

AnimationWorld

Directing A Play Performed by Sprites

Tuesday, December 4, 2007

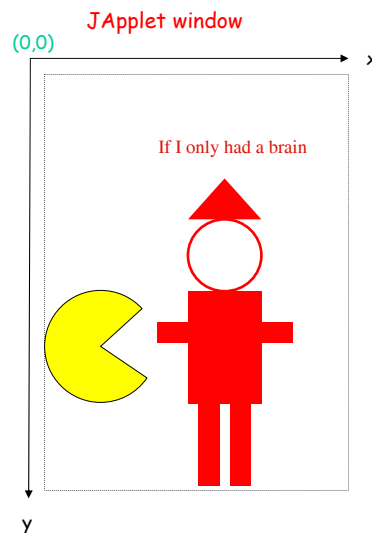


CS111 Computer Programming

Department of Computer Science
Wellesley College

Review: Applet Graphics

```
import java.awt.*;
import javax.swing.*; // for JApplet
public class PacmanEatsTinman extends JApplet {
    public void paint (Graphics g)
    // Draw Tinman
    g.setColor(Color.red);
    g.drawString("If I only had a brain", 80, 60);
    g.drawOval(100,100,50,50); // Head
    Polygon hat = new Polygon();
    hat.addPoint(100,100);
    hat.addPoint(125,70);
    hat.addPoint(150,100);
    g.fillPolygon(hat);
    g.fillRect(100,150,50,80); // Body
    g.fillRect(105,230,15,50); // Leg
    g.fillRect(130,230,15,50); // Leg
    g.fillRect(75,175,100,20); // Arms
    // Draw Pacman
    g.setColor(Color.yellow);
    g.fillArc(0,150,80,80,45,270); // Body
}
```



Animation 23-2

Review: Graphics contract

```
public void setColor(Color c)
    Sets this graphics context's current color to the
    specified color.
public void drawLine(int x1, int y1, int x2, int y2)
    Draws a line, using the current color, between the points
    (x1, y1) and (x2, y2) in this graphics context's
    coordinate system.
public void drawRect(int x, int y, int width, int height)
    Draws the outline of the specified rectangle.
public void fillRect(int x, int y, int width, int height)
    Fills the specified rectangle.
public void drawOval(int x, int y, int width, int height)
    Draws the outline of an oval.
public void fillOval(int x, int y, int width, int height)
    Fills the specified oval.
public void drawString(String str, int x, int y)
    Draws the text given by the specified string, using this
    graphics context's current font and color.
...
```

Animation 23-3

CS111 Museum

Do you want your CS111 efforts to be immortalized?

Create a museum quality project!

<http://cs.wellesley.edu/~cs111/museum/>

Animation 23-4

A New Microworld: AnimationWorld*

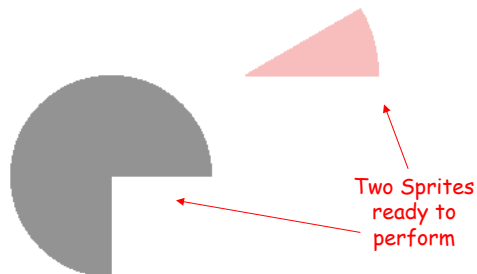
Animation World uses Java Graphics to create animations.
There are 2 main parts to AnimationWorld:

1. *Sprites* are the actresses in an animation. We create a cast of Sprites to act in our play. Each sprite knows how to perform in a particular way.
2. *Animations* are the plays in which the sprites act. An animation shows a sequence of numbered frames. In each frame, each sprite in the animation is asked to draw itself. Between frames, each sprite is asked to update its state. Displaying the sequence of frames shows a "movie" of the sprites' performances.

*Created by Elaine Yang (CS111 lab instructor, 1999-2000)

Animation 23-5

Our First Animation: Rotators



Animation 23-6

Creating new Sprites

```
public class Rotator extends Sprite {
    // Instance variables describing the state of the sprite
    ...

    // Constructors for creating sprite instances
    ...

    // Instance methods *must* include the following 3 methods:

    // Describes how to draw this sprite in a particular frame
    public void drawState(Graphics g) {
        ...}

    // Describes how to update the state of the Sprite between frames
    public void updateState() {
        ...}

    // Describes how to reset the sprite back to its initial state
    public void resetState() {
        ...}
}
```

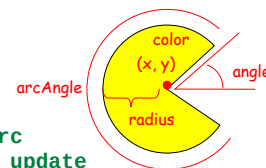
Animation 23-7

The Rotator Class

```
public class Rotator extends Sprite {
    // Instance variables describing the state of a rotator.
    // All of these remain constant during animation except angle.
    private int x; // x-coord of arc center
    private int y; // y-coord of arc center
    private int radius; // radius of arc
    private int arcAngle; // angle of arc
    private Color color; // color of arc
    private int angle; // current startAngle of arc
    private int deltaAngle; // change in angle on update

    // Constructor method to create a rotator
    public Rotator(int new_x, int new_y, int new_radius,
                  int new_arcAngle, int new_deltaAngle,
                  Color new_color) {

        x = new_x;
        y = new_y;
        radius = new_radius;
        arcAngle = new_arcAngle;
        deltaAngle = new_deltaAngle;
        color = new_color;
        angle = 0; // initial startAngle of arc
        // ... Instance methods go here ...
    }
}
```



Animation 23-8

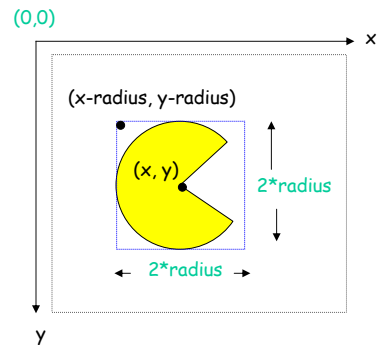
Rotator Instance Methods

```
public class Rotator extends Sprite {
    // Instance variables and constructors ...

    public void drawState (Graphics g) {
        g.setColor(color);
        g.fillArc(x-radius,y-radius,2*radius,2*radius,
            angle,arcAngle);
    }

    public void updateState () {
        angle = angle + deltaAngle;
    }

    public void resetState () {
        angle = 0;
    }
}
```

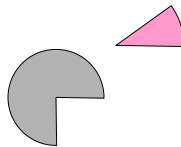


Animation 23-9

Casting

```
public class RotatorAnimation extends Animation {

    // Constructor method
    public RotatorAnimation () {
        this.addSprite(new Rotator(400,200,100,30,5,Color.pink));
        this.addSprite(new Rotator(300,275,75,270,-2,Color.gray));
        this.setNumberFrames(Animation.NO_MAX_FRAMES);
    }
}
```

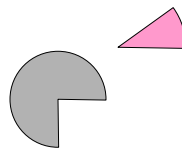


```
// Recall the Rotator constructor contract:
public Rotator(int new_x, int new_y, int new_radius,
    int new_arcAngle, int new_deltaAngle,
    Color new_color);
```

Animation 23-10

Casting

```
public class RotatorAnimation extends Animation {  
    // Constructor method  
    public RotatorAnimation () {  
        Sprite r1 = new Rotator(400,200,100,30,5,Color.pink));  
        Sprite r2 = new Rotator(300,275,75,270,-2,Color.gray));  
        this.addSprite(r2);  
        this.addSprite(r1);  
        // What happens if sprites are added in opposite order?  
        this.setNumberFrames(Animation.NO_MAX_FRAMES);  
    }  
}
```



```
// Recall the Rotator constructor contract:  
public Rotator(int new_x, int new_y, int new_radius,  
               int new_arcAngle, int new_deltaAngle,  
               Color new_color);
```

Animation 23-11

Rotator Updates

Frame	r1.angle	r2.angle
1	0	0
2	5	-2
3	10	-4
4	15	-6
5	20	-8

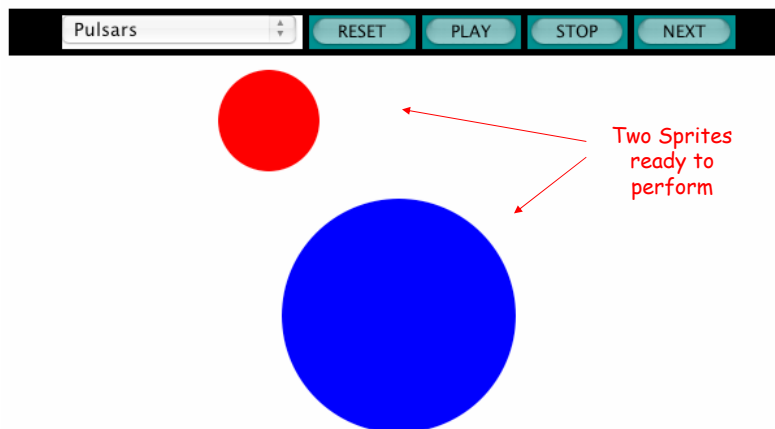
Animation 23-12

Other Animation Controls

```
public class RotatorAnimation2 extends Animation {  
  
    // Constructor method  
    public RotatorAnimation2 () {  
        Sprite r1 = new Rotator(400,200,100,30,5,Color.pink));  
        Sprite r2 = new Rotator(300,275,75,270,-2,Color.gray));  
        this.addSprite(r1);  
        this.addSprite(r2);  
        this.setNumberFrames(100); // default is 60;  
                                   // Animation.NO_MAX_FRAMES is infinite  
        this.setNumberRepeats(2); // default is 0;  
                                   // Animation.FOREVER loops forever  
        this.setInactive(r2,40);  
        this.setActive(r2,80);  
        this.setHidden(r1,20);  
        this.setVisible(r1,60);  
        this.setFps(6); // default is 12  
    }  
}
```

Animation 23-13

Pulsars

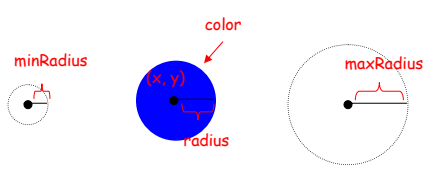


Animation 23-14

The Pulsar Class

```
public class Pulsar extends Sprite {
    // instance variables
    private int x;
    private int y;
    private int radius;
    private int minRadius;
    private int maxRadius;
    private int changeRadius;
    private Color color;

    // constructors
    public Pulsar(int new_x, int new_y, int new_minRadius,
                 int new_maxRadius, int new_changeRadius,
                 Color new_color) {
        x = new_x;
        y = new_y;
        radius = new_minRadius;
        minRadius = new_minRadius;
        maxRadius = new_maxRadius;
        changeRadius = new_changeRadius;
        color = new_color;
    }
}
```

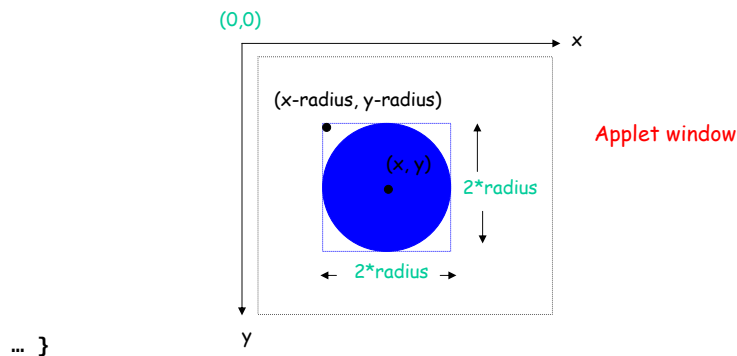


Animation 23-15

Pulsar instance methods

```
public class Pulsar extends Sprite {
    // instance variables and constructors ...

    public void drawState(Graphics g) {
        g.setColor(color);
        g.fillOval(x-radius, y-radius, 2*radius, 2*radius);
    }
}
```



Animation 23-16

Updating the Pulsar

```
public class Pulsar extends Sprite {  
    // instance variables and constructors ...  
  
    public void drawState(Graphics g) {  
        g.setColor(color);  
        g.fillOval(x-radius,y-radius,2*radius,2*radius);  
    }  
  
    public void updateState() {  
        radius = radius + changeRadius;  
        if (radius > maxRadius) {  
            radius = maxRadius;  
            changeRadius = -changeRadius;  
        } else if (radius < minRadius) {  
            radius = minRadius;  
            changeRadius = -changeRadius;  
        }  
    }  
    ... }  
}
```

Animation 23-17

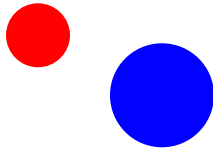
Resetting state

```
public class Pulsar extends Sprite {  
    // instance variables and constructors ...  
  
    public void drawState(Graphics g) {  
        g.setColor(color);  
        g.fillOval(x-radius,y-radius,2*radius,2*radius);  
    }  
  
    public void updateState() {  
        radius = radius + changeRadius;  
        if (radius > maxRadius) {  
            radius = maxRadius;  
            changeRadius = -changeRadius;  
        } else if (radius < minRadius) {  
            radius = minRadius;  
            changeRadius = -changeRadius;  
        }  
    }  
  
    public void resetState() {  
        this.radius = minRadius;  
    }  
}
```

Animation 23-18

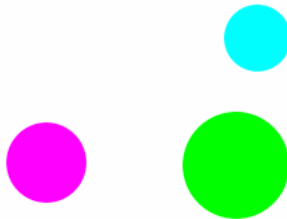
Casting

```
public class PulsarAnimation extends Animation {  
  
    // constructor  
    public PulsarAnimation() {  
        this.addSprite(new Pulsar(200,50,20,40,1,Color.red));  
        this.addSprite(new Pulsar(300,200,0,200,5,Color.blue));  
        this.setFps(12);  
    }  
}  
  
public Pulsar(int new_x, int new_y, int new_minRadius,  
              int new_maxRadius, int new_changeRadius,  
              Color new_color)  
{...}
```



Animation 23-19

Bouncing balls



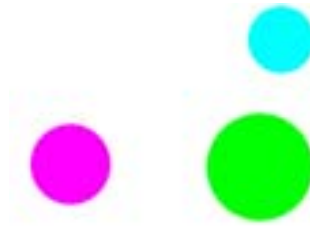
Animation 23-20

Mobile sprites

```
public class BouncingBall extends Sprite {

    // Instance variables
    private int radius;
    private int initX;
    private int initY;
    private int x;
    private int y;
    private int changeX;
    private int changeY;
    private Color color;
    // Constructor method
    public BouncingBall(int new_initX, int new_initY,
                       int new_radius, int new_changeX,
                       int new_changeY, Color new_color) {

        initX = new_initX;
        initY = new_initY;
        radius = new_radius;
        changeX = new_changeX;
        changeY = new_changeY;
        color = new_color;
        resetState();
    }
}
```



Animation 23-21

Rendering & resetting the Sprite

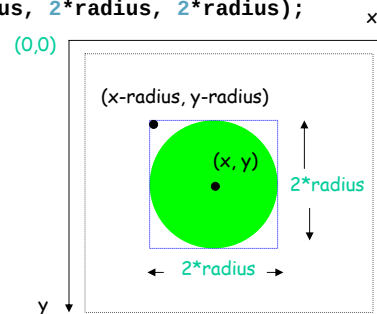
```
public class BouncingBall extends Sprite {

    // Instance variables and constructor method
    ...

    // Instance methods

    public void drawState(Graphics g) {
        g.setColor(color);
        g.fillOval(x-radius, y-radius, 2*radius, 2*radius);
    }

    public void resetState() {
        this.x = this.initX;
        this.y = this.initY;
    }
}
```

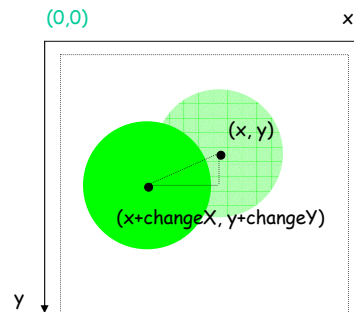


Animation 23-22

The ball moves by (changeX, changeY)

```
public class BouncingBall extends Sprite {

    ...
    public void updateState() {
        x = x + changeX;
        y = y + changeY;
        if (x < radius) {
            x = radius;
            changeX = -changeX;
        }
        if (x + radius >= width) {
            x = width - radius;
            changeX = -changeX;
        }
        if (y < radius) {
            y = radius;
            changeY = -changeY;
        }
        if (y + radius >= height) {
            y = height - radius;
            changeY = -changeY;
        }
    }
}
```

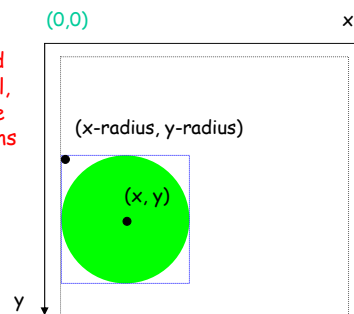


Animation 23-23

Ouch!

```
public class BouncingBall extends Sprite {

    ...
    public void updateState() {
        x = x + changeX;
        y = y + changeY;
        if (x < radius) {
            x = radius;
            changeX = -changeX;
        }
        if (x + radius >= width) {
            x = width - radius;
            changeX = -changeX;
        }
        if (y < radius) {
            y = radius;
            changeY = -changeY;
        }
        if (y + radius >= height) {
            y = height - radius;
            changeY = -changeY;
        }
    }
}
```

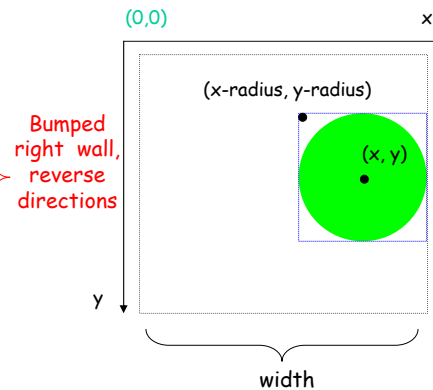


Animation 23-24

Cut that out

```
public class BouncingBall extends Sprite {
```

```
    ...
    public void updateState() {
        x = x + changeX;
        y = y + changeY;
        if (x < radius) {
            x = radius;
            changeX = -changeX;
        }
        if (x + radius >= width) {
            x = width - radius;
            changeX = -changeX;
        }
        if (y < radius) {
            y = radius;
            changeY = -changeY;
        }
        if (y + radius >= height) {
            y = height - radius;
            changeY = -changeY;
        }
    }
}
```

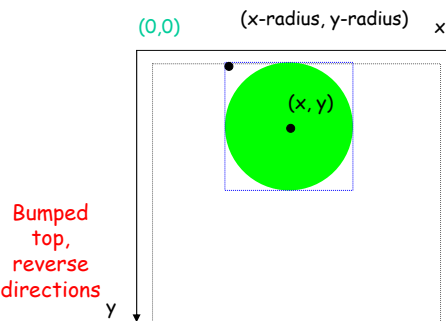


Animation 23-25

Hitting the ceiling

```
public class BouncingBall extends Sprite {
```

```
    ...
    public void updateState() {
        x = x + changeX;
        y = y + changeY;
        if (x < radius) {
            x = radius;
            changeX = -changeX;
        }
        if (x + radius >= width) {
            x = width - radius;
            changeX = -changeX;
        }
        if (y < radius) {
            y = radius;
            changeY = -changeY;
        }
        if (y + radius >= height) {
            y = height - radius;
            changeY = -changeY;
        }
    }
}
```



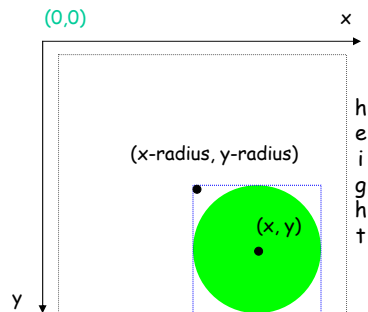
Animation 23-26

Bottoming out

```
public class BouncingBall extends Sprite {
```

```
    ...
    public void updateState() {
        x = x + changeX;
        y = y + changeY;
        if (x < radius) {
            x = radius;
            changeX = -changeX;
        }
        if (x + radius >= width) {
            x = width - radius;
            changeX = -changeX;
        }
        if (y < radius) {
            y = radius;
            changeY = -changeY;
        }
        if (y + radius >= height) {
            y = height - radius;
            changeY = -changeY;
        }
    }
}
```

Bumped bottom, reverse directions



Animation 23-27

The show must go on

```
public class BouncingBallAnimation extends Animation {
```

```
    // Constructor method for BouncingBallAnimation
```

```
    public BouncingBallAnimation() {
        this.addSprite(new BouncingBall(50,50,30,2,2,Color.magenta));
        this.addSprite(new BouncingBall(150,150,25,3,-3,Color.cyan));
        this.addSprite(new BouncingBall(250,250,40,1,4,Color.green));
        this.setFps(12);
    }
}
```



```
    // Constructor method for BouncingBall
```

```
    public BouncingBall(int new_initX, int new_initY,
        int new_radius, int new_changeX,
        int new_changeY, Color new_color)
    {...}
```

Animation 23-28