Example:

Applet which displays a simulated *trackball* in the upper half of its window.

By dragging the trackball using the mouse, you change its state, given by its x-y position relative to the window boundaries, expressed as a percentage from 0 to 100.

The relative x-position is displayed by a *bar chart* at the lower left of the window, and the relative y-pos is displayed by a *pie chart* at the lower right.

```java
public class BarChart extends Applet implements Observer {

    public BarChart(Subject subject) {
        subject.addObserver(this);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.black);
        g.drawString("Relative X position:", 0, 10);
        g.setColor(Color.red);
        g.fillRect(0, 15, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(15, 0, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(0, 15, 15, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(15, 0, 15, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(0, 15, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(0, 15, theSize, 10);
        g.setColor(Color.red);
        g.fillRect(0, 15, theSize, 10);
    }

    public void update(Observable s, Object arg) {
        Point p = (Point) arg;
        theSize = p.x;
        repaint();
    }

    private int theSize = 10;
}
```
public class PieChart extends Applet implements Observer {

    public PieChart(Subject subject) {
        subject.addObserver(this);
    }

    public void paint(Graphics g) {
        g.setColor(Color.black);
        g.drawString("Relative Y position:", 0, 10);
        g.setColor(Color.blue);
        g.fillArc(0, 15, 50, 50, 0, 360*theSize/100);
    }

    public void update(Observable s, Object arg) {
        Point p = (Point) arg;
        theSize = p.y;
        repaint();
    }

    private int theSize = 10;
}

public class Trackball extends Canvas implements Subject, MouseListener, MouseMotionListener {

    public Trackball() {
        setBackground(Color.lightGray);
        addMouseMotionListener(this);
    }
}
public Point getState() {
    int width = getSize().width;
    int height = getSize().height;
    int relx = (int) (100.0 * (double) theX / ((double) width - DIAMETER) + 0.5);
    int rely = (int) (100.0 * (double) theY / ((double) height - DIAMETER) + 0.5);
    return new Point(relx, rely);
}

public void paint(Graphics g) {
    g.setColor(Color.black);
    g.drawRect(0, 0, getSize().width -1, getSize().height -1);
    g.drawRect(1, 1, getSize().width -3, getSize().height -3);
    g.drawRect(2, 2, getSize().width -5, getSize().height -5);
    g.setColor(Color.darkGray);
    g.fillOval(theX, theY, DIAMETER, DIAMETER);
}

public void addObserver(Observer o) {
    theSubject.addObserver(o);
}

public void mouseReleased(MouseEvent event) {
    isDragging = false;
}

public void mouseClicked(MouseEvent event) {
}

public void mouseEntered(MouseEvent event) {
}

public void mouseExited(MouseEvent event) {
}

public void mousePressed(MouseEvent event) {
}

public void mouseMoved(MouseEvent event) {
}
public void mouseDragged (MouseEvent event) {
    int x = theX + DIAMETER/2;
    int y = theY + DIAMETER/2;
    int px = event.getX();
    int py = event.getY();
    int d = (int) Math.sqrt((x -px)*(x -px) + (y -py)*(y -py));
    if (isDragging || d < DIAMETER/2) {
        isDragging = true;
        theX = px - DIAMETER/2;
        theY = py - DIAMETER/2;
        int width = getSize().width;
        int height = getSize().height;
        if (theX < 0) theX = 0;
        if (theY < 0) theY = 0;
        if (theX > width - DIAMETER)
            theX = width - DIAMETER;
        if (theY > height - DIAMETER)
            theY = height - DIAMETER;

        theSubject.setChanged();
        theSubject.notifyObservers(getState());
        repaint( );
    }
}

// The TrackBall extends Canvas but also needs the state and behavior of an Observable. It creates a captive Observable object which will do its bidding, and that implements the Subject interface that can handle addObserver().

private OwnedSubject theSubject = new OwnedSubject();

private static final int DIAMETER = 20;
private int theX = DIAMETER;
private int theY = DIAMETER;
private boolean isDragging = false;
import java.applet, awt, awt.event.*;

public class **MVC extends Applet** {

    public void init() {
        setLayout(new GridLayout(2, 1));
        theBarChart = new BarChart(theTrack);
        thePieChart = new PieChart(theTrack);
        Panel south = new Panel();
        south.setLayout(new GridLayout(1, 2));
        south.add(theBarChart);
        south.add(thePieChart);
        add(theTrack);
        add(south);
    }

    private Trackball theTrack = new Trackball();
    private Applet theBarChart;
    private Applet thePieChart;
}

// OwnedSubject combines the properties of an Observable and a Subject.
import java.util.Observable;

public class **OwnedSubject extends Observable implements Subject** {

    public void setChanged() {
        super.setChanged();
    }
}

// The promise made by the Subject interface is that it can handle the addObserver method
import java.util.Observer;

public **interface Subject** {
    public void addObserver(Observer o);
}
Example:

- When the applet Test is executed, a GUI with a text field and a button is displayed.

- The applet’s init() instantiates the Clock c, adds the applet into the Clock’s list of Observers (addObserver(this)), defines a new thread for the Clock (clockT), and getsthe date & time when the Clock was created (c.getDate()).

- When the user clicks the button, the Clock’s method c.setAlarm() is called which sets the variable alarmSet to true.

- When the Clock awakens, it gets a new time & date (d.newDate()) and calls setChanged() to indicate that the state of the Clock has changed.

- Method notifyObservers(d.toString()) automatically invoke the update method of all its observers.

- Method update() displays the new date and time in the text field and a message in the status bar area.

```java
public class Test extends Applet implements Observer {

    clock c;
    Label aLabel;
    TextField aText;
    Button alarmOn;
    Thread clockT;

    public void init() {
        c = new Clock;
        c.addObserver(this);
        clockT = new Thread(c);

        aLabel = new Label("Current Date & Time: ");
        aText = new TextField(c.getDate(), 25);
        aText.setEditable(false);
        alarmOn = new Button("Alarm on");
        alarmOn.addActionListener(this);
        add(aLabel); add(aText); add(alarmOn);
        clockT.start();
    }
```

public void actionPerformed(ActionEvent e) {
    showStatus("Alarm set at " + c.setAlarm());
}

public void update(Observable ob, Object arg) {
    aText.setText(arg.toString());
    showStatus("Alarm sounded, see new time");
}

class Clock extends Observable implements Runnable {
    Date d;
    boolean alarmSet;

    public Clock() {d = new Date();}

    public String getDate() { return d.toString(); }

    public String setAlarm() {
        alarmSet = true;
        d = new Date();
        return d.toString();
    }

    public void run() {
        while (true) {
            if (alarmSet) {
                try {
                    Thread.sleep(10000); }
                catch (InterruptedException e) {
                    System.err.println("Sleep interrupted"); }
                alarmSet = false;
                d = new Date(); // get new time
                setChanged();
                notifyObservers(d.toString());
            }
        }
    }
}