terminate(Integer.parseInt(args[0])));
          gc.run();
        }
      }
```

GreenHouseControls.java

```
//: c08:GreenhouseControls.java
//
// This produces a specific application of the
// control system, all in a single class. Inner
// classes allow you to encapsulate different
// functionality for each type of event.

/*
 *  Uses the Command Pattern...
 */

//
// From 'Thinking in Java, 3rd ed.' (c) Bruce Eckel 2002
// www.BruceEckel.com. See copyright notice in CopyRight.txt.


package controller;

public class GreenhouseControls
    extends Controller {
 // static Test monitor = new Test();

      // default states
  private boolean light = false;
  private boolean water = false;
  private String thermostat = "Day";

  // ------------------------------------
  class LightOn extends Event {
    public LightOn(long eventTime) {
      super(eventTime, "Light(on)");
    }
    public void action() {
      //  code here to turn on light.
      light = true;
    }
  }
  // ------------------------------------
  class LightOff extends Event {
    public LightOff(long eventTime) {
      super(eventTime, "Light(off)");
    }
```

```
  public void action() {
    // control code here to turn off the light.
    light = false;
  }
}
// ---------------------------------
class WaterOn extends Event {
  public WaterOn(long eventTime) {
    super(eventTime, "Water(on)");
  }
  public void action() {
    // Put hardware control code here
    water = true;
  }
}
// ---------------------------------
class WaterOff extends Event {
  public WaterOff(long eventTime) {
    super(eventTime, "Water(off)");
  }
  public void action() {
    // Put hardware control code here
    water = false;
  }
}
// ---------------------------------
class ThermostatNight extends Event {
  public ThermostatNight(long eventTime) {
    super(eventTime, "Thermostat(night setting)");
  }
  public void action() {
    // Put hardware control code here
    thermostat = "Night";
  }
}
// ---------------------------------
class ThermostatDay extends Event {
  public ThermostatDay(long eventTime) {
    super(eventTime, "Thermostat(day setting)");
  }
  public void action() {
    // Put hardware control code here
    thermostat = "Day";
  }
}
// ---------------------------------
// An example of an action() that inserts a
// new one of itself into the event list:
private int rings;
class Bell extends Event {
  public Bell(long eventTime) {
    super(eventTime, "[Ring Bell]");
  }
  public void action() {
    // Ring every 2 seconds, 'rings' times:
    System.out.println("Bing!");
    if(--rings > 0)
      addEvent(new Bell(System.currentTimeMillis() + 2000));
  }
}
// ---------------------------------  ------------------------
class Restart extends Event {
  public Restart(long eventTime) {
```

```
      super(eventTime, "Restart");
    }
    public void action() {
      long tm = System.currentTimeMillis();
      // configuration information could come from a file (XML...)
      rings = 5;

      addEvent(new ThermostatNight(8000));
      addEvent(new LightOn      (tm +  100));
      addEvent(new LightOff     (tm +  200));
      addEvent(new WaterOn      (tm +  300));
      addEvent(new WaterOff     (tm +  800));
      addEvent(new Bell         (tm +  900));
      addEvent(new ThermostatDay(tm + 1000));

      // Can even add a Restart object!
      addEvent(new Restart      (tm + 2000));
    }
    public String description() {
      return "Restarting system";
    }
  }
  // ----------------------------------
  public static void main(String[] args) {
    GreenhouseControls gc = new GreenhouseControls();

    long tm = System.currentTimeMillis();

    System.out.println("Setup...");
    gc.addEvent(gc.new Restart(tm));
    System.out.println("Starting...");

    gc.run();
  }
} ///:~
```