Chapter 7

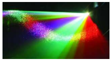
A Static Visual Content System

The Design and Implementation of Multimedia Software

David Bernstein

Jones and Bartlett Publishers

www.jbpub.com



About this Chapter

- Consider a comic book character that has one `BufferedImage` as its head and another as its body:

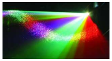
  One could create two `sampled.Content` objects and put them in one `Visualization`.

  But the character could not be treated as a coherent whole.

- Consider an illustration of a house using described content in which the door and roof must be different colors:

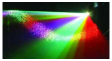
  One could create a `described.Content` object for each part of the house and add them all to a single `Visualization`.

  But the house could not be treated as a coherent whole.



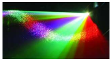
Requirements

F7.1 Support visual content that has multiple component parts.



What's Next

We need to consider some high-level design alternatives.

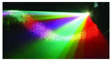


A Top-Down Approach

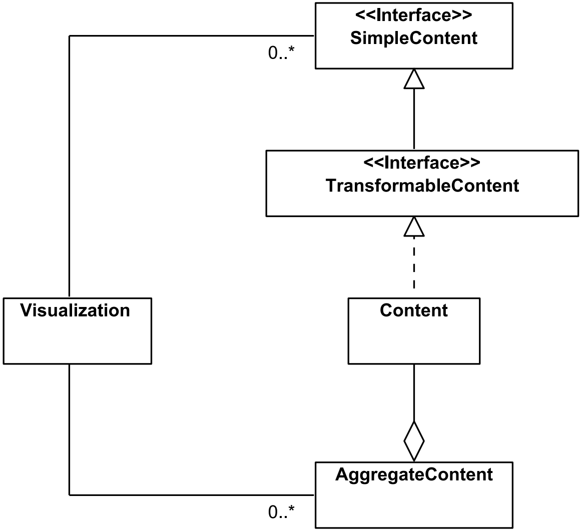
1. Consider design alternatives at a high level of abstraction.

   Ignore the differences between `statik.sampled.Content` and `statik.described.Content` objects and consider only `statik.Content` objects.

2. Add the details needed to support the differences between `statik.sampled.Content` and `statik.described.Content`.

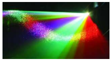


Alternative 1



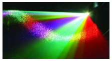
Alternative 1 - Shortcomings

What are the shortcomings?



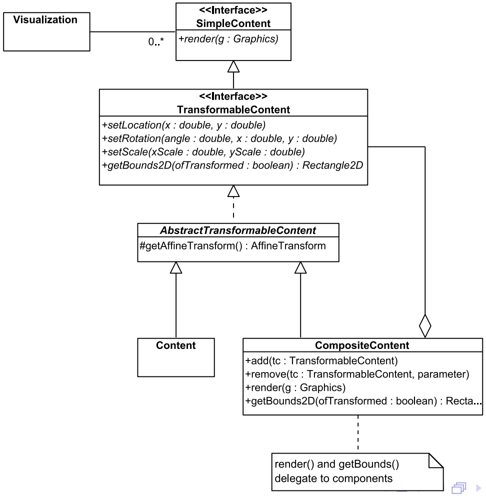
Alternative 1 - Shortcomings

Both the `Visualization` and `VisualizationView` classes must now distinguish between `Content` objects and `AggregateContent` objects making them more complicated, more repetitive, and less cohesive.



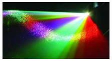
Alternative 2

Use the Composite Pattern


Visualization
0..*
<<Interface>>
SimpleContent
+render(g : Graphics)
<<Interface>>
TransformableContent
+setLocation(x : double, y : double)
+setRotation(angle : double, x : double, y : double)
+setScale(xScale : double, yScale : double)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
AbstractTransformableContent
#getAffineTransform() : AffineTransform
Content
CompositeContent
+add(tc : TransformableContent)
+remove(tc : TransformableContent, parameter)
+render(g : Graphics)
+getBounds2D(ofTransformed : boolean) : Recta...
render() and getBounds()
delegate to components

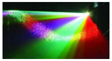
Alternative 2 - Advantages

- Allows both the `Visualization` and `VisualizationView` classes to treat `SimpleContent` objects and `CompositeContent` objects in exactly the same way.
- Enables the creation of content that is very complicated.



What's Next

We need to consider some lower-level design alternatives.



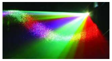
Alternative 2.1

- Approach:

  Make three 'copies' of the design, one for `statik.sampled.Content`, one for `statik.described.Content`, and one for either/both.

- Shortcomings:

  What are the shortcomings?



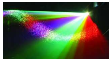
Alternative 2.1

- Approach:

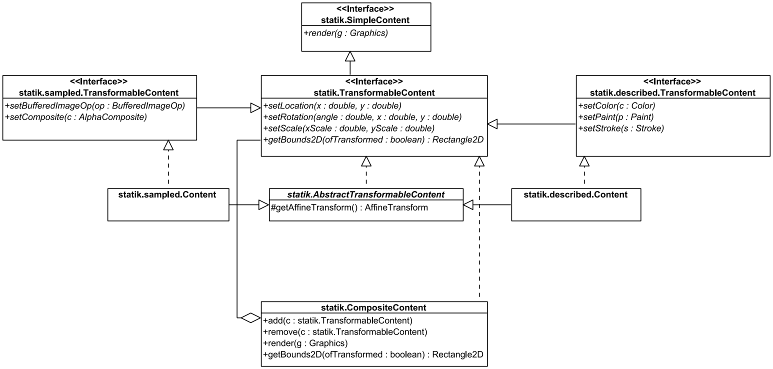
  Make three 'copies' of the design, one for `statik.sampled.Content`, one for `statik.described.Content`, and one for either/both.

- Shortcomings:

  The `statik.sampled.CompositeContent`, `statik.described.CompositeContent`, and `statik.CompositeContent` classes all must have `add()`, `remove()`, `render()`, and `getBounds2D()` methods that contain (virtually) identical code.



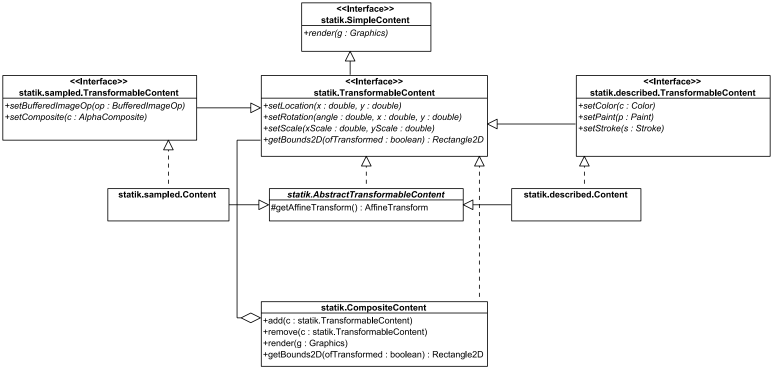
Alternative 2.2



<<Interface>>
statik.SimpleContent
+render(g : Graphics)
<<Interface>>
statik.sampled.TransformableContent
+setBufferedImageOp(op : BufferedImageOp)
+setComposite(c : AlphaComposite)
<<Interface>>
statik.TransformableContent
+setLocation(x : double, y : double)
+setRotation(angle : double, x : double, y : double)
+setScale(xScale : double, yScale : double)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
<<Interface>>
statik.described.TransformableContent
+setColor(c : Color)
+setPaint(p : Paint)
+setStroke(s : Stroke)
statik.sampled.Content
statik.AbstractTransformableContent
#getAffineTransform() : AffineTransform
statik.described.Content
statik.CompositeContent
+add(c : statik.TransformableContent)
+remove(c : statik.TransformableContent)
+render(g : Graphics)
+getBounds2D(ofTransformed : boolean) : Rectangle2D

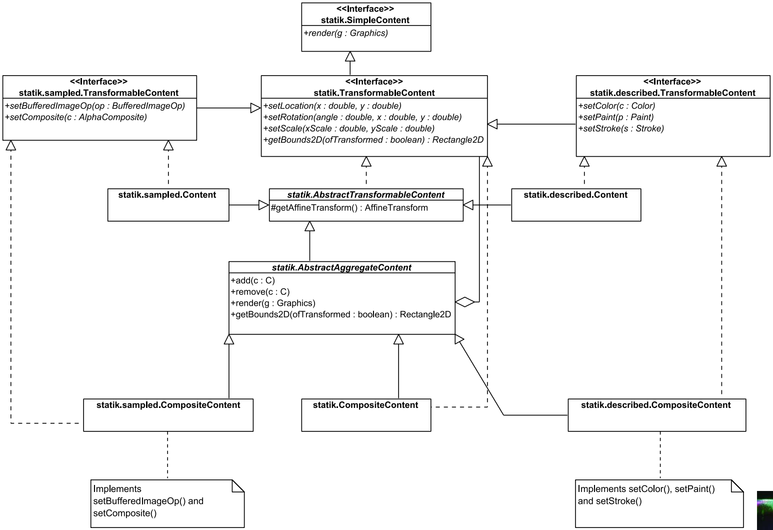
What are the shortcomings?

Alternative 2.2

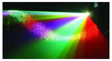

<<Interface>>
statik.SimpleContent
+render(g : Graphics)
<<Interface>>
statik.sampled.TransformableContent
+setBufferedImageOp(op : BufferedImageOp)
+setComposite(c : AlphaComposite)
<<Interface>>
statik.TransformableContent
+setLocation(x : double, y : double)
+setRotation(angle : double, x : double, y : double)
+setScale(xScale : double, yScale : double)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
<<Interface>>
statik.described.TransformableContent
+setColor(c : Color)
+setPaint(p : Paint)
+setStroke(s : Stroke)
statik.sampled.Content
statik.AbstractTransformableContent
#getAffineTransform() : AffineTransform
statik.described.Content
statik.CompositeContent
+add(c : statik.TransformableContent)
+remove(c : statik.TransformableContent)
+render(g : Graphics)
+getBounds2D(ofTransformed : boolean) : Rectangle2D

It doesn’t allow operations that are either specific to `statik.sampled.Content` or `statik.described.Content` objects to be applied to `statik.CompositeContent` objects.

Alternative 2.3



<<Interface>>
statik.SimpleContent
+render(g : Graphics)
<<Interface>>
statik.sampled.TransformableContent
+setBufferedImageOp(op : BufferedImageOp)
+setComposite(c : AlphaComposite)
<<Interface>>
statik.TransformableContent
+setLocation(x : double, y : double)
+setRotation(angle : double, x : double, y : double)
+setScale(xScale : double, yScale : double)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
<<Interface>>
statik.described.TransformableContent
+setColor(c : Color)
+setPaint(p : Paint)
+setStroke(s : Stroke)
statik.sampled.Content
statik.AbstractTransformableContent
#getAffineTransform() : AffineTransform
statik.described.Content
statik.AbstractAggregateContent
+add(c : C)
+remove(c : C)
+render(g : Graphics)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
statik.sampled.CompositeContent
statik.CompositeContent
statik.described.CompositeContent
Implements setBufferedImageOp() and setComposite()
Implements setColor(), setPaint() and setStroke()



Alternative 2.3 - Advantages and Disadvantages

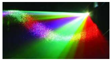
- Advantage:

  Allows for operations that are specific to `statik.sampled.Content` and `statik.described.Content`

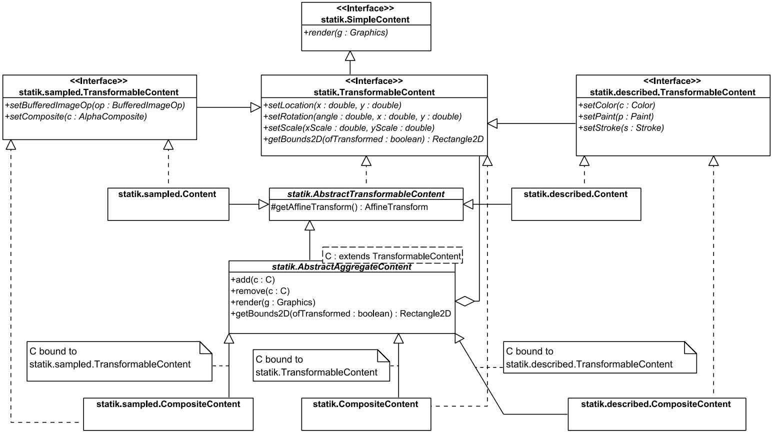
  Eliminates code duplication by moving the common code to the abstract parent.

- Disadvantage:

  Since the abstract parent manages the collection, the collection must contain `statik.TransformableContent` objects, rather than the more specific `statik.sampled.Content` and `statik.described.Content` objects.



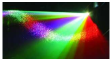
Alternative 2.4



<<Interface>>
statik.SimpleContent
+render(g : Graphics)
<<Interface>>
statik.sampled.TransformableContent
+setBufferedImageOp(op : BufferedImageOp)
+setComposite(c : AlphaComposite)
<<Interface>>
statik.TransformableContent
+setLocation(x : double, y : double)
+setRotation(angle : double, x : double, y : double)
+setScale(xScale : double, yScale : double)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
<<Interface>>
statik.described.TransformableContent
+setColor(c : Color)
+setPaint(p : Paint)
+setStroke(s : Stroke)
statik.sampled.Content
statik.AbstractTransformableContent
#getAffineTransform() : AffineTransform
statik.described.Content
C : extends TransformableContent
statik.AbstractAggregateContent
+add(c : C)
+remove(c : C)
+render(g : Graphics)
+getBounds2D(ofTransformed : boolean) : Rectangle2D
C bound to statik.sampled.TransformableContent
C bound to statik.TransformableContent
C bound to statik.described.TransformableContent
statik.sampled.CompositeContent
statik.CompositeContent
statik.described.CompositeContent

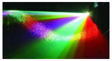
AbstractAggregateContent - Structure

```
package visual.statik;

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

public abstract class AbstractAggregateContent<C extends TransformableContent>
                extends  AbstractTransformableContent
{
  // A LinkedList is used to order the components.
  // Since we don't expect many components to be removed
  // this doesn't raise efficiency concerns.
  //
  // Alternatively, we could have a z-order.
  protected LinkedList<C>        components;

  public AbstractAggregateContent()
  {
    super();
    components = new LinkedList<C>();
  }
}
```



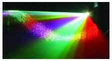
AbstractAggregateContent - Content Management

```
public void add(C component)
{
  components.add(component);
}

public Iterator<C> iterator()
{
  return components.iterator();
}

public void remove(C component)
{
  components.remove(component);
}
```



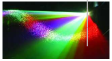
AbstractAggregateContent - Rendering

```
public void render(Graphics g)
{
  Iterator<C>   i;
  C             component;

  i = components.iterator();
  while (i.hasNext())
  {
    component = i.next();

    component.setLocation(x, y);
    component.setRotation(angle, xRotation, yRotation);
    component.setScale(xScale, yScale);

    component.render(g);
  }
}
```



AbstractAggregateContent - Bounds

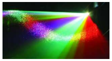
```
public Rectangle2D getBounds2D(boolean ofTransformed)
{
  double                maxX, maxY, minX, minY, rx, ry;
  Iterator<C>           i;
  Rectangle2D           bounds;
  TransformableContent  component;

  maxX = Double.NEGATIVE_INFINITY;
  maxY = Double.NEGATIVE_INFINITY;
  minX = Double.POSITIVE_INFINITY;
  minY = Double.POSITIVE_INFINITY;

  i = components.iterator();
  while (i.hasNext())
  {
    component = i.next();
    bounds    = component.getBounds2D(ofTransformed);

    if (bounds != null)
    {
      rx = bounds.getX();
      ry = bounds.getY();
      if (rx < minX) minX = rx;
      if (ry < minY) minY = ry;

      rx = bounds.getX() + bounds.getWidth();
      ry = bounds.getY() + bounds.getHeight();
      if (rx > maxX) maxX = rx;
```



AbstractAggregateContent - Bounds (cont.)

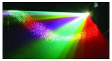
```
        if (ry > maxY) maxY = ry;
      }
    }
    // Note: We could, instead, use an object pool
    return new Rectangle2D.Double(minX, minY, maxX-minX, maxY-minY);
  }
```

For simplicity, this implementation does not store bounds, it calculates them when needed using the `getBounds2D()` method in the `TransformableContent` objects. Alternatively, this class could calculate the bounds whenever they change.



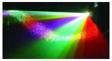
CompositeContent

```
package visual.statik;

public class      CompositeContent
       extends    AbstractAggregateContent<TransformableContent>
       implements TransformableContent
{
  public CompositeContent()
  {
    super();
  }
}
```



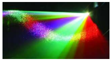
CompositeContent - Structure

```
package visual.statik.sampled;

import java.awt.Composite;
import java.awt.image.BufferedImageOp;
import java.util.Iterator;

import visual.statik.AbstractAggregateContent;

public class      CompositeContent
       extends    AbstractAggregateContent<TransformableContent>
       implements TransformableContent
{
  public CompositeContent()
  {
    super();
  }
}
```



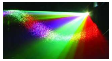
CompositeContent - setBufferedImageOp()

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

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



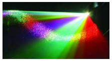
CompositeContent - setComposite()

```
public void setComposite(Composite c)
{
  Iterator<TransformableContent>  i;

  i = iterator();
  while (i.hasNext())
  {
    i.next().setComposite(c);
  }
}
```



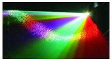
CompositeContent - Structure

```
package visual.statik.described;

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

import visual.statik.AbstractAggregateContent;

public class      CompositeContent
       extends    AbstractAggregateContent<TransformableContent>
       implements TransformableContent
{
  public CompositeContent()
  {
    super();
  }
}
```



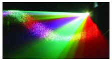
CompositeContent - setColor()

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

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



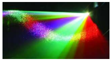
CompositeContent - setPaint()

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

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



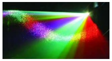
CompositeContent - setStroke()

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

  i = iterator();
  while (i.hasNext())
  {
    i.next().setStroke(stroke);
  }
}
```

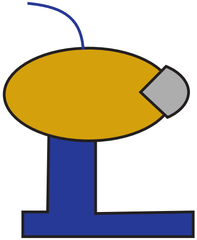


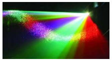
What's Next

We need to consider some examples.

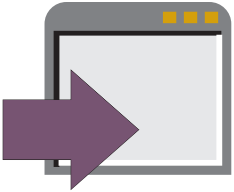


Introducing Buzzy



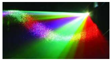


Introducing Buzzy – Demonstration



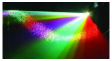
In `examples/chapter:`

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



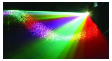
Buzzy - Structure

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

import visual.statik.described.*;

public class BoringBuzzy extends CompositeContent
{
}
```



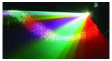
Buzzy - Constructor

```
public BoringBuzzy()
{
  super();

  Color black  = Color.BLACK;
  Color gold   = new Color(0xc2,0xa1,0x4d);
  Color gray   = new Color(0xaa,0xaa,0xaa);
  Color purple = new Color(0x45,0x00,0x84);

  BasicStroke stroke = new BasicStroke();
}
```



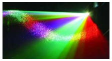
Buzzy - Body

```
    Path2D.Float bodyShape = new Path2D.Float();
    bodyShape.moveTo( 20, 50);
    bodyShape.lineTo( 20, 70);
    bodyShape.lineTo( 20, 90);
    bodyShape.lineTo( 10, 90);
    bodyShape.lineTo( 10,100);
    bodyShape.lineTo( 80,100);
    bodyShape.lineTo( 80, 90);
    bodyShape.lineTo( 40, 90);
    bodyShape.lineTo( 40, 70);
    bodyShape.lineTo( 40, 50);
    bodyShape.closePath();
    Content body = new Content(bodyShape, black, purple, stroke);
    add(body);
```



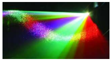
Buzzy - Head

```
    CompositeContent head = new CompositeContent();
    add(head);
```



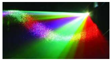
Buzzy - Hair (?)

```
    QuadCurve2D.Float hairShape = new QuadCurve2D.Float(10,2,40,10,30,25);
    Content hair = new Content(hairShape, purple, null, stroke);
    head.add(hair);
```



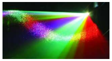
Buzzy - Helmet

```
    Arc2D.Float helmetShape = new Arc2D.Float(2,20,70,40,2,360,Arc2D.OPEN);
    Content helmet=new Content(helmetShape, black, gold, stroke);
    head.add(helmet);
```

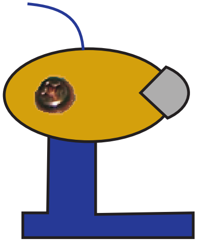


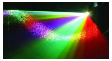
Buzzy - Visor

```
Arc2D.Float visorShape = new Arc2D.Float(40,25,35,30,315,90,Arc2D.PIE);
Content visor=new Content(visorShape, black, gray, stroke);
head.add(visor);
```



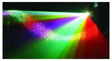
Buzzy with a Fancy Helmet





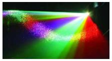
Buzzy with a Fancy Helmet - Structure

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


public class FancyBuzzy extends visual.statik.CompositeContent
{
}
```



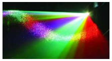
Buzzy with a Fancy Helmet - Declarations

```
    ResourceFinder                        finder;
    visual.statik.CompositeContent        head;
    visual.statik.described.Content       body, hair, helmet, visor;
    visual.statik.sampled.Content         screw;
    visual.statik.sampled.ContentFactory  factory;
```



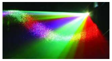
Buzzy with a Fancy Helmet - Logo

```
    finder  = ResourceFinder.createInstance(new resources.Marker());
    factory = new visual.statik.sampled.ContentFactory(finder);
    screw = factory.createContent("screw.png", 4);
```

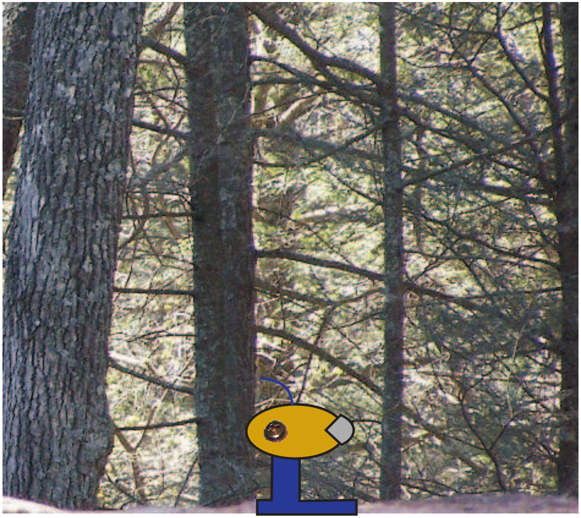


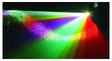
Buzzy with a Fancy Helmet - Head

```
    head = new visual.statik.CompositeContent();
    head.add(hair);
    head.add(helmet);
    head.add(screw);
    head.add(visor);

    add(head);
```



Buzzy in the Woods

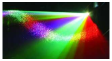




Buzzy in the Woods – Demonstration

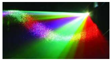
In `examples/chapter`:

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



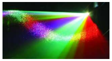
Buzzy in the Woods - The Woods

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

    Content woods  = factory.createContent("woods.png", 3);
    woods.setLocation(0,0);
    woods.setBufferedImageOp(opFactory.createBlurOp(3));
```



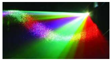
Buzzy in the Woods - Buzzy

```
    FancyBuzzy buzzy  = new FancyBuzzy();
    buzzy.setLocation(200, 318);
```



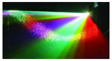
Buzzy in the Woods - The Visualization

```
    visualization = new Visualization();
    view          = visualization.getView();

    view.setBounds(0,0,471,418);
    view.setSize(471,418);

    visualization.add(woods);
    visualization.add(buzzy);
```

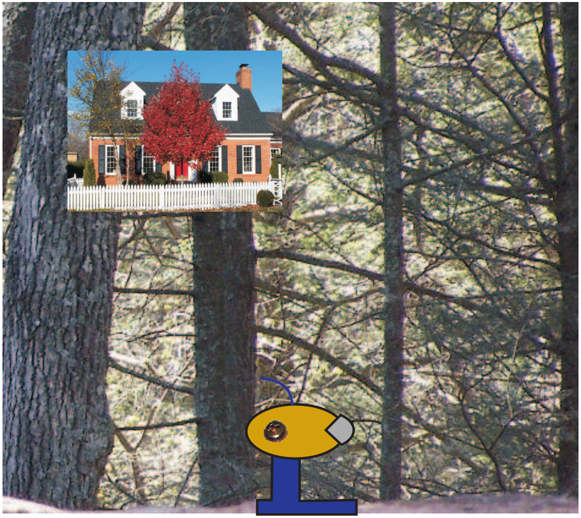


Buzzy in the Woods - The Content Pane

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



Picture-in-a-Picture



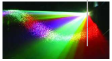
Picture-in-a-Picture – Demonstration

In `examples/chapter`:

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

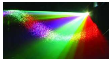
Picture-in-a-Picture - Structure

```
import javax.swing.*;

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

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

  public StaticPIPDemo(String[] args, int width, int height)
  {
    super(args, width, height);
  }
  public void init()
  {
    BufferedImageOpFactory        opFactory;
    ResourceFinder                finder;
    Visualization                 model1, model2;
    VisualizationRenderer         renderer1, renderer2;
    VisualizationView             view1, view2;

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



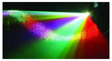
Picture-in-a-Picture - Structure (cont.)

```
    ContentFactory factory    = new ContentFactory(finder);
    opFactory = BufferedImageOpFactory.createFactory();
  }
}
```



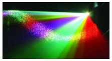
Picture-in-a-Picture - One Visualization

```
    Content woods  = factory.createContent("woods.png", 3);
    woods.setLocation(0,0);
    woods.setBufferedImageOp(opFactory.createBlurOp(3));

    FancyBuzzy buzzy  = new FancyBuzzy();
    buzzy.setLocation(200, 318);

    model1 = new Visualization();
    model1.add(woods);
    model1.add(buzzy);

    view1     = model1.getView();
    renderer1 = view1.getRenderer();
    view1.setRenderer(new ScaledVisualizationRenderer(renderer1,
        471.0,
        418.0));
    view1.setBounds(0,0,471,418);
    view1.setSize(471,418);
```



Picture-in-a-Picture - The Other Visualization

```
    Content house = factory.createContent("house.png", 3);
    house.setLocation(0,0);

    model2 = new Visualization();
    model2.add(house);

    view2     = model2.getView();
    renderer2 = view2.getRenderer();
    view2.setRenderer(new ScaledVisualizationRenderer(renderer2,
        525.0,
        375.0));
    view2.setBounds(50,50,160,120);
    view2.setSize(160,120);
```



Picture-in-a-Picture - The Content Pane

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