AnimationWorld

Directing A Play Performed by Sprites

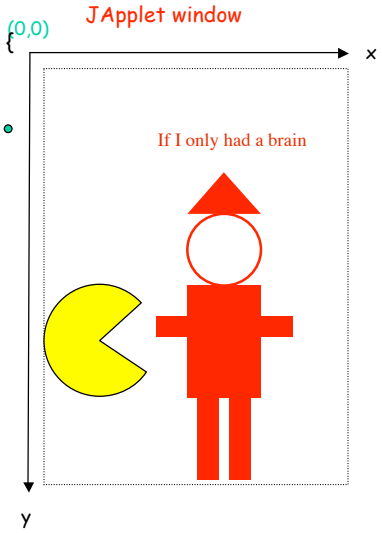
Tuesday, December 5, 2006

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
  }
}
```



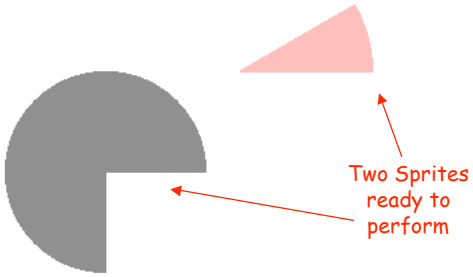
Review: Graphics contract (partial)

```
public void setColor(Color c)
public void drawString(String str, int x, int y)
public void drawLine(int x1, int y1, int x2, int y2)
public void drawRect(int x, int y, int width, int height)
public void fillRect(int x, int y, int width, int height)
public void drawOval(int x, int y, int width, int height)
public void fillOval(int x, int y, int width, int height)
public void drawPolygon(Polygon p);
public void fillPolygon (Polygon p);
public void drawArc (int x, int y, int width, int height,
                     int startAngle, int arcAngle);
public void fillArc (int x, int y, int width, int height,
                     int startAngle, int arcAngle);
```


A New Microworld: AnimationWorld

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

Our First Animation: Rotators

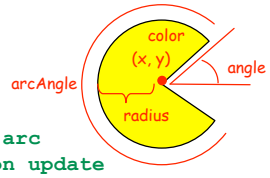


Creating new Sprites

```
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() {
    ...}

}
```

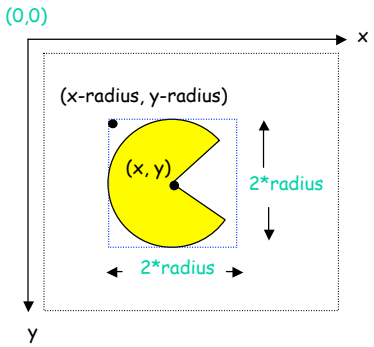
The Rotator Class

```
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 x, int y, int radius, int arcAngle,
                 int deltaAngle, Color color) {
    this.x = x;
    this.y = y;
    this.radius = radius;
    this.arcAngle = arcAngle;
    this.deltaAngle = deltaAngle;
    this.color = color;
    this.angle = 0; // initial startAngle of arc
  // … Instance methods go here …
  }
}
```



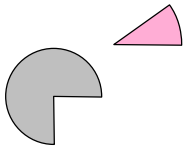
Rotator Instance Methods

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

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

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

  public void resetState () {
    angle = 0;
  }

}
```



Casting

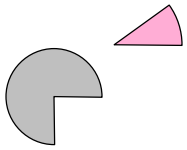
```
public class RotatorAnimation extends Animation {

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



```
    // Recall the Rotator constructor contract:
    public Rotator(int x, int y, int radius, int arcAngle,
                   int deltaAngle, Color color);
```


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(r1);
    this.addSprite(r2);
    // What happens if sprites are added in opposite order?
    this.setNumberFrames(Animation.NO_MAX_FRAMES);
  }
}
```



```
    // Recall the Rotator constructor contract:
    public Rotator(int x, int y, int radius, int arcAngle,
                   int deltaAngle, Color color);
```

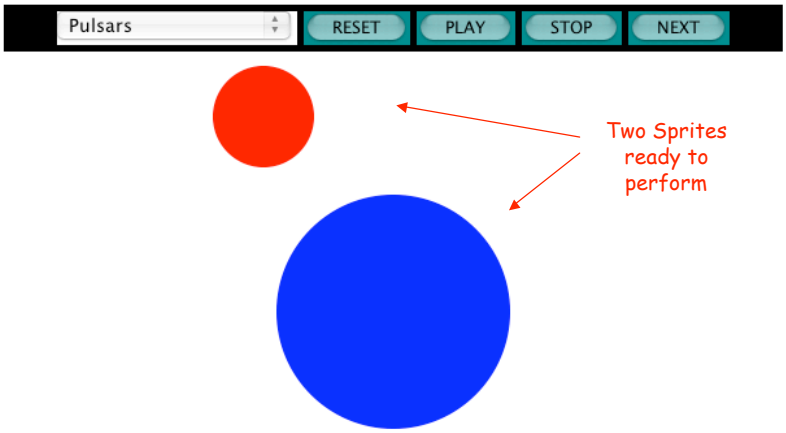
Rotator Updates

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

Other Animation Controls

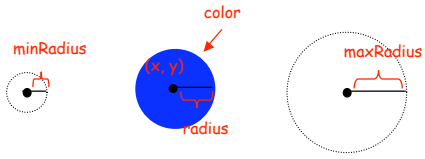
```
public class RotatorAnimation2 extends Animation {
  // Constructor method
  public RotatorAnimation2 () {
    Sprite r1 = new Rotator(400,200,100,30,5,Color.pink));
    Sprite r2 = new Rotator(300,275,75,270,-2,Color.gray));
    this.addSprite(r1);
    this.addSprite(r2);
    this.setNumberFrames(100); // 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
  }
}
```

Pulsars



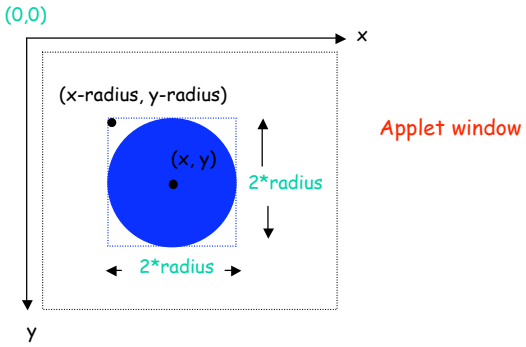
The Pulsar Class

```
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 x, int y, int minRadius, int maxRadius,
                   int changeRadius, Color color)
     {
        this.x = x;
        this.y = y;
        this.radius = minRadius;
        this.minRadius = minRadius;
        this.maxRadius = maxRadius;
        this.changeRadius = changeRadius;
        this.color = color;
     }
}
```



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);
      }
```



Applet window

```
... }
```


Updating the Pulsar

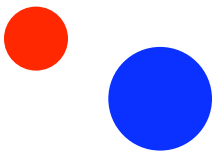
```
public class Pulsar extends Sprite
{
     // instance variables and constructors ...

     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;
          }
     }

... }
```

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;
     }
}
```

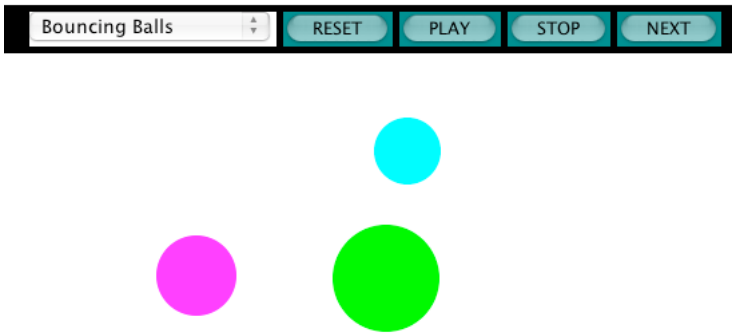

Casting

```
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);
     }
}
```



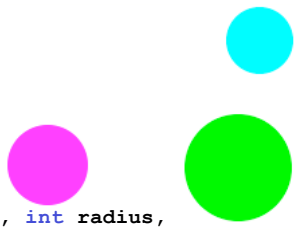
```
public Pulsar(int x, int y, int minRadius, int maxRadius,
              int changeRadius, Color color)
{…}
```

Bouncing balls



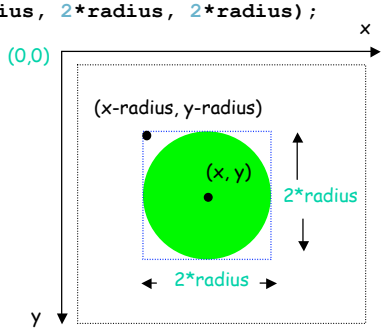
Mobile sprites

```
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 ix, int iy, int radius,
                         int changeX, int changeY, Color color)
     {
          this.initX = ix;
          this.initY = iy;
          this.radius = radius;
          this.changeX = changeX;
          this.changeY = changeY;
          this.color = color;
          this.resetState();
     }
}
```



Rendering & resetting the Sprite

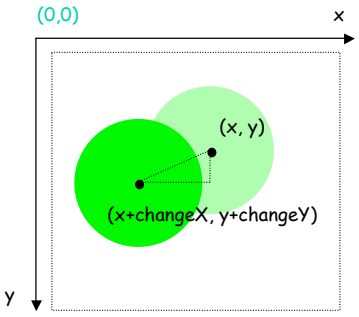
```
public class BouncingBall extends Sprite
{
     // Instance variables and constructor method
      ...
    // Instance methods
    public void drawState(Graphics g)
    {
          g.setColor(color);
          g.fillOval(x-radius, y-radius, 2*radius, 2*radius);
    }
     public void resetState()
     {
          this.x = this.initX;
          this.y = this.initY;
      }
}
```



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;
          }
     }
}
```

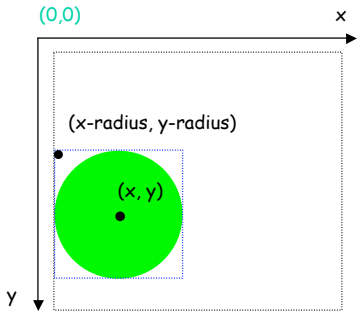
Move ball



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;
      }
    }
}
```

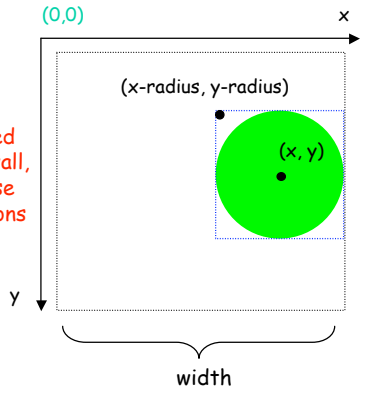
Bumped left wall, reverse directions



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;
      }
    }
}
```

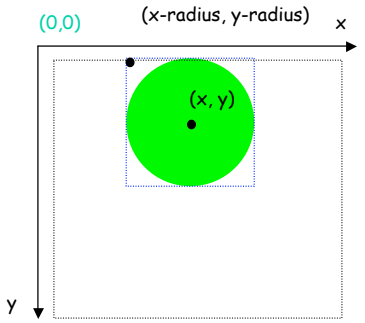
Bumped right wall, reverse directions



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;
       }
    }
}
```

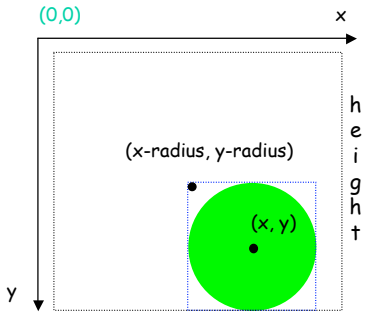
Bumped top, reverse directions



Bottoming out

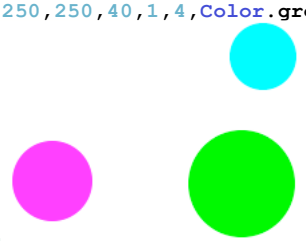
```
public class BouncingBall extends Sprite
{
    ...
    public void updateState()
    {
       x = x + changeX;
       y = y + changeY;
       if (x < radius) {
          x = radius;
          changeX = -changeX;
       }
       if (x + radius >= width) {
          x = width - radius;
          changeX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

Bumped bottom, reverse directions



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 ix, int iy, int radius,
                        int changeX, int changeY, Color color)
    {…}
```