Animation

In this part of the Java 2D games tutorial, we will work with animation.

Animation is a rapid display of sequence of images which creates an illusion of movement. We will animate a star on our Board. We will implement the movement in three basic ways. We will use a Swing timer, a standard utility timer and a thread.

Swing timer

In the first example we will use a Swing timer to create animation. This is the easiest but also the least effective way of animating objects in Java games.

Star.java

```
package star;

import javax.swing.JFrame;

public class Star extends JFrame {

    public Star() {

        add(new Board());

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(280, 240);
        setLocationRelativeTo(null);
        setTitle("Star");
        setResizable(false);
        setVisible(true);

    }

    public static void main(String[] args) {
        new Star();
    }
}
```

This is the main class for the code example.

Board.java

```
package star;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Board extends JPanel implements ActionListener {
```

```
    Image star;
    Timer timer;
    int x, y;

    public Board() {
        setBackground(Color.BLACK);

        ImageIcon ii =
            new ImageIcon(this.getClass().getResource("star.png"));
        star = ii.getImage();

        setDoubleBuffered(true);

        x = y = 10;
        timer = new Timer(25, this);
        timer.start();
    }


    public void paint(Graphics g) {
        super.paint(g);

        Graphics2D g2d = (Graphics2D)g;
        g2d.drawImage(star, x, y, this);
        Toolkit.getDefaultToolkit().sync();
        g.dispose();
    }


    public void actionPerformed(ActionEvent e) {

        x += 1;
        y += 1;

        if (y > 240) {
            y = -45;
            x = -45;
        }
        repaint();
    }
}
```

```
setDoubleBuffered(true);
```

Our JPanel component will use a buffer to paint. This means that all drawing will be done in memory first. Later the off-screen buffer will be copied to the screen. In this example, I didn't notice any differences.

```
timer = new Timer(25, this);
timer.start();
```

Here we create a Swing `Timer` class. We start the timer. Every 25 ms the timer will call the `actionPerformed()` method. In order to use the `actionPerformed()` method, we must implement the `ActionListener` interface.

```
g2d.drawImage(star, x, y, this);
```

In the `paint()` method, we draw the star.

```
Toolkit.getDefaultToolkit().sync();
```

We must synchronize the painting on Linux systems. Otherwise, the animation would not be smooth.

```
public void actionPerformed(ActionEvent e) {
    x += 1;
    y += 1;

    if (y > 240) {
        y = -45;
        x = -45;
    }
    repaint();
}
```

In the `actionPerformed()` method we increase the x, y values. Then we call the `repaint()` method. This way we regularly repaint the `Board` thus making the animation.

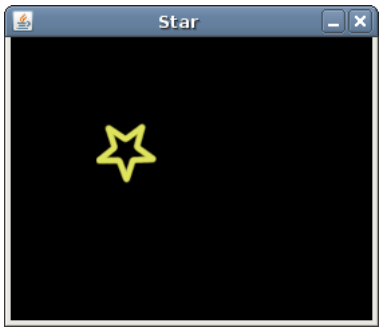

Figure: Star

Utility timer

This is very similar to the previous way. We use the java.util.Timer instead of the javax.Swing.Timer. For Java Swing games this way should be more accurate.

Star.java

```
package star2;

import javax.swing.JFrame;

public class Star extends JFrame {

    public Star() {

        add(new Board());

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(280, 240);
        setLocationRelativeTo(null);
        setTitle("Star");
        setResizable(false);
        setVisible(true);

    }

    public static void main(String[] args) {
        new Star();
    }
}
```

The main class.

Board.java

```
package star2;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Toolkit;

import java.util.Timer;
import java.util.TimerTask;

import javax.swing.ImageIcon;
import javax.swing.JPanel;


public class Board extends JPanel  {

    Image star;
    Timer timer;
    int x, y;

    public Board() {
        setBackground(Color.BLACK);

        ImageIcon ii = new ImageIcon(this.getClass().getResource("star.png"));
        star = ii.getImage();

        setDoubleBuffered(true);

        x = y = 10;
        timer = new Timer();
        timer.scheduleAtFixedRate(new ScheduleTask(), 100, 10);
    }
```

```
    public void paint(Graphics g) {
        super.paint(g);

        Graphics2D g2d = (Graphics2D)g;
        g2d.drawImage(star, x, y, this);
        Toolkit.getDefaultToolkit().sync();
        g.dispose();
    }


    class ScheduleTask extends TimerTask {

        public void run() {
            x += 1;
            y += 1;

            if (y > 240) {
                y = -45;
                x = -45;
            }
            repaint();
        }
    }
}
```

In this example, the timer will regularly call the `run()` method of the `ScheduleTask` class.

```
timer = new Timer();
timer.scheduleAtFixedRate(new ScheduleTask(), 100, 10);
```

Here we create a timer. And schedule a task at 10 ms interval. There is 100 ms initial delay.

```
public void run() {
...
}
```

Each 10 ms the timer will call this `run()` method.

Thread

Animating objects using a thread is the most effective way of animation.

Star.java

```
package star3;

import javax.swing.JFrame;

public class Star extends JFrame {

    public Star() {

        add(new Board());

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(280, 240);
        setLocationRelativeTo(null);
        setTitle("Star");
        setResizable(false);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Star();
    }
}
```

This is the main class.

Board.java

```
package star2;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Toolkit;

import javax.swing.ImageIcon;
```

```
import javax.swing.JPanel;


public class Board extends JPanel implements Runnable {

    private Image star;
    private Thread animator;
    private int x, y;

    private final int DELAY = 50;


    public Board() {
        setBackground(Color.BLACK);
        setDoubleBuffered(true);

        ImageIcon ii = new ImageIcon(this.getClass().getResource("star.png"));
        star = ii.getImage();

        x = y = 10;
    }

    public void addNotify() {
        super.addNotify();
        animator = new Thread(this);
        animator.start();
    }

    public void paint(Graphics g) {
        super.paint(g);

        Graphics2D g2d = (Graphics2D)g;
        g2d.drawImage(star, x, y, this);
        Toolkit.getDefaultToolkit().sync();
        g.dispose();
    }


    public void cycle() {

        x += 1;
        y += 1;

        if (y > 240) {
            y = -45;
            x = -45;
        }
    }

    public void run() {

        long beforeTime, timeDiff, sleep;

        beforeTime = System.currentTimeMillis();

        while (true) {

            cycle();
            repaint();

            timeDiff = System.currentTimeMillis() - beforeTime;
            sleep = DELAY - timeDiff;

            if (sleep < 0)
                sleep = 2;
            try {
                Thread.sleep(sleep);
            } catch (InterruptedException e) {
                System.out.println("interrupted");
            }

            beforeTime = System.currentTimeMillis();
        }
    }
}
```

In the previous examples, we executed a task at specific intervals. In this example, the animation will take place inside a thread. The `run()` method is called only once. That's why we have a while loop in the method. From this method, we call the `cycle()` and the `repaint()` methods.

```
public void addNotify() {
    super.addNotify();
    animator = new Thread(this);
    animator.start();
}
```

The `addNotify()` method is called after our `JPanel` has been added to the `JFrame` component. This method is often used for various initialization tasks.

We want our game run smoothly. At constant speed. Therefore we compute the system time.

```
timeDiff = System.currentTimeMillis() - beforeTime;
sleep = DELAY - timeDiff;
```

The `cycle()` and the `repaint()` methods might take different time at various while cycles. We calculate the time both methods run and subtract it from the DELAY constant. This way we want to ensure that each while cycle runs a constant time. In our case, 50ms each cycle.

This part of the Java 2D games tutorial covered animation.

Home ǂ Contents ǂ Top of Page