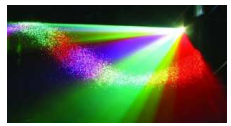
Chapter 9
Described Dynamic Visual Content

The Design and Implementation of Multimedia Software

David Bernstein

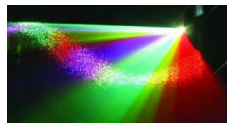
Jones and Bartlett Publishers

www.jbpub.com



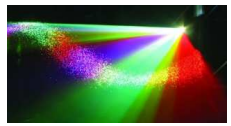
About this Chapter

- This chapter considers ways in which one can describe the way the visual 'stream' changes over time.
- This chapter uses the analogy of the theater (and acting).



What's Next

We need some instant gratification.

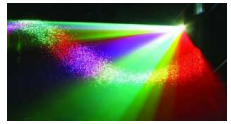


Requirements

F9.1 Manage a collection of sprites.

F9.2 Repeatedly inform each sprite that it should perform the next task in its script.

F9.3 Render the sprites.



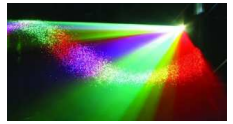
Alternative 1 

- Approach:

  Add code to the `Visualization` class.

- Shortcomings:

  What are the shortcomings?



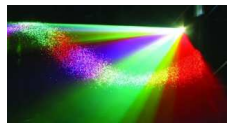
Alternative 1 

- Approach:

  Add code to the `Visualization` class.

- Shortcomings:

  Complexity – There is no reason that someone who is interested in static visual content should have to understand features that are required to work with dynamic visual content.



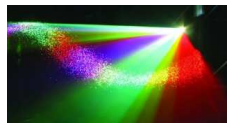
Alternative 2

- Approach:

  Use the decorator pattern.

- Shortcomings:

  What are the shortcomings?



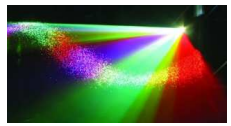
Alternative 2 

- Approach:

  Use the decorator pattern.

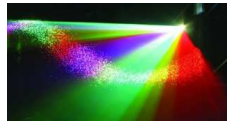
- Shortcomings:

  It is hard to imagine a situation in which, at run time, one would want to add these kinds of capabilities to a `Visualization` object.



Alternative 3

Create a `Stage` class that specializes the `Visualization` class.



Comparison to the `Screen` Class

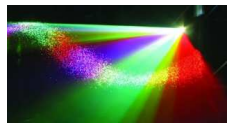
- Similarities:

  Addition of a `Metronome`.

  The ability to respond to 'ticks'.

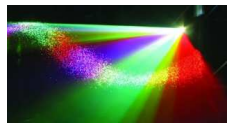
- Differences:

  No notion of a frame (since sprites might need to change their behavior at any time).



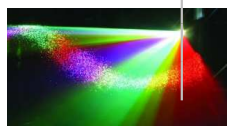
Sprite

```
package visual.dynamic.described;

public interface Sprite extends event.MetronomeListener,
                                visual.statik.TransformableContent
{
}
```



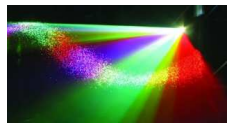
Stage – Structure

```
package visual.dynamic.described;

import java.awt.*;

import event.*;
import visual.*;

public class Stage extends    Visualization
                   implements MetronomeListener
{
  private boolean               shouldRestart;
  private int                   timeStep, restartTime;
  private Metronome             metronome;

  public Stage(int timeStep)
  {
    this(timeStep, new Metronome(timeStep));
  }

  public Stage(int timeStep, Metronome metronome)
  {
    super();

    this.timeStep     =  timeStep;
    shouldRestart     =  false;
    restartTime       = -1;
    this.metronome    = metronome;
    setBackground(Color.WHITE);
```



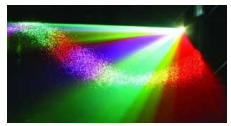
Stage – Structure (cont.)

```
    // The first listener is notified last
    metronome.addListener(this);
  }
}
```



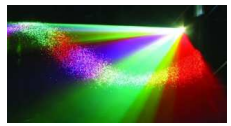
Stage – Metronome

```
public void setRestartTime(int restartTime)
{
  if (restartTime < 0)
  {
    this.restartTime = -1;
    shouldRestart = false;
  }
  else
  {
    this.restartTime = restartTime;
    shouldRestart = true;
  }
}

public void start()
{
  metronome.start();
}

public void stop()
{
  metronome.stop();
}
```



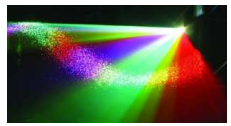
Stage – Managing Sprite Objects

```
public void add(Sprite sprite)
{
  // Make the Sprite a MetronomeListener
  metronome.addListener(sprite);

  // Treat the Sprite as a SimpleContent and
  // add it to the Visualization
  super.add(sprite);
}

public void remove(Sprite sprite)
{
  metronome.removeListener(sprite);
  super.remove(sprite);
}
```



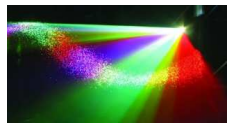
Stage – handleTick()

```
public void handleTick(int time)
{
  if ((shouldRestart) && (time > restartTime))
  {
    metronome.setTime(-timeStep);
  }

  repaint();
}
```

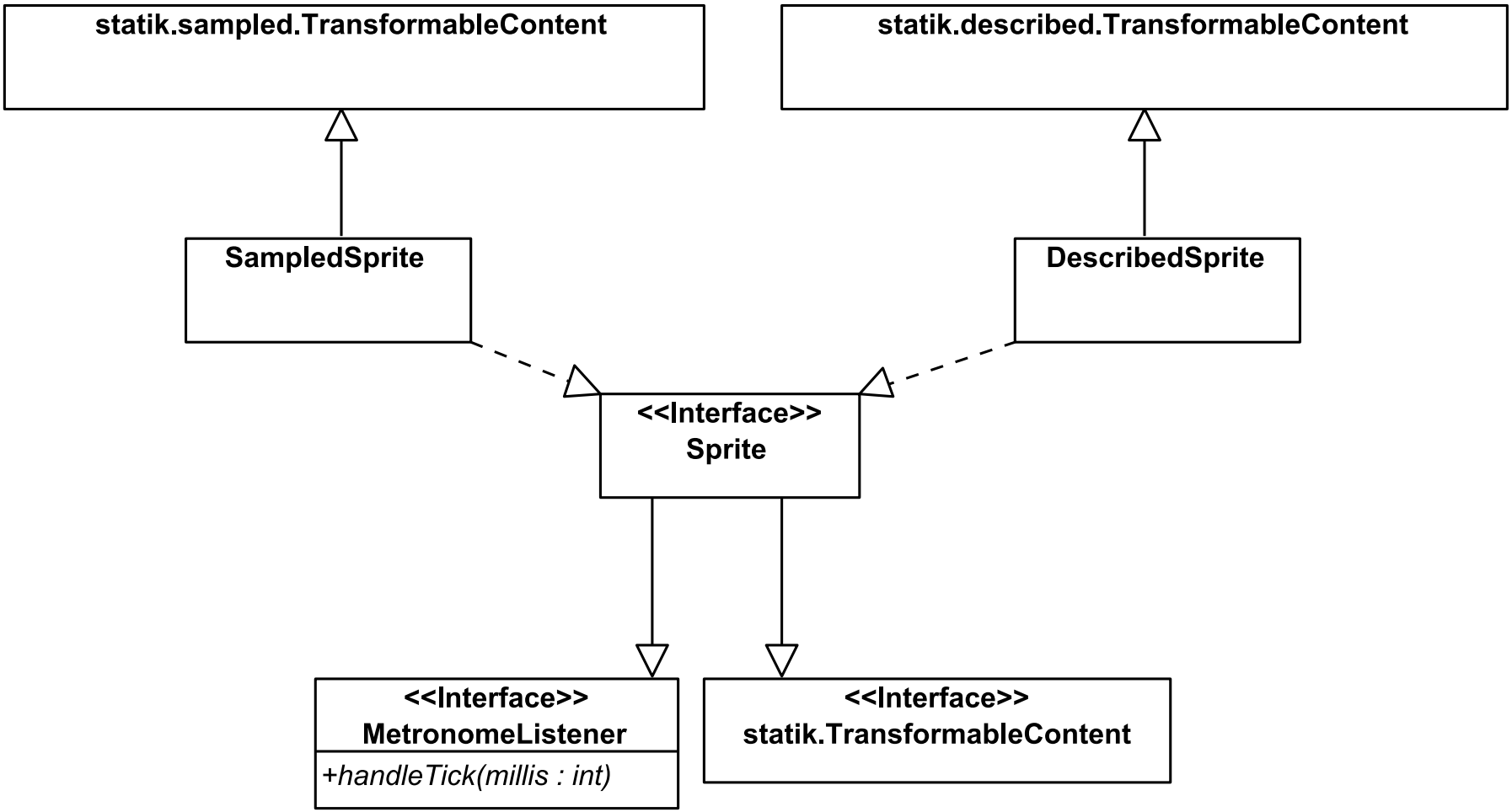


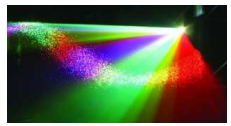
Satisfying Requirements 9.2 and 9.3

- A class that implements the `Sprite` interface must have a `handleTick()`.
- A class that implements the `Sprite` interface must have a `render()` method.

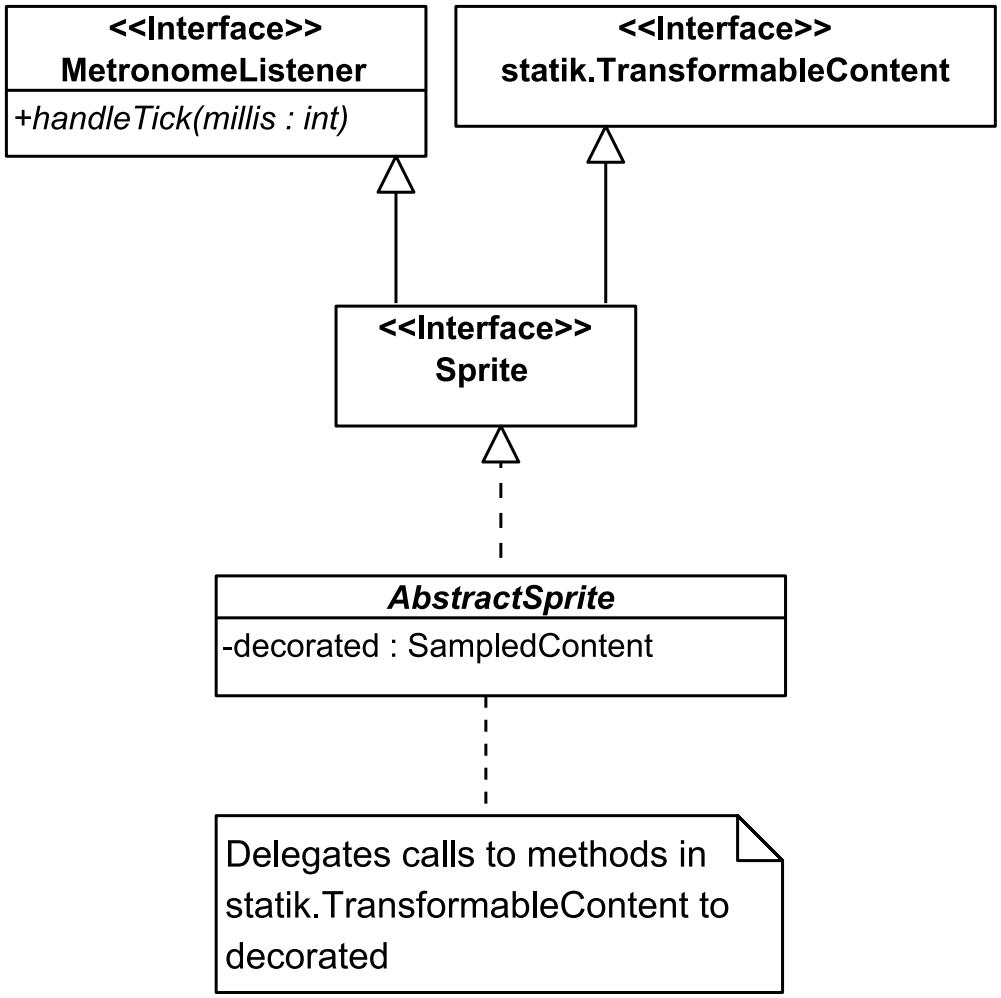


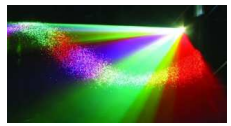
Alternative 1





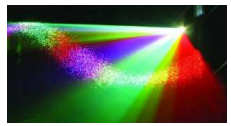
Alternative 2





Advantages of Alternative 2

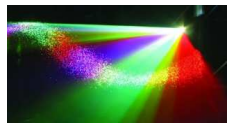
What are the advantages?



Advantages of Alternative 2

Can decorate different `SimpleContent` objects in the same way (e.g., `FallingSprite` could decorate `SimpleContent` that looks like a leaf, a raindrop, a snowflake, etc).

Can associate a different `SimpleContent` object with a particular `Sprite` at different points in time (e.g., a `WalkingPersonSprite` might use different `sampled.Content` objects to represent its legs at different points in the walking process).



The Next Question to Address

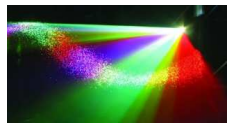
- The Question:

  How to incorporate a ‘script’ in objects that implement the `Sprite` interface.

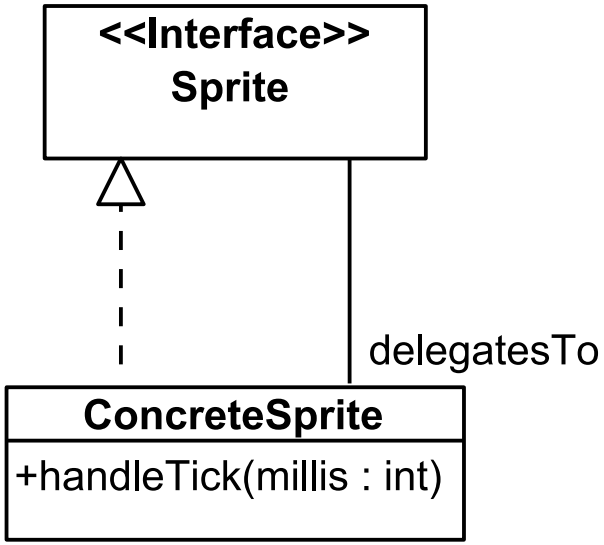
- Common Approaches:

  Use ‘rules’

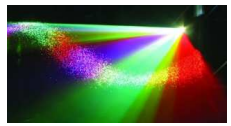
  Interpolate between known states



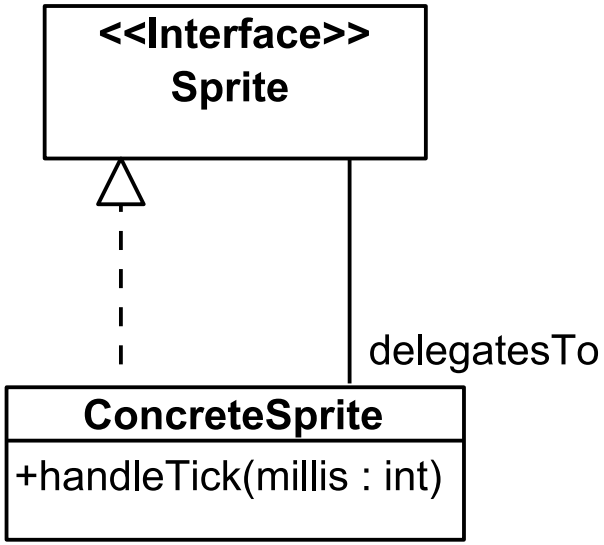
Alternative 1



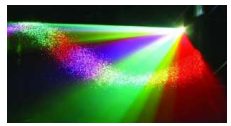
What are the shortcomings?



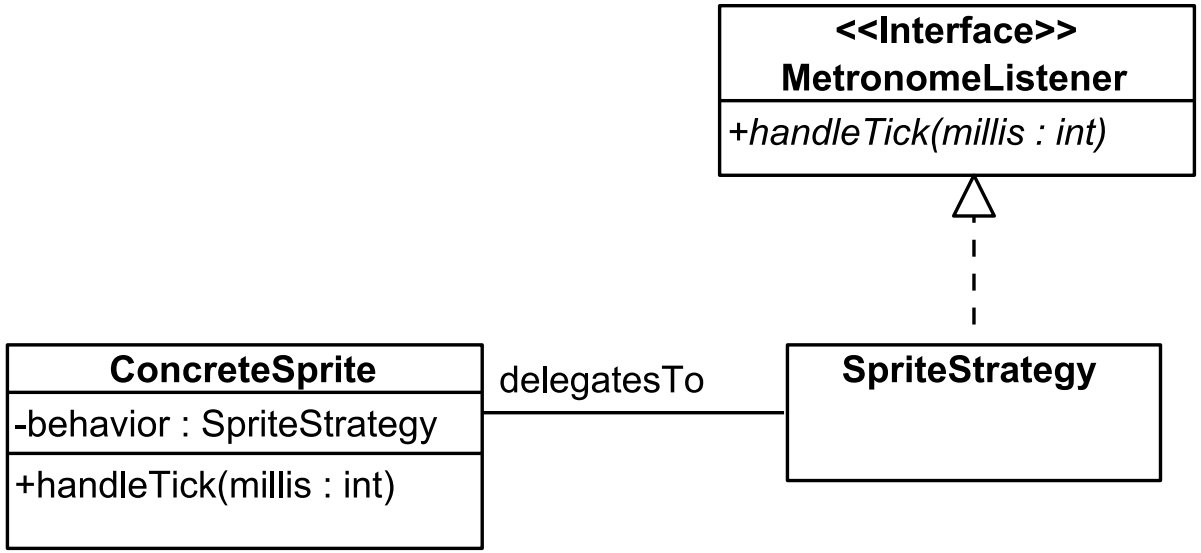
Alternative 1

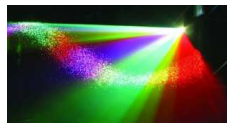


It is a little confusing since a `ConcreteSprite` decorates an `AbstractSprite` which, in turn, decorates a `TransformableContent` object.

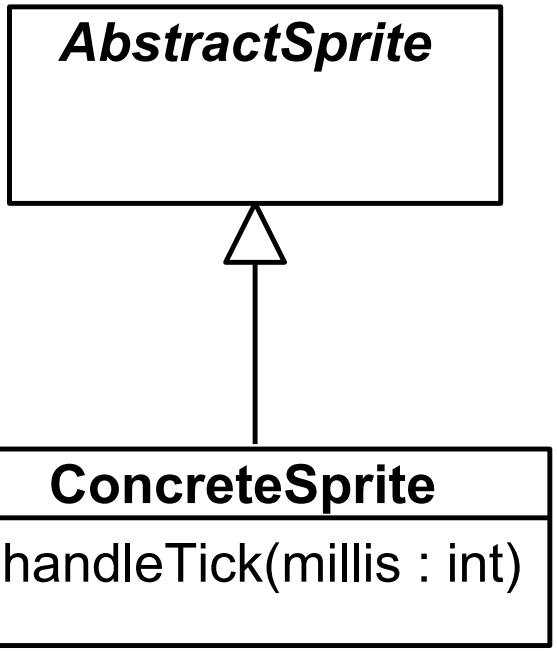


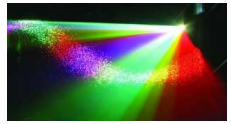
Alternative 2





Alternative 3





Comparing Alternatives 2 and 3

- Thoughts:

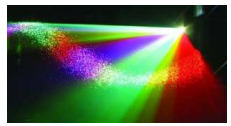
  Both have a lot to offer.

  Specialization is simpler.

  It seems unlikely that the system will need to change a rule-based sprite to an interpolating sprite at run-time.

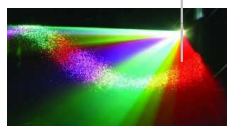
- Fortunately:

  One could use the strategy pattern in the future without breaking any ‘legacy’ classes that used specialization.



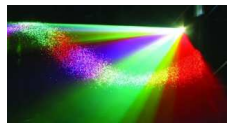
AbstractSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;

import visual.statik.TransformableContent;

public abstract class      AbstractSprite
                implements Sprite
{
  protected boolean     rotationPoint, visible;
  protected double      angle, rotationX, rotationY;
  protected double      scaleX, scaleY, x, y;

  public AbstractSprite()
  {
    super();

    x      = 0.0;
    y      = 0.0;
    angle  = 0.0;
    scaleX = 1.0;
    scaleY = 1.0;

    rotationPoint = false;
    rotationX     = 0.0;
    rotationY     = 0.0;
```



AbstractSprite – Structure (cont.)

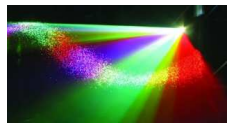
```
  }
}
```



AbstractSprite – Some Abstract Methods

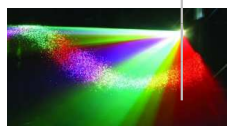
```
public abstract void handleTick(int time);
```

```
protected abstract TransformableContent getContent();
```



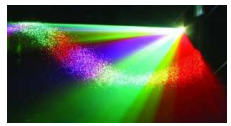
AbstractSprite – Setters

```
public void setLocation(double x, double y)
{
  this.x = x;
  this.y = y;
}

public void setRotation(double r, double x, double y)
{
  rotationPoint = true;
  this.angle    = r;
  this.x        = x;
  this.y        = y;
}

public void setRotation(double r)
{
  rotationPoint = false;
  this.angle    = r;
}

public void setScale(double sx, double sy)
{
  scaleX = sx;
  scaleY = sy;
}

public void setScale(double s)
{
```



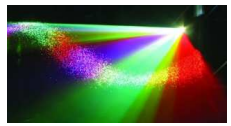
AbstractSprite – Setters (cont.)

```
    setScale(s, s);
  }

  public void setVisible(boolean v)
  {
    visible = v;
  }
}
```



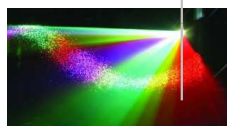
AbstractSprite – getBounds()

```
public Rectangle2D getBounds2D(boolean ofTransformed)
{
  return getContent().getBounds2D(ofTransformed);
}

public Rectangle2D getBounds2D()
{
  return getBounds2D(true);
}
```



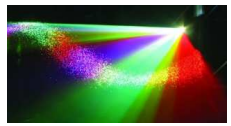
AbstractSprite – Rendering

```
public void render(Graphics g)
{
  double                  rx, ry;
  Rectangle2D             bounds;
  TransformableContent    tc;

  if (visible)
  {
    tc = getContent();

    if (tc != null)
    {
      // Find the point to rotate around
      if (rotationPoint)
      {
        rx = rotationX;
        ry = rotationY;
      }
      else
      {
        bounds = tc.getBounds2D(false);
        rx     = bounds.getWidth()/2.0;
        ry     = bounds.getHeight()/2.0;
      }

      // Transform
      tc.setLocation(x, y);
      tc.setRotation(angle, rx, ry);
```



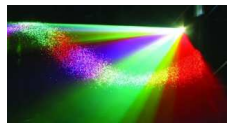
AbstractSprite – Rendering (cont.)

```
        tc.setScale(scaleX, scaleY);

        // Render
        tc.render(g);
      }
    }
  }
```



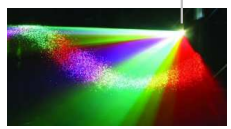
What's Next?

- We need to create specializations of the `AbstractSprite` class.
- For example, let's consider a simple rule-based sprite that 'floats' from the top of the `Stage` to the bottom of the `Stage`.



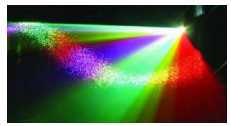
FloatingSprite – Structure

```
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class FloatingSprite extends AbstractSprite
{
  private double                    maxX, maxY, x, y;
  private Random                    rng;
  private TransformableContent      content;

  public FloatingSprite(TransformableContent content,
      double width, double height)
  {
    super();
    this.content = content;
    maxX         = width;
    maxY         = height;

    rng = new Random();

    x = rng.nextDouble()*maxX;
    y = 0.0;
    setLocation(x, y);

    setVisible(true);
```



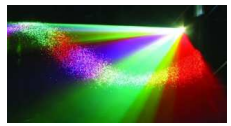
FloatingSprite – Structure (cont.)

```
  }
}
```



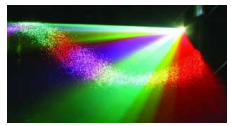
FloatingSprite – getContent()

```
public TransformableContent getContent()
{
  return content;
}
```



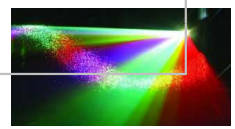
FloatingSprite – handleTick()

```
public void handleTick(int time)
{
  double        n;

  n = rng.nextDouble();
  if      (n < 0.80)   y += 2.0;
  else if (n > 0.90)   y -= 1.0;

  n = rng.nextDouble();
  if      (n < 0.20) x -= 1.0;
  else if (n > 0.80) x += 1.0;

  // Check if at the bottom
  if (y > maxY)
  {
    y  = 0.0;
    x = rng.nextDouble()*maxX;
  }

  setLocation(x, y);
}
```

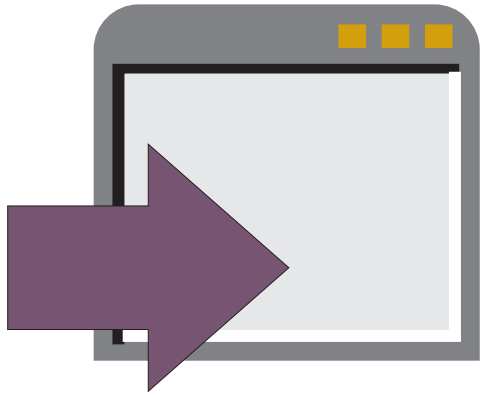


FloatingSpriteDemo

```
FloatingSprite                  sprite;
ResourceFinder                  finder;
TransformableContent            content;

int width   = 640;
int height  = 480;

finder  = ResourceFinder.createInstance(new resources.Marker());
ContentFactory factory = new ContentFactory(finder);

// The Stage
Stage stage = new Stage(50);
stage.setBackground(new Color(255, 255, 255));
VisualizationView stageView = stage.getView();
stageView.setBounds(0,0,width,height);

// The Sprite
content = factory.createContent("snowflake.png", 4, false);
sprite  = new FloatingSprite(content, width, height);
stage.add(sprite);

// The content pane
JPanel contentPane = (JPanel)getContentPane();
contentPane.add(stageView);

// Start the dynamics
stage.start();
```

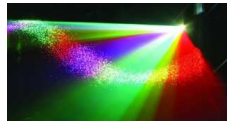


FloatingSpriteDemo – Demonstration



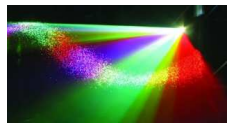
In examples/chapter:

```
java -cp multimedia2.jar:examples.jar FloatingSpriteDemo
```



What's Next

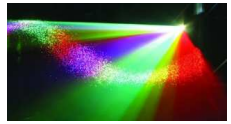
We need to consider the encapsulation of rule-based dynamics.



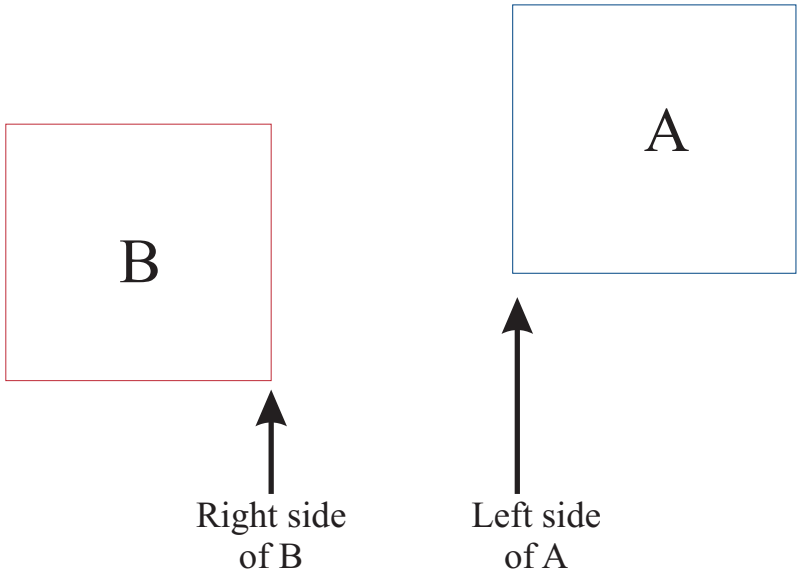
Requirements

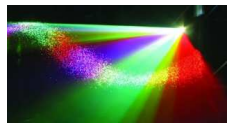
F9.4 Allow one sprite to interact with another.

F9.5 Allow the user to interact with sprites.



Determining if Rectangles Do Not Intersect

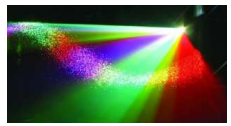




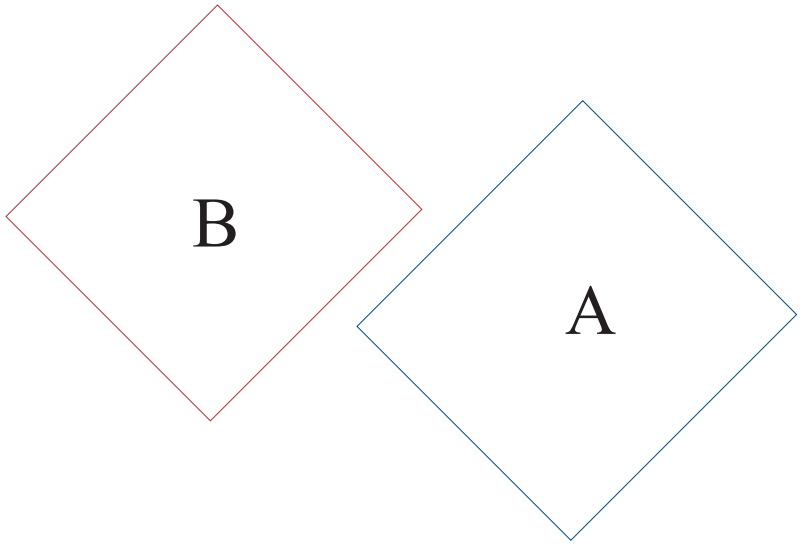
Determining if Rectangles Do Not Intersect (cont.)

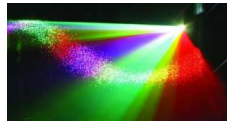
Letting `rightA`, `leftA`, `topA` and `botA` denote the right, left, top, and bottom of A, and `rightB`, `leftB`, `topB` and `botB` denote the right, left, top, and bottom of B, the expression to use to test if A and B **do not** intersect is:

```
(rightA < leftB) || (leftA > rightB) || (botA < topB) || (topA > botB)
```

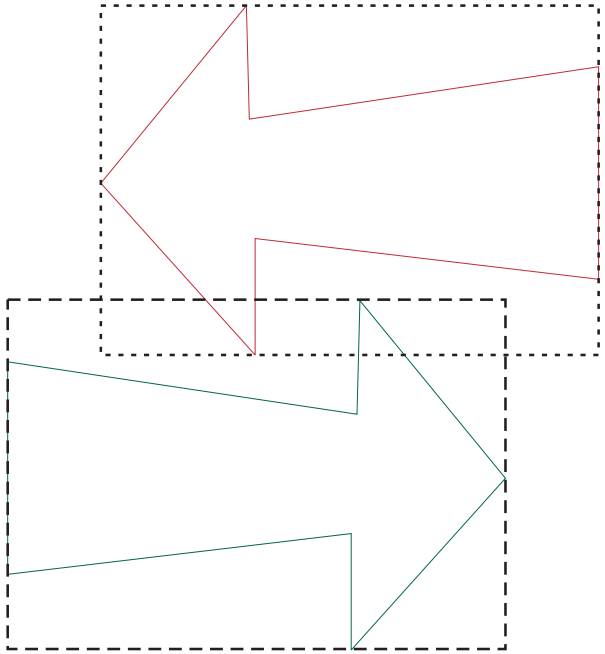


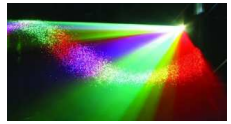
Intersection of Non-Rectangular, Convex Sprites





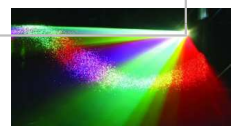
Using Bounding Boxes





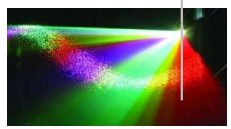
AbstractSprite – intersects()

```
public boolean intersects(Sprite s)
{
  boolean          retval;
  double           maxx, maxy, minx, miny;
  double           maxx0, maxy0, minx0, miny0;
  Rectangle2D      r;

  retval = true;

  r = getBounds2D(true);
  minx = r.getX();
  miny = r.getY();
  maxx = minx + r.getWidth();
  maxy = miny + r.getHeight();

  r = s.getBounds2D(true);
  minx0 = r.getX();
  miny0 = r.getY();
  maxx0 = minx0 + r.getWidth();
  maxy0 = miny0 + r.getHeight();

  if ( (maxx < minx0) || (minx > maxx0) ||
       (maxy < miny0) || (miny > maxy0) ) retval = false;

  return retval;
}
```



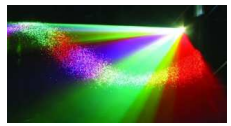
RuleBasedSprite

```
package visual.dynamic.described;

import java.util.*;

import visual.statik.TransformableContent;

public abstract class RuleBasedSprite extends AbstractSprite
{
  protected ArrayList<Sprite>          antagonists;
  protected TransformableContent       content;

  public RuleBasedSprite(TransformableContent content)
  {
    super();

    antagonists = new ArrayList<Sprite>();
    this.content = content;
    setVisible(true);
  }

  public void addAntagonist(Sprite antagonist)
  {
    antagonists.add(antagonist);
  }

  public TransformableContent getContent()
  {
    return content;
```



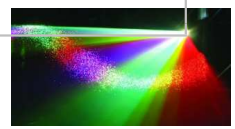
RuleBasedSprite (cont.)

```
  }

  public abstract void handleTick(int time);


  public void removeAntagonist(Sprite antagonist)
  {
    antagonists.remove(antagonist);
  }
}
```



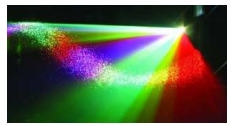
Fish – Structure

```
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class Fish extends RuleBasedSprite
{
  protected double      initialSpeed, maxX, maxY, speed, x, y;

  private static final int     INITIAL_LOCATION = -320;
  private static final Random  rng = new Random();

  public Fish(TransformableContent content,
      double width, double height, double speed)
  {
    super(content);
    maxX = width;
    maxY = height;

    x     = rng.nextDouble()*maxX;
    y     = rng.nextInt()*maxY;

    this.initialSpeed = speed;
    this.speed        = speed;
  }
}
```



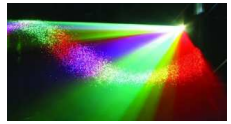
Fish – updateLocation()

```
protected void updateLocation()
{
  x += speed;

  if (x > (int)maxX)
  {
    x     = INITIAL_LOCATION;
    y     = rng.nextDouble()*maxY;
    speed = initialSpeed;
  }

  // Set the location
  setLocation(x, y);
}
```



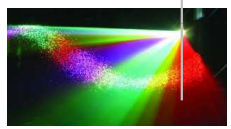
Fish – handleTick()

```
public void handleTick(int time)
{
  Iterator<Sprite>  i;
  Sprite            shark;

  i = antagonists.iterator();
  while (i.hasNext())
  {
    shark = i.next();
    if (intersects(shark)) speed = 20.;
  }

  updateLocation();
}
```



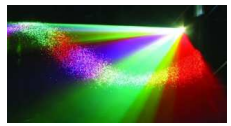
FishTankDemo

```
int width  = 640;
int height = 480;

finder       = ResourceFinder.createInstance(new resources.Marker());
ContentFactory factory = new ContentFactory(finder);
ImageFactory imageFactory = new ImageFactory(finder);

// The Stage
Stage stage = new Stage(50);
stage.setBackground(Color.blue);
Content content = factory.createContent("ocean.png", 3, false);
stage.add(content);
stageView = stage.getView();
stageView.setBounds(0,0,width,height);

// The Shark
content = factory.createContent("shark.png", 4, false);
Fish shark = new Fish(content, width, height, 8.);
stage.add(shark);

// The school of Fish
// (Use the same BufferedImage object for all Fish)
image   = imageFactory.createBufferedImage("fish.png", 4);
for (int i=0; i<10; i++)
{
  content = factory.createContent(image, false);
  Fish fish = new Fish(content, width, height, 3.);
  fish.addAntagonist(shark);
```

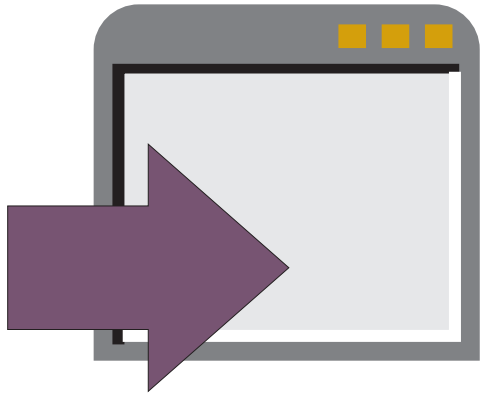


FishTankDemo (cont.)

```
    stage.add(fish);
  }

  // The content pane
  JPanel contentPane = (JPanel)getContentPane();
  contentPane.add(stageView);

  stage.start();
```

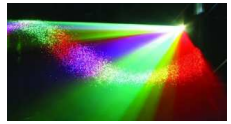


FishTankDemo – Demonstration



In examples/chapter:

```
java -cp multimedia2.jar:examples.jar FishTankDemo
```



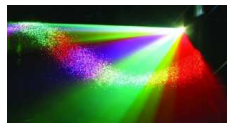
Sprites with Multiple Pieces of Content

- The Objective:

  Make a fish appear to move it's tail.

- What's Needed?

  What's Needed?



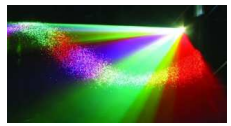
Sprites with Multiple Pieces of Content

- The Objective:

  Make a fish appear to move it's tail.

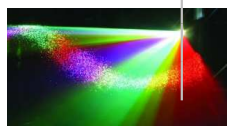
- What's Needed?

  Different `Content` objects for different states.



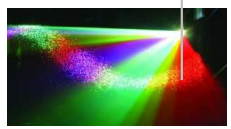
A Sprite with Multiple Pieces of Content

```
import java.util.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class SwimmingFish extends RuleBasedSprite
{
        protected double      initialSpeed, maxX, maxY, speed, x, y;
        protected int         lastTime, millisPerState, state, stateChange;
        protected int         timeInState;
        protected TransformableContent[]  contents;



        private static final int     INITIAL_LOCATION = -320;
        private static final Random  rng = new Random();

        public SwimmingFish(TransformableContent[] contents,
                        double width, double height, double speed)
        {
                super(contents[0]);

                this.contents = contents;

                maxX = width;
                maxY = height;

                x    = rng.nextDouble()*maxX;
```



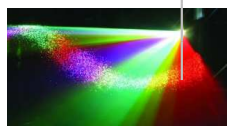
A Sprite with Multiple Pieces of Content (cont.)

```
        y    = rng.nextInt()*maxY;

        this.initialSpeed = speed;
        this.speed        = speed;
        state             = 0;
        lastTime          = 0;
        timeInState       = 0;
        stateChange       = 1;
}


public TransformableContent getContent()
{
        return contents[state];
}


public void handleTick(int time)
{
        Iterator<Sprite>  i;
        Sprite            shark;

        i = antagonists.iterator();
        while (i.hasNext())
        {
                shark = i.next();
                if (intersects(shark)) speed = 20.;
```



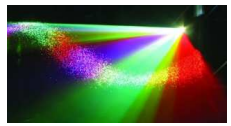
A Sprite with Multiple Pieces of Content (cont.)

```
        }

        millisPerState    = 500 - (int)(speed*20);

        timeInState += (time - lastTime);
        if (timeInState > millisPerState)
        {
                timeInState = 0;
                state += stateChange;
                if      (state == 2) stateChange = -1;
                else if (state == 0) stateChange =  1;
        }
        lastTime = time;

        updateLocation();
}


protected void updateLocation()
{
        x += speed;

        if (x > (int)maxX)
        {
                x     = INITIAL_LOCATION;
                y     = rng.nextDouble()*maxX;
                speed = initialSpeed;
```

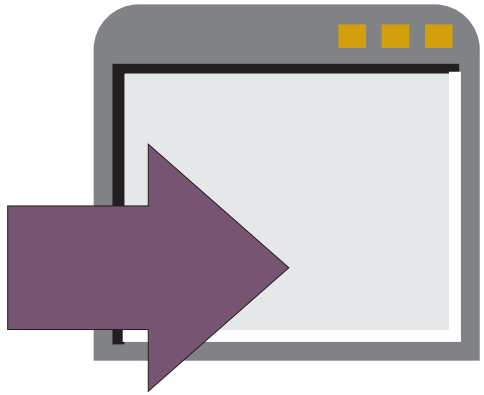


A Sprite with Multiple Pieces of Content (cont.)

```
                }

                // Set the location
                setLocation(x, y);
        }
}
```

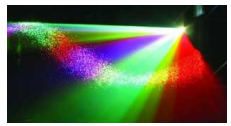


SwimmingFishTankDemo – Demonstration



In extras:

```
java -cp multimedia2.jar:examples.jar SwimmingFishTankDemo
```



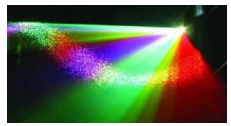
Sprites that Move Together

- The Objective:

  Different `Sprite` objects move together (e.g., exhaust coming out of a bus).

- What's Needed?

  What's Needed?



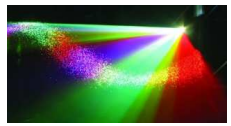
Sprites that Move Together

- The Objective:

  Different `Sprite` objects move together (e.g., exhaust coming out of a bus).

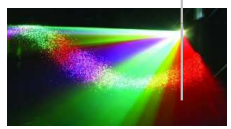
- What's Needed?

  One `Sprite` object that "controls" other `Sprite` objects.



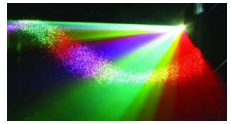
Sprites that Move Together – Code

```
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class BigBus extends RuleBasedSprite
{
  private double              maxX, x, y;
  private Exhaust[]           exhaust;

  public BigBus(TransformableContent content,
      double width, double height,
      Stage stage)
  {
    super(content);

    exhaust = new Exhaust[15];
    for (int i=0; i<exhaust.length; i++)
    {
      exhaust[i] = new Exhaust();
      stage.add(exhaust[i]);
    }

    x    =   0.0;
    y    = 300.0;
    maxX = width;
  }

  public void handleTick(int millis)
  {
```



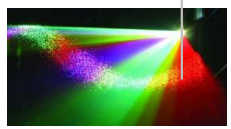
Sprites that Move Together – Code (cont.)

```
    // Move the bus
    x = x + 1;
    setLocation(x, y);
    if (x > maxX+50)
    {
      setVisible(false);
      for (int i=0; i<exhaust.length; i++)
        exhaust[i].setVisible(false);
    }

    // Set the origin for the Exhaust objects
    for (int i=0; i<exhaust.length; i++)
      exhaust[i].setOrigin(x, y+63);

    // Inform the Exhaust objects that a tick has occurred
    for (int i=0; i<exhaust.length; i++)
      exhaust[i].handleTick(millis);
  }
}
```



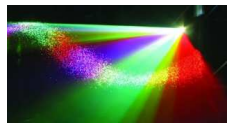
Sprites that Move Together – Code (cont.)

```
import java.awt.*;
import java.awt.geom.*;
import java.util.Random;

import visual.dynamic.described.*;
import visual.statik.described.*;

public class Exhaust extends RuleBasedSprite
{
    private double        originX, originY;
    private int           count, length, slope;

    private static final  int      DIAMETER = 5;
    private static final  Random   rng      = new Random();

    public Exhaust()
    {
       super(new Content(new Ellipse2D.Float(0,0,DIAMETER,DIAMETER),
                         Color.BLACK,
                         Color.GRAY,
                         new BasicStroke()
                         )
             );

       length = rng.nextInt(15);
       count  = -1;
```

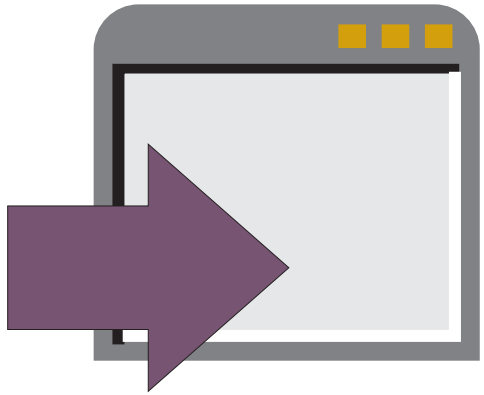


Sprites that Move Together – Code (cont.)

```
    }

    public void handleTick(int millis)
    {
       count++;

       if (count >= length)
       {
          count = 0;
          setLocation(originX, originY);
       }
       else
       {
          slope  = rng.nextInt(4) - 1;
          setLocation(originX-count, originY-(count*slope));
       }
    }

    public void setOrigin(double x, double y)
    {
       originX = x - DIAMETER/2;
       originY = y - DIAMETER/2;
    }
}
```

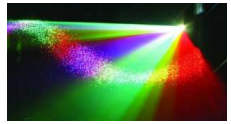


Sprites that Move Together – Demonstration



In `extras`:

```
java -cp multimedia2.jar:examples.jar BigBusDemo
```



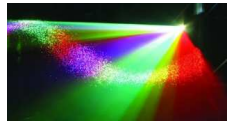
Satisfying Requirement 9.5

- Interested `Sprite` objects must be able to observe user-generated events.

  They can implement the `KeyListener` interface and/or the `MouseListener` and `MouseMotionListener` interfaces.

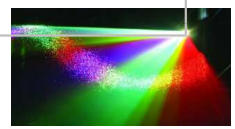
- A subject is needed.

  The `VisualizationView` class extends the `JComponent` class, and the `JComponent` class provides this functionality.



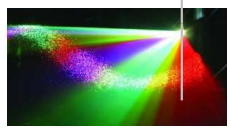
Visualization – Key Listeners

```
public void addKeyListener(KeyListener kl)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.addKeyListener(kl);
  }
}

public synchronized void removeKeyListener(
    KeyListener kl)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.removeKeyListener(kl);
  }
}
```



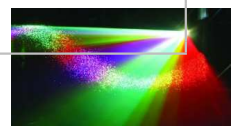
Visualization – Mouse Listeners

```
public void addMouseListener(MouseListener ml)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.addMouseListener(ml);
  }
}

public void addMouseMotionListener(
    MouseMotionListener mml)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.addMouseMotionListener(mml);
  }
}

public synchronized void removeMouseListener(
```



Visualization – Mouse Listeners (cont.)

```
    MouseListener ml)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.removeMouseListener(ml);
  }
}

public synchronized void removeMouseMotionListener(
    MouseMotionListener mml)
{
  Iterator<VisualizationView>  i;
  VisualizationView            view;

  i = getViews();
  while (i.hasNext())
  {
    view = i.next();
    view.removeMouseMotionListener(mml);
  }
}
```



An Example

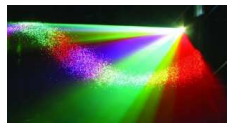
- The Setting:

  An amazingly addictive (and/or unbearably stupid) balloon popping game.

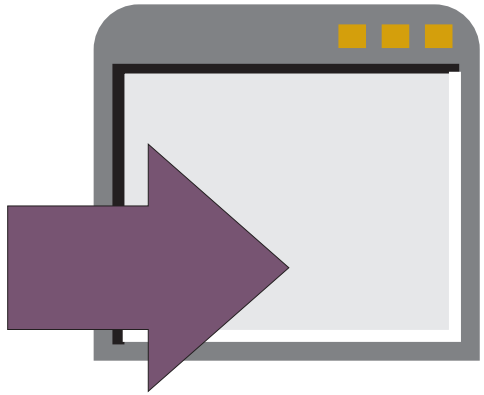
- The Participants:

  `Cupola`

  `Balloon`

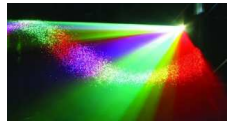


An Example – Demonstration



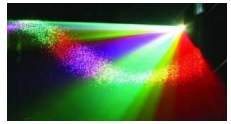
In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar BalloonDemo
```



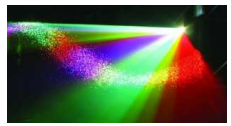
Cupola – Structure

```
import java.awt.event.*;
import java.awt.geom.*;

import visual.dynamic.described.*;
import visual.statik.TransformableContent;

public class Cupola extends    RuleBasedSprite
                    implements MouseMotionListener
{
  private double     left, top;

  public Cupola(TransformableContent content,
      double stageWidth, double stageHeight)
  {
    super(content);
    Rectangle2D        bounds;

    bounds = content.getBounds2D(false);
    top    = (stageHeight - bounds.getHeight() - 34);
    left   = (stageWidth  - bounds.getWidth())/2.0;

    setLocation(left, top);
  }
}
```



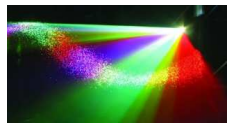
Cupola – MouseMotionListener

```
public void mouseDragged(MouseEvent evt)
{
  mouseMoved(evt);
}

public void mouseMoved(MouseEvent evt)
{
  this.left = (double)evt.getX();
}
```



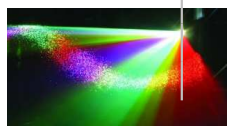
Cupola – handleTick()

```
public void handleTick(int time)
{
  setLocation(left, top);
}
```



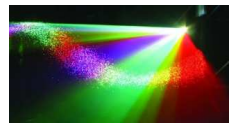
Balloon – handleTick()

```
public void handleTick(int time)
{
  Sprite   cupola;

  // Wait for the initial rendering
  if (time < 1000) return;

  // Check for an intersection
  cupola = null;
  if (antagonists.size() > 0) cupola = antagonists.get(0);


  if ((cupola != null) && (intersects(cupola)))
  {
    speed = 0;
    setVisible(false);
  }

  // Update the location
  top += speed;

  if (top > maxY)
  {
    left   = rng.nextInt(maxX);
    top    = minY;
    speed  = 1 + rng.nextInt(10);
  }
```



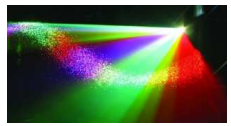
Balloon – handleTick() (cont.)

```
    // Set the location
    setLocation(left, top);
  }
```



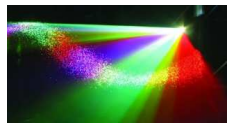
What's Next

We need to consider the encapsulation of key-time dynamics.



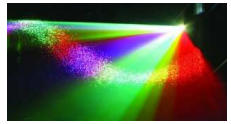
Jobs in Traditional Cel Animation

- Drawing backgrounds.
- Drawing key/important frames.
- Drawing all of the frames in between the key frames.



Requirements

F9.6 Support the description of dynamic behavior using key-times and tweening.



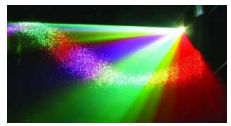
Alternative 1

- Approach:

  Store the attributes of the `TransformableContent` objects at each key time in the `TransformableContent` objects themselves.

- Shortcomings:

  What are the shortcomings?



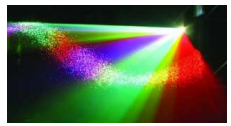
Alternative 1

- Approach:

  Store the attributes of the `TransformableContent` objects at each key time in the `TransformableContent` objects themselves.

- Shortcomings:

  It makes it difficult to interpolate between the key times.



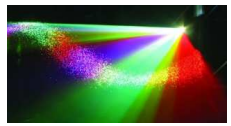
Alternative 2

- Approach:

  Keep the attributes for each of the key times external to the `TransformableContent` objects.

- Advantages:

  What are the advantages?



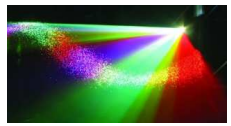
Alternative 2

- Approach:

  Keep the attributes for each of the key times external to the `TransformableContent` objects.

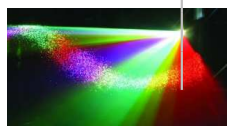
- Advantages:

  The `Sprite` has easy access to all of the information it needs to calculate the attributes at the in-between times.



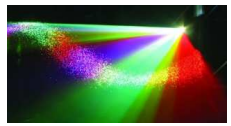
TweeningSprite – Structure

```
package visual.dynamic.described;

import java.awt.geom.*;
import java.util.*;

public abstract class TweeningSprite extends AbstractSprite
{
  protected ArrayList<Integer>  keyTimes;
  protected ArrayList<Point2D>  locations;
  protected ArrayList<Double>   rotations, scalings;
  private   double              frac;
  private   int                 currentIndex, endState, lastTime;
  private   int                 nextIndex, nextKT;

  public static final int REMAIN         = 0;
  public static final int REMOVE         = 1;

  public TweeningSprite()
  {
    super();

    keyTimes  = new ArrayList<Integer>();
    locations = new ArrayList<Point2D>();
    rotations = new ArrayList<Double>();
    scalings  = new ArrayList<Double>();
    endState  = REMAIN;
```



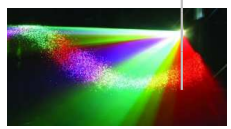
TweeningSprite – Structure (cont.)

```
    initialize();
  }
}
```



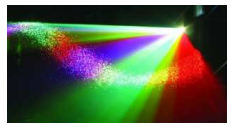
TweeningSprite – addKeyTime()

```
protected int addKeyTime(int keyTime, Point2D location,
    Double rotation, Double scaling)
{
  boolean     keepLooking;
  int         existingKT, i;

  existingKT  = -1;
  keepLooking = true;

  i = 0;
  while ((i < keyTimes.size()) && keepLooking)
  {
    existingKT = ((Integer)keyTimes.get(i)).intValue();

    if (existingKT >= keyTime) keepLooking = false;
    else                       i++;
  }

  if ((existingKT == i) && !keepLooking)  // Duplicate
  {
    i = -1;
  }
  else
  {
    keyTimes.add(i, new Integer(keyTime));
    locations.add(i, location);
```



TweeningSprite – addKeyTime() (cont.)

```
      rotations.add(i, rotation);
      scalings.add(i, scaling);
    }

    return i;
  }
```

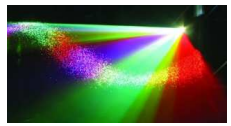


Linear Interpolation

Letting a_t denote the value of the attribute at the previous/current key time and a_{t+1} denote the value of the attribute at the next key time, the in-between value, $b(\lambda)$, is then given by:

$$b(\lambda) = (1 - \lambda)a_t + \lambda a_{t+1} \tag{1}$$

where $\lambda \in [0, 1]$ denotes the interpolation fraction.



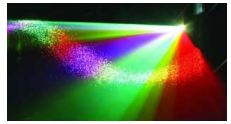
Linear Interpolation (cont.)

Note that (1) implies:

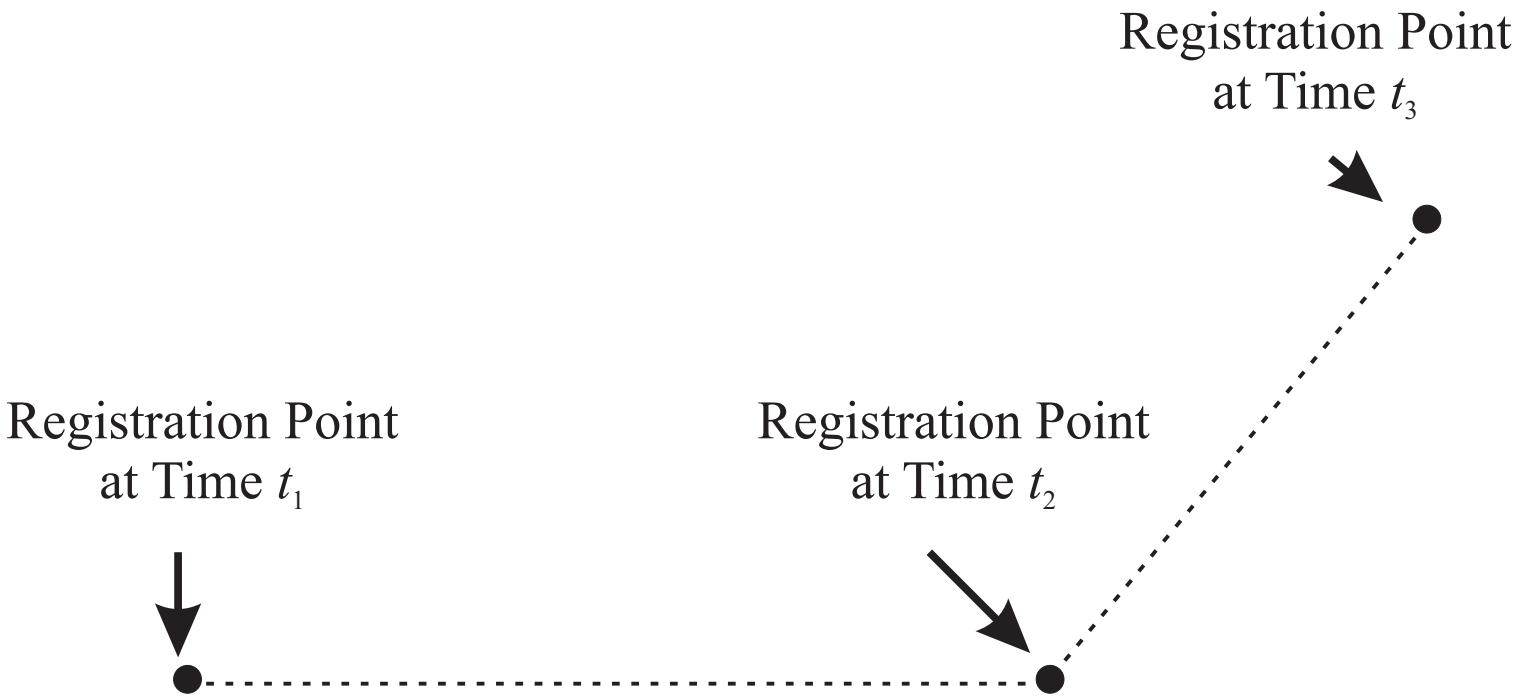
$$b(\lambda) = a_t - \lambda a_t + \lambda a_{t+1} \quad (2)$$

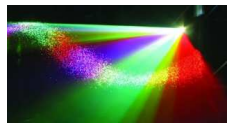
$$= a_t + \lambda(a_{t+1} - a_t) \quad (3)$$

which is the more widely-used form.



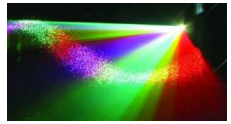
Location Tweening



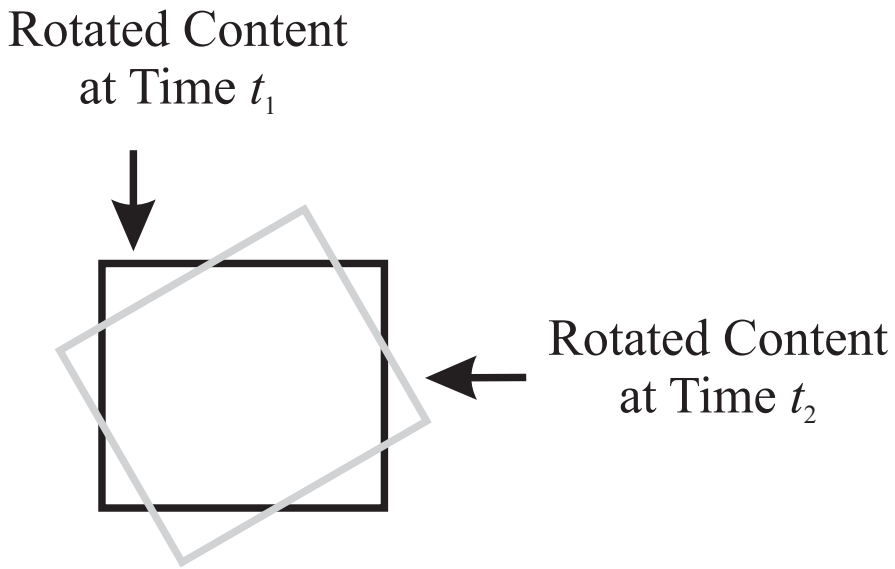


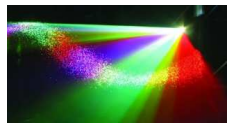
TweeningSprite – tweenLocation

```
protected void tweenLocation(int currentIndex, int nextIndex,
    double frac)
{
  double          x, y;
  Point2D         currentKTLocation, nextKTLocation;

  currentKTLocation = locations.get(currentIndex);
  nextKTLocation    = locations.get(nextIndex);

  x = currentKTLocation.getX() +
      frac*(nextKTLocation.getX()- currentKTLocation.getX());
  y = currentKTLocation.getY() +
      frac*(nextKTLocation.getY() - currentKTLocation.getY());

  setLocation(x, y);
}
```



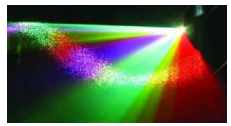
Pure Rotation Tweening



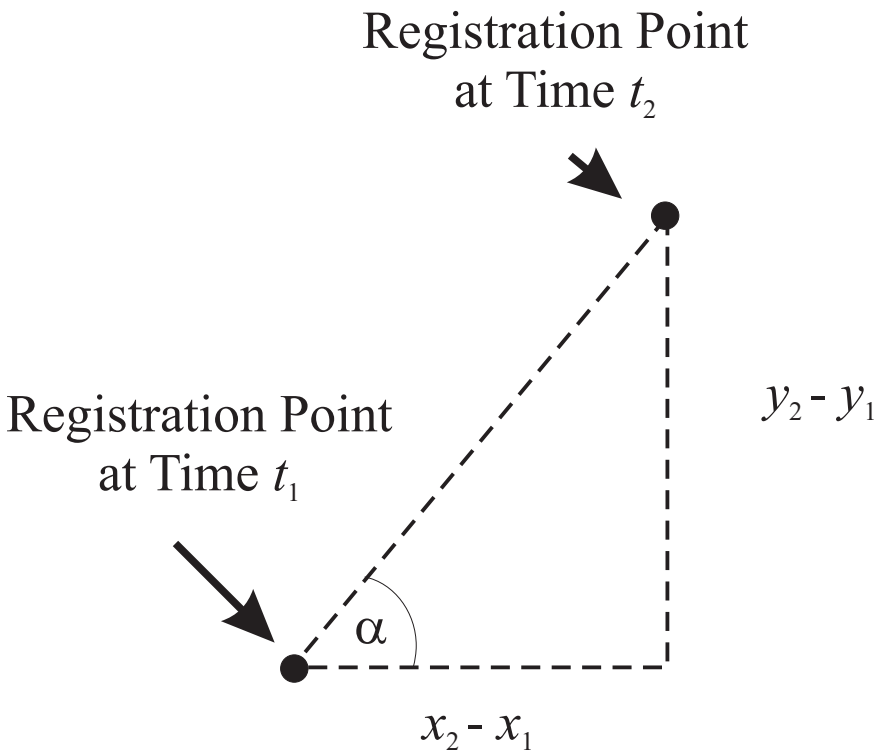


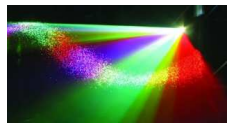
TweeningSprite – Pure Rotation Tweening

```
      currentKTRotation = rotation.doubleValue();

      rotation = rotations.get(nextIndex);
      if (rotation == null) nextKTRotation = currentKTRotation;
      else                  nextKTRotation = rotation.doubleValue();

      r = currentKTRotation + frac*(nextKTRotation-currentKTRotation);
    }
```



Aligned Rotation Tweening





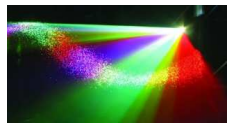
Pure Rotation Tweening (cont.)

Thinking of the current segment as the hypotenuse of a right triangle, the difference in y values defines the length of the side opposite the angle of interest (denoted by α), and the difference in x values defines the length of the adjacent side. Hence:

$$\tan(\alpha) = \frac{y_2 - y_1}{x_2 - x_1} \tag{4}$$

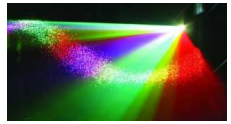
It follows that:

$$\alpha = \text{atan}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{5}$$



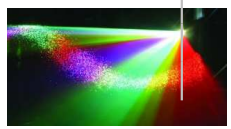
TweeningSprite – Aligned Rotation Tweening

```
currentKTLocation = locations.get(currentIndex);
nextKTLocation    = locations.get(nextIndex);

r=Math.atan((nextKTLocation.getY()-currentKTLocation.getY())/
    (nextKTLocation.getX()-currentKTLocation.getX()) );
```



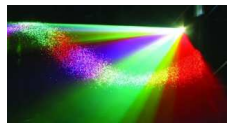
An Example: Airplane

```
import java.awt.geom.*;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class Airplane extends SampledSprite
{
    public Airplane()
    {
        super();
        Content             content;
        ContentFactory      factory;

        factory = new ContentFactory(ResourceFinder.createInstance(new resources.Marker()));
        content = factory.createContent("airplane.png", 4);
        addKeyTime(  500,   0.0, 350.0, -0.75, 1.0, content);
        addKeyTime( 2000, 100.0, 200.0, -0.30, 1.0, null);
        addKeyTime( 4000, 200.0,  50.0,  0.00, 1.0, null);
        addKeyTime( 6000, 300.0,  50.0,  0.20, 1.0, null);
        addKeyTime( 8000, 400.0, 200.0,  0.00, 1.0, null);
        addKeyTime( 8500, 500.0, 200.0,  0.00, 1.0, null);
        setEndState(REMOVE);
    }

    private void addKeyTime(int time, double x, double y,
                            double r, double s, Content c)
    {
```

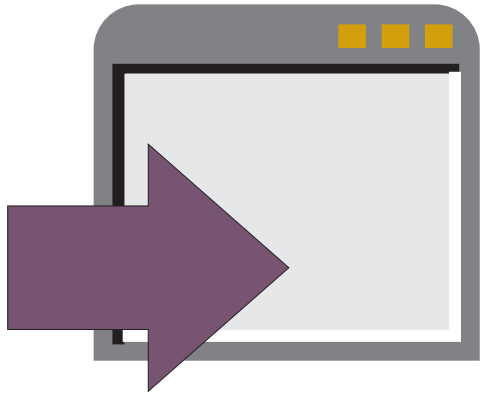


An Example: `Airplane` (cont.)

```
        addKeyTime(time, new Point2D.Double(x, y), new Double(r),
                    new Double(s), c);
    }
}
```

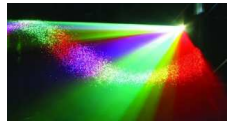


Airplane – Demonstration



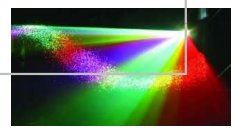
In examples/chapter:

```
java -cp multimedia2.jar:examples.jar AirplaneDemo
```

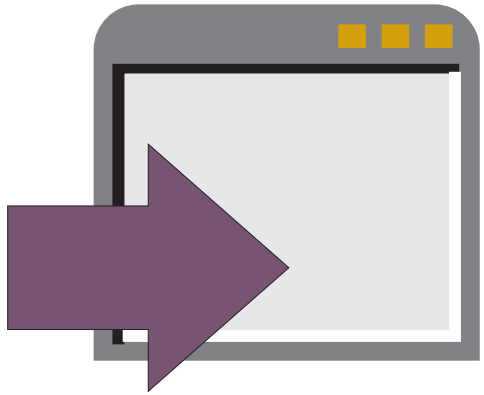


Another Example: BuzzyOnMars

```
import java.awt.geom.*;

import visual.dynamic.described.DescribedSprite;
import visual.statik.described.*;

public class BuzzyOnMars extends DescribedSprite
{
  public BuzzyOnMars()
  {
    super();
    BuzzyStanding       buzzy;

    buzzy = new BuzzyStanding();
    addKeyTime(  500,   0.0, 380.0,  0.00, 1.0, buzzy);
    addKeyTime( 2000, 180.0, 380.0,  0.00, 1.0, null);
    addKeyTime( 4000, 180.0,  75.0,  0.20, 1.0, null);
    addKeyTime( 6000, 640.0,  20.0,  6.48, 1.0, null);
    setEndState(REMOVE);
  }

  private void addKeyTime(int time, double x, double y,
      double r, double s, AggregateContent c)
  {
    addKeyTime(time, new Point2D.Double(x, y), new Double(r),
        new Double(s), c);
  }
}
```

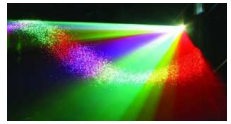


BuzzyOnMars – Demonstration



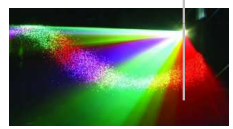
In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar BuzzyOnMarsDemo
```



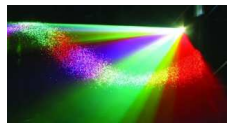
A Last Example: BusOnRoute

```
import java.awt.geom.Point2D;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class BusOnRoute extends SampledSprite
{
  public BusOnRoute()
  {
    super();
    Content               content;
    ContentFactory        factory;
    ResourceFinder        finder;

    finder  = ResourceFinder.createInstance(new resources.Marker());
    factory = new ContentFactory(finder);
    content = factory.createContent("bus.png");
    addKeyTime( 0, 164, 210, content);
    addKeyTime( 1, 310, 255, null);
    addKeyTime( 2, 314, 234, null);
    addKeyTime( 3, 401, 231, null);
    addKeyTime( 4, 419, 269, null);
    addKeyTime( 5, 353, 340, null);
    addKeyTime( 6, 430, 367, null);
    addKeyTime( 7, 420, 418, null);
    addKeyTime( 8, 450, 421, null);
    addKeyTime( 9, 454, 386, null);
```

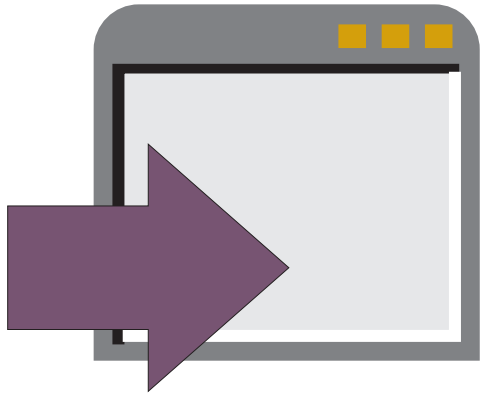


A Last Example: `BusOnRoute` (cont.)

```
    addKeyTime(10, 512, 393, null);
    addKeyTime(11, 487, 338, null);
    addKeyTime(12, 554, 323, null);
    addKeyTime(13, 500, 238, null);
    addKeyTime(14, 577, 206, null);
    addKeyTime(15, 632, 155, null);
    addKeyTime(16, 480, 151, null);
    addKeyTime(19, 301,  88, null);
    addKeyTime(21, 233, 149, null);
    addKeyTime(22, 147, 181, null);
    addKeyTime(30, 164, 210, null);
    setEndState(REMAIN);
  }

  private void addKeyTime(int time, int x, int y,
      Content content)
  {
    addKeyTime(time*1000, new Point2D.Double((double)x, (double)y),
        null, new Double(1.0), content);
  }
}
```

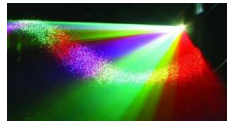


BusOnRoute – Demonstration



In examples/chapter:

```
java -cp multimedia2.jar:examples.jar BusOnRouteDemo
```



Tweening the Visual Content

- Sampled:

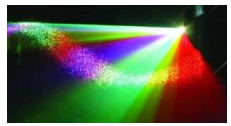
  Specify a raster for each key frame.

  Tweening from one to the next.

- Described:

  Specifying a shape (or shapes) for each key frame.

  Tweening from one to the next.

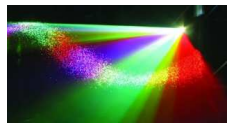


Tweening Sampled Static Content

- Use an object that has two component `statik.Content` objects.

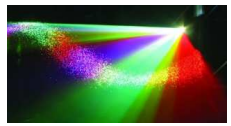
  `statik.CompositeContent` provides this capability but it a little easier to use a simple (i.e., non-hierarchical) collection.

- Combine them with alpha blending.



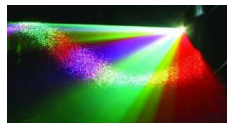
sampled.AggregateContent – Structure

```
package visual.statik.sampled;

import java.awt.*;
import java.awt.image.*;
import java.util.Iterator;

public class       AggregateContent
       extends     visual.statik.AbstractAggregateContent<Content>
       implements TransformableContent
{
  public AggregateContent()
  {
    super();
  }
}
```



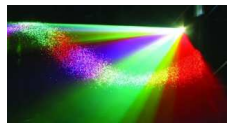
sampled.AggregateContent – setBufferedImageOp()

```
public void setBufferedImageOp(BufferedImageOp op)
{
  Iterator<Content>  i;

  i = iterator();
  while (i.hasNext())
  {
    i.next().setBufferedImageOp(op);
  }
}
```



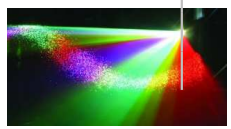
sampled.AggregateContent – setComposite()

```
public void setComposite(Composite c)
{
  Content     content;

  content = components.getLast();
  content.setComposite(c);
}
```



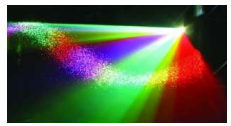
SampledSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.ArrayList;

import visual.statik.sampled.AggregateContent;
import visual.statik.sampled.Content;

public class SampledSprite extends TweeningSprite
{
  protected   AggregateContent      tweened;
  protected   ArrayList<Content>    content;

  public SampledSprite()
  {
    super();

    content = new ArrayList<Content>();
  }

  public void addKeyTime(int keyTime, Point2D location,
      Double rotation, Double scaling, Content c)
  {
    int         index;
```



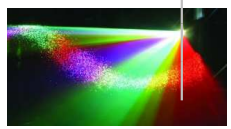
SampledSprite – Structure (cont.)

```
    index = super.addKeyTime(keyTime, location, rotation, scaling);

    if (index >= 0)
    {
      // If c is null then re-use the last Content
      if (c==null) c = content.get(index-1);

      content.add(index, c);
    }
  }
}
```



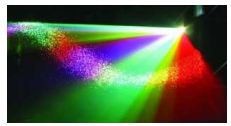
SampledSprite – getContent()

```
protected visual.statik.TransformableContent getContent()
{
  AggregateContent      aggregate;
  Content               currentContent, nextContent;
  float                 alpha;
  int                   current, next;
  visual.statik.TransformableContent   result;

  result  = null;
  current = getKeyTimeIndex();
  next    = getNextKeyTimeIndex();

  if (visible && (current >= 0))
  {
    currentContent = content.get(current);
    nextContent    = content.get(next);

    if ((nextContent != null) &&
        (currentContent != nextContent))
    {
      aggregate = new AggregateContent();
      aggregate.add(currentContent);
      aggregate.add(nextContent);

      // Setup alpha blending
      alpha = (float)getInterpolationFraction();

      aggregate.setComposite(
```



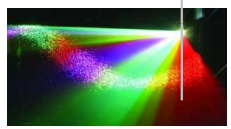
SampledSprite – getContent() (cont.)

```
            AlphaComposite.getInstance(
                AlphaComposite.SRC_OVER,
                alpha));

        result = aggregate;
      }
      else
      {
        result = currentContent;
      }
    }
    return result;
  }
```



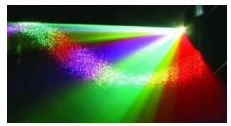
A Crystal Ball

```
import java.awt.geom.Point2D;

import io.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class CrystalBall extends SampledSprite
{
  public CrystalBall()
  {
    super();

    ResourceFinder finder  = ResourceFinder.createInstance(new resources.Marker());
    ContentFactory factory = new ContentFactory(finder);

    Content content = factory.createContent("crystalball01.png");
    addKeyTime(  500,   0.0, 350.0, -0.75, content);
    addKeyTime( 4000, 100.0, 200.0, -0.30, null);

    content = factory.createContent("crystalball02.png");
    addKeyTime( 7500, 200.0,  50.0,  0.00, content);

    setEndState(REMAIN);
  }

  private void addKeyTime(int time, double x, double y,
      double r, Content content)
  {
```

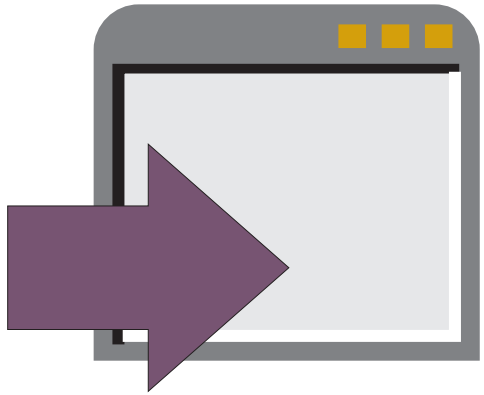


A Crystal Ball (cont.)

```
    addKeyTime(time, new Point2D.Double(x, y), new Double(r),
        new Double(1.0), content);
  }
}
```

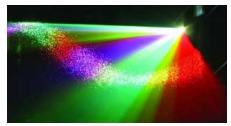


A Crystal Ball – Demonstration

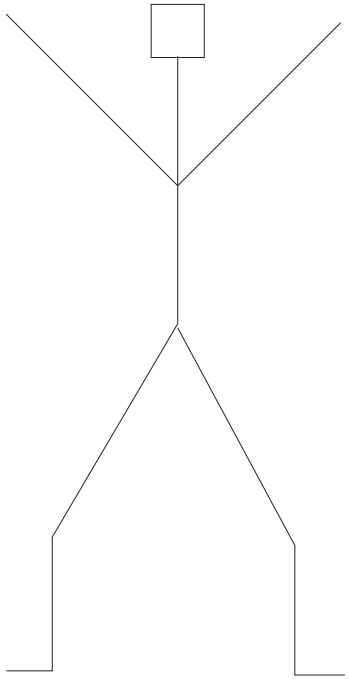


In `examples/chapter`:

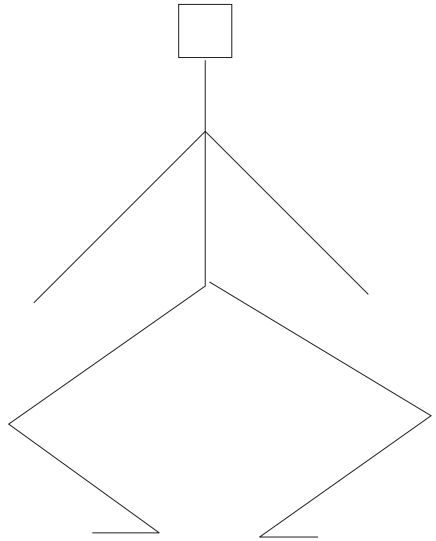
```
java -cp multimedia2.jar:examples.jar CrystalBallDemo
```



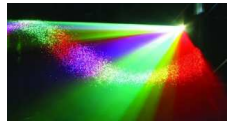
Shape Tweening



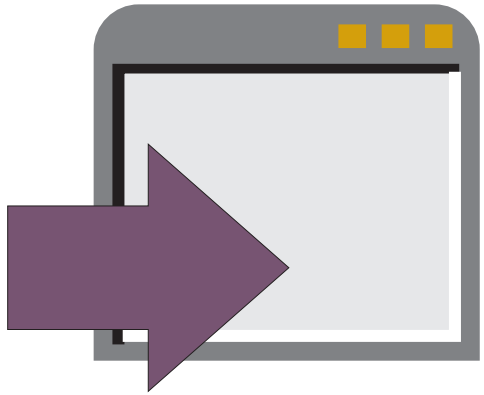
Shape
at Time t_1



Shape
at Time t_2

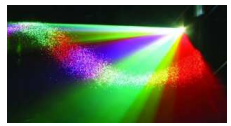


Shape Tweening – Demonstration



In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar BuzzyJumpingDemo
```



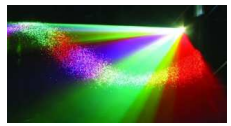
Shape Tweening (cont.)

- Most Common Approach:

  Tween the location of each of the points that defines the shape.

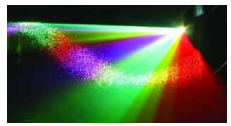
- A Helpful Participant:

  The `PathIterator` interface which provides access to "move to" and "draw to" segments.



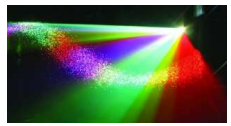
described.Content – PathIterator

```
public PathIterator getPathIterator(boolean transformed)
{
  if (transformed)
    return transformedShape.getPathIterator(IDENTITY);
  else
    return originalShape.getPathIterator(IDENTITY);
}
```



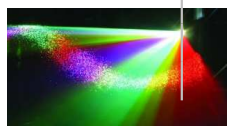
described.AggregateContent – Structure

```
package visual.statik.described;

import java.awt.*;
import java.util.Iterator;

public class       AggregateContent
       extends     visual.statik.AbstractAggregateContent<Content>
       implements TransformableContent
{
  public AggregateContent()
  {
    super();
  }
}
```



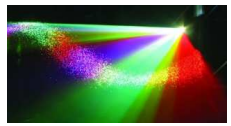
described.AggregateContent – Setters

```
public void setColor(Color color)
{
  Iterator<Content>  i;

  i = iterator();
  while (i.hasNext())
  {
    i.next().setColor(color);
  }
}

public void setPaint(Paint paint)
{
  Iterator<Content>  i;

  i = iterator();
  while (i.hasNext())
  {
    i.next().setPaint(paint);
  }
}

public void setStroke(Stroke stroke)
{
  Iterator<Content>  i;

  i = iterator();
  while (i.hasNext())
```



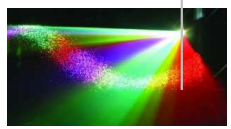
described.AggregateContent – Setters (cont.)

```
    {
      i.next().setStroke(stroke);
    }
  }
```



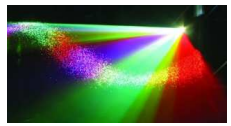
DescribedSprite – Structure

```
package visual.dynamic.described;

import java.awt.*;
import java.awt.geom.*;
import java.util.ArrayList;
import java.util.Iterator;

import visual.statik.described.*;

public class DescribedSprite extends TweeningSprite
{
  protected   AggregateContent              tweened;
  protected   ArrayList<AggregateContent>   content;

  public DescribedSprite()
  {
    content = new ArrayList<AggregateContent>();
    tweened = new AggregateContent();
  }

  public void addKeyTime(int keyTime, Point2D location,
      Double rotation, Double scaling,
      AggregateContent ctc)
  {
    int         index;
```



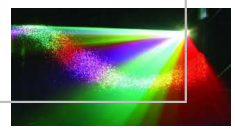
DescribedSprite – Structure (cont.)

```
    index = super.addKeyTime(keyTime, location, rotation, scaling);

    if (index >= 0)
    {
      // If ctc is null then re-use the last CompositeContent
      if (ctc == null) ctc = content.get(index-1);

      content.add(index, ctc);
    }

  }
}
```



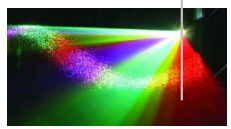
DescribedSprite – getContent()

```
public visual.statik.TransformableContent getContent()
{
  int                    current, next;
  AggregateContent       currentCTC, nextCTC;


  current = getKeyTimeIndex();
  next    = getNextKeyTimeIndex();

  tweened = null;

  if (current >= 0)
  {
    currentCTC = content.get(current);
    nextCTC    = content.get(next);

    tweened    = currentCTC;

    if (currentCTC != nextCTC)
    {
      tweenShape(currentCTC, nextCTC, getInterpolationFraction());
    }
  }

  return tweened;
}
```



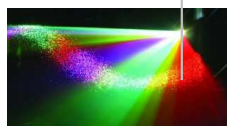
DescribedSprite – tweenShape()

```
protected void tweenShape(AggregateContent a,
    AggregateContent b,
    double frac)
{
  Color                       color;
  float[]                     coords, coordsA, coordsB;
  GeneralPath                 gp;
  int                         seg;
  Iterator<Content>           iterA, iterB;
  PathIterator                piA, piB;
  Paint                       paint;
  Content                     shapeA, shapeB;
  Stroke                      stroke;


  tweened = new AggregateContent();

  coordsA = new float[6];
  coordsB = new float[6];
  coords  = new float[6];

  iterA = a.iterator();
  iterB = b.iterator();

  // Loop over all of the TransformableContent objects
  // in the AggregateContent
  while (iterA.hasNext())
  {
```



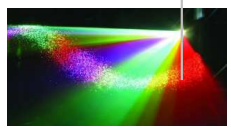
DescribedSprite – tweenShape() (cont.)

```
shapeA = iterA.next();
if (iterB.hasNext()) shapeB = iterB.next();
else                 shapeB = shapeA;

piA = shapeA.getPathIterator(false);
piB = shapeB.getPathIterator(false);

gp = new GeneralPath();
gp.setWindingRule(piA.getWindingRule());


// Loop over all of the segments in the
// TransformableContent object
while (!piA.isDone())
{
  seg = piA.currentSegment(coordsA);
  if (piB.isDone()) // Use the coordinates of the first shape
  {
    for (int i=0; i < coordsA.length; i++)
      coords[i] = coordsA[i];
  }
  else           // Interpolate the coordinates
  {
    piB.currentSegment(coordsB);

    for (int i=0; i < coordsA.length; i++)
    {
```



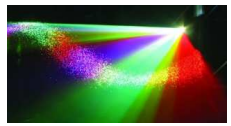
DescribedSprite – tweenShape() (cont.)

```
      coords[i] = coordsA[i] +
          (float)frac*(coordsB[i] - coordsA[i]);
    }
  }

  // Add to the General Path object
  if      (seg == PathIterator.SEG_MOVETO)
  {
    gp.moveTo(coords[0], coords[1]);
  }
  else if (seg == PathIterator.SEG_LINETO)
  {
    gp.lineTo(coords[0], coords[1]);
  }
  else if (seg == PathIterator.SEG_QUADTO)
  {
    gp.quadTo(coords[0], coords[1], coords[2], coords[3]);
  }
  else if (seg == PathIterator.SEG_CUBICTO)
  {
    gp.curveTo(coords[0], coords[1],
        coords[2], coords[3],
        coords[4], coords[5]);
  }
  else if (seg == PathIterator.SEG_CLOSE)
  {
    gp.closePath();
```



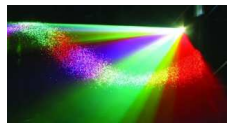
DescribedSprite – tweenShape() (cont.)

```
        }

        piA.next();
        piB.next();
      }

      paint  = shapeA.getPaint();
      color  = shapeA.getColor(); // This could also be tweened
      stroke = shapeA.getStroke();

      tweened.add(new Content(gp, color, paint, stroke));
    }
  }
```



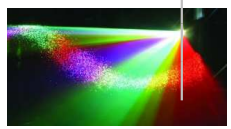
What's Next

We need to consider the complete system.



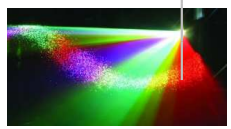
A JumboTron

```
import java.awt.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class DynamicJumboTronDemo extends JApplication
{
  public static void main(String[] args)
  {
    JApplication demo = new DynamicJumboTronDemo(args, 640, 480);
    invokeInEventDispatchThread(demo);
  }

  public DynamicJumboTronDemo(String[] args, int width, int height)
  {
    super(args, width, height);
  }

  public void init()
  {
    ResourceFinder                finder;
    ScaledVisualizationRenderer   renderer2;
    VisualizationView             view1, view2;
```



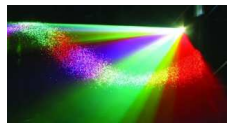
A JumboTron (cont.)

```
finder  = ResourceFinder.createInstance(new resources.Marker());
ContentFactory factory = new ContentFactory(finder);

// The Stage for Buzzy
Stage stage = new Stage(10);
stage.setBackground(Color.white);
stage.setRestartTime(7000);
view1 = stage.getView();
view1.setBounds(0,0,640,480);

renderer2 = new ScaledVisualizationRenderer(
    new PlainVisualizationRenderer(), 640.0, 480.0);
view2 = new VisualizationView(stage, renderer2);
view2.setBounds(50,50,160,120);
stage.addView(view2);

Content mars = factory.createContent("mars.png");
stage.add(mars);

// Buzzy
BuzzyOnMars buzzy = new BuzzyOnMars();
stage.add(buzzy);

// The content pane
JPanel contentPane = (JPanel)getContentPane();
contentPane.add(view2);
contentPane.add(view1);
```

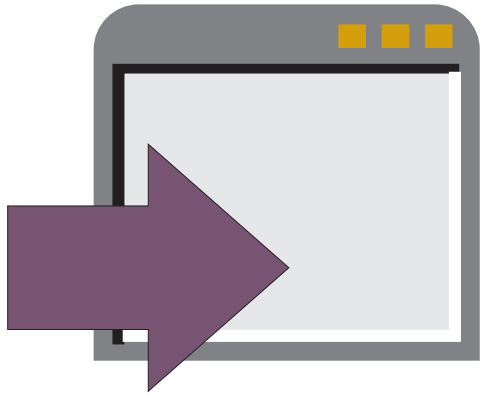


A JumboTron (cont.)

```
    stage.start();
  }
}
```

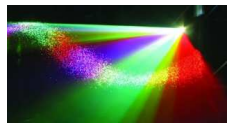


A JumboTron – Demonstration



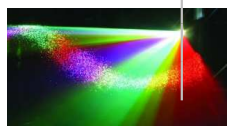
In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar DynamicJumboTronDemo
```



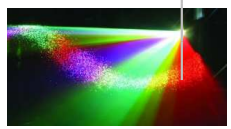
Picture-in-Picture

```
import java.awt.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class DynamicPIPDemo extends JApplication
{
  public static void main(String[] args)
  {
    JApplication demo = new  DynamicPIPDemo(args, 640, 480);
    invokeInEventDispatchThread(demo);
  }

  public  DynamicPIPDemo(String[] args, int width, int height)
  {
    super(args, width, height);
  }

  public void init()
  {
    ResourceFinder                finder;
    Stage                         stage1, stage2;
    VisualizationView             view1, view2;
```



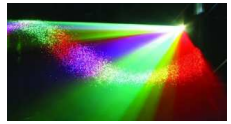
Picture-in-Picture (cont.)

```
finder  = ResourceFinder.createInstance(new resources.Marker());
ContentFactory factory = new ContentFactory(finder);

// The Stage for Buzzy
stage1 = new Stage(10);
stage1.setBackground(Color.white);
stage1.setRestartTime(7000);
view1 = stage1.getView();
view1.setRenderer(new ScaledVisualizationRenderer(
    view1.getRenderer(),
    640.0, 480.0));
view1.setBounds(0,0,640,480);

Content mars = factory.createContent("mars.png");
stage1.add(mars);

// Buzzy
BuzzyOnMars buzzy = new BuzzyOnMars();
stage1.add(buzzy);

// The stage for the airplane
stage2 = new Stage(10);
view2 = stage2.getView();
view2.setRenderer(new ScaledVisualizationRenderer(
    view2.getRenderer(),
    640.0, 480.0));
view2.setBounds(50,50,160,120);
```

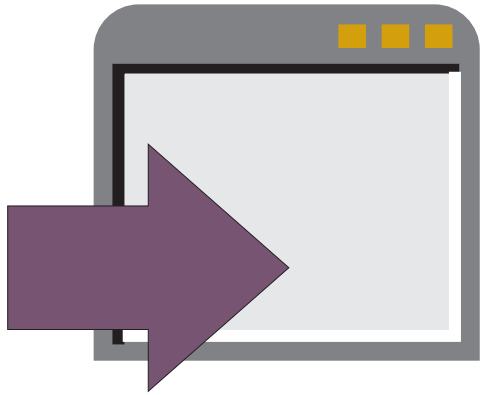


Picture-in-Picture (cont.)

```
    view2.setSize(160,120);
    view2.setBackground(Color.white);
    stage2.setRestartTime(12000);

    // The Airplane
    Airplane plane = new Airplane();
    stage2.add(plane);

    // The content pane
    JPanel contentPane = (JPanel)getContentPane();
    contentPane.add(view2);
    contentPane.add(view1);

    stage1.start();
    stage2.start();
  }
}
```

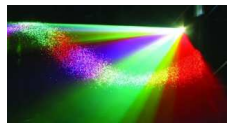


Picture-in-Picture – Demonstration



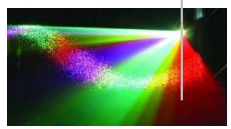
In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar DynamicPIPDemo
```



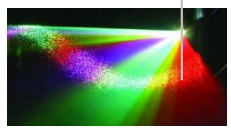
A Diptych

```
import java.awt.*;
import javax.swing.*;

import app.*;
import io.*;
import visual.*;
import visual.dynamic.described.*;
import visual.statik.sampled.*;

public class DynamicDiptychDemo extends JApplication
{
  public static void main(String[] args)
  {
    JApplication demo = new DynamicDiptychDemo(args, 320, 480);
    invokeInEventDispatchThread(demo);
  }

  public DynamicDiptychDemo(String[] args, int width, int height)
  {
    super(args, width, height);
  }

  public void init()
    {
       ContentFactory                factory;
       ResourceFinder                finder;
       VisualizationRenderer         renderer2;
```



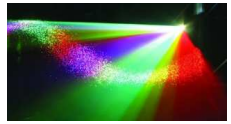
A Diptych (cont.)

```
// The Stage for Buzzy
Stage stage = new Stage(10);
stage.setBackground(Color.WHITE);
stage.setRestartTime(7000);
VisualizationView view1 = stage.getView();
view1.setRenderer(new PartialVisualizationRenderer(
                          view1.getRenderer(),
                          0.0, 0.0));
view1.setBounds(0,0,320,480);

renderer2 = new PartialVisualizationRenderer(
                 new PlainVisualizationRenderer(), 320.0, 0.0);
VisualizationView view2 = new VisualizationView(stage, renderer2);
view2.setBounds(0,0,320,480);
stage.addView(view2);

finder = ResourceFinder.createInstance(new resources.Marker());
factory = new ContentFactory(finder);

Content mars = factory.createContent("mars.png");
stage.add(mars);

// Buzzy
BuzzyOnMars buzzy = new BuzzyOnMars();
stage.add(buzzy);

// The content pane for the main window
```

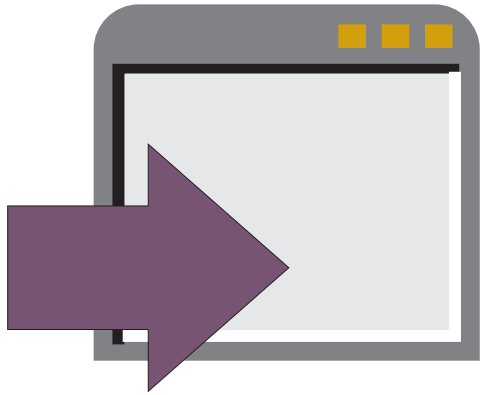


A Diptych (cont.)

```
        JPanel contentPane = (JPanel)getContentPane();
        contentPane.add(view1);

        // The content pane for the other window
        JFrame window2 = new JFrame();
        window2.setSize(320,480);
        window2.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        contentPane = (JPanel)window2.getContentPane();
        contentPane.add(view2);
        window2.setVisible(true);

        stage.start();
    }
}
```

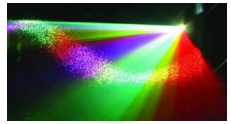


A Diptych – Demonstration



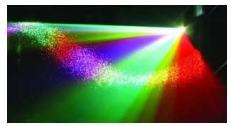
In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar DynamicDiptychDemo
```



What's Next

We need to consider other interesting things we can do (that are no in the textbook).



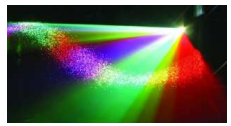
Adding Special Effects to Sampled Dynamic Visual Content

- The Objective:

  Add “sprites” to a “movie”.

- What’s Needed?:

  What’s Needed?



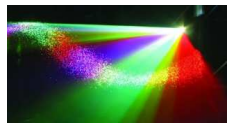
Adding Special Effects to Sampled Dynamic Visual Content

- The Objective:

  Add "sprites" to a "movie".

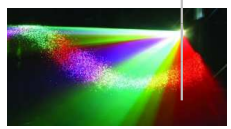
- What's Needed?:

  The `Screen` object's `Visualization` and the `Stage` need to render to the same `VisualizationView`.



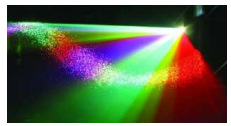
SpecialEffectsRenderer

```
package visual.dynamic;

import java.awt.*;

import visual.*;

public class      SpecialEffectsRenderer
       implements VisualizationRenderer
{
  protected Visualization           stage;
  protected VisualizationRenderer   decorated;

  public SpecialEffectsRenderer(
      VisualizationRenderer decorated,
      Visualization         stage)
  {
    this.decorated = decorated;
    this.stage     = stage;
  }

  public void postRendering(Graphics          g,
      Visualization     model,
      VisualizationView view)
  {
    decorated.postRendering(g, model, view);
  }

  public void preRendering(
```



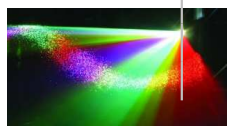
SpecialEffectsRenderer (cont.)

```
      Graphics           g,
      Visualization      model,
      VisualizationView  view)
  {
    decorated.preRendering(g, model, view);
  }

  public void render(
      Graphics           g,
      Visualization      model,
      VisualizationView  view)
  {
    decorated.render(g, model, view);
    decorated.render(g, stage, view);
  }
}
```



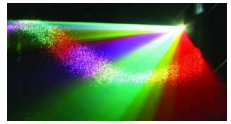
SpecialEffectsScreen

```
package visual.dynamic;

import visual.*;
import visual.dynamic.described.*;
import visual.dynamic.sampled.*;

public class SpecialEffectsScreen extends Screen
{
  SpecialEffectsRenderer    renderer;
  Visualization             stage;

  public SpecialEffectsScreen()
  {
    super();
    stage.setView(getView());
  }

  public SpecialEffectsScreen(int frameRate)
  {
    super(frameRate);
    stage.setView(getView());
  }

  public void add(Sprite sprite)
  {
    // Make the Sprite a MetronomeListener
    metronome.addListener(sprite);
```



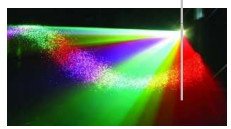
SpecialEffectsScreen (cont.)

```
    // Treat the Sprite as a SimpleContent and
    // add it to the Visualization
    stage.add(sprite);
  }

  protected VisualizationView createDefaultView()
  {
    stage = new Visualization();

    renderer = new SpecialEffectsRenderer(
        new ScreenRenderer(
            new PlainVisualizationRenderer()),
        stage);

    return new VisualizationView(this, renderer);
  }
}
```



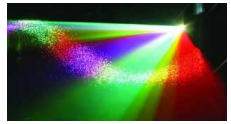
A Special Effect - A Bee

```
import java.awt.geom.*;

import io.*;
import visual.dynamic.described.*;
import visual.dynamic.sampled.*;
import visual.statik.sampled.*;

public class Bee extends SampledSprite
{
  public Bee()
  {
    super();
    Content             content;
    ContentFactory      factory;
    ResourceFinder      finder;

    finder  = ResourceFinder.createInstance(new resources.Marker());
    factory = new ContentFactory(finder);
    content = factory.createContent("bee.png", 4);
    addKeyFrame(    1, 173.0, 118.0,  0.00, 0.20, content);
    addKeyFrame(   45, 166.0, 120.0,  0.00, 0.35, null);
    addKeyFrame(  100, 148.0, 105.0,  0.00, 0.50, null);
    addKeyFrame(  115, 230.0,  90.0,  0.00, 0.75, null);
    addKeyFrame(  200, 245.0, 143.0,  0.00, 1.00, null);

    setEndState(REMOVE);
  }
```



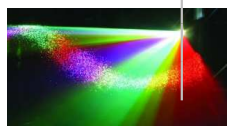
A Special Effect - A Bee (cont.)

```
  private void addKeyFrame(int frame, double x, double y,
      double r, double s, Content c)
  {
    int     time;

    time = frame * Screen.DEFAULT_FRAME_DELAY;

    addKeyTime(time, new Point2D.Double(x, y), new Double(r),
        new Double(s), c);
  }
}
```



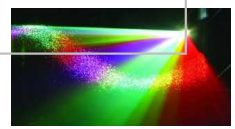
An Example

```
import javax.swing.*;

import app.*;
import visual.*;
import io.ResourceFinder;
import visual.dynamic.*;
import visual.statik.*;
import visual.statik.sampled.*;

public class SpecialEffectsDemo extends JApplication
{
  public static void main(String[] args)
  {
    JApplication demo = new SpecialEffectsDemo(args, 640, 480);
    invokeInEventDispatchThread(demo);
  }

  public SpecialEffectsDemo(String[] args, int width, int height)
  {
    super(args, width, height);
  }



  public void init()
  {
    ResourceFinder                    finder;
```

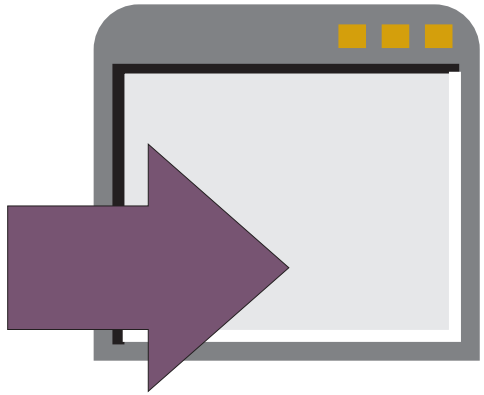


An Example (cont.)

```
    SpecialEffectsScreen screen = new SpecialEffectsScreen(20);
    screen.setRepeating(true);

    VisualizationView view = screen.getView();
    view.setBounds(0,0,320,240);

    JPanel contentPane = (JPanel)getContentPane();
    contentPane.add(view);

    finder = ResourceFinder.createInstance(new resources.Marker());

    String[] names = finder.loadResourceNames("scribble.txt");
    ContentFactory factory = new ContentFactory(finder);
    SimpleContent[] frames = factory.createContents(names, 4);

    for (int i=0; i<frames.length; i++)
    {
      screen.add(frames[i]);
    }

    Bee bee = new Bee();
    screen.add(bee);

    screen.start();
  }
}
```

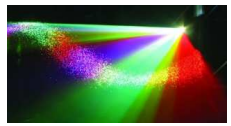


Special Effects – Demonstration

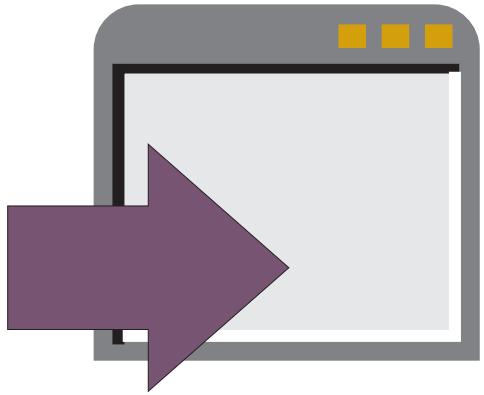


In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar SpecialEffectsDemo -Xmx256m
```



Putting it All Together – Demonstration



In `examples/chapter`:

```
java -cp multimedia2.jar:examples.jar SpecialEffectsPIPDemo -Xmx256m
```