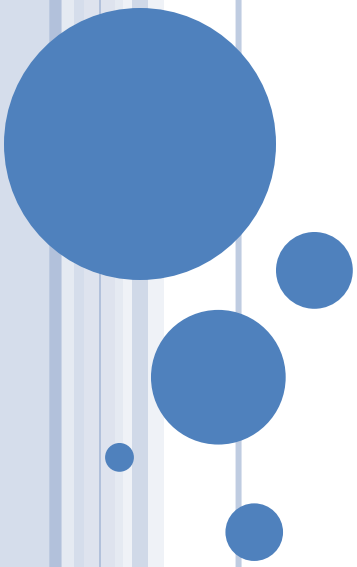


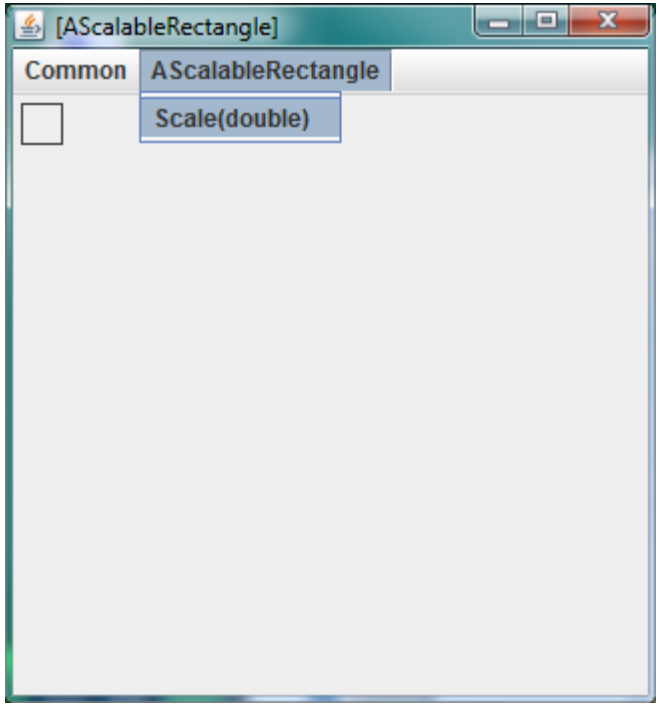
Comp 401
Composite Design Pattern

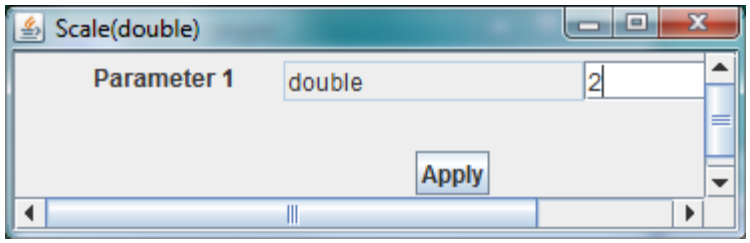
Instructor: Prasun Dewan

Prerequisite

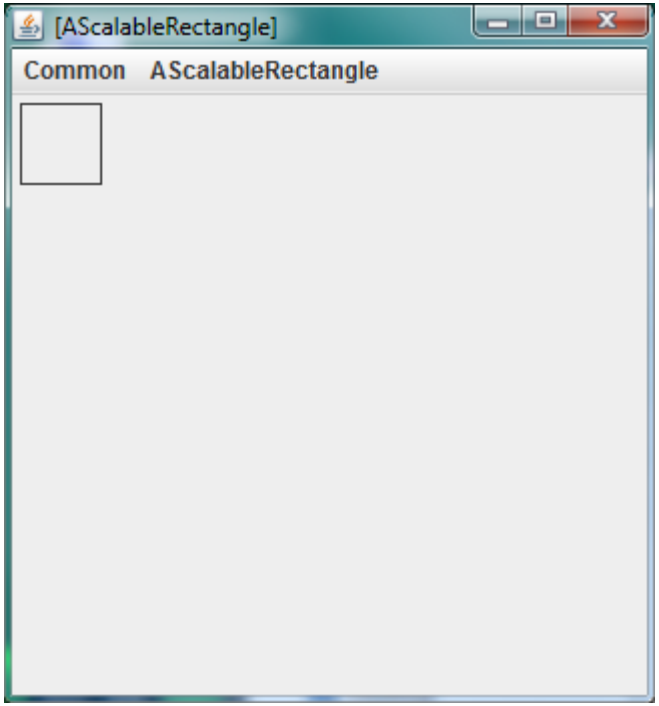
- Composite Objects Shapes
- Inheritance Multiple.
- Recursion

Scalable Rectangle





Scalable Rectangle



Scalable Shape

```
public interface ScalableShape {
  public int getX();
  public int getY();
  public int getWidth();
  public int getHeight();
  public void scale(double fraction);
}
```

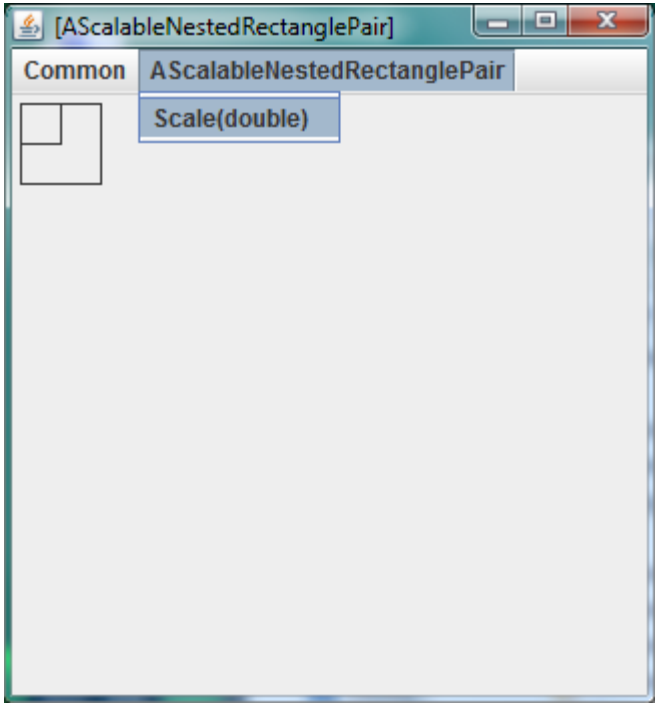
Scalable Shape Implementation

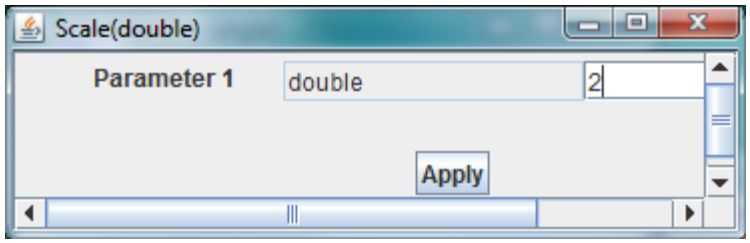
```
public class AScalableShape implements ScalableShape {
int x, y, width, height;
  public AScalableShape(int theX, int theY,
                                int theWidth, int theHeight) {
        x = theX;
        y = theY;
        width = theWidth;
        height = theHeight;
  }
  public int getX() {return x;}
  public int getY() {return y;}
  public int getWidth() {return width;}
  public int getHeight() { return height;}
  public void setHeight(int newVal) {height = newVal;}
  public void setWidth(int newVal) {width = newVal;}
  public void scale(double fraction){
        width =  (int) (width*fraction);
        height = (int) (height*fraction);
  }
}
```

Scalable Rectangle Implementation

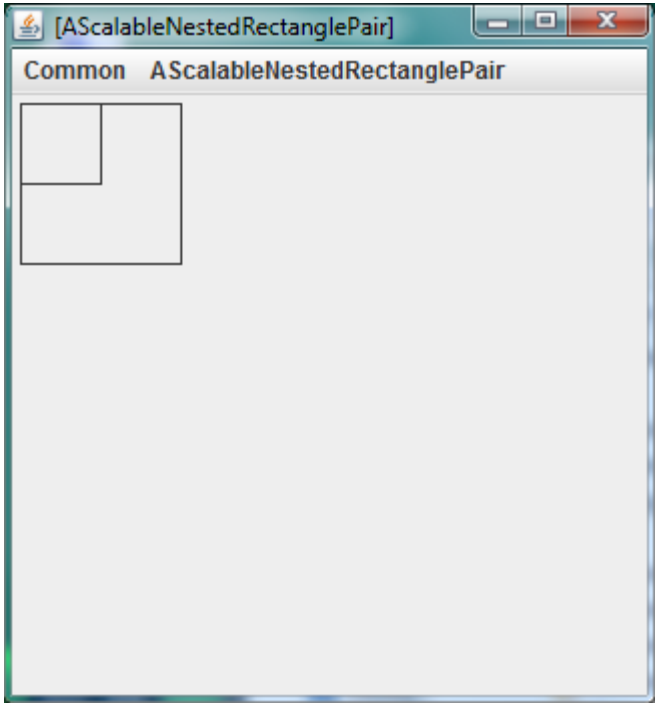
```
@StructurePattern(StructurePatternNames.RECTANGLE_PATTERN)
public class AScalableRectangle extends AScalableShape {
  public AScaleableRectangle(int theX, int theY,
                             int theWidth, int theHeight) {
    super(theX, theY, theWidth, theHeight);
  }
}
```

Scalable Nested Rectangle Pair





Scalable Nested Rectangle Pair



Nester Pair Interface

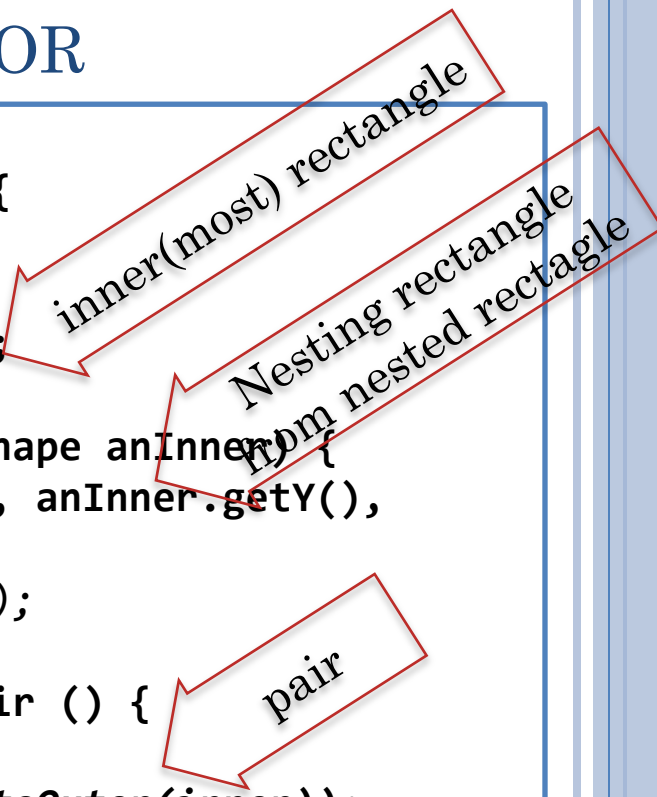
```
public interface ScalableNestedShapePair  {
  public ScalableShape getInner();
  public ScalableShape getOuter() ;
  public void scale(double fraction);
}
```

Nester Pair Implementation

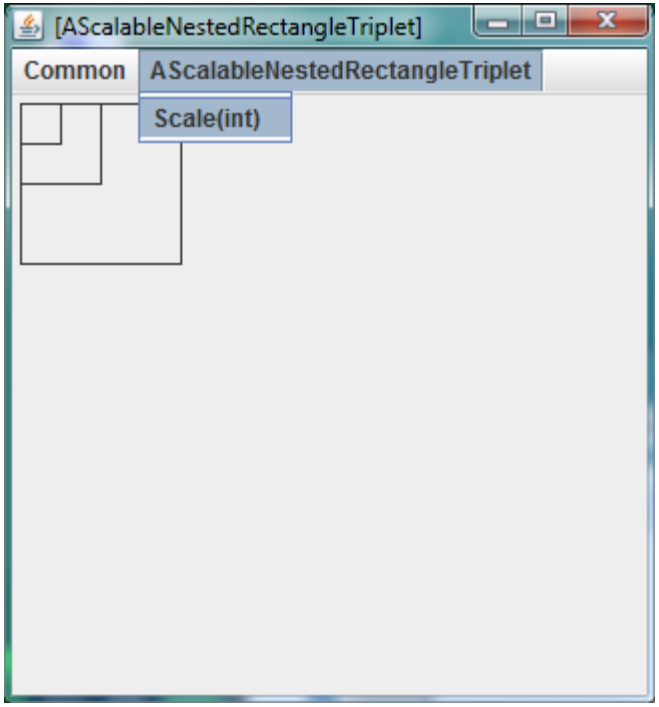
```
public class AScalableNestedShapePair
                  implements ScalableNestedShapePair {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedShapePair(ScalableShape theInner,
                                  ScalableShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

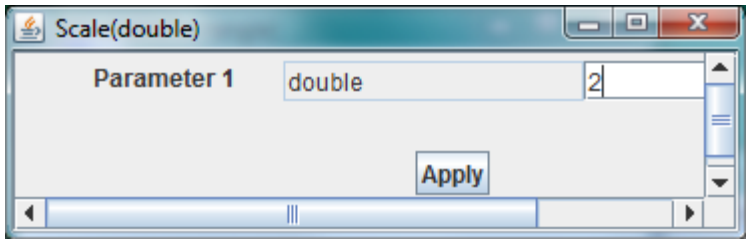
Nested Pair Creator

```
public class ScalableNestedRectanglePairCreator {
  public static final int RELATIVE_SIZE = 2;
  public static ScalableShape innermost() {
    return new AScalableRectangle (0, 0, 20, 20);
  }
  public static ScalableShape toOuter (ScalableShape anInner) {
    return new AScalableRectangle(anInner.getX(), anInner.getY(),
                anInner.getWidth()*RELATIVE_SIZE,
                anInner.getHeight()*RELATIVE_SIZE);
  }
  public static ScalableNestedShapePair createPair () {
    ScalableShape inner = innermost();
    return new AScalableNestedShapePair (inner, toOuter(inner));
  }
  public static void main (String[] args) {
    ObjectEditor.edit(createPair());
  }
}
```

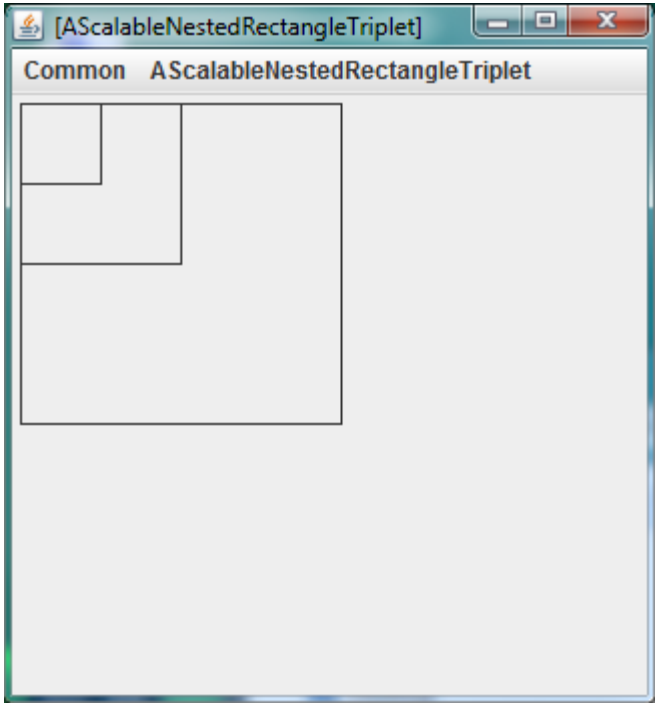


Scalable Nested Rectangle Triplet





Scalable Nested Rectangle Triplet



Triplet Interface

```
public interface ScalableNestedShapeTriplet  {
  public ScalableNestedShapePair getInner();
  public ScalableShape getOuter() ;
  public void scale(int percentage);
}
```

Reusing ScalableNestedShapePair

Triplet Implementation

```
public class AScalableNestedShapeTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedShapeTriplet(
                        ScalableNestedShapePair theInner,
                        ScalableShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Triplet Main

```
public class ScalableNestedRectangleTripletCreator
                extends ScalableNestedRectanglePairCreator {
  public static ScalableNestedShapeTriplet createTriplet () {
    ScalableNestedShapePair pair = createPair();
    return new AScalableNestedShapeTriplet(pair,
                                           toOuter(pair.getOuter()) );
  }
  public static void main (String[] args) {
    ObjectEditor.edit(createTriplet());
  }
}
```

Triplet outer from pair outer

Reusing ScalableNestedRectanglePairCreator

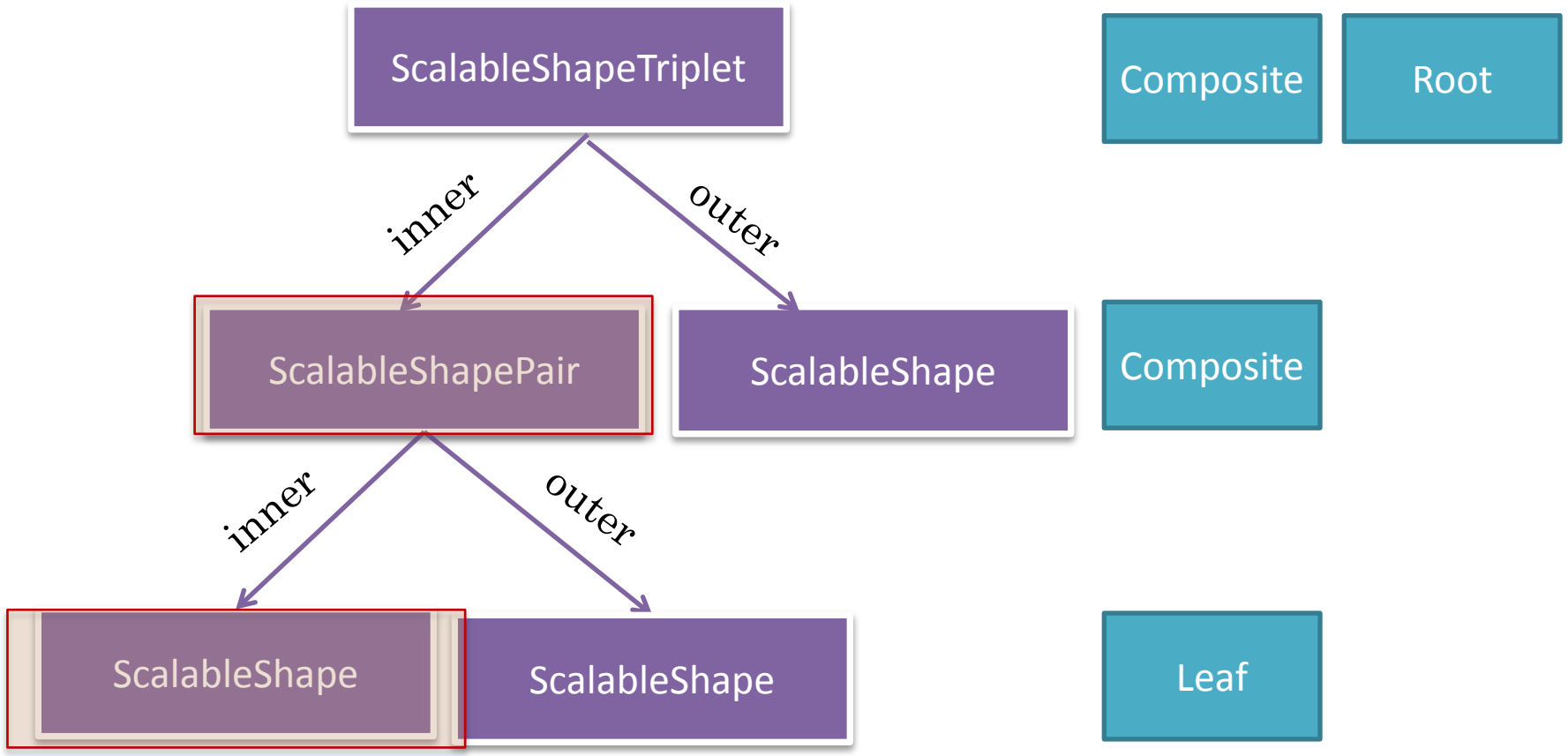
Composition Problems

Each time we add another rectangle, we create a new class and interface

Infinite number of nesting possibilities

What if we do not know the number of nester rectangles at program writing time?

Problem in Logical Structure Tree

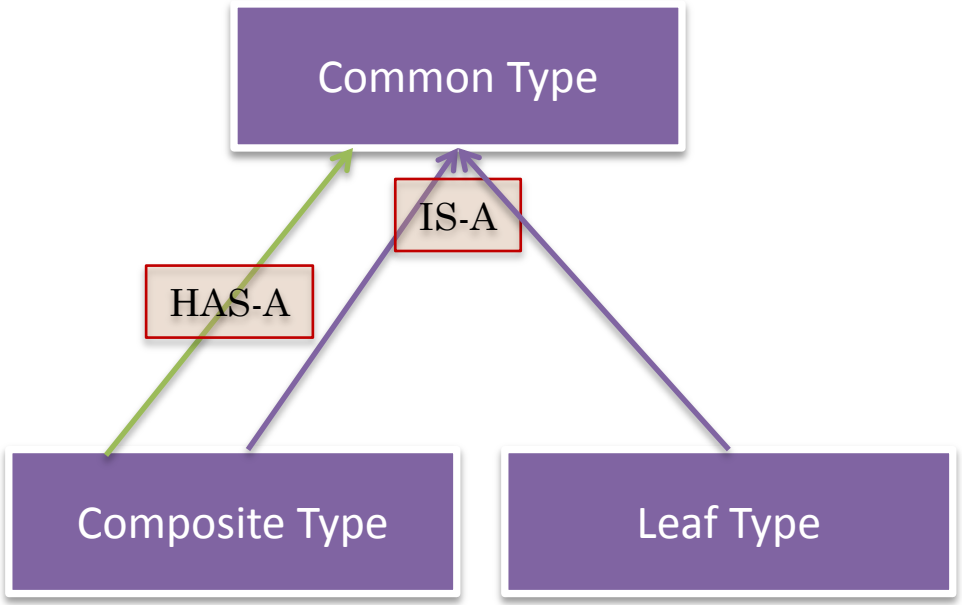


Problem: Type of inner property changes each time we add a new nesting level

Leaf node is not the same as composite node as it does not have children

But they can share a common type

Key: Composite Design Pattern



Single common type for all composite nodes and types

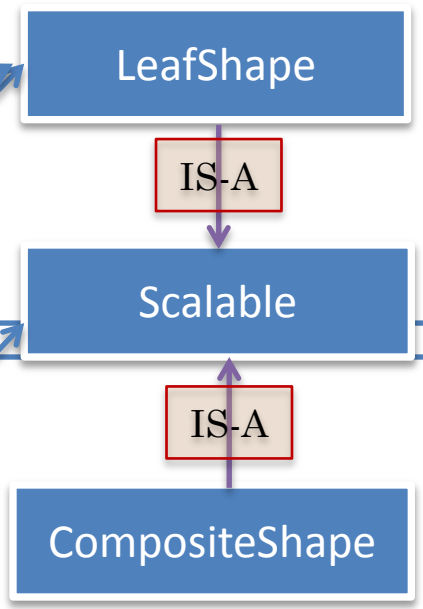
Composite node has-a and is-a common type

Common, Leaf and Composite Type?

```
public interface ScalableShape {
  public int getX();
  public int getY();
  public int getWidth();
  public int getHeight();
  public void scale(double fraction);
}
```

```
public interface ScalableNestedShapePair  {
  public ScalableShape getInner();
  public ScalableShape getOuter() ;
  public void scale(double fraction);
}
```

```
public interface ScalableNestedShapeTriplet  {
  public ScalableNestedShapePair getInner();
  public ScalableShape getOuter() ;
  public void scale(int percentage);
}
```



Common, Leaf and Composite Types

```
public interface LeafShape extends Scalable {
  public int getX();
  public int getY();
  public int getWidth();
  public int getHeight();
}
```

IS-A

```
public interface Scalable {
  public void scale(double fraction);
}
```

IS-A

```
public interface CompositeShape extends Scalable {
  public Scalable getInner();
  public LeafShape getOuter();
}
```

Making outer LeafShape rather than Scalable (or Object) allows us to perform more operations on us. Make type as specific “as it needs to be”.

Scalable Shape Implementation

```
public class AScalableShape implements ScalableShape {
  int x, y, width, height;
  public AScalableShape(int theX, int theY,
                        int theWidth, int theHeight) {
        x = theX;
        y = theY;
        width = theWidth;
        height = theHeight;
  }
  public int getX() {return x;}
  public int getY() {return y;}
  public int getWidth() {return width;}
  public int getHeight() { return height;}
  public void setHeight(int newVal) {height = newVal;}
  public void setWidth(int newVal) {width = newVal;}
  public void scale(double fraction){
        width =  (int) (width*fraction);
        height = (int) (height*fraction);
  }
}
```

LeafShape Implementation

```
public class ALeafShape implements LeafShape {
  int x, y, width, height;
  public AScalableShape(int theX, int theY,
                        int theWidth, int theHeight) {
        x = theX;
        y = theY;
        width = theWidth;
        height = theHeight;
  }
  public int getX() {return x;}
  public int getY() {return y;}
  public int getWidth() {return width;}
  public int getHeight() { return height;}
  public void setHeight(int newVal) {height = newVal;}
  public void setWidth(int newVal) {width = newVal;}
  public void scale(double fraction){
        width =  (int) (width*fraction);
        height = (int) (height*fraction);
  }
}
```

LeafRectangle Implementation

```
@StructurePattern(StructurePatternNames.RECTANGLE_PATTERN)
public class ALeafRectangle extends ALeafShape {
  public ALeafRectangle(int theX, int theY,
                        int theWidth, int theHeight) {
    super(theX, theY, theWidth, theHeight);
  }
}
```

Nester Pair Implementation

```
public class AScalableNestedShapePair
                  implements ScalableNestedShapePair {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedShapePair(ScalableShape theInner,
                                  ScalableShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Composite Shape Implementation

```
public class ACompositeShape
                implements CompositeShape{
  Scalable inner;
  LeafShape outer;
  public AScalableNestedShapePair(Scalable theInner,
                                  LeafShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public Scalable getInner() {
    return inner;
  }
  public LeafShape getOuter() {
    return outer;
  }
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Composite Pair Creator

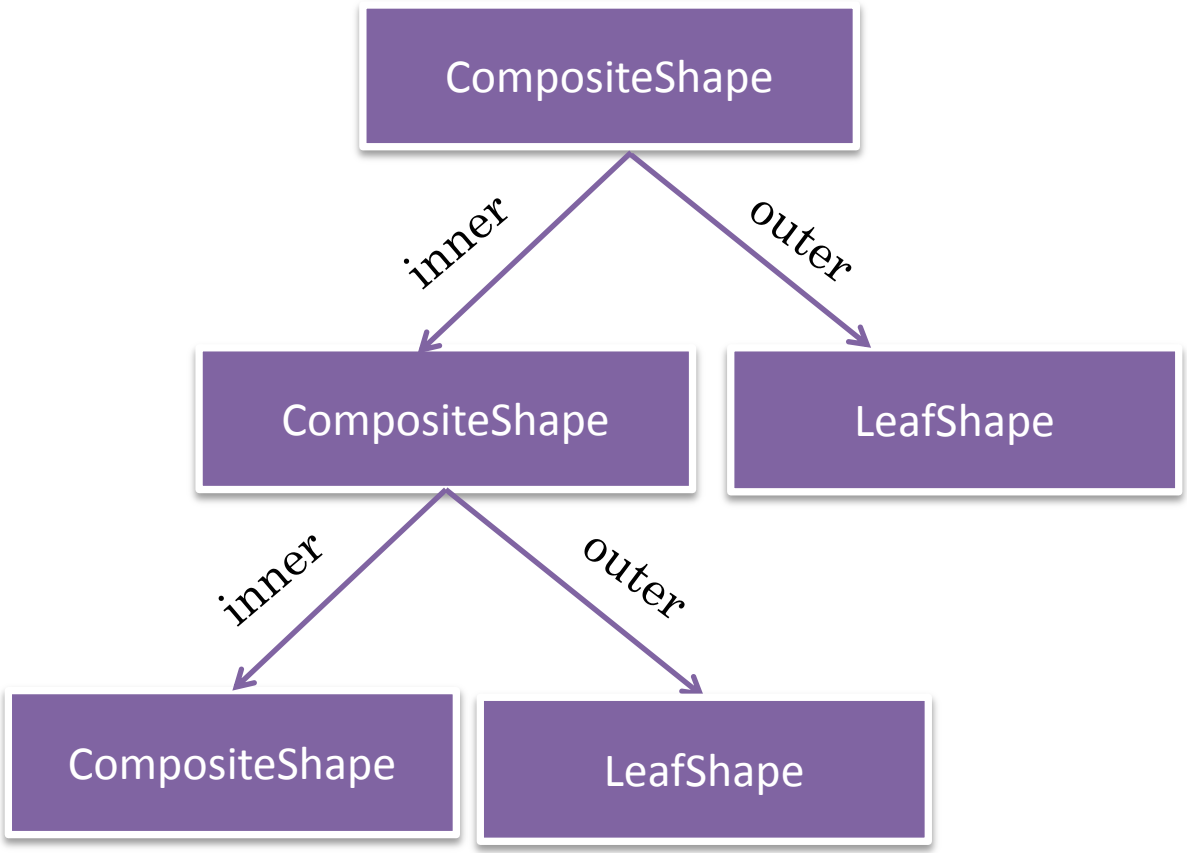
```
public class CompositeScalableNestedPairCreator {
  public static final int RELATIVE_SIZE = 2;
  public static LeafShape innermost() {
    return new ALeafRectangle (0, 0, 20, 20);
  }
  public static LeafShape toOuter (LeafShape anInner) {
    return new ALeafRectangle(anInner.getX(), anInner.getY(),
    anInner.getWidth()*RELATIVE_SIZE,
    anInner.getHeight()*RELATIVE_SIZE);
  }
  public static CompositeShape createPair () {
    LeafShape inner = innermost();
    return new ACompositeShape (inner, toOuter(inner));
  }
  public static void main (String[] args) {
    ObjectEditor.edit(createPair());
  }
}
```

Composite Triplet Creator

```
public class CompositeScalableNestedTripletCreator
                extends CompositeScalableNestedPairCreator {
  public static CompositeShape createTriplet () {
    CompositeShape pair = createPair();
    return new ACompositeShape(pair, toOuter(pair.getOuter()) );
  }
  public static void main (String[] args) {
    ObjectEditor.edit(createTriplet());
  }
}
```

Making use of fact that outer is LeafShape rather than just a Scalable

Recursive Logical Structure



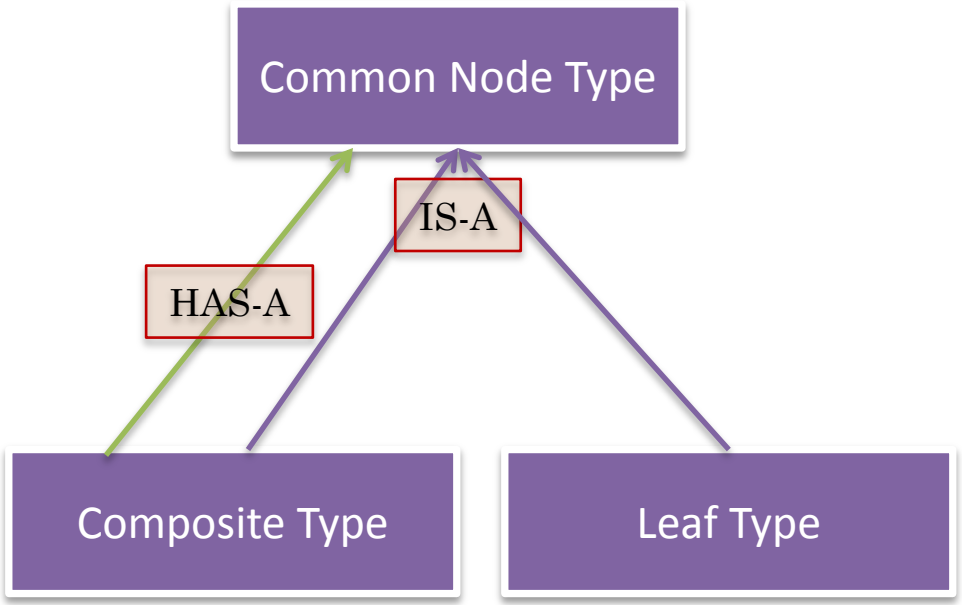
Recursive Structure

Number of levels is not fixed

Composite parent and children share not only common interface but also implementation

Recursive structure → Composite design pattern

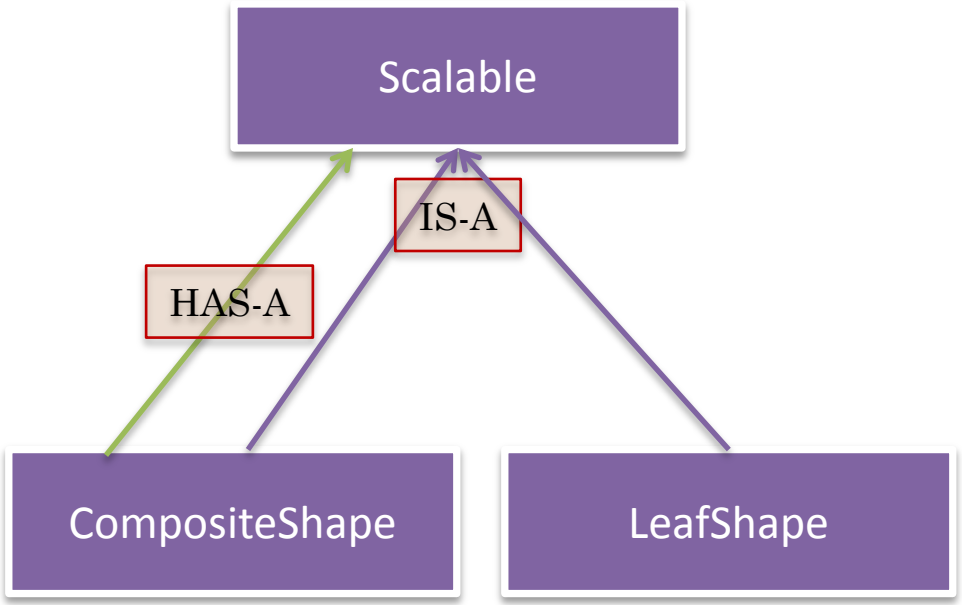
Composite Design Pattern



Single common node type for all composite nodes and types

Composite node has-a and is-a common type

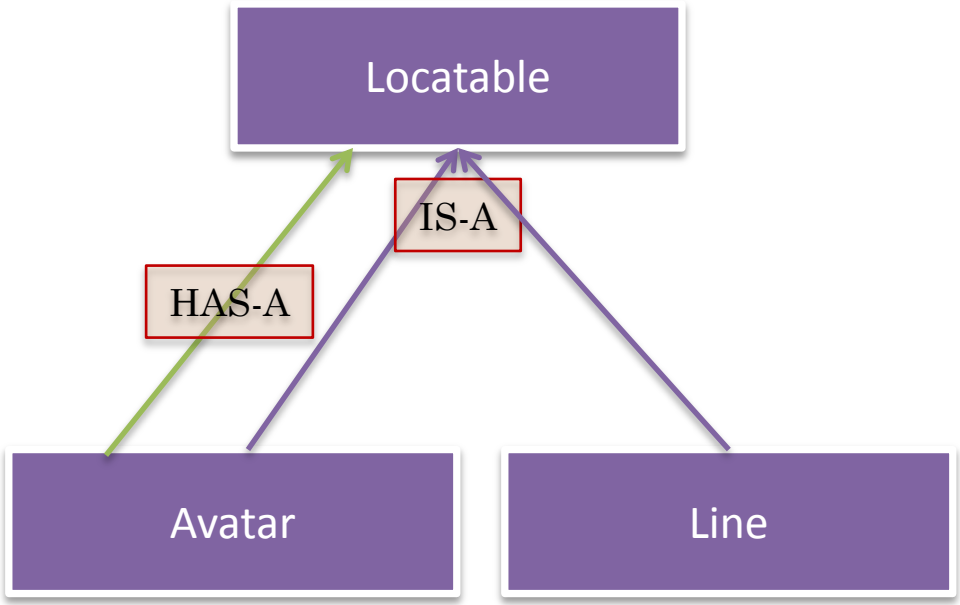
Composite Design Pattern in Example



Single common node type for all scalable objects

CompositeShape has-a and is-a Scalable

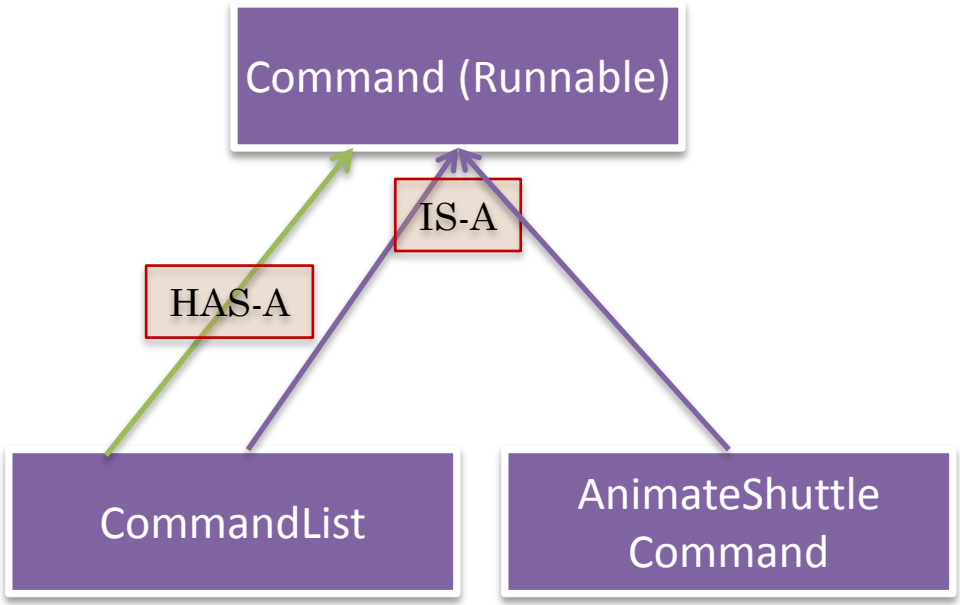
Composite Design Pattern in Graphics Assignment(s)



Single common node type for all objects with a location

Avatar has-a and is-a Locatable

Composite Design Pattern in Interpreter Assignment(s)

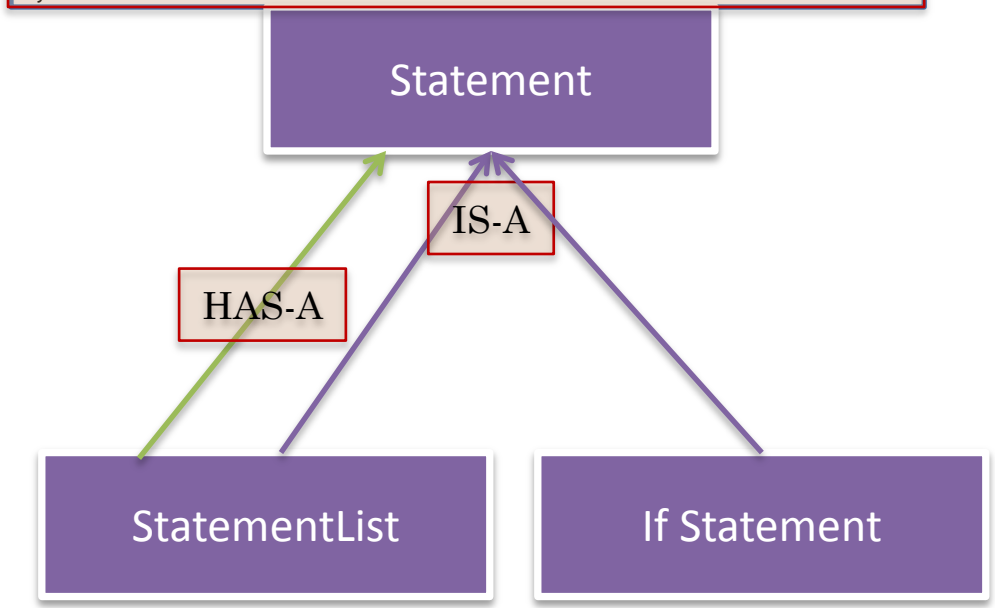


Single common node type for all command objects

CommandList has-a and is-a Command

Composite Design Pattern in Statements

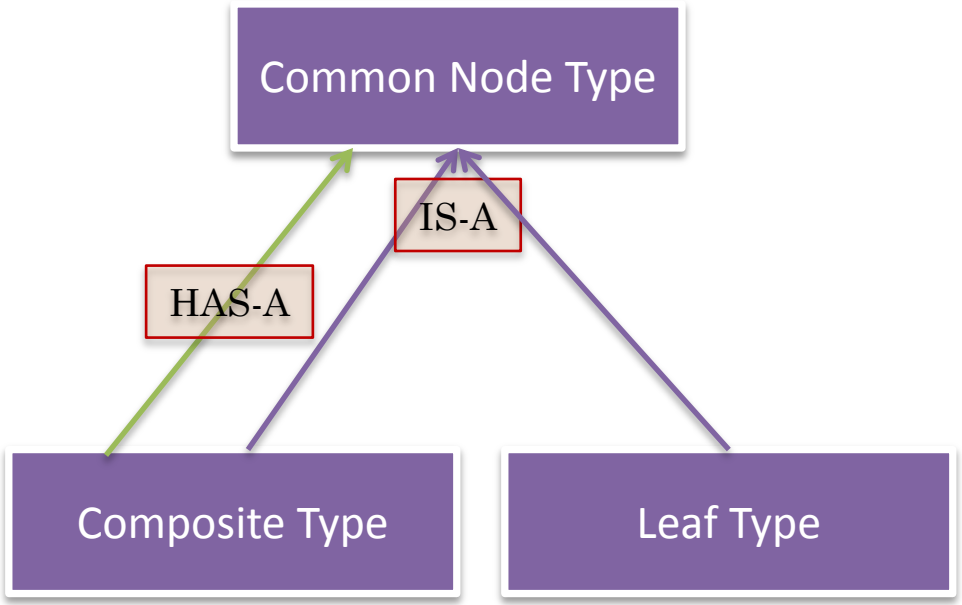
```
{
    width =  (int) (width*fraction);
    height = (int) (height*fraction);
}
```



Single common node type for all statement objects

StatementList has-a and is-a Statement

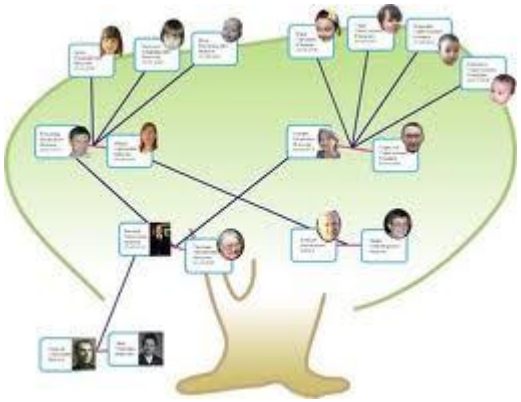
Composite Design Pattern (Review)

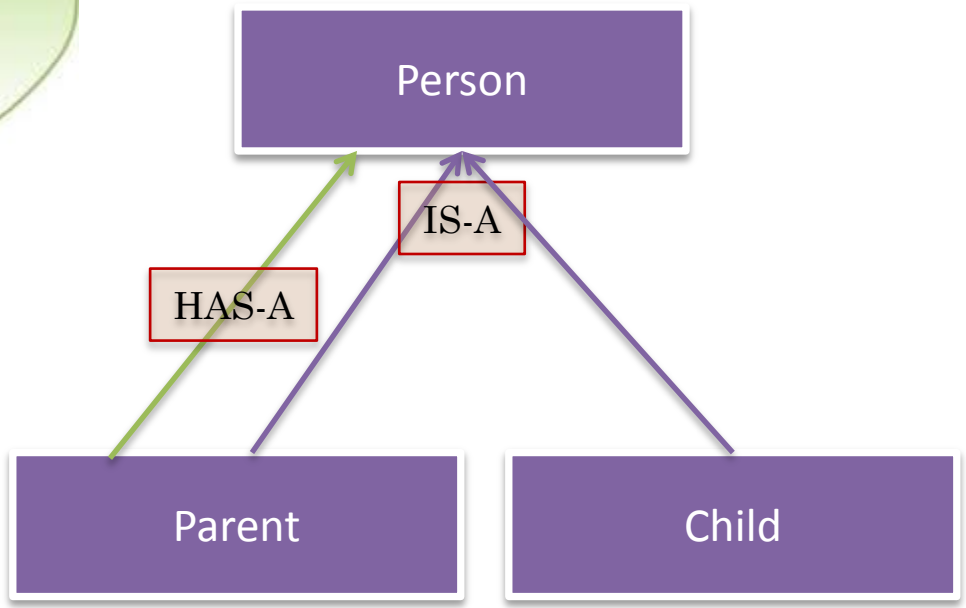


Single common node type for all composite nodes and types

Composite node has-a and is-a common type

Composite Design Pattern in Family Tree

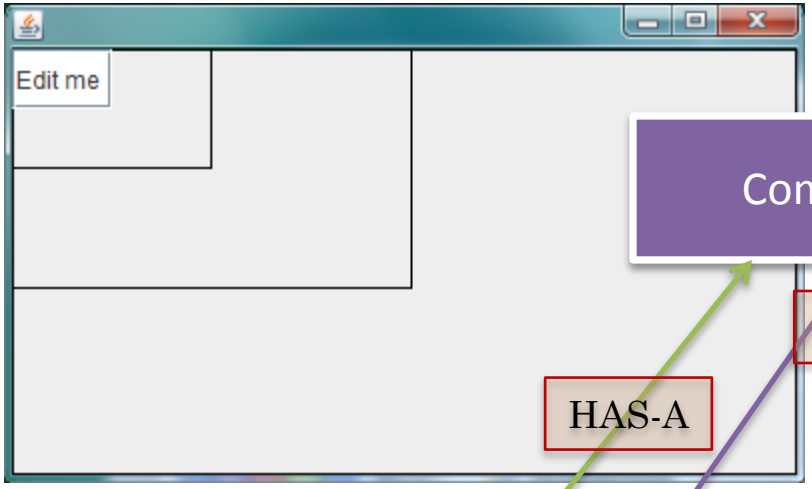




Single common node type for all people

Parent has-a and is-a person

Composite Design Pattern in Toolkit



Component, Container and JPanel methods used here?

Single common node type for all UI components

Container has-a and is-a Component

Relevant Component Methods

```
public void setSize (int width, int height) {
     …
}
public Dimension setSize (Dimension size) {
     …
}
public Dimension getSize () {
     …
}
public int getWidth() {
     …
}
public int getHeight() {
     …
}
```

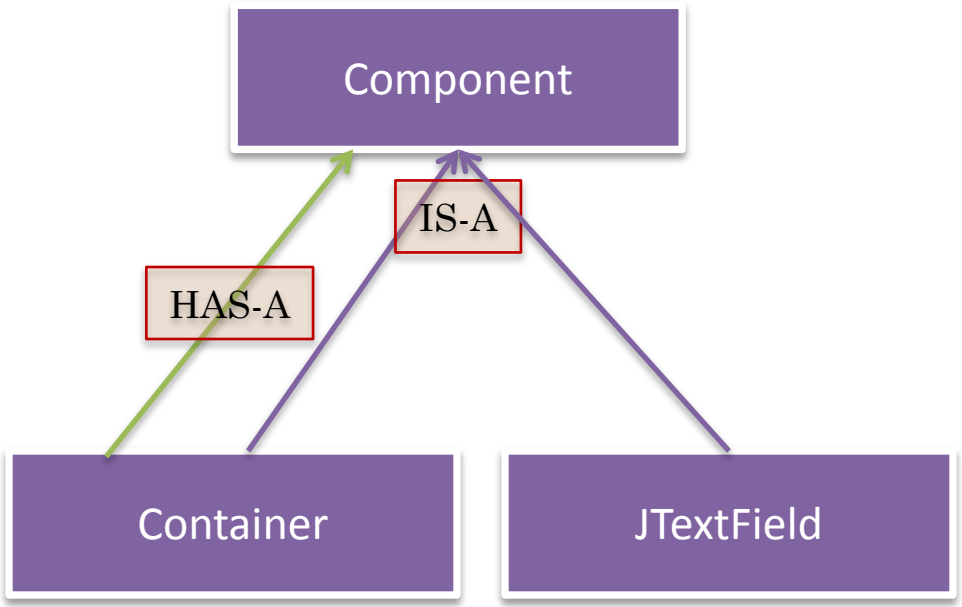
Relevant Container Methods

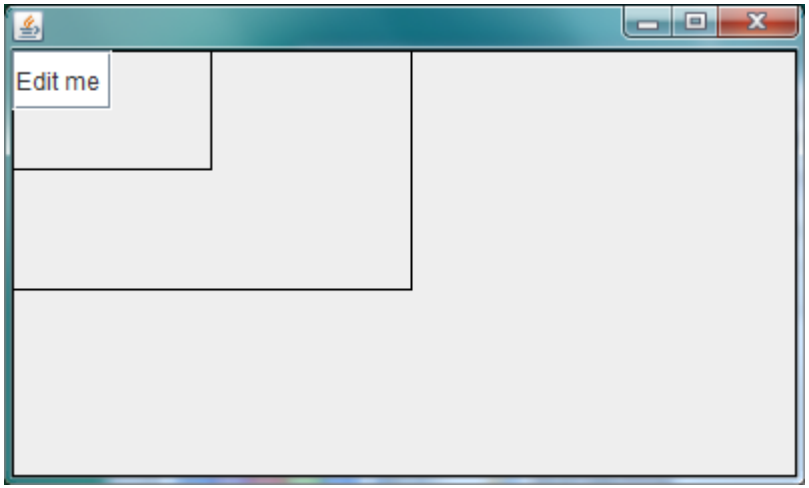
```
// allows a dynamic number of components
 public void add(Component component) {
      …
 }
 // automatically manages the size
 public void setLayout (Layout layout) {
      …
 }
```

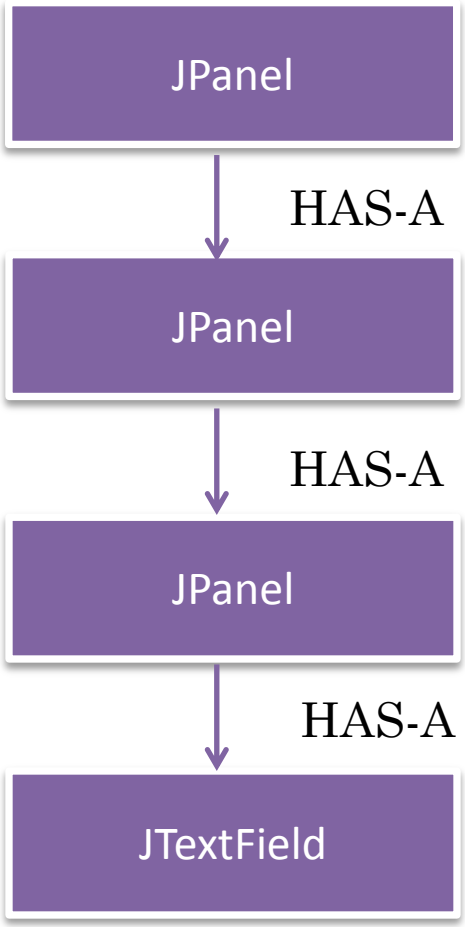
Relevant JPanel Methods

```
public void setBorder(Border border) {
    …
}
```

Example Structure







Composite User

```
public class ACompositeSwingComponentCreator {
  static final int RELATIVE_SIZE = 2;
  static Border border = new LineBorder(Color.black);
  static final Dimension LEAF_SIZE = new Dimension (50, 30);
  static int NUM_LEVELS = 3;
  public static void main(String[] args) {
    Component leaf = createLeaf(LEAF_SIZE);
    Component root = createComponentTree(leaf);
    displayInWindow(root);
  }
  public static Component createComponentTree(Component leaf) {
    Component retVal = leaf;
    for (int i = 1; i <= NUM_LEVELS; i++) {
      retVal = nestComponent(retVal);
    }
    return retVal;
}
```

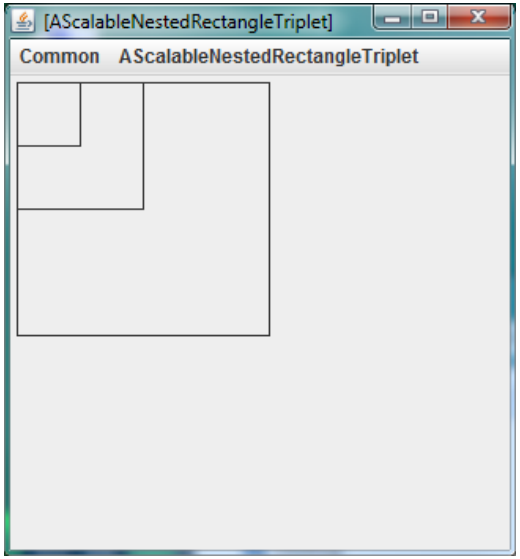
Using Component and Container Methods

```
public static Component createLeaf(Dimension size) {
  Component retVal = new JTextField("Edit me");
  retVal.setSize(size);
  return retVal;
}
public static Container nestComponent(Component inner) {
  JPanel retVal = new JPanel();
  retVal.setBorder(border); // show outline
  retVal.setSize(
    inner.getWidth() * RELATIVE_SIZE,
    inner.getHeight()* RELATIVE_SIZE);
  retVal.setLayout(null); // do not mess with the size
  retVal.add(inner);
  return retVal;
}
```

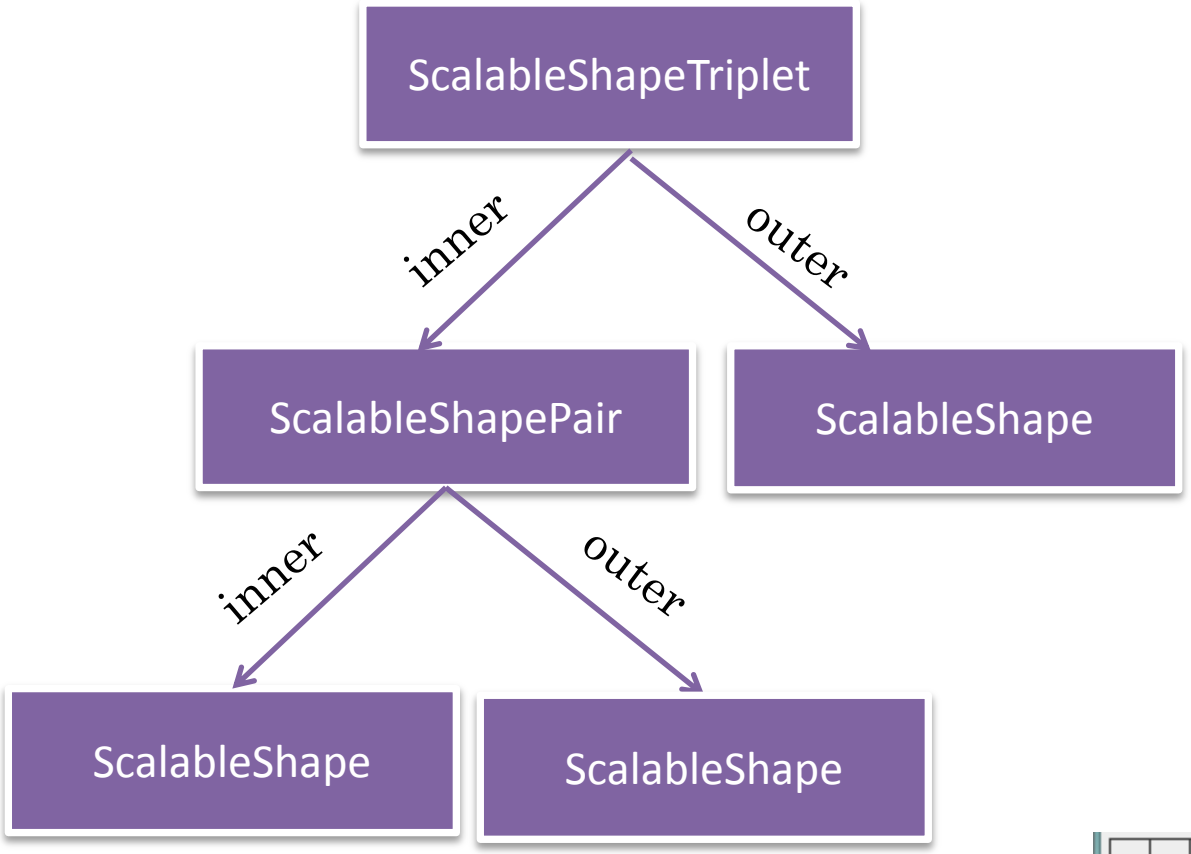
Displaying on the Screen

```
  public static void displayInWindow(Component aComponent) {
    JFrame frame = new JFrame();
    frame.add(aComponent);
    frame.setSize(aComponent.getSize());
    frame.setVisible(true);
  }
}
```

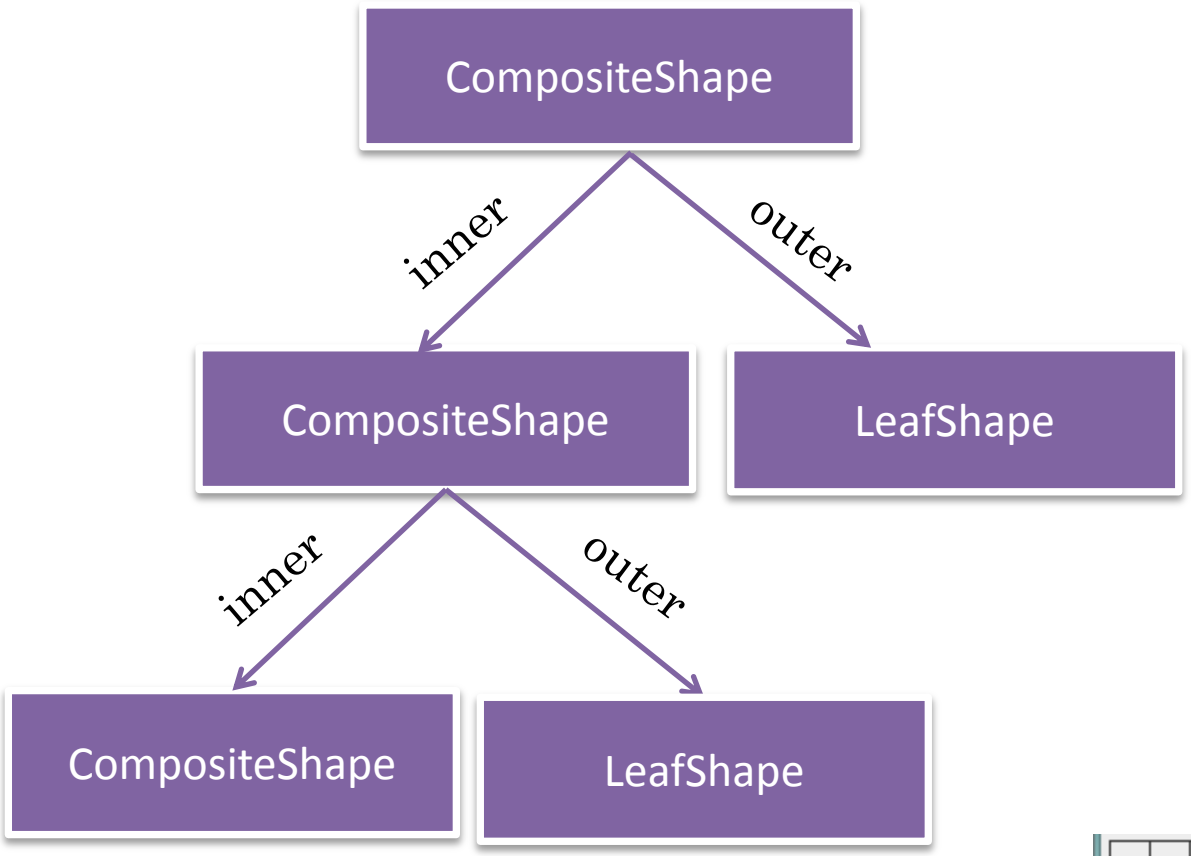
Scalable Nested Rectangle Triplet (Revisited)



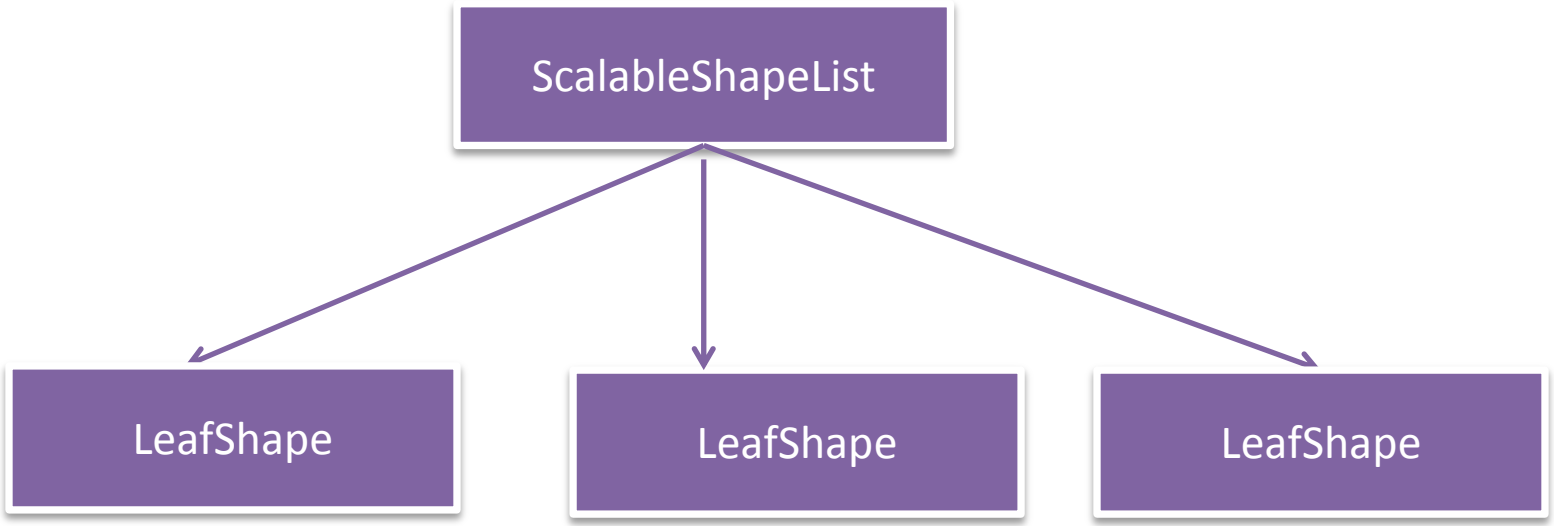
Non Composite Multi-Level Tree



Recursive Multi-Level Tree with Composite Pattern



Non Recursive, Flat, Logical Structure



ScalableShapeList

```
public interface ScalableShapeList extends Scalable {
  public int size();
  public LeafShape get(int index);
  public void add(LeafShape aLeafShape);
}
```

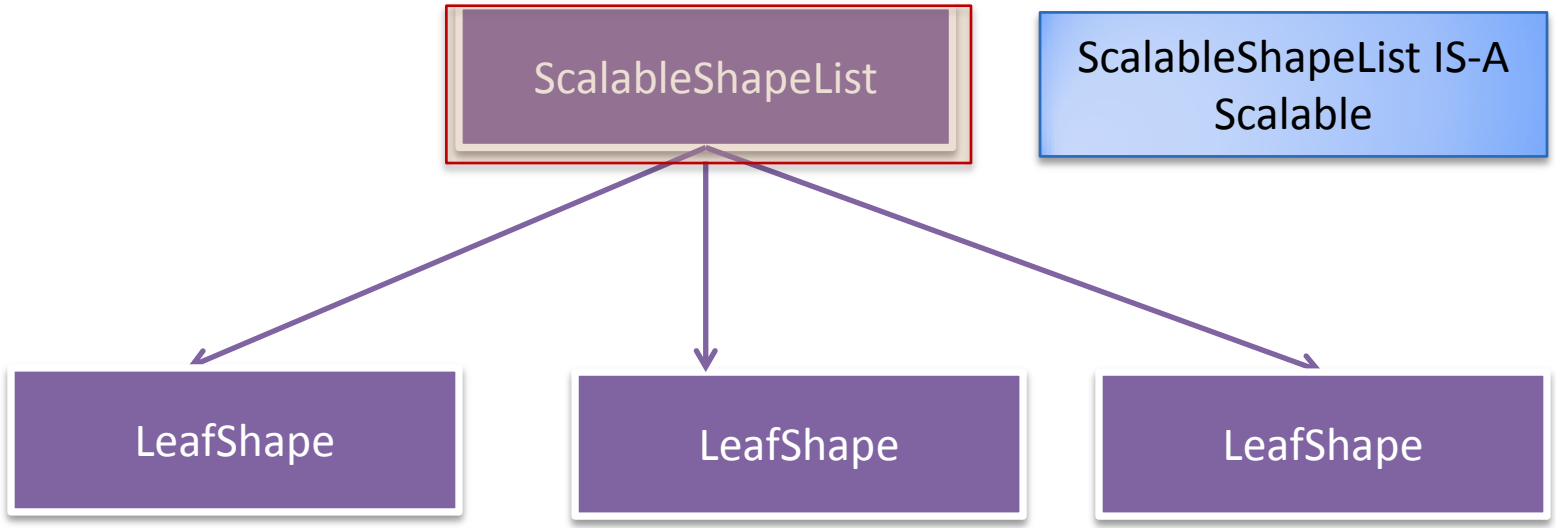
AScalableShapeList

```
public class AScalableShapeList implements ScalableShapeList{
  List list = new ArrayList();
  public void scale(double fraction) {
    for (int index=0; index < list.size(); index++) {
      ((LeafShape) (list.get(index))).scale(fraction);
    }
  }
  public void add(LeafShape aLeafShape) {
    list.add(aLeafShape);
  }
  public int size() {
    return list.size();
  }
  public LeafShape get(int index) {
    return (LeafShape) list.get(index);
  }
}
```

ScalableShapeListCreator

```
public class ScalableShapeListCreator extends
                CompositeScalableNestedPairCreator {
  public static ScalableShapeList createShapeList (int numShapes){
    ScalableShapeList shapeList = new AScalableShapeList();
    LeafShape curShape = innermost();
    for (int curElementNum = 0; curElementNum < numShapes;
         curElementNum ++) {
      shapeList.add(curShape);
      curShape = toOuter(curShape);
    }
    return shapeList;
  }
  public static void main (String[] args) {
    ObjectEditor.edit(createShapeList(3));
  }
}
```

Non Recursive, Flat, Logical Structure Pattern



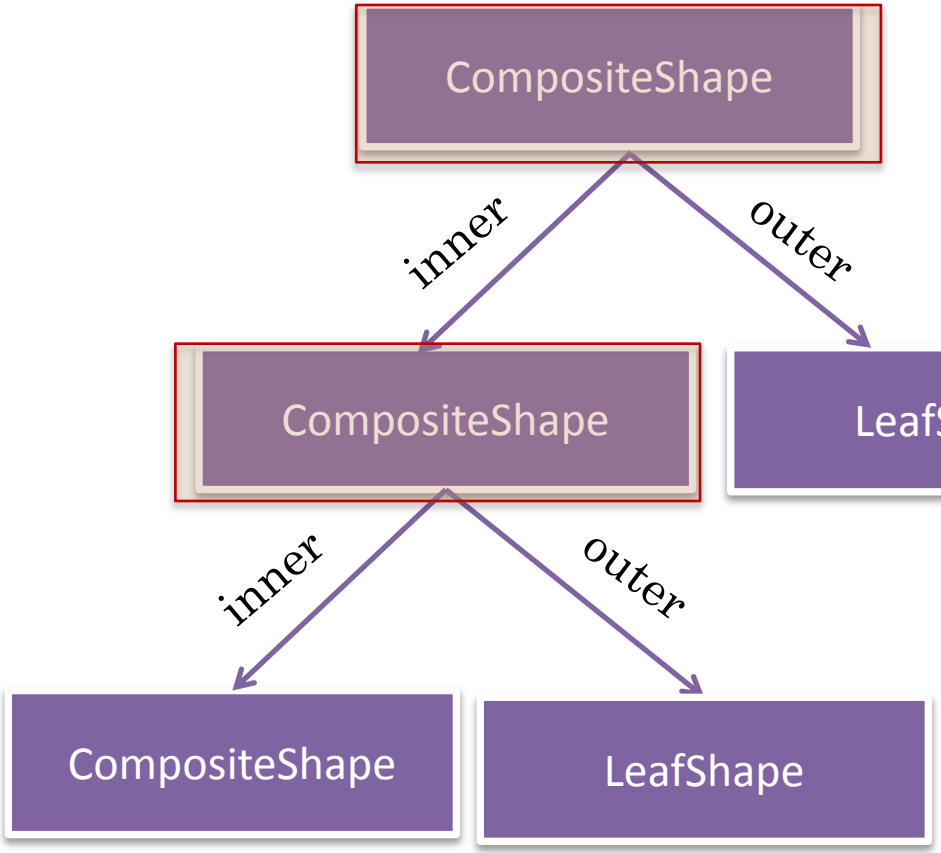
Equivalent to previous structures?

Same leaf nodes

Only two levels and one group

Cannot perform a single operation on a subgroups of leaf nodes

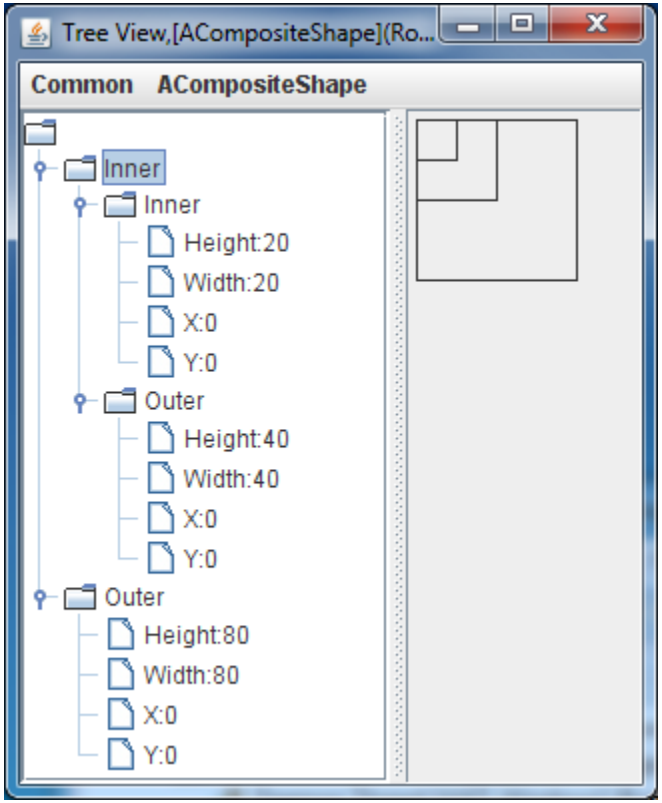
Multiple Levels and Groups



Each composite node groups all leaf (and composite) nodes in the subtree below it

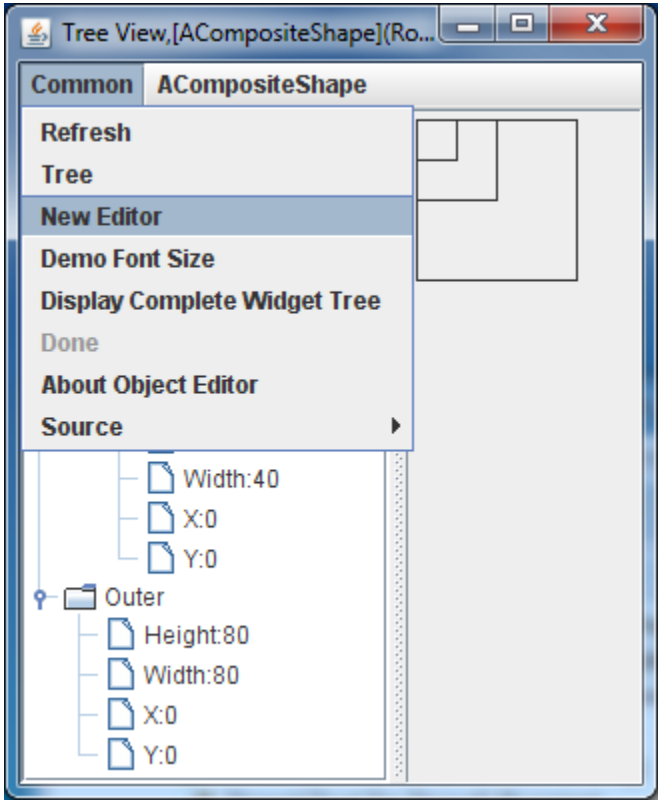
Can perform one operation to get, set, mutate the entire sub tree of a composite node

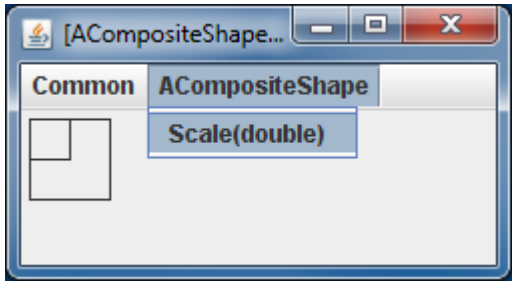
3-Level Tree

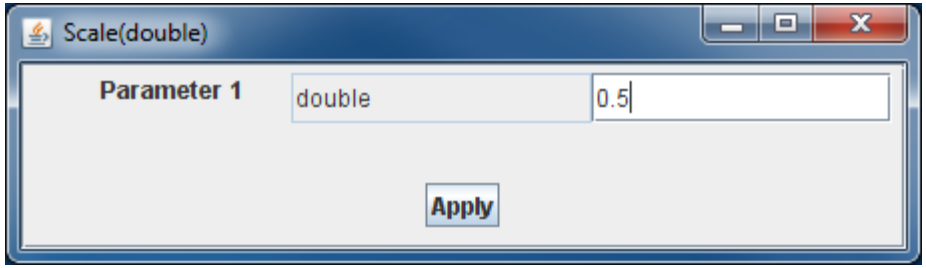


Going to do operations on the inner property of the root node (root.inner)

3-Level Tree: Displaying root.inner

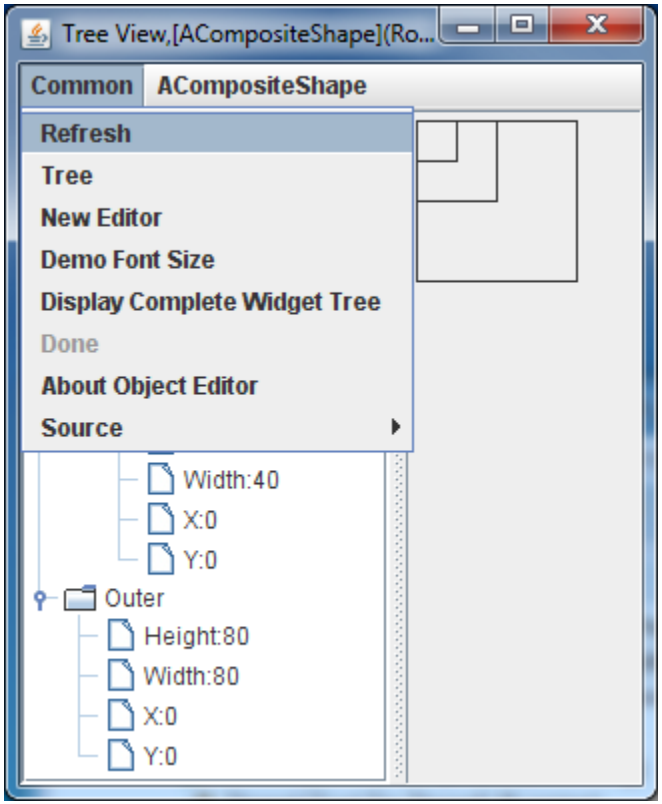


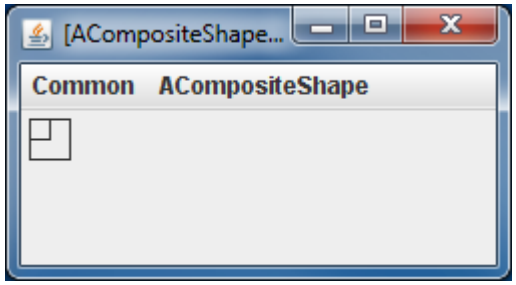




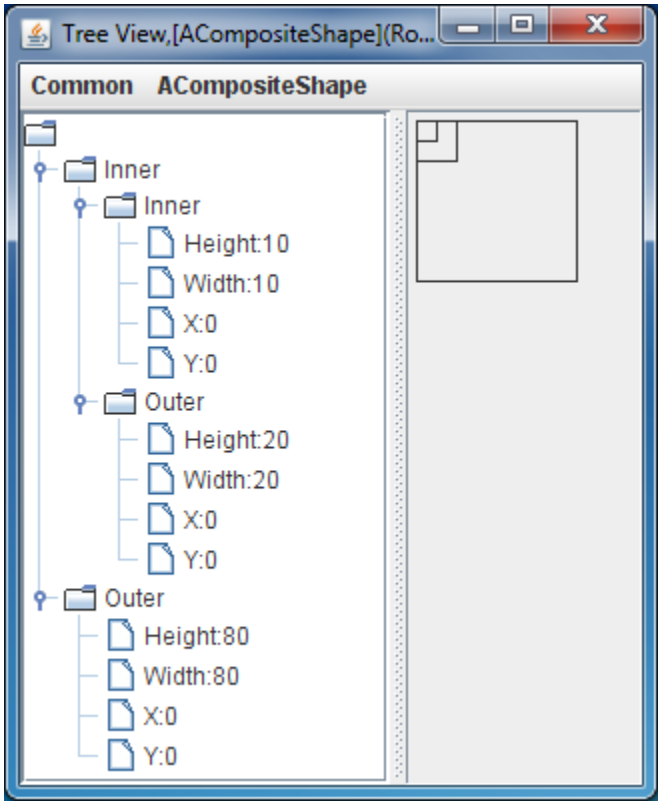
Another operation: Scaling root.inner

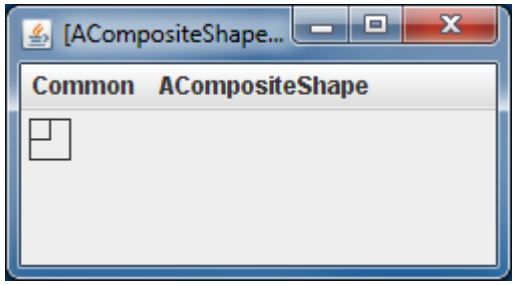
Inner Pair Scales in Right Window



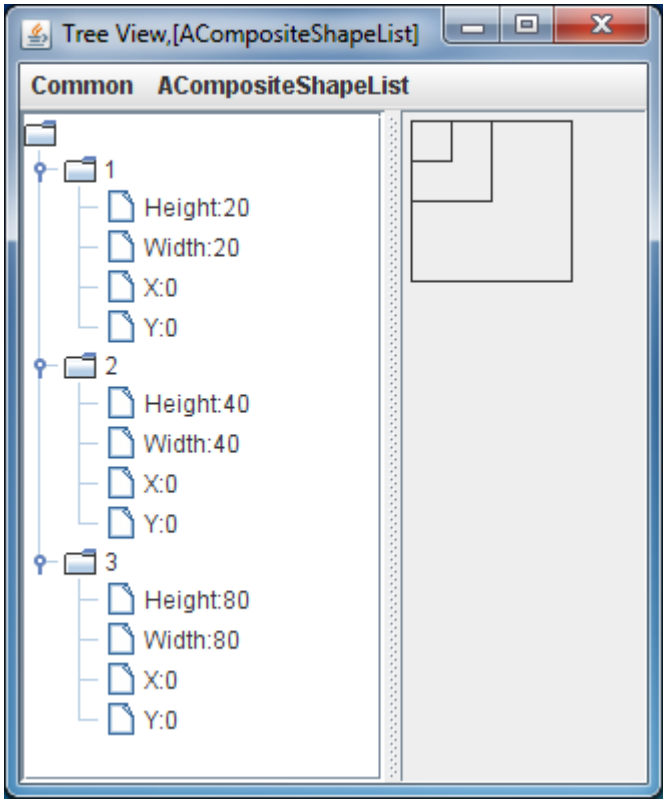


Also in Left Window after Refresh





Flat Structure



Can do operations on root and leaves but no intermediate nodes exist

In this example cannot display or scale inner.outer as such a node does not exist

Can create a new list with first two elements but that requires new copy

As in real life flat vs heirarchical structures can be created with pros and cons

Composite Pattern and Recursive Structure

- Composite design pattern useful for creating composite and leaf nodes in a composite (tree/DAG/graph) structure
- Composite type is-a and has-a type
- Must identify a type common to leaf and composite nodes
- Common type implemented differently by lead and composite nodes
- Leads to recursive structures
  - Used in many contexts
- Some recursive structures can be replaced by lists in some applications

Extra Slides

Nester Pair Implementation Revisited

```
public class AScalableNestedShapePair
                  implements ScalableNestedShapePair {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedShapePair(ScalableShape theInner,
                                  ScalableShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Pair imposes no constraint on the inner and outer shapes

Triplet Implementation

```
public class AScalableNestedShapeTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedShapeTriplet(
                        ScalableNestedShapePair theInner,
                        ScalableShape theOuter) {
    inner = theInner;
    outer = theOuter;
  }
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Triplet imposes no constraint on the inner and outer shapes

Another Pair

```
public class AnotherScalableNestedShapePair
                  implements ScalableNestedShapePair  {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedRectanglePair(ScalableShape theInner) {
    inner = theInner;
    outer =  new AScalableRectangle(inner.getX(), inner.getY(),
                                    inner.getWidth()*RELATIVE_SIZE,
                                    inner.getHeight()*RELATIVE_SIZE);
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Another Triplet

```
public class AnotherScalableNestedShapeTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedRectangleTriplet(
                          ScalableShape theInnerMost) {
    inner = new AScalableNestedRectanglePair(theInnerMost);
    outer = new AScalableRectangle(
              inner.getOuter().getX(), inner.getOuter().getY(),
              inner.getOuter().getWidth()*
                 ScalableNestedShapePair.RELATIVE_SIZE,
              inner.getOuter().getHeight()*
                 ScalableNestedShapePair.RELATIVE_SIZE);
}
```

Triplet Implementation

```
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Triplet Main

```
public class ScalableNestedRectangleTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle = new AScalableRectangle (0, 0, 20, 20);
    ScalableNestedShapeTriplet triplet =
      new AScalableNestedRectangleTriplet(rectangle);
    ObjectEditor.edit(triplet);
  }
}
```

Another Pair

```
public class AnotherScalableNestedShapePair
                  implements ScalableNestedShapePair  {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedRectanglePair(ScalableShape theInner) {
    inner = theInner;
    outer =  new AScalableRectangle(inner.getX(), inner.getY(),
                                    inner.getWidth()*RELATIVE_SIZE,
                                    inner.getHeight()*RELATIVE_SIZE);
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Need getters for properties of bounding rectangle of inner shape

Need the scale method of inner shape

Controlling Triplet

```
public class AnotherScalableNestedShapeTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedRectangleTriplet(
                        ScalableShape theInnerMost) {
    inner = new AScalableNestedRectanglePair(theInnerMost);
    outer = new AScalableRectangle(
              inner.getOuter().getX(), inner.getOuter().getY(),
              inner.getOuter().getWidth()*
                 ScalableNestedShapePair.RELATIVE_SIZE,
              inner.getOuter().getHeight()*
                 ScalableNestedShapePair.RELATIVE_SIZE);
```

Need getters for properties of bounding rectangle of **outer rectangle of inner object**

Inner object could implement shape methods to read properties of the outer rectangle it creates

Need getters for properties of bounding rectangle of inner **shape**

Triplet Implementation

```
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Need the scale method of inner shape

Pair Implementation

```
public class AScalableNestedRectanglePair
                  implements ScalableNestedShapePair  {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedRectanglePair(ScalableShape theInner) {
    inner = theInner;
    outer =  new AScalableRectangle(inner.getX(), inner.getY(),
                                    inner.getWidth()*RELATIVE_SIZE,
                                    inner.getHeight()*RELATIVE_SIZE);
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Need getters for properties of bounding rectangle of inner shape

Need the scale method of inner shape

Triplet Implementation

```
public class AScalableNestedRectangleTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedRectangleTriplet(
                        ScalableShape theInnerMost) {
    inner = new AScalableNestedRectanglePair(theInnerMost);
    outer = new AScalableRectangle(
              inner.getOuter().getX(), inner.getOuter().getY(),
              inner.getOuter().getWidth()*
                 ScalableNestedShapePair.RELATIVE_SIZE,
              inner.getOuter().getHeight()*
                 ScalableNestedShapePair.RELATIVE_SIZE);
```

Need getters for properties of bounding rectangle of **outer rectangle of inner object**

Inner object could implement shape methods to read properties of the outer rectangle it creates

Need getters for properties of bounding rectangle of inner **shape**

Implementing Composite Node

```
public class ACompositeScalableNestedPair
                implements NestedShapePair, ScalableShape {
  ScalableShape inner, outer;
  public ACompositeScalableNestedPair(ScalableShape theInner) {
    inner = theInner;
    outer = new AScalableRectangle(
                inner.getX(), inner.getY(),
                inner.getWidth()*RELATIVE_SIZE,
                inner.getHeight()*RELATIVE_SIZE);
  }
```

Implementing Composite Node

```
public int getX() {
  return outer.getX();
}
public int getY() {
  return outer.getY();
}
public int getWidth() {
  return outer.getWidth();
}
public int getHeight() {
  return outer.getHeight();
}
public void scale(double fraction){
  outer.scale(fraction);
  inner.scale(fraction);
}
public ScalableShape getInner() {
  return inner;
}
public ScalableShape getOuter() {
  return outer;
}
}
```

Properties of bounding box of (outer rectangle of) composite shape

Scales the shape

Extra methods in NestedShapePair

Composite Pair/Triplet Creator

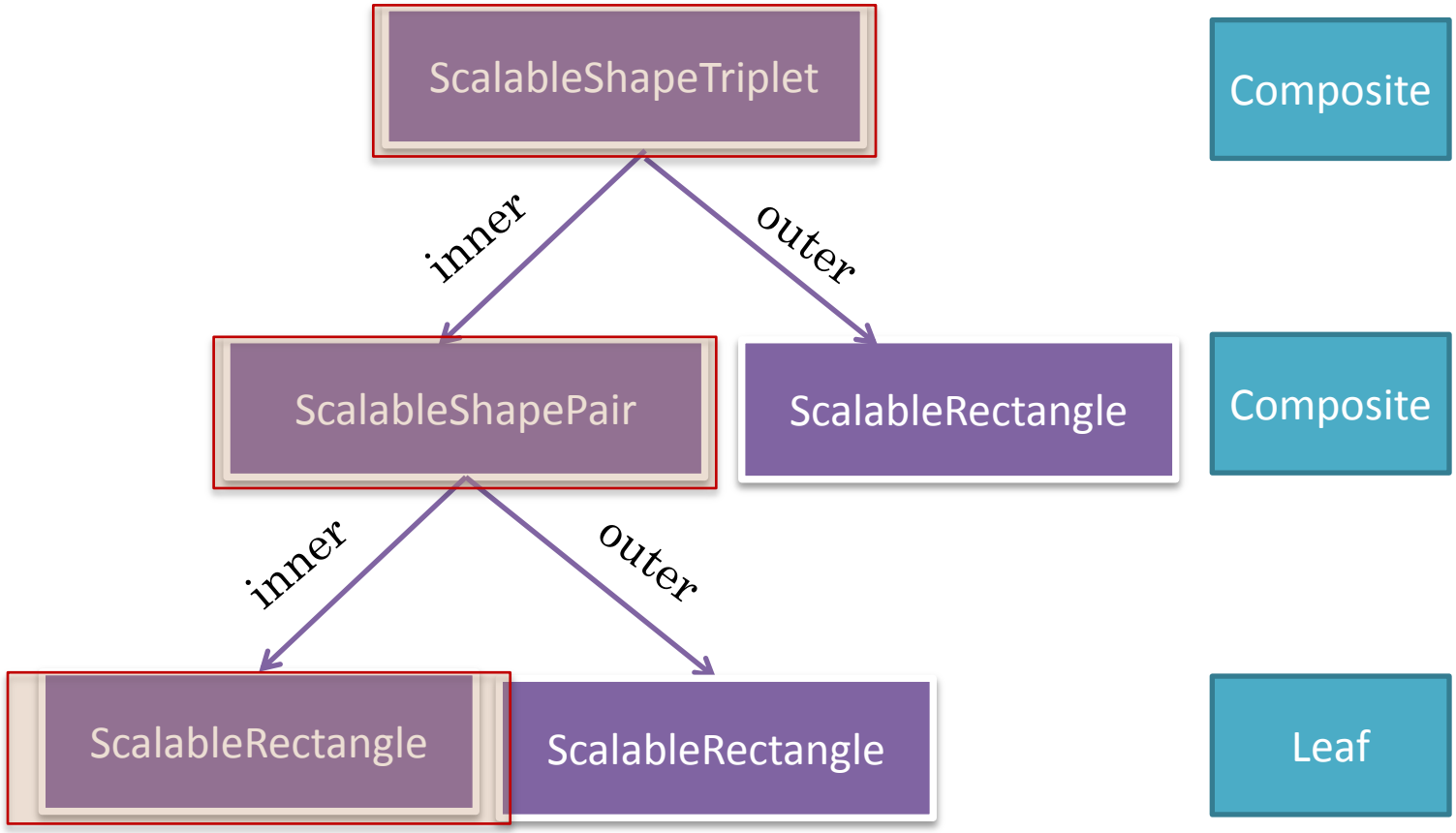
```
public class ACompositeScalableNestedPairCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectangle));
  }
}
```

```
public class ACompositeScalableNestedTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ScalableShape rectanglePair =
      new ACompositeScalableNestedPair(rectangle);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectanglePair));
  }
}
```

Composite Triplet Creator

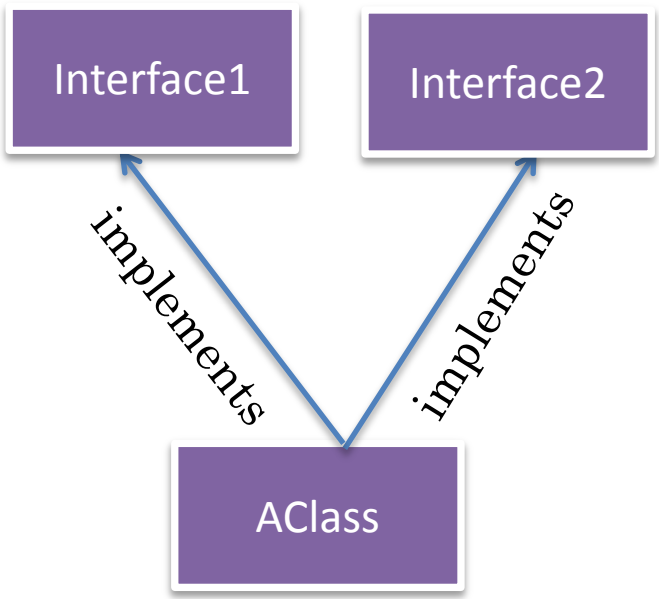
```
public class ACompositeScalableNestedTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ScalableShape rectanglePair =
      new ACompositeScalableNestedPair(rectangle);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectanglePair));
  }
}
```

Problem in Logical Structure Tree



Leaf node does not have components so cannot have the same interface as composite node

Implementing Multiple Interfaces



A class must provide method bodies for all interfaces it implements

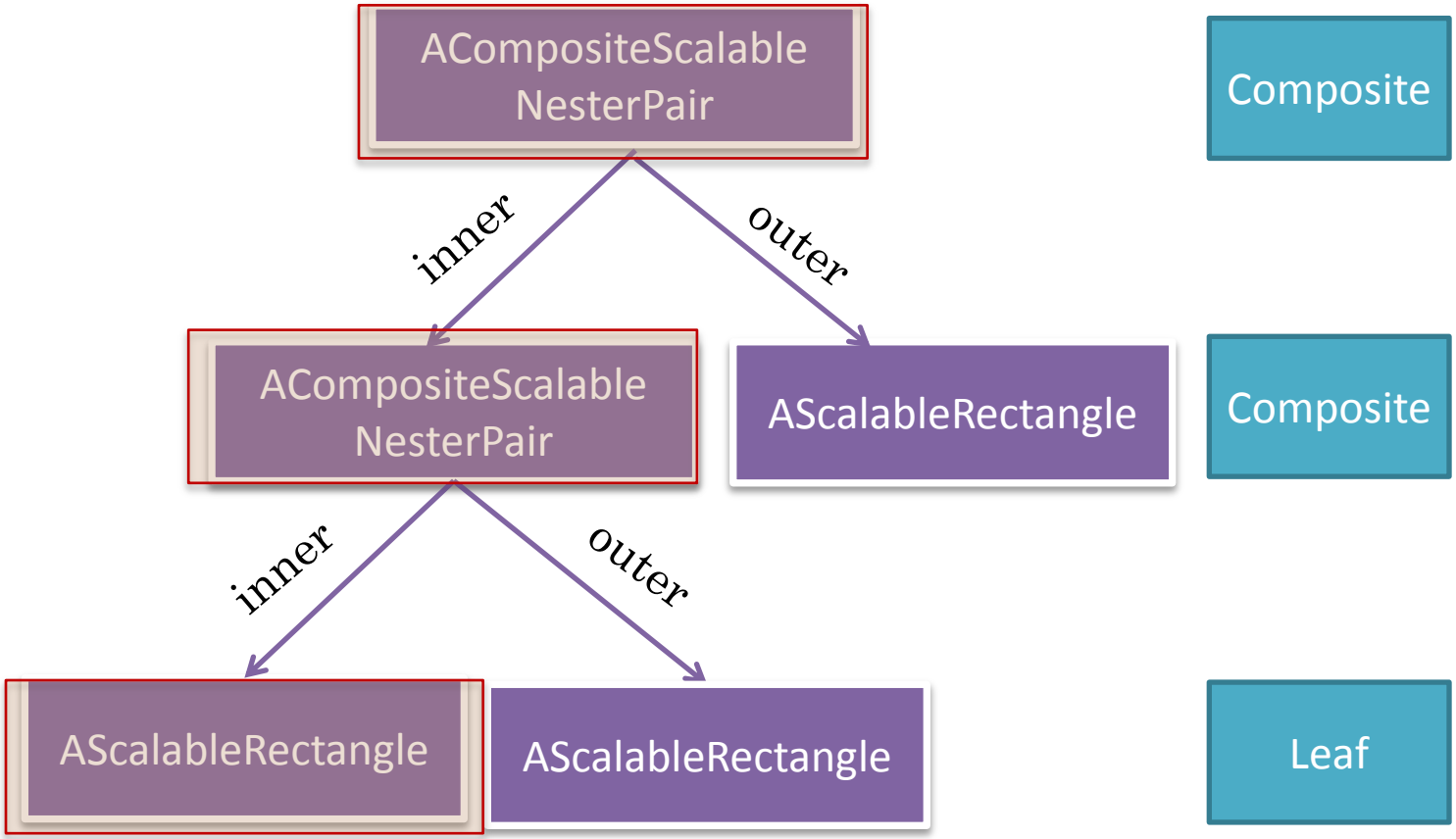
An instance of it can be typed using any of the interfaces it implements

A la a teaching assistant following duties of both a student and an employee

```
public class AClass implements Interface1, Interface2 { …}
```

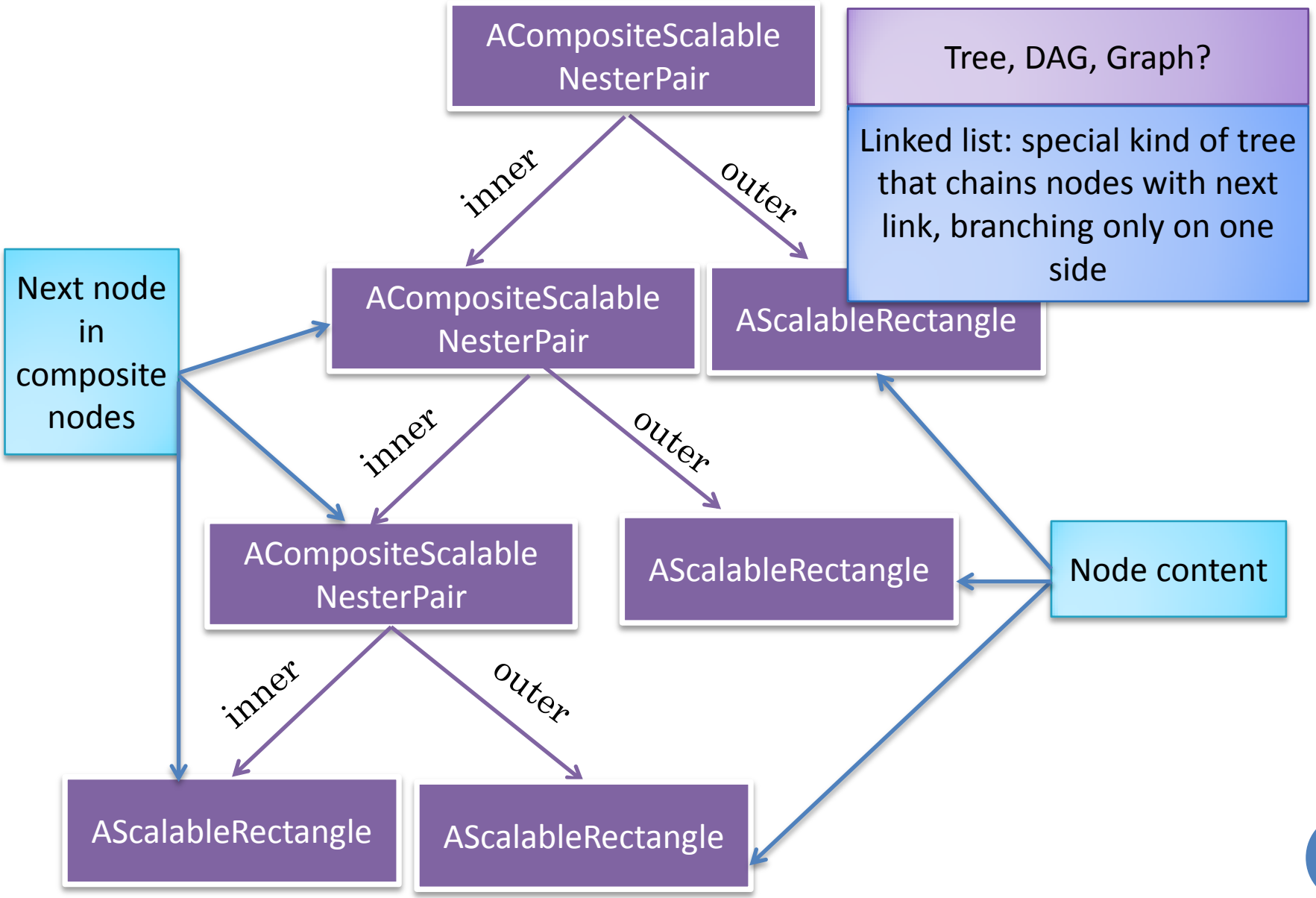
```
Interface1 anInterface1 = new AClass();
Interface2 anInterface2 = new AClass();
```

Problem in Logical Structure



The inner property is of some type implemented by both AScalableRectangle and ACompositeScalableNestedPair

Kind of Logical Structure



Triplet Implementation

```
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Triplet Main

```
public class ScalableNestedRectangleTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle = new AScalableRectangle (0, 0, 20, 20);
    ScalableNestedShapeTriplet triplet =
      new AScalableNestedRectangleTriplet(rectangle);
    ObjectEditor.edit(triplet);
  }
}
```

Reusing AScalableNestedShapePair

Triplet Main

```
public class ACompositeScalableNestedTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
            new AScalableRectangle (0, 0, 20, 20);
    ScalableShape rectanglePair =
            new ACompositeScalableNestedPair(rectangle);
    ObjectEditor.edit(
            new ACompositeScalableNestedPair(rectanglePair));
  }
}
```

Reusing AScalableNestedShapePair

Pair Implementation

```
public class AScalableNestedRectanglePair
                  implements ScalableNestedShapePair  {
  ScalableShape inner;
  ScalableShape outer;
  public AScalableNestedRectanglePair(ScalableShape theInner) {
    inner = theInner;
    outer =  new AScalableRectangle(inner.getX(), inner.getY(),
                                    inner.getWidth()*RELATIVE_SIZE,
                                    inner.getHeight()*RELATIVE_SIZE);
  }
  public ScalableShape getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
  @Override
  public void scale(double percentage) {
    inner.scale(percentage);
    outer.scale(percentage);
  }
}
```

Need getters for properties of bounding rectangle of inner shape

Need the scale method of inner shape

Triplet Implementation

```
public class AScalableNestedRectangleTriplet
                implements ScalableNestedShapeTriplet  {
  ScalableNestedShapePair inner;
  ScalableShape outer;
  public AScalableNestedRectangleTriplet(
                        ScalableShape theInnerMost) {
    inner = new AScalableNestedRectanglePair(theInnerMost);
    outer = new AScalableRectangle(
              inner.getOuter().getX(), inner.getOuter().getY(),
              inner.getOuter().getWidth()*
                 ScalableNestedShapePair.RELATIVE_SIZE,
              inner.getOuter().getHeight()*
                 ScalableNestedShapePair.RELATIVE_SIZE);
```

Need getters for properties of bounding rectangle of **outer rectangle of inner object**

Inner object could implement shape methods to read properties of the outer rectangle it creates

Need getters for properties of bounding rectangle of inner **shape**

Triplet Implementation

```
  public void scale(int percentage){
    outer.scale(percentage);
    inner.scale(percentage);
  }
  public ScalableNestedShapePair getInner() {
    return inner;
  }
  public ScalableShape getOuter() {
    return outer;
  }
}
```

Need the scale method of inner shape

Implementing Composite Node

```
public class ACompositeScalableNestedPair
                implements NestedShapePair, ScalableShape {
  ScalableShape inner, outer;
  public ACompositeScalableNestedPair(ScalableShape theInner) {
    inner = theInner;
    outer = new AScalableRectangle(
                inner.getX(), inner.getY(),
                inner.getWidth()*RELATIVE_SIZE,
                inner.getHeight()*RELATIVE_SIZE);
  }
```

Implementing Composite Node

```
public int getX() {
  return outer.getX();
}
public int getY() {
  return outer.getY();
}
public int getWidth() {
  return outer.getWidth();
}
public int getHeight() {
  return outer.getHeight();
}
public void scale(double fraction){
  outer.scale(fraction);
  inner.scale(fraction);
}
public ScalableShape getInner() {
  return inner;
}
public ScalableShape getOuter() {
  return outer;
}
}
```

Properties of bounding box of (outer rectangle of) composite shape

Scales the shape

Extra methods in NestedShapePair

Composite Pair/Triplet Creator

```
public class ACompositeScalableNestedPairCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectangle));
  }
}
```

```
public class ACompositeScalableNestedTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ScalableShape rectanglePair =
      new ACompositeScalableNestedPair(rectangle);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectanglePair));
  }
}
```

Composite Triplet Creator

```
public class ACompositeScalableNestedTripletCreator {
  public static void main (String[] args) {
    ScalableShape rectangle =
      new AScalableRectangle (0, 0, 20, 20);
    ScalableShape rectanglePair =
      new ACompositeScalableNestedPair(rectangle);
    ObjectEditor.edit(
      new ACompositeScalableNestedPair(rectanglePair));
  }
}
```

Relevant Component Methods

```
public void setSize (int width, int height) {
     …
}
public Dimension setSize (Dimension size) {
     …
}
public Dimension getSize () {
     …
}
public int getWidth() {
     …
}
public int getHeight() {
     …
}
```

Like Shape methods (setters instead of scale)

Relevant Container Methods

```
// allows a dynamic number of components
 public void add(Component component) {
      …
 }
 // automatically manages the size
 public void setLayout (Layout layout) {
      …
 }
```

Like composite pair except a dynamic number of components

Java Composite User

```
public class ACompositeSwingComponentCreator {
  static final int RELATIVE_SIZE = 2;
  static Border border = new LineBorder(Color.black);
  static final Dimension LEAF_SIZE = new Dimension (50, 30);
  static int NUM_LEVELS = 3;
  public static void main(String[] args) {
    Component leaf = createLeaf(LEAF_SIZE);
    Component root = createComponentTree(leaf);
    displayInWindow(root);
  }
  public static Component createComponentTree(Component leaf) {
    Component retVal = leaf;
    for (int i = 1; i <= NUM_LEVELS; i++) {
      retVal = nestComponent(retVal);
    }
    return retVal;
}
```

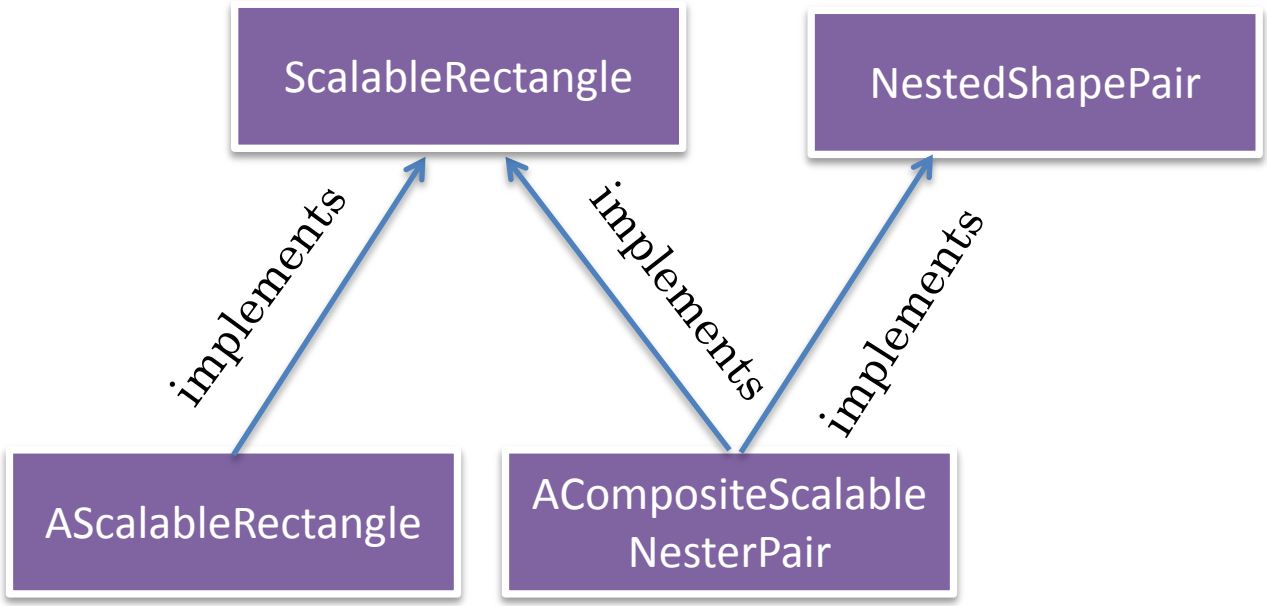
Using Component and Container Methods

```
public static Component createLeaf(Dimension size) {
  Component retVal = new JTextField("Edit me");
  retVal.setSize(size);
  return retVal;
}
public static Container nestComponent(Component inner) {
   JPanel retVal = new JPanel();
   retVal.setBorder(border); // show outline
   retVal.setSize(
     inner.getWidth() * RELATIVE_SIZE,
     inner.getHeight()* RELATIVE_SIZE);
   retVal.setLayout(null); // do not mess with the size
   retVal.add(inner);
   return retVal;
}
```

Displaying on the Screen

```
  public static void displayInWindow(Component aComponent) {
    JFrame frame = new JFrame();
    frame.add(aComponent);
    frame.setSize(aComponent.getSize());
    frame.setVisible(true);
  }
}
```

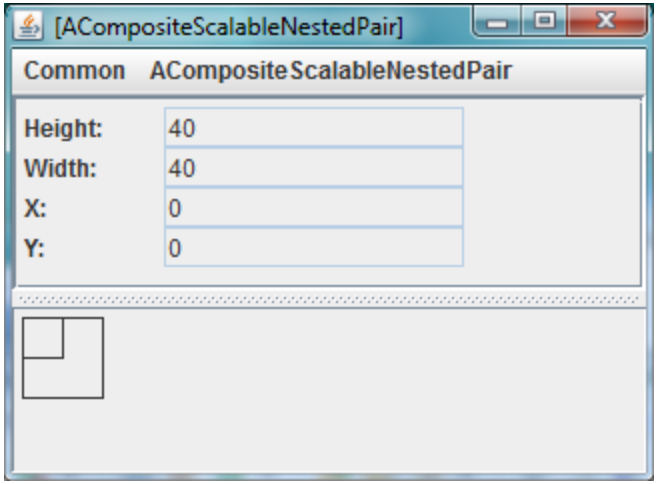
Composite Design Pattern in Problem

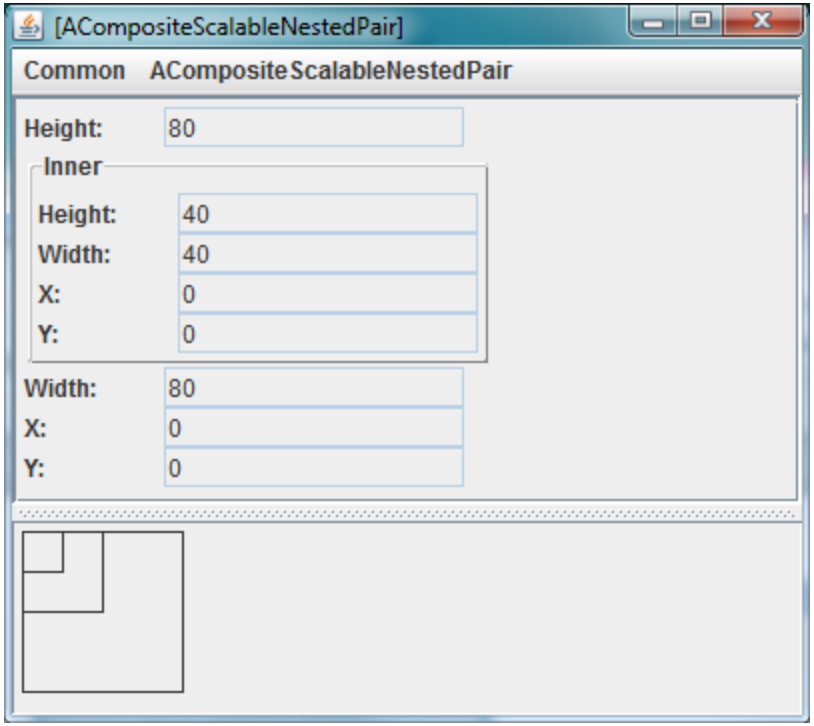


Usually implemented with inheritance

Can do without inheritance

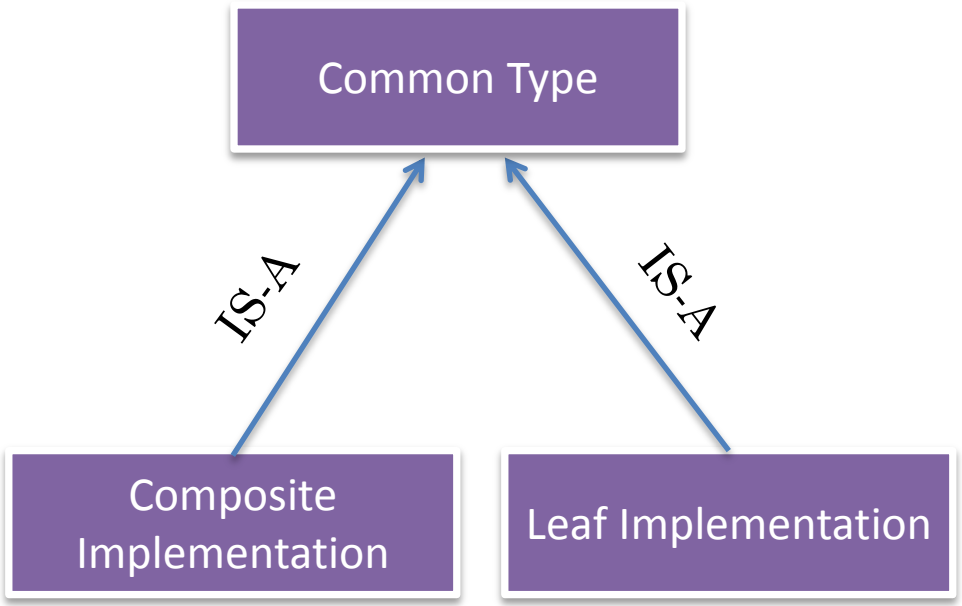
User Interfaces





Can use @Visible annotation to get rid of main panel properties

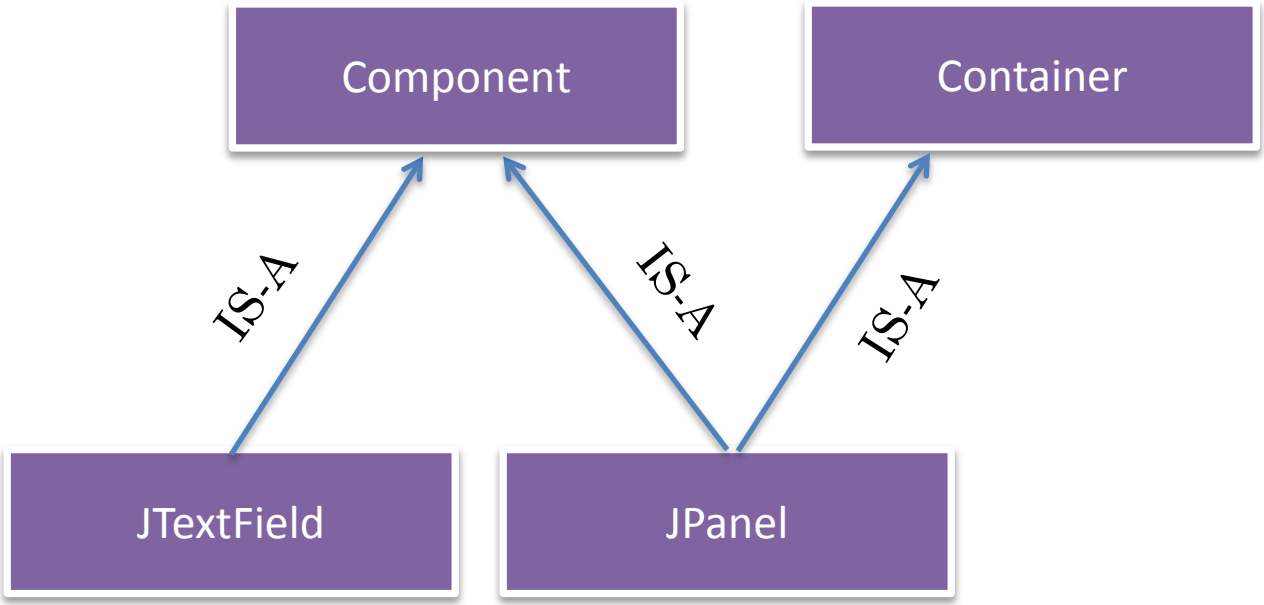
Key: Composite Design Pattern



Usually implemented with inheritance

Can do without inheritance

Java Toolkit Example



Usually implemented with inheritance

Can do without inheritance

Java Swing Example