Java - Dates and Threads

Thread example

Java uses threads to allow 1 event to break into another. If an action is to occur a number of times with a loop or to occur until some condition is satisfied then that action will take over the computer. Any other planned events will have to wait until the loop is finished. A thread can break into such an action with another section of code.

In this example a circle is redrawn to give the effect of a ball bouncing in a box. Without the use of a thread there would be no way to interrupt the bouncing. Any controls placed on the applet would be disabled. The following code should run as typed.

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Bounce extends Applet implements ActionListener
{ //this class is used to create new Ball objects and display them
        private Button start;
        private Ball ball
        public void init()
        {
                start = new Button("Start");
                add(start);
                start.addActionListener(this);
        }

        public void actionPerformed(ActionEvent event)
        {
                Graphics g = getGraphics();
                if(event.getSource()==start)
                {
                        ball = new Ball(g);
                        ball.start(); //not run see the Ball Class
                }

        }
}

class Ball extends Thread  //extend Thread to make use of Thread methods
{
        private Graphics g;
        private boolean moving;
        private int x=7, xChange =7;
        private int y=0, yChange =2;
        private int diameter =10;
        private int rectLeftX =0, rectRightX=100;
```

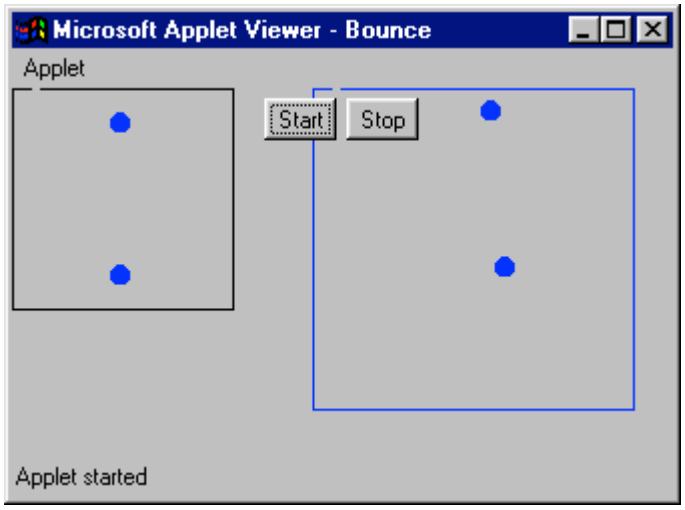
```
        private int rectTopY=0, rectBottomY=100;

        public Ball(Graphics graphics)
        {  //constructor
                g=graphics;
                moving=true;
        }
        public void run()  //the Thread equivalent of init()
        {       //draw rectangle so ball edge just touches
                g.drawRect(rectLeftX,rectTopY,rectRightX-
rectLeftX+diameter,rectBottomY-rectTopY+diameter); //all on 1 line
                while(moving)
                {
                        //paint over last ball
                        g.setColor(Color.lightGray);
                        g.fillOval(x,y,diameter,diameter);
                        //code for hitting edges
                        if(x+xChange<=rectLeftX)
                                xChange = -xChange;
                        if(x+xChange>=rectRightX)
                                xChange = -xChange;
                        if(y+yChange<=rectTopY)
                                yChange = -yChange;
                        if(y+yChange>=rectBottomY)
                                yChange = -yChange;
                        //move ball
                        x=x+xChange;
                        y=y+yChange;
                        //repaint ball
                        g.setColor(Color.blue);
                        g.fillOval(x,y,diameter,diameter);
                        try
                        {//allow other actions – this is the gap for other code to cut in
                                Thread.sleep(50);
                        }
                        catch(InterruptedException e)
                        {//have to catch this exception
                                g.drawString("Sleep exception",100,100);
                        }
                }
        }

}
```

The above code will keep creating new blue balls as long as the Button start is clicked. The key method is run(), this goes on and on while moving is true but allows other methods to cut in during the Thread.Sleep() call. As run() is a method that goes on and on it is not possible to pass parameters from outside while the Thread runs. The Thread could be killed and another object created from that Class with new parameters.

In the Ball example the oval is filled in blue. That colour cannot be changed while the Thread runs. By adjusting the constructor to accept a Color value a new Thread object could be created with a different fill.

Modifications

- Add a Button to stop the Balls moving. It is only necessary to stop 1 Ball if there are multiple Balls in the rectangle.
- Create a 2nd Ball that bounces within another rectangle when start is pressed. This could be achieved by extending the original Ball Class or overloading the constructor of the Ball Class.



Time in Java

Threads will be used to display the system time as a set of numbers and moving hands on a clock. Before that some explanation of how to get the system date is required.

The easy way is to import java.util.Date; and make use of a Date object. There are a number of handy Date methods that will supply the hours and so forth as useful integers.

```
import java.util.Date; //for Date
private Date now;
now = new Date();
int min=now.getMinutes();
```

Unfortunately these methods are depreciated because the Java authors decided that they could not cover all the World's possible ways of displaying dates. There are 2 ways around the problem.

The Calendar Class appears to have attributes similar to Date but on closer observation most of the contents of this Class are constants used to set dates from user input rather than to get the system date.

```
private Calendar rightNow;
rightNow = Calendar.getInstance(); //create Calendar object
String time = String.valueOf(rightNow.getTime())
```

The above code will get the current system time and put it into a String for displaying on an Applet or in a TextField.
Note that the following enticing code lines do not work:

```
String hours = String.valueOf(rightNow.HOUR_OF_DAY);
String  mins = String.valueOf(rightNow.MINUTE);
String  secs = String.valueOf(rightNow.SECOND);
```

Another approach uses the SimpleDateFormat Class to get the time from a Date object.

```
import java.text.*;  //for SimpleDateFormat
SimpleDateFormat formatter = new SimpleDateFormat ("EEE MMM dd hh:mm:ss
yyyy z");  //all on one line
// the full format with day, month, date, hour, minute, second, and time zone.
//the number of characters affects how items are displayed such as Mon or Monday
String time = formatter.format(now); //display the Date object now in the above
format
```

Appropriate formats for minutes and seconds can be assigned to Strings such as:

```
SimpleDateFormat formatter = new SimpleDateFormat("s",Locale.getDefault());
//note s is not a variable like secs
String secs = formatter.format(now);
```

The above code may cause a problem with BST times as the additional hour may not be factored in. This does not occur with NT at College but may do under Windows 95+. One solution is to set up a method to scan through the String created by SimpleDateFormat.formatter() and add an hour if the '+' character is found.

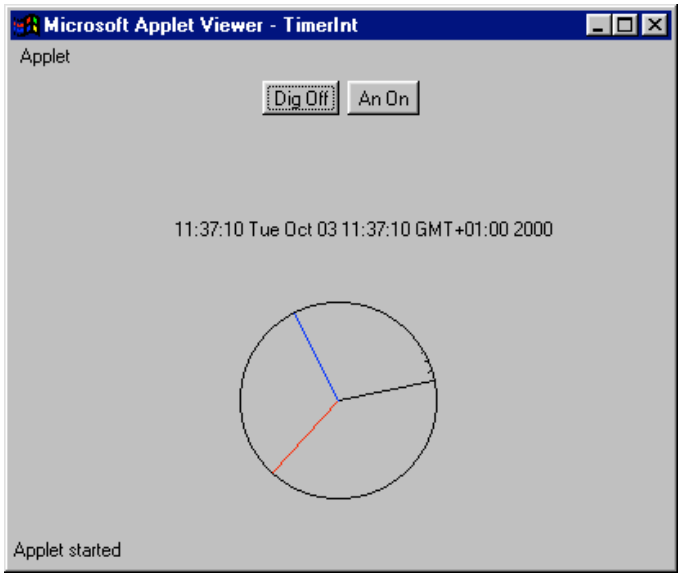
Set up a pair of Classes.

- One to host the business end and create instances of other Classes, this will need Buttons to start and stop the time Thread.
- Another that extends Thread causing the time displayed to be updated every second - Thread.sleep(1000); the time should be displayed as the complete time and as separate Strings for hours, minutes and seconds.

When the time is correctly displayed extend the time Thread to display the time as an analogue clock. The hours, minutes and seconds Strings will come in handy here. Adjust the Buttons so either clock can be started or stopped independently.

The analogue clock will need a centre to draw the hands from and a radius for each hand. The angle to move each hand can be found from the time and the position using Sin and Cos, unfortunately Java angles are measured in radians.

The circle that forms the clock can be created with drawOval for the outside and fillOval for a filled centre. The fill will have to be a little smaller that the rim or it will overdraw it.




```
g.drawOval(topX,topY,radius*2,radius*2);//outside of clock circle
g.setColor(Color.lightGray);//inside of clock
g.fillOval(topX+2,topY+2,(radius*2)-2,(radius*2)-2);//slightly smaller
```

The initial string values for hours, minutes and seconds will need to be converted to double or float values for plotting. When drawing the hands there are 3 points to take care of:

- There are 60 minutes or seconds in an hour but 360^{o} in a full circle.
  dAngle = dAngle*6; *//6*60=360 - for secs and mins*
- Converting from degrees to radians:
  dAngle = dAngle * 3.14/180;
- Angles are drawn from the 3.00 position, 12.00 would be a better start line for a clock:
  dAngle= dAngle - (3.14/2);

Plotting the lines can be handled by Java.Math;

```
int yPos = (int)(Math.sin(dAngle)*60);
int yPos = (int)(Math.sin(dAngle)*60);
```

The hour hand will be more complex as there are 12 hours to divide in to 360^{o} and the effect of the minutes on the hour hand will have to be taken care of.