Comp 401
Object and Shape Composition

Instructor: Prasun Dewan

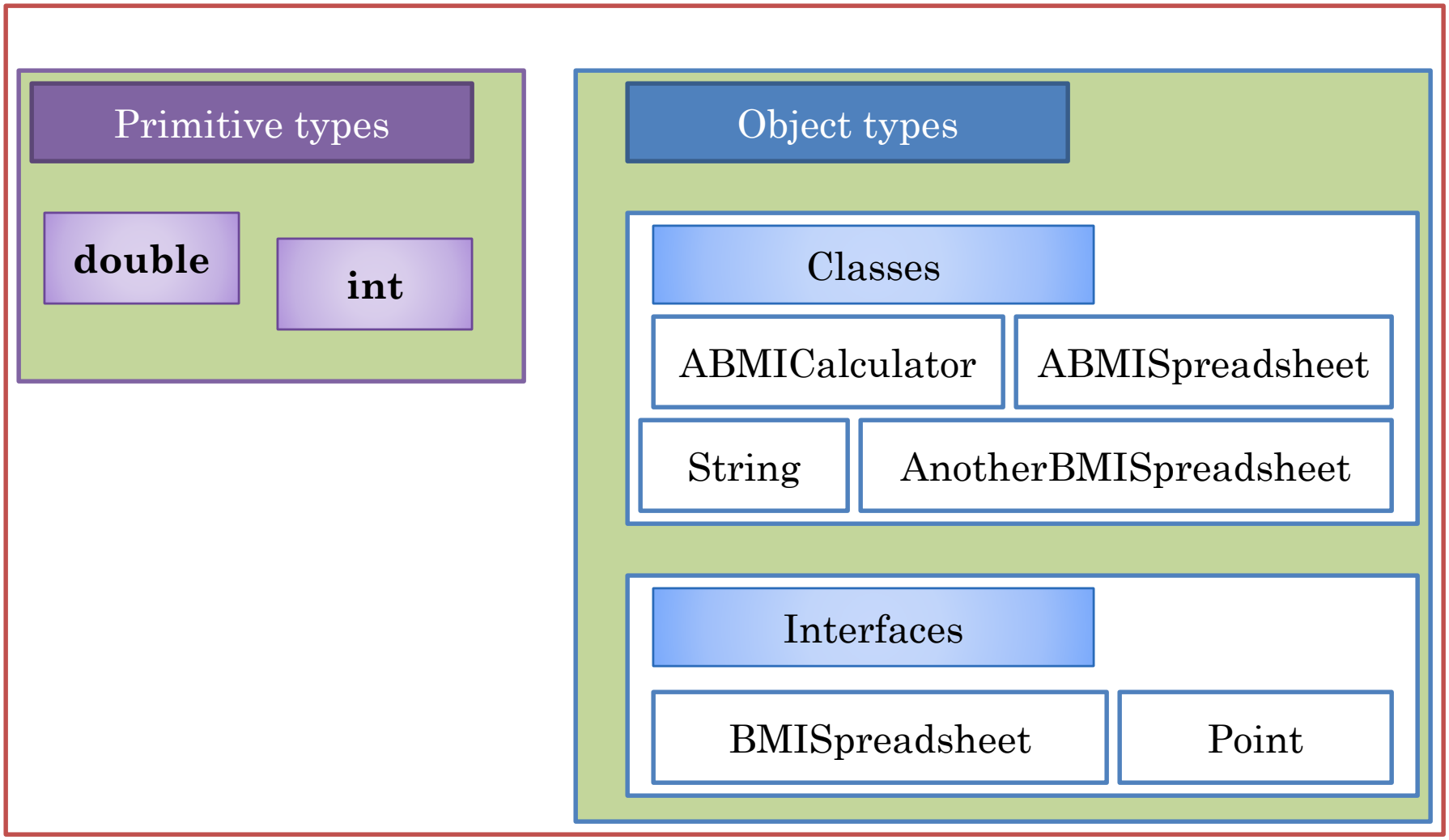
Prerequisite

- Interfaces
- Graphics

More on Objects

- Structure vs. atomic types
- Physical vs. logical structure

Primitive vs. Object Types



type = set of operations

Line Interface with Primitive Properties

```
public interface Line {
        public int getX();
        public void setX(int newX);
        public int getY();
        public void setY(int newY);
        public int getWidth();
        public void setWidth(int newVal);
        public int getHeight();
        public void setHeight(int newHeight);
}
```

Object rather than primitive properties?

LineWithObjectProperty Interface

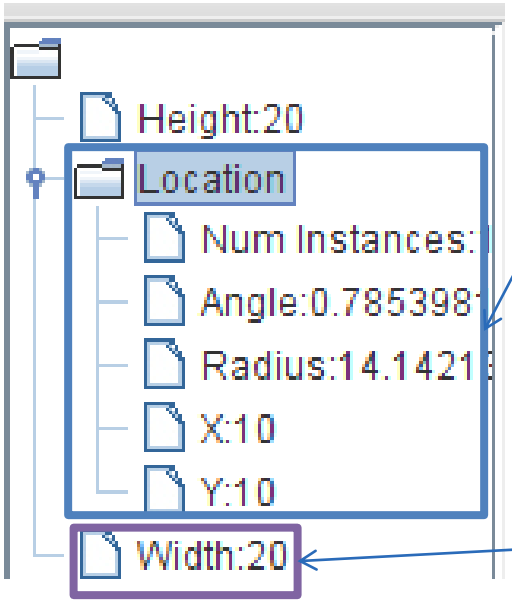
```
public interface LineWithObjectProperty {
        public Point getLocation();
        public void setLocation(Point newLocation);
        public int getWidth();
        public void setWidth(int newVal);
        public int getHeight();
        public void setHeight(int newHeight);
}
```

Object with object property == **Composite Object**

Primitive vs. Object Properties

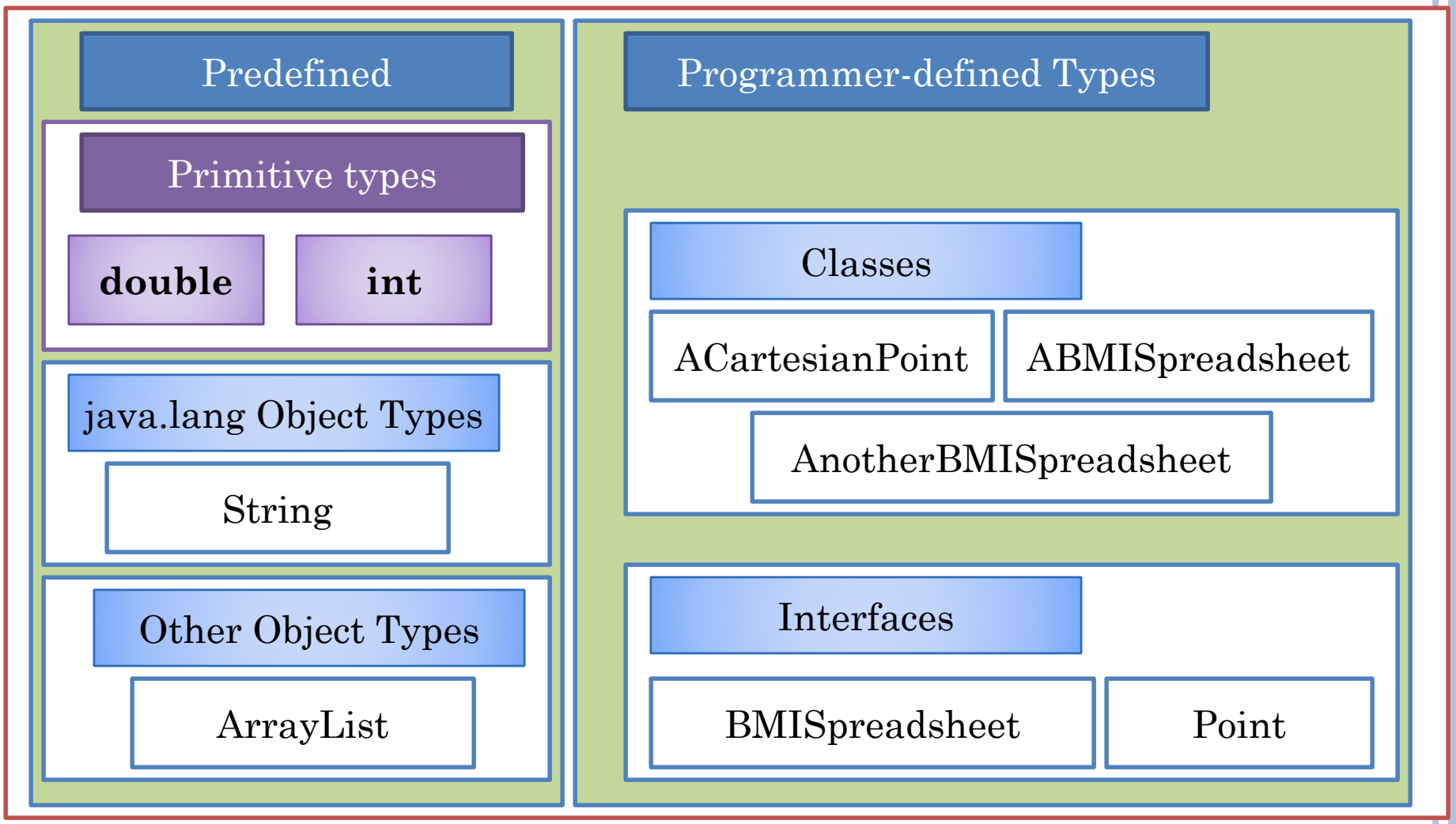
```
public class ALineWithObjectProperty implements LineWithObjectProperty {
        int width, height;
        Point location;
        public ALineWithObjectProperty (Point initLocation, int initWidth, int
initHeight) {
                location = initLocation;
                width = initWidth;
                height = initHeight;
        }
        public ALineWithObjectProperty() { }
        public Point getLocation() {return location;}
        public void setLocation(Point newVal) {location = newVal;}
        public int getWidth() {return width;}
        public void setWidth(int newVal) {width = newVal;}
        public int getHeight() {return height;}
        public void setHeight(int newHeight) {height = newHeight;}
}
```

Predefined, Primitive vs. Programmer-Defined, Object



Point	Object	Programmer-defined	Structured
int	Primitive	Predefined	Atomic

Programmer-defined vs. Predefined

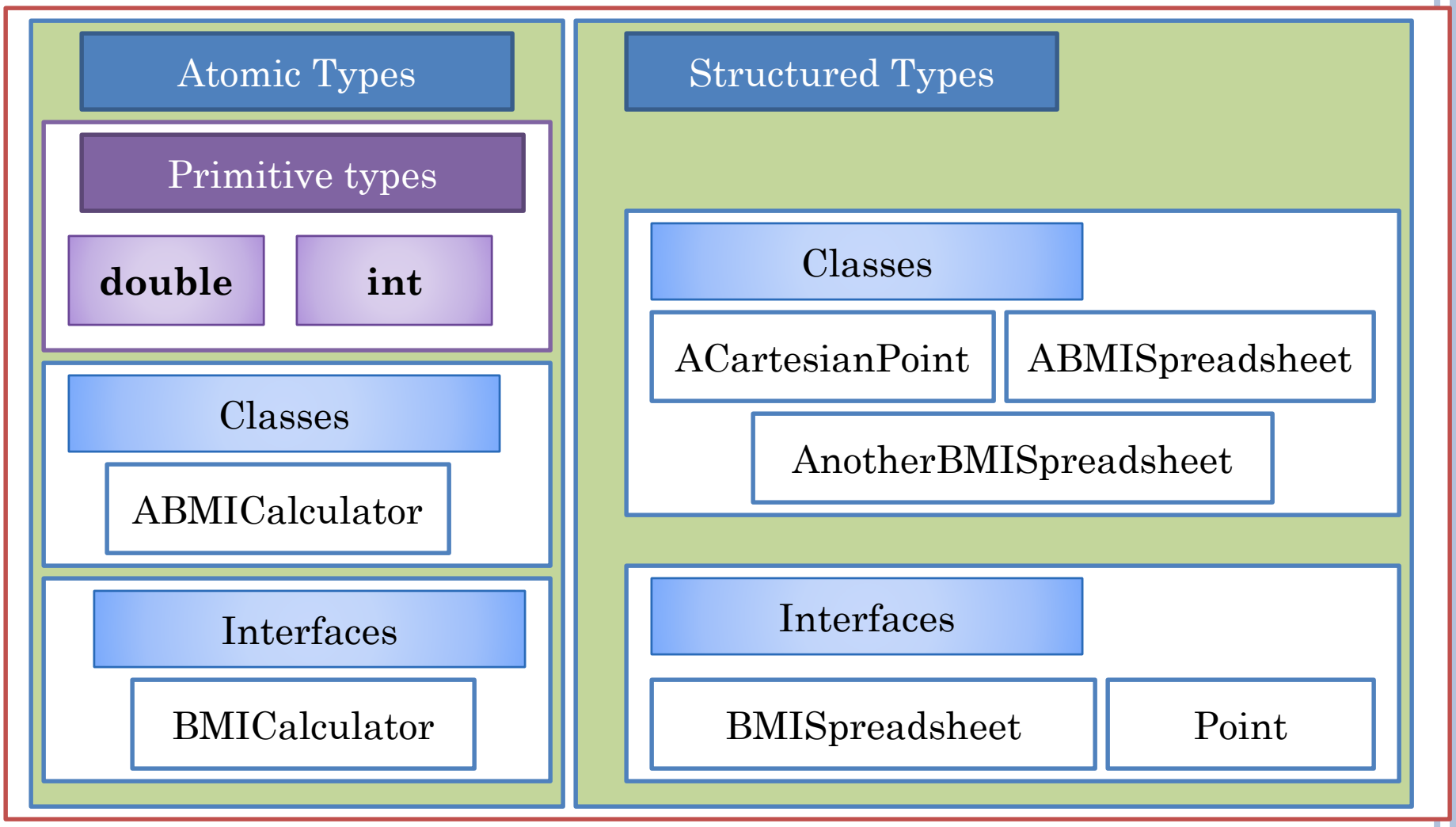


Java.lang types do not have to be imported

Programmer-defined vs. Predefined Types

- Programmer-defined interface/class (Point/ACartesianPoint) is programmer-defined type
- Programmer-defined types in Java must be object (or enum) types
- Some object types are predefined
  - Could be predefined by Java (in java.lang)
    - String
    - Integer
  - Could be predefined by Java library
    - ArrayList
    - Vector
- All primitive types are predefined

Structured vs. Atomic Types



Instances of structure type decomposed into one or more smaller values

Decomposing ALineWithObjectProperty Instance

```
public class ALineWithObj          rty implements LineWithObjectProperty {
        int width, height;
        Point location;
        public ALineWithObjectProperty (Point intLocation, int initWidth, int
initHeight) {
                location = initLocation;
                width = initWidth;
                height = initHeight;
        }
        public ALineWithObjectProperty() { }
        public Point getLocation() {return location;}
        public void setLocation(Point newVal) {location = newVal;}
        public int getWidth() {return width;}
        public void setWidth(int newVal) {width = newVal;}
        public int getHeight() {return height;}
        public void setHeight(int newHeight) {height = n          t;}
}
```

Initialized to 0s

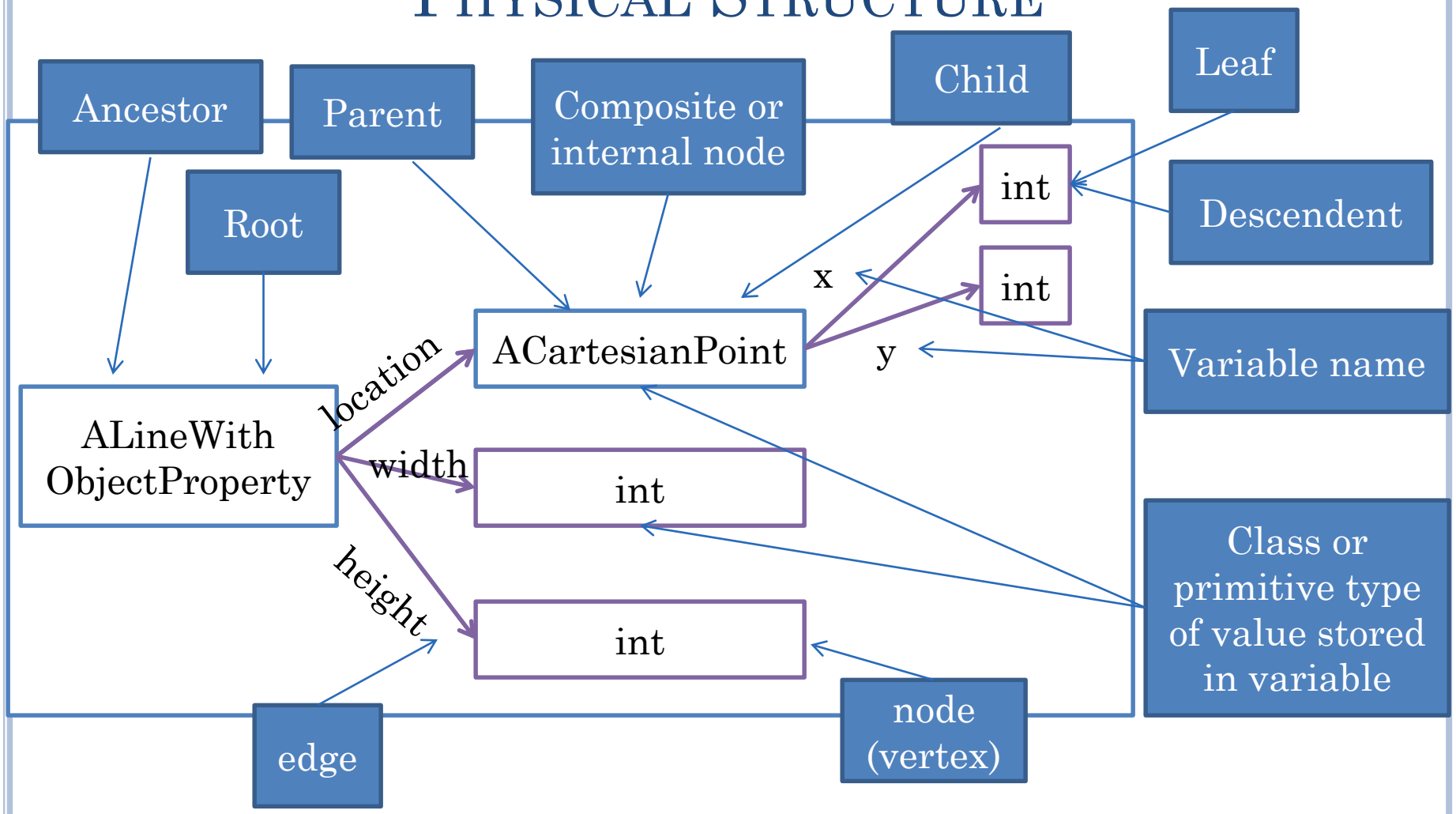
Initialized to null

Composition illustrated

```
new ALineWithObjectProperty(new ACartesianPoint (10, 10), 20, 20)
```

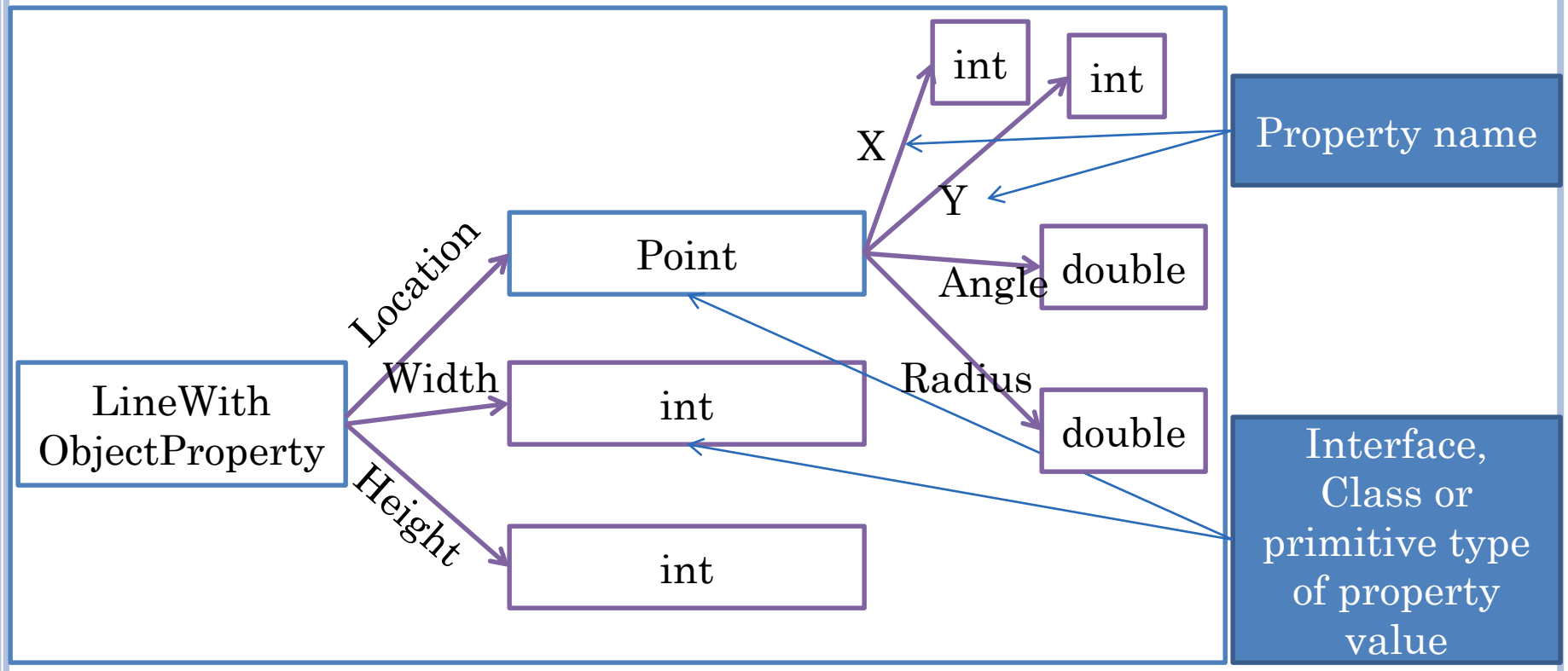
```
new ALineWithObjectProperty()
```

Physical Structure



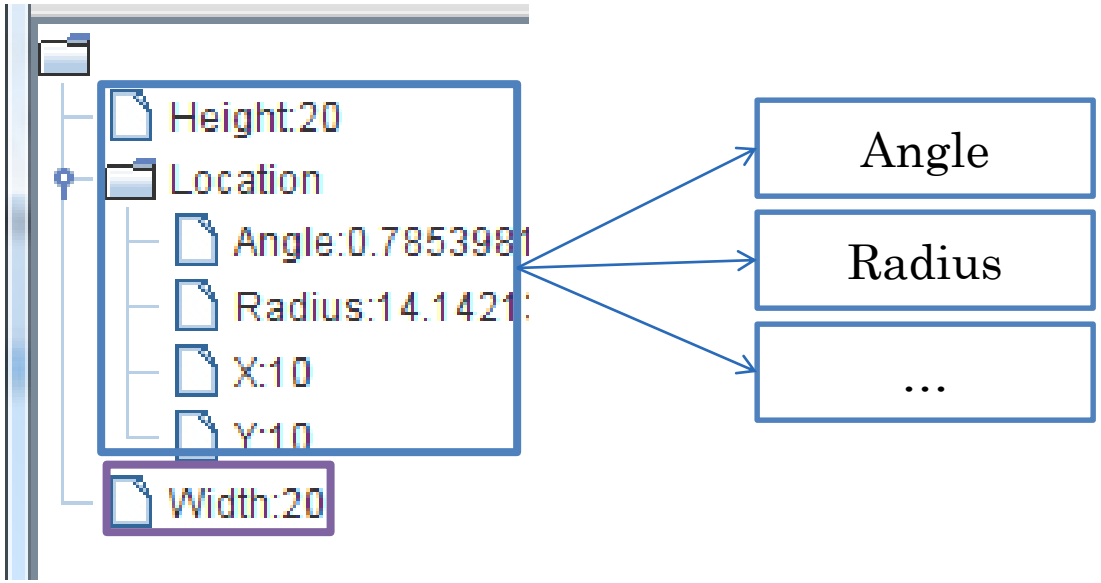
new ALineWithObjectProperty(new ACartesianPoint (10, 10), 20, 20)

Logical Structure



new ALineWithObjectProperty(**new** ACartesianPoint (10, 10), 20, 20)

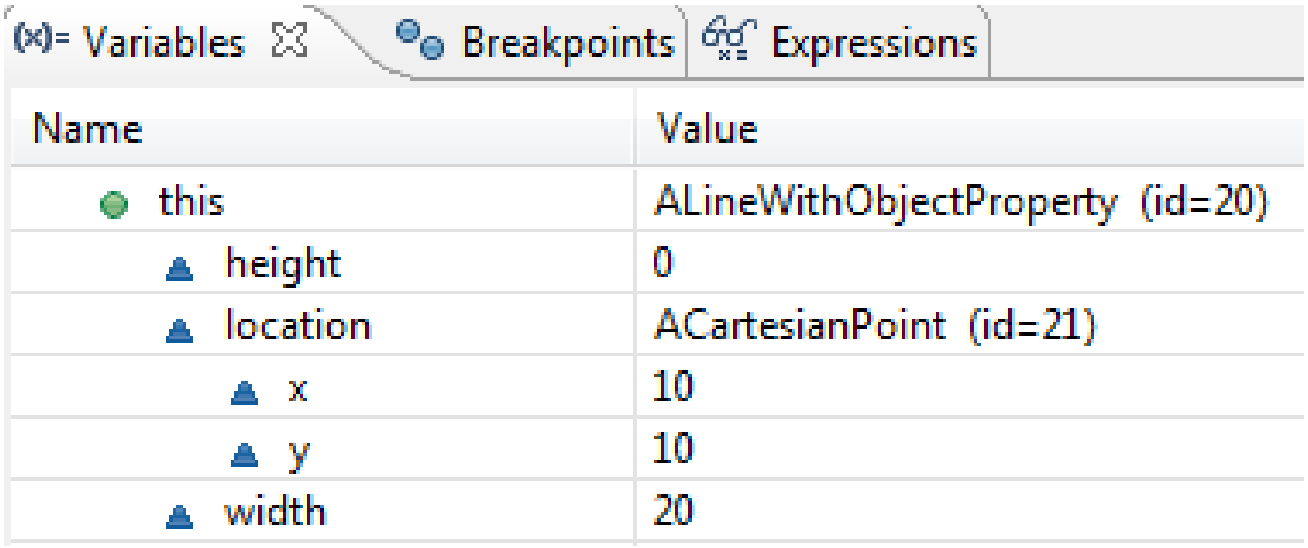
Logical Structure Shown by ObjectEditor



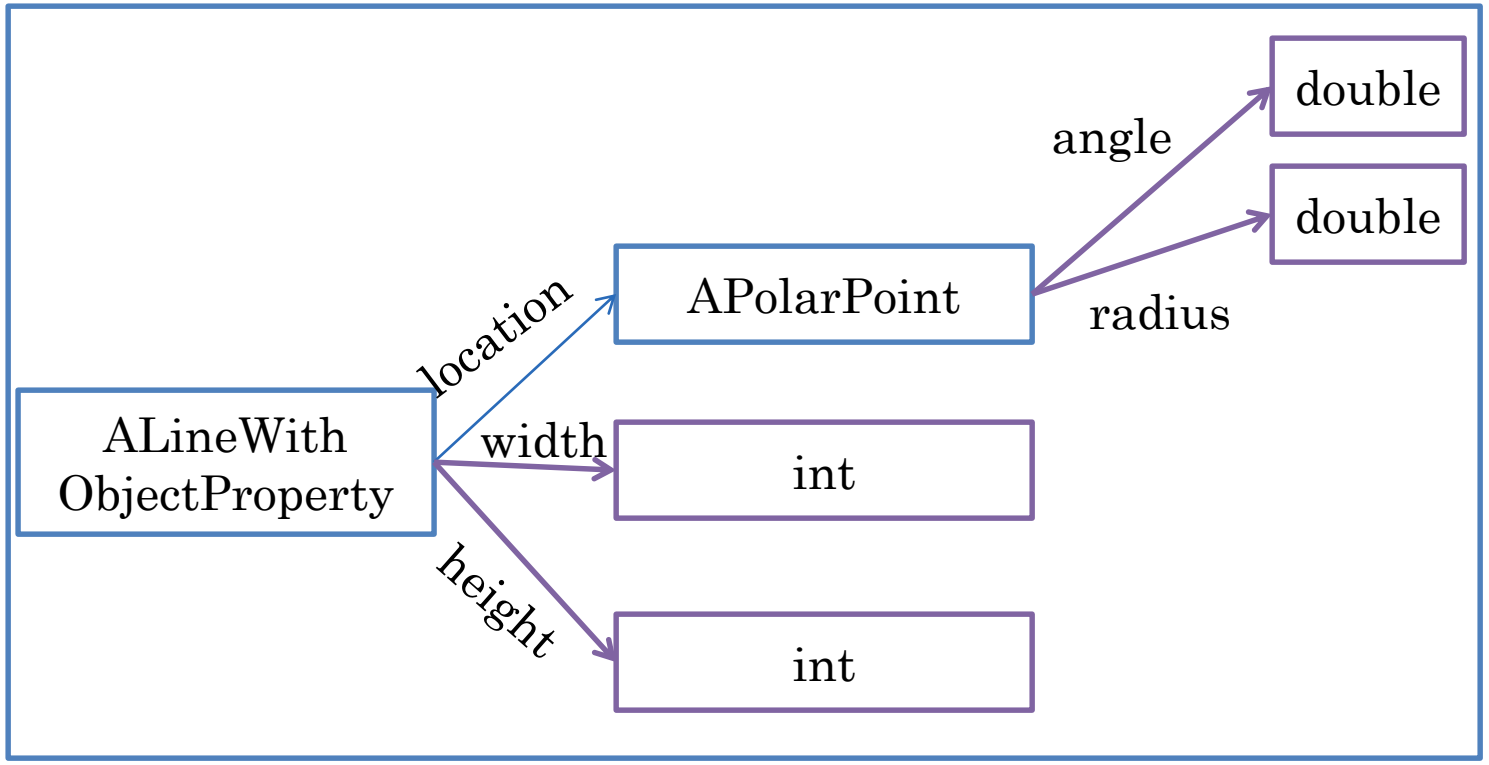
ObjectEditor cannot see instance variables

Tool for showing physical structure?

Physical Structure Shown by Debugger

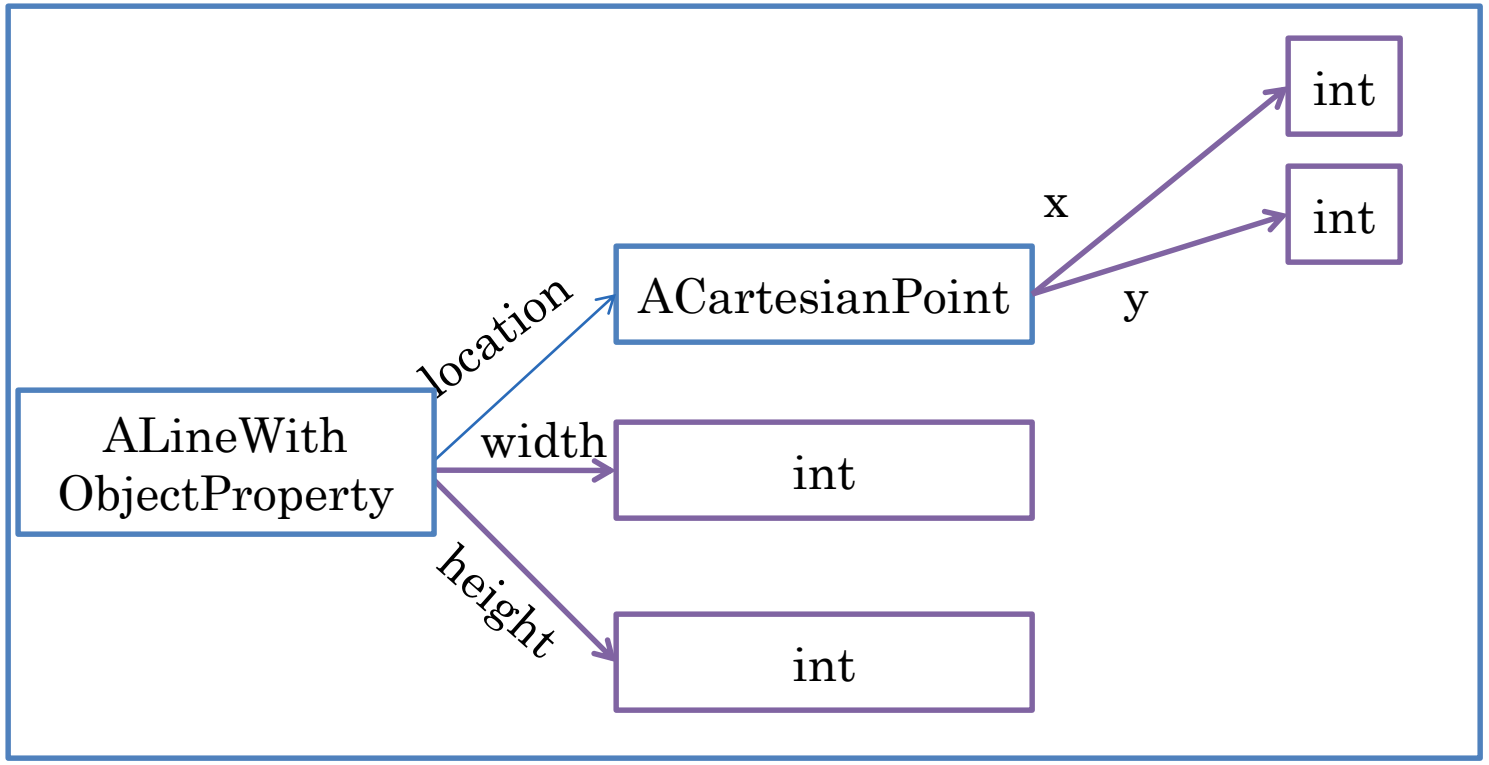


Instance-Specific Structure



new ALineWithObjectProperty(**new** APolarPoint (14.01, 0.78), 20, 20)

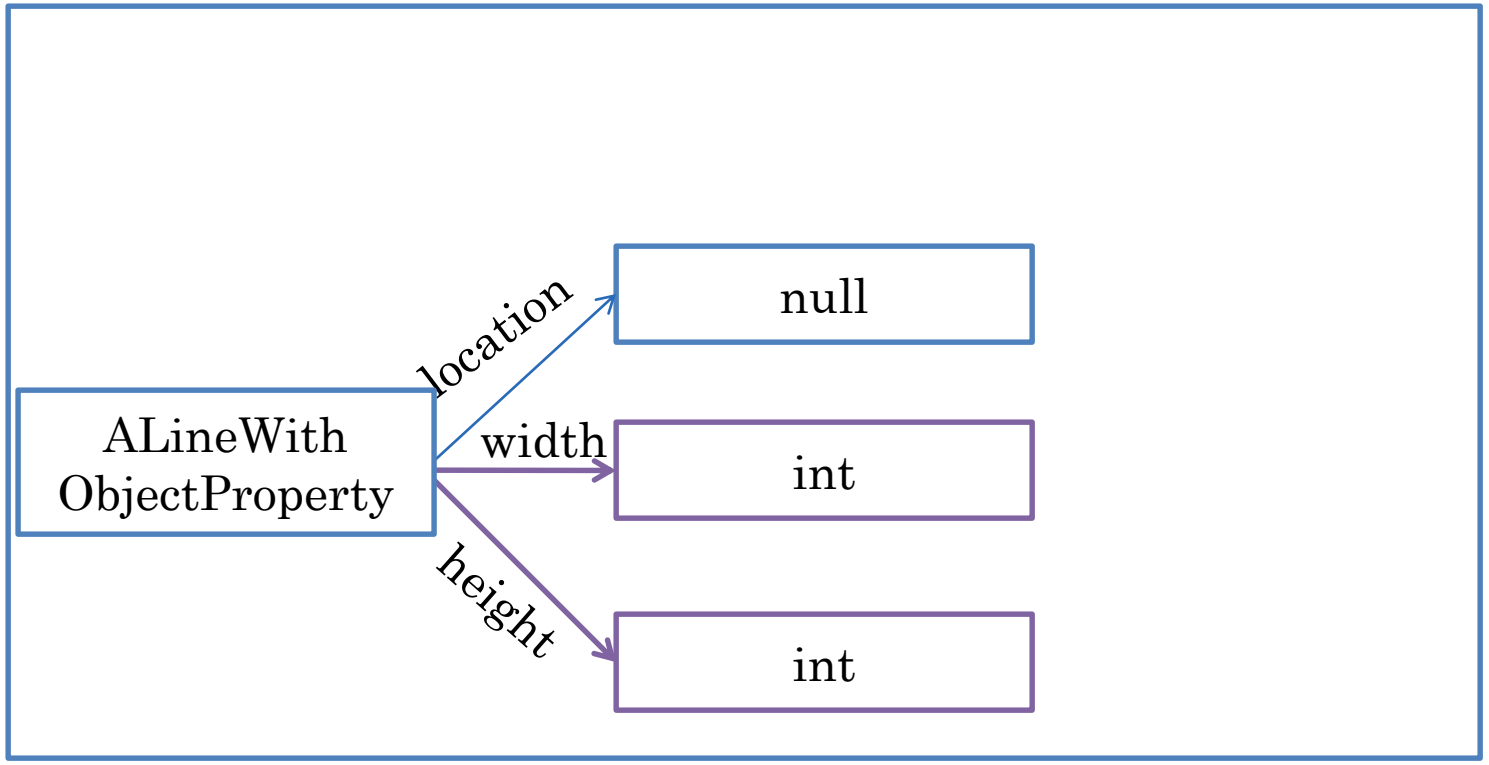
Instance-Specific Structure



new ALineWithObjectProperty(**new** ACartesianPoint (10, 10), 20, 20)

Structures of instances of same class can be different!

Null Component




```
E***Null value stored as location of:lectures.composition.AnAnotherLine@17d2f0e. Assuming coordinates of 0, 0
```

new ALineWithObjectProperty(**null**, 20, 20)

Object (but not primitive) properties can lead to null pointer exceptions

Computer Data Structures

- Connect items called **nodes** with directed lines called **edges**.
- Start with node with no incoming edge called **root**
- Nodes with no outgoing edge called **leafs**
- Non leaf or root node called **internal node** or **composite** nodes
- If a node a has a direct edge to another node b, then a is a **parent** of b and b is a **child** of a
- If a is a parent of b and b is a parent of c, then a is an **ancestor** of c and c is a **descendent** of a

Two Ways to Decompose Objects

- Logical and physical structures decompose objects
- Physical Decomposition
  - components of objects are its instance variables
    - x, y, location, width, size
  - the component can be further decomposed
    - location into x, y
- Logical Decomposition
  - components of object are its properties
    - Location, X, Y, Radius, Angle, Width, Height
  - each of these can be further decomposed
    - Location can be decomposed into X, Y, Angle, Radius

Drawing the Physical Structure of a Value of Some Type

- Start with the name of the type of root value
  - type can be class or primitive type
  - cannot be interface because no associated physical representation
- For each instance variable of the value
  - draw an edge
  - end point of edge is primitive type/class of value assigned to variable if value is non-null, and null otherwise
  - label of edge is variable name
- Follow the same algorithm to physically decompose value of each instance variable
- Stop at value of atomic type or a value already drawn in the figure (in case of recursive structures where a component points to an ancestor)

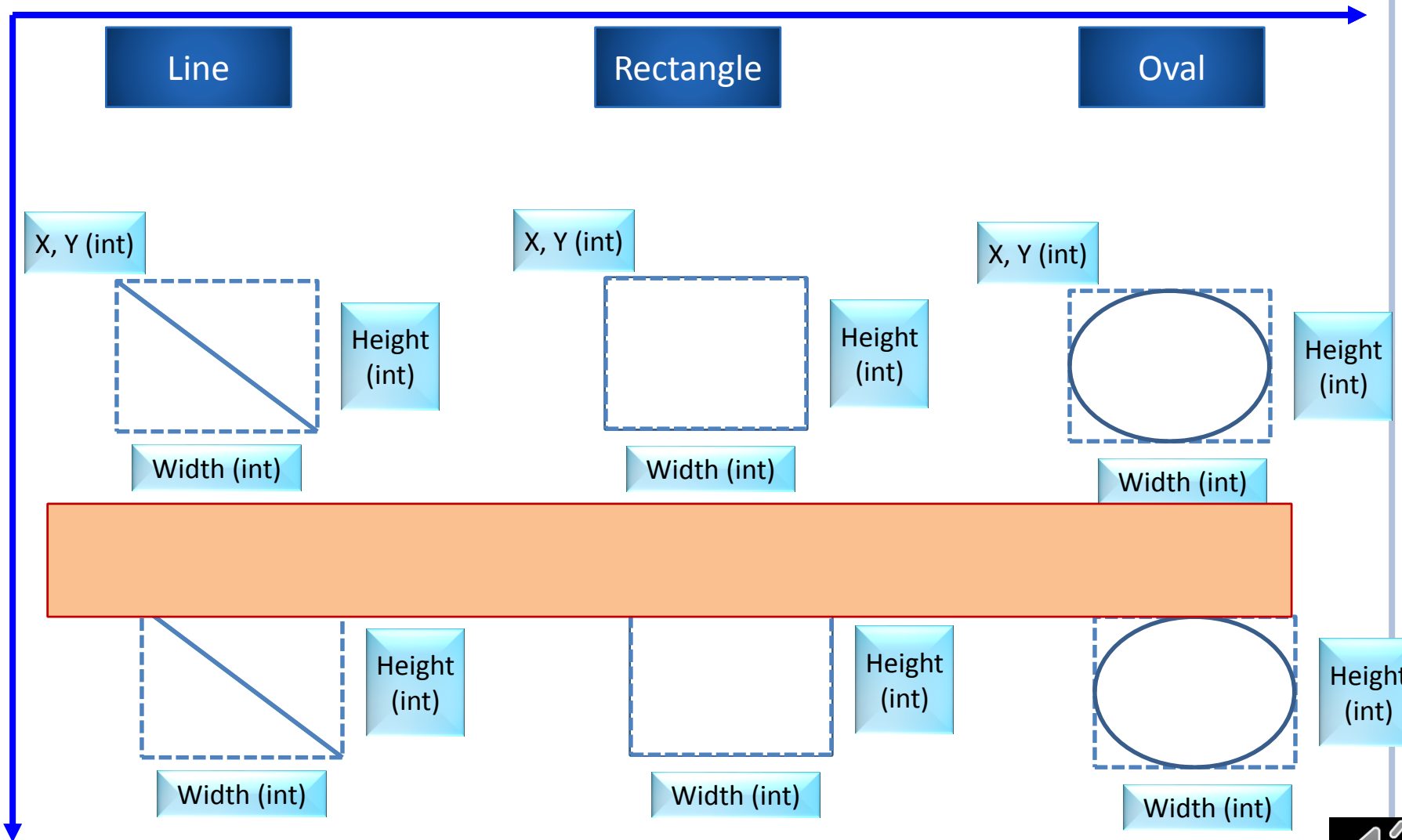
Drawing the Logical Structure of a Value of Some Type

- Start with the name of the class/primitive type of root value
- For each property defined by the type
  - draw an edge
  - end point of edge is primitive type/class of property value if not null and null otherwise
  - label of edge is property name
- Follow the same algorithm to logically decompose value of each property
- Stop at value of atomic type, or value drawn in the figure (in case of recursive structures where a component points to an ancestor)

Atomic vs. Structured Types

- All primitive types are atomic
- Object types with >= 1 properties are logically structured
- Object types with >= 1 instance variables are physically structured
- Some object types may be physically or logically atomic

ObjectEditor Shape Rules



The type of Location is any type following point rules given ea

XY-based String and Shape Shape Rules

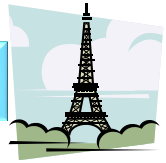
String Shape

X, Y (int)

Text hello

Image Shape

X, Y (int)

ImageFileName 

Text hello

ImageFileName 

Creating Other Geometric Objects

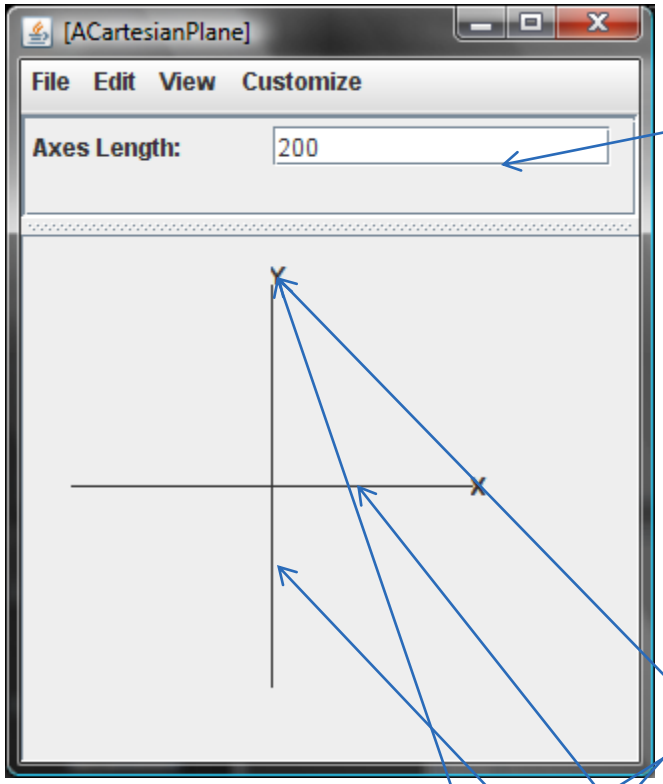
- Polygon, triangle, ….
- Compose the existing geometric objects into a logically structured object
  - Point, Line, Rectangle, Oval, TextBox, Icon
  - Arc, Curves added in latest ObjectEditor
- Any graphical image can be created from triangles!

Lines of Composition

Lines can be composed into triangles

Triangles can be composed to create arbitrary computer graphics

ACartesianPlane



Text Property

Axes Length (int)

XAxis (Line)

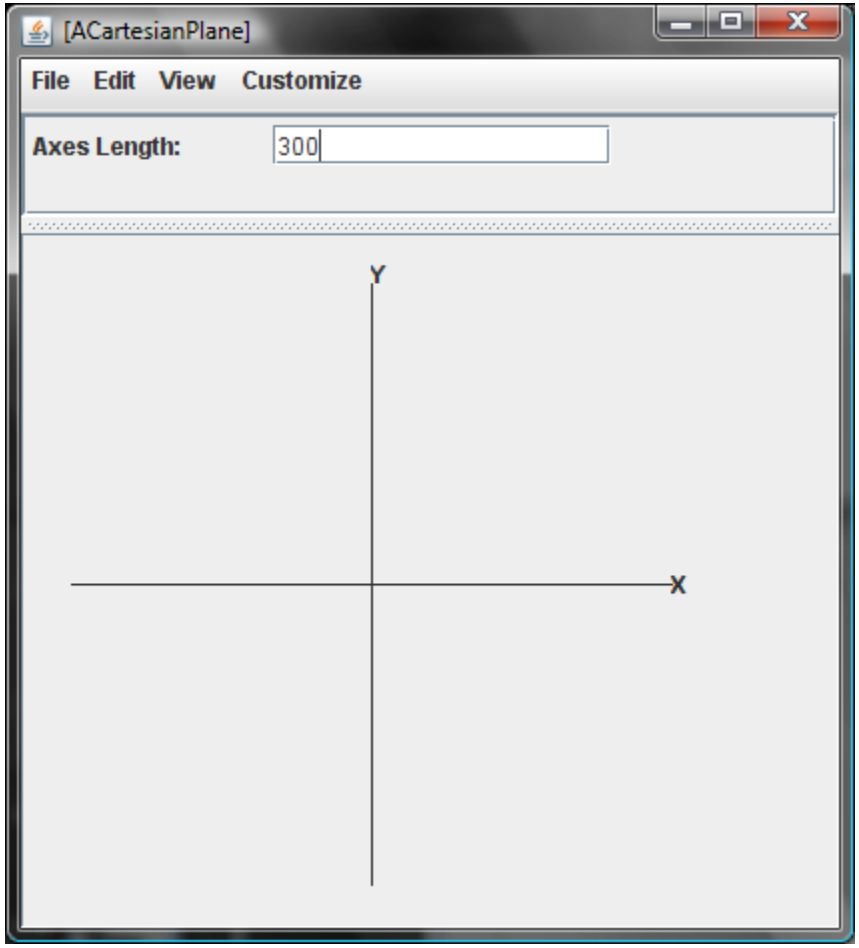
YAxis (Line)

XLabel (Label)

YLabel (Label)

Graphics Property

Changing the Length



Axes Length (int)

XAxis (Line)

YAxis (Line)

XLabel (StringShape)

YLabel (StringShape)

Cartesian plane retains the same origin

Other properties are dependent on the axes length property

ACartesianPlane : Constructor

```
public class ACartesianPlane implements CartesianPlane {
  int originX, originY;
  int axesLength;
  Line xAxis;
  Line yAxis;
  StringShape xLabel;
  StringShape yLabel;
  public ACartesianPlane (int theAxesLength,
                          int theOriginX, int theOriginY ) {
    axesLength = theAxesLength;
    originX = theOriginX;
    originY = theOriginY;
    xAxis = new ALine(toXAxisX(), toXAxisY(), axesLength, 0);
    yAxis = new ALine(toYAxisX(), toYAxisY(), 0, axesLength);
    xLabel = new AStringShape ("X", toXLabelX(), toXLabelY());
    yLabel = new AStringShape ("Y", toYLabelX(), toYLabelY());
 }
```

ACartesianPlane: Properties

```
public Line getXAxis() {return xAxis;}
public Line getYAxis() {return yAxis;}
public StringShape getXLabel() {return xLabel;}
public StringShape getYLabel() {return yLabel;}
public int getAxesLength() {return axesLength;}
public void setAxesLength(int anAxesLength) {
  axesLength = anAxesLength;
  xAxis.setWidth(axesLength);
  yAxis.setHeight(axesLength);
  xAxis.setX(toXAxisX());
  xAxis.setY(toXAxisY());
  yAxis.setX(toYAxisX());
  yAxis.setY(toYAxisY());
  xLabel.setX(toXLabelX());
  xLabel.setY(toXLabelY());
  yLabel.setX(toYLabelX());
  yLabel.setY(toYLabelY());
 }
```

ACartesianPlane: Dependencies

```
int toXAxisX() {
  return originX - axesLength/2;
}
int toXAxisY() {
  return originY;
}
int toYAxisX() {
  return originX;
}
int toYAxisY() {
  return originY - axesLength/2;
}
int toXLabelX() {
  return originX + axesLength/2;
}
int toXLabelY() {
  return originY;
}
int toYLabelX() {
  return originX;
}
int toYLabelY() {
  return originY - axesLength/2;
}
```

Code Duplication:Constructor vs. Set

```
  public ACartesianPlane (int theAxesLength,
                          int theOriginX, int theOriginY ) {
    axesLength = theAxesLength;
    originX = theOriginX;
    originY = theOriginY;
    xAxis = new ALine(toXAxisX(), toXAxisY(), axesLength, 0);
    yAxis = new ALine(toYAxisX(), toYAxisY(), 0, axesLength);
    xLabel = new AStringShape ("X", toXLabelX(), toXLabelY());
    yLabel = new AStringShape ("Y", toYLabelX(), toYLabelY());
 }
```

```
public void setAxesLength(int anAxesLength) {
    axesLength = anAxesLength;
    xAxis.setWidth(axesLength);
    yAxis.setHeight(axesLength);
    xAxis.setX(toXAxisX());
    xAxis.setY(toXAxisY());
    yAxis.setX(toYAxisX());
    yAxis.setY(toYAxisY());
    xLabel.setX(toXLabelX());
    xLabel.setY(toXLabelY());
    yLabel.setX(toYLabelX());
    yLabel.setY(toYLabelY());
   }
```

In ACartesianPlane constructor, could instantiate dependent objects with null constructors or constructors with wrong values and then simply call setAxesLength

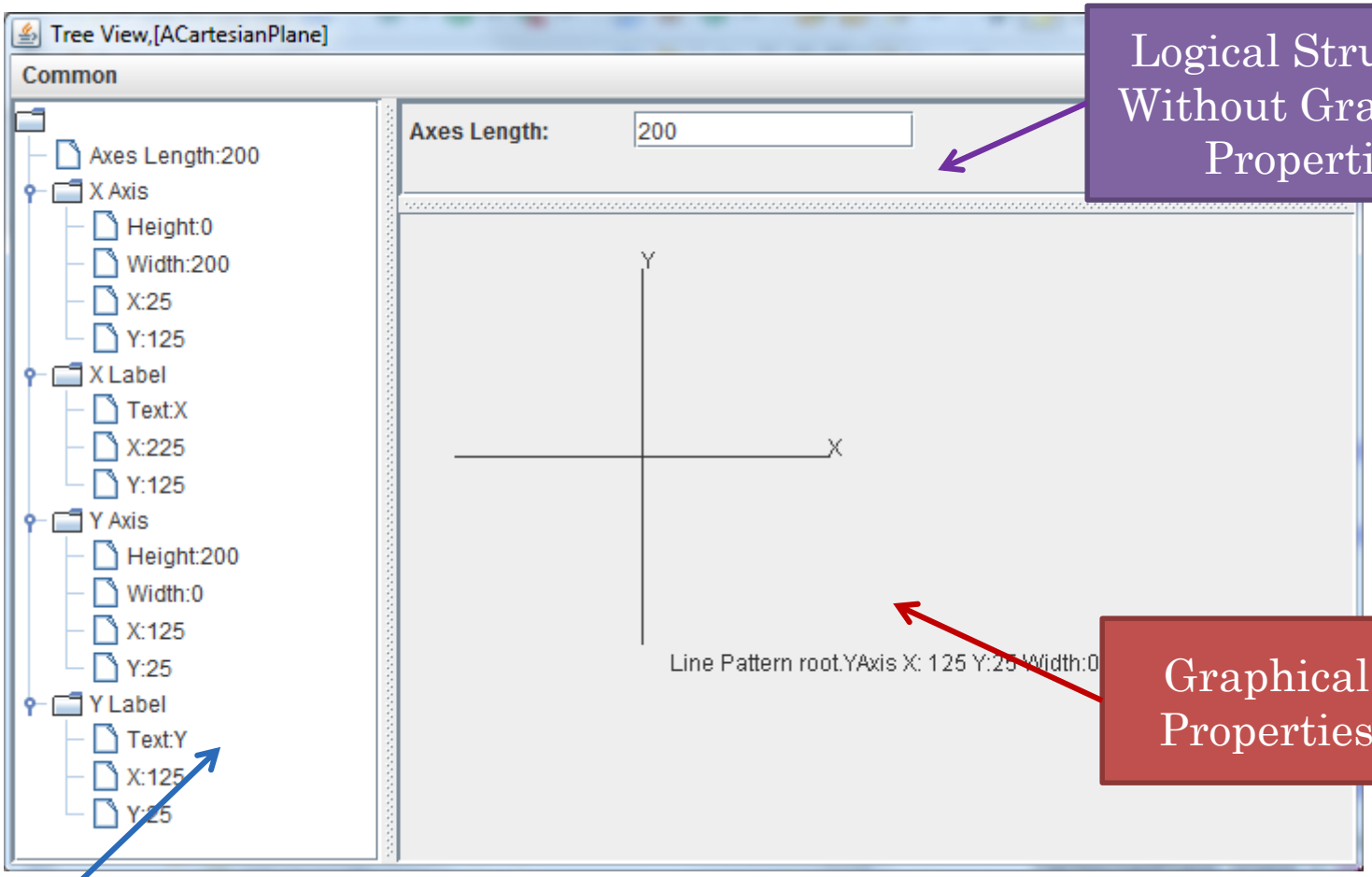
No Code Duplication

```
  public ACartesianPlane (int theAxesLength,
                          int theOriginX, int theOriginY ) {
    originX = theOriginX;
    originY = theOriginY;
    xAxis = new ALine();
    yAxis = new ALine();
    xLabel = new AStringShape ();
    yLabel = new AStringShape ();
    setAxesLength(theAxesLength);
  }
```

```
public void setAxesLength(int anAxesLength) {
    axesLength = anAxesLength;
    xAxis.setWidth(axesLength);
    yAxis.setHeight(axesLength);
    xAxis.setX(toXAxisX());
    xAxis.setY(toXAxisY());
    yAxis.setX(toYAxisX());
    yAxis.setY(toYAxisY());
    xLabel.setX(toXLabelX());
    xLabel.setY(toXLabelY());
    yLabel.setX(toYLabelX());
    yLabel.setY(toYLabelY());
   }
```

In ACartesianPlane constructor, could instantiate dependent objects with null constructors or constructors with wrong values and then simply call setAxesLength

ACartesianPlane Logical Structure



Complete Logical Structure

```
p          oid main (String[] args) {
           ane plane = new ACartesianPlane(200, 125, 125);
           r.edit(plane);
}
```

ACartesianPlane : Constructor

```
public class ACartesianPlane implements CartesianPlane {
  int originX, originY;
  int axesLength;
  Line xAxis;
  Line yAxis;
  StringShape xLabel;
  StringShape yLabel;
  public ACartesianPlane (int theAxesLength,
                          int theOriginX, int theOriginY ) {
    axesLength = theAxesLength;
    originX = theOriginX;
    originY = theOriginY;
    xAxis = new ALine(toXAxisX(), toXAxisY(), axesLength, 0);
    yAxis = new ALine(toYAxisX(), toYAxisY(), 0, axesLength);
    xLabel = new AStringShape ("X", toXLabelX(), toXLabelY());
    yLabel = new AStringShape ("Y", toYLabelX(), toYLabelY());
 }
```

ACartesianPlane: Properties

```
public Line getXAxis() {return xAxis;}
public Line getYAxis() {return yAxis;}
public StringShape getXLabel() {return xLabel;}
public StringShape getYLabel() {return yLabel;}
public int getAxesLength() {return axesLength;}
public void setAxesLength(int anAxesLength) {
  axesLength = anAxesLength;
  xAxis.setWidth(axesLength);
  yAxis.setHeight(axesLength);
  xAxis.setX(toXAxisX());
  xAxis.setY(toXAxisY());
  yAxis.setX(toYAxisX());
  yAxis.setY(toYAxisY());
  xLabel.setX(toXLabelX());
  xLabel.setY(toXLabelY());
  yLabel.setX(toYLabelX());
  yLabel.setY(toYLabelY());
 }
```

Inefficient CartesianPlane

```
public class AnInefficientCartesianPlane implements CartesianPlane {
  int originX, originY;
  int axesLength;
  public AnInefficientCartesianPlane (int theAxesLength,
                int theOriginX, int theOriginY ) {
    axesLength = theAxesLength;
    originX = theOriginX;
    originY = theOriginY;
  }
  public Line getXAxis() {
    return new ALine(toXAxisX(), toXAxisY(), axesLength, 0);
  }
  public Line getYAxis() {
    return new ALine(toYAxisX(), toYAxisY(), 0, axesLe
  }
  public StringShape getXLabel() {
    return new AStringShape ("X", toXLabelX(), toXLabelY());
  }
  public StringShape getYLabel() {
    return new AStringShape ("Y", toYLabelX(), toYLabelY());
  }
  public void setAxesLength(int anAxesLength) {
    axesLength = anAxesLength;
  }
  …
```

New objects constructed on each refresh!

Much simpler setAxesLength

Uniting Structure Graphics and Types

Structured Types

APlottedShuttle

ACartesianPlane

ALine

AStringShape

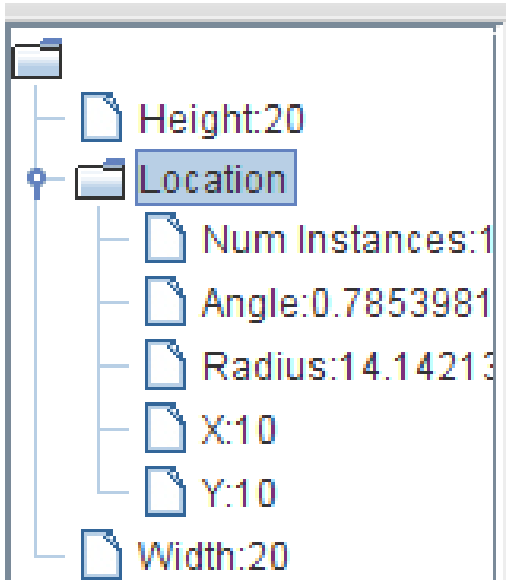
AnImageWithHeight

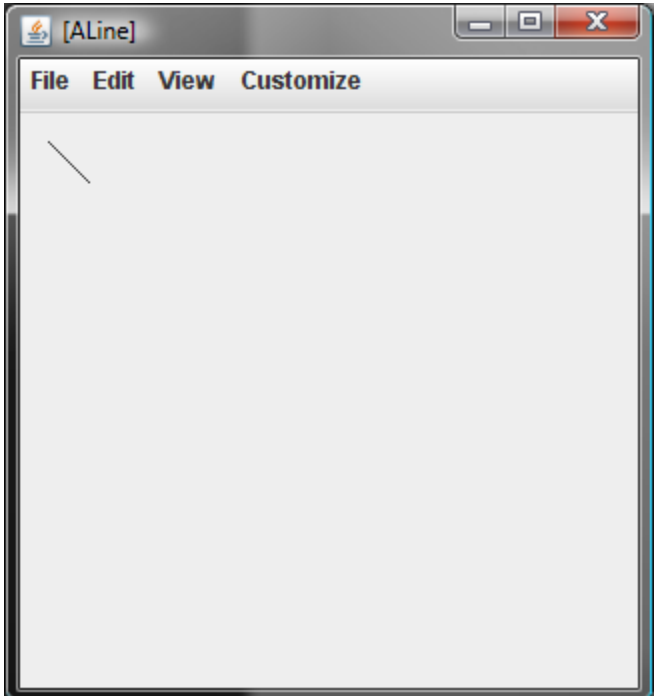
Composite Object/Shape vs. Atomic Object/Shape

Composite object	Object with one or more object logical components
Non composite object	Object with zero or more primitive logical components
Atomic (wrapper) object	Object with one primitive logical component
Composite Shape	Object with one or more shape (geometric/graphic) components
Atomic Shape	Shape that cannot be decomposed into component shapes

Atomic Shape, Composite Object

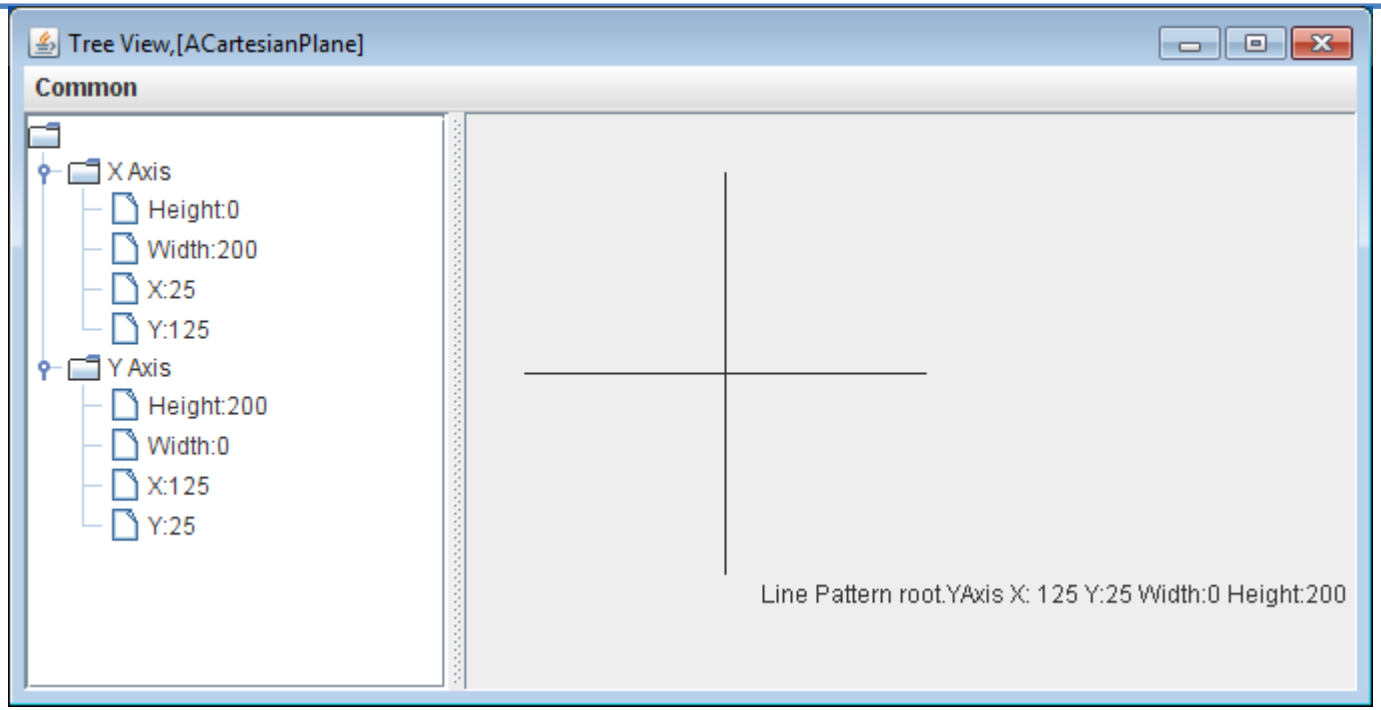
```
public interface LineWithObjectProperty {
        public Point getLocation();
        public void setLocation(Point newLocation);
        public int getWidth();
        public void setWidth(int newVal);
        public int getHeight();
        public void setHeight(int newHeight);
}
```





Composite Shapes and object

```
public interface CartesianPlane {
    public Line getXAxis();
    public Line getYAxis();
}
```



Mutable Object Properties in Atomic Shapes

```
public interface LineWithObjectProperty {
        public Point getLocation();
        public int getWidth();
        public void setWidth(int newVal);
        public int getHeight();
        public void setHeight(int newHeight);
}
```

None in Composite Shapes

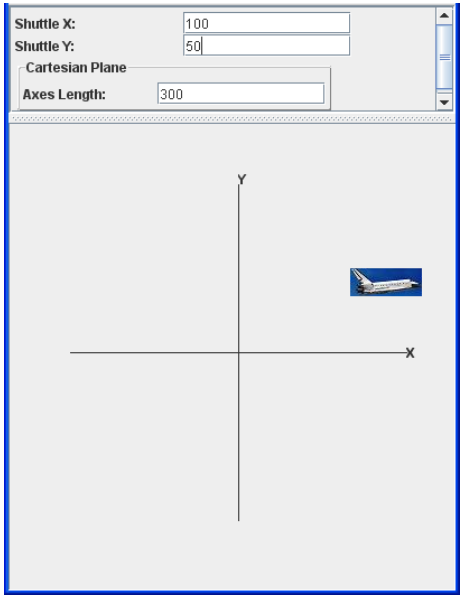
```
public interface CartesianPlane {
  public int getAxesLength();
  public void setAxesLength(int newVal);
  public Label getXLabel();
  public Label getYLabel();
  public CartesianPlane getCartesianPlane();
}
```

Changes in composite shape usually take place through changes in their atomic descendants

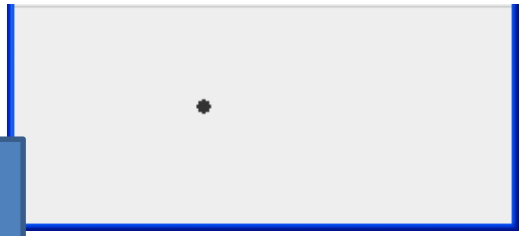
```
public interface ShuttleLocation {
  public FancyCartesianPlane getCartesianPlane();
  public ImageLabel getShuttleLabel();
  public int getShuttleX();
  public void setShuttleX(int newVal);
  public int getShuttleY();
  public void setShuttleY(int newVal);
}
```

ObjectEditor Graphics Rules

```
public interface ShuttleLocation {
  public FancyCartesianPlane
        getCartesianPlane();
  public ImageLabel getShuttleLabel();
  public int  getShuttleX();
  public void setShuttleX(int newVal);
  public int  getShuttleY();
  public void setShuttleY(int newVal);
}
```




```
public interface NotAPoint {
  public FancyCartesianPlane
        getCartesianPlane();
  public ImageLabel getShuttleLabel();
  public int  getX();
  public void setX(int r
  public int  getY();
  public void setY(int r
  public Point getLocati
}
```



Type interpreted as a Point as its name contains "Point" and has X and Y Properties

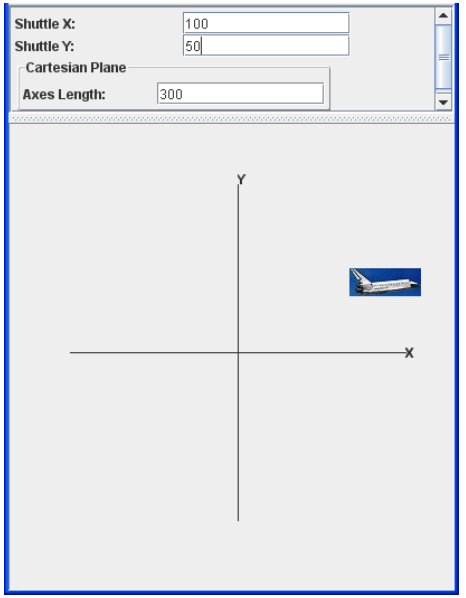
ObjectEditor Point Rules

- An object is recognized as a point representation if:
  - Its interface or class has the string "Point" in its name or has a Point annotation
  - It has (read-only) int properties, X and Y, representing Cartesian window coordinates
  - Can have additional properties

```
@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point {
        public int getX();
        public int getY();
        public double getAngle();
        public double getRadius();
}
```

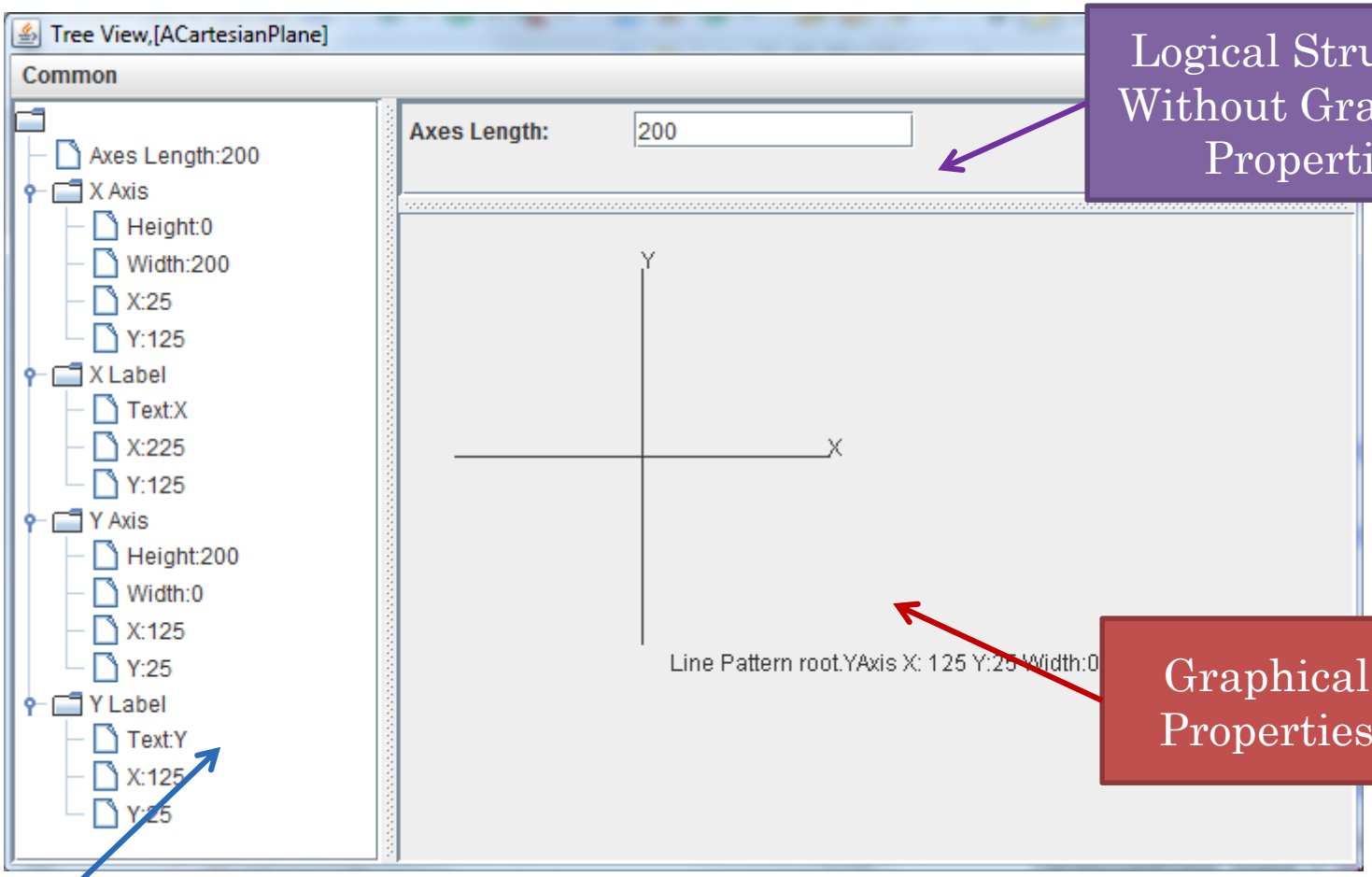
IS_ATOMIC_SHAPE CLASS ANNOTATION

```
import util.annotations.IsAtomicShape;
@IsAtomicShape(false)
//same as AShuttleLocation except interface name
public class AShuttleLocationImplementingABadlyNamedInterface
                              implements NotAPoint {
```



IsAtomicShape(false) before class name says do not interpret the class as an atomic shape (it can be a composition)

Composite Shape (Review)



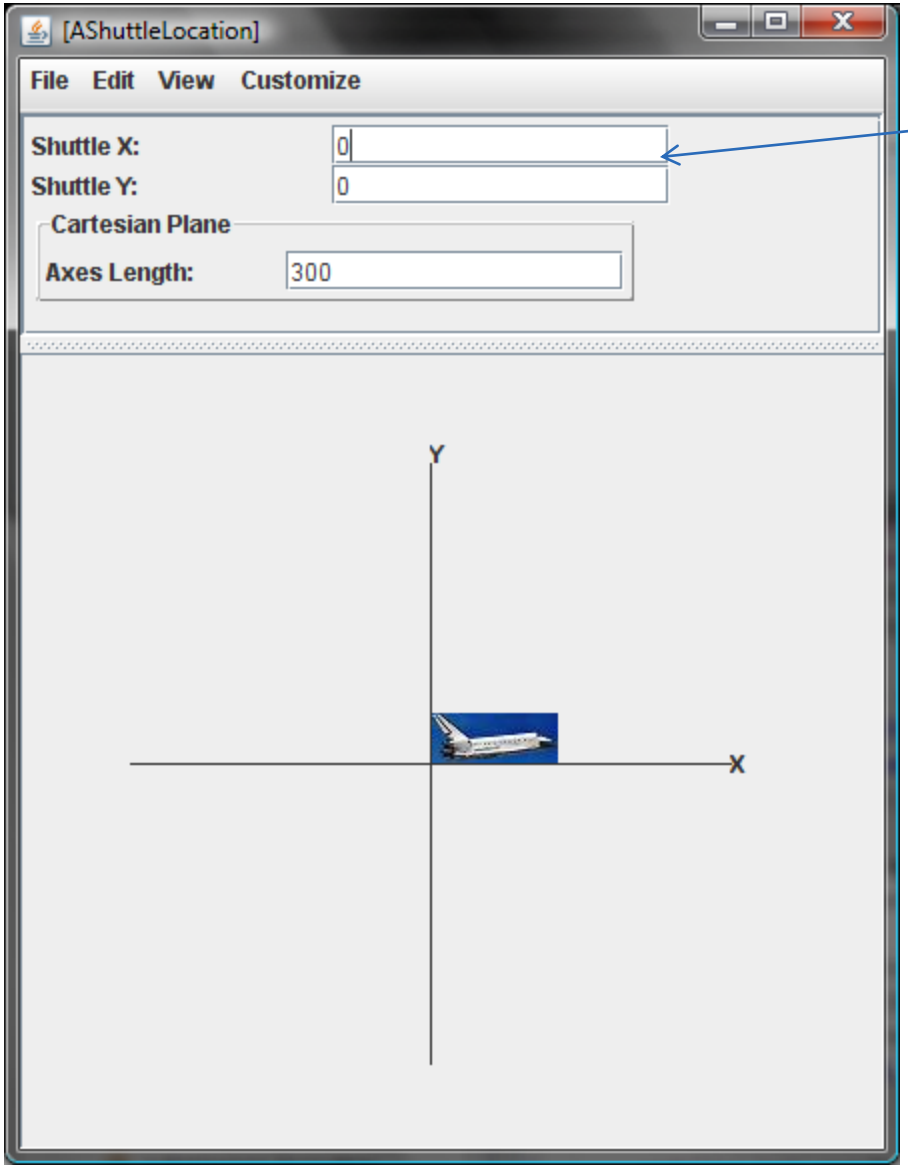
Logical Structure Without Graphical Properties

Graphical Properties

Complete Logical Structure

```
p          oid main (String[] args) {
                ane plane = new ACartesianPlane(200, 125, 125);
                r.edit(plane);
}
```

Plotted Shuttle



Cartesian, not Java coordinates, require translation

ShuttleX (int)

ShuttleY (int)

ShuttleImage (ImageWithHeight)

Axes Length (int)

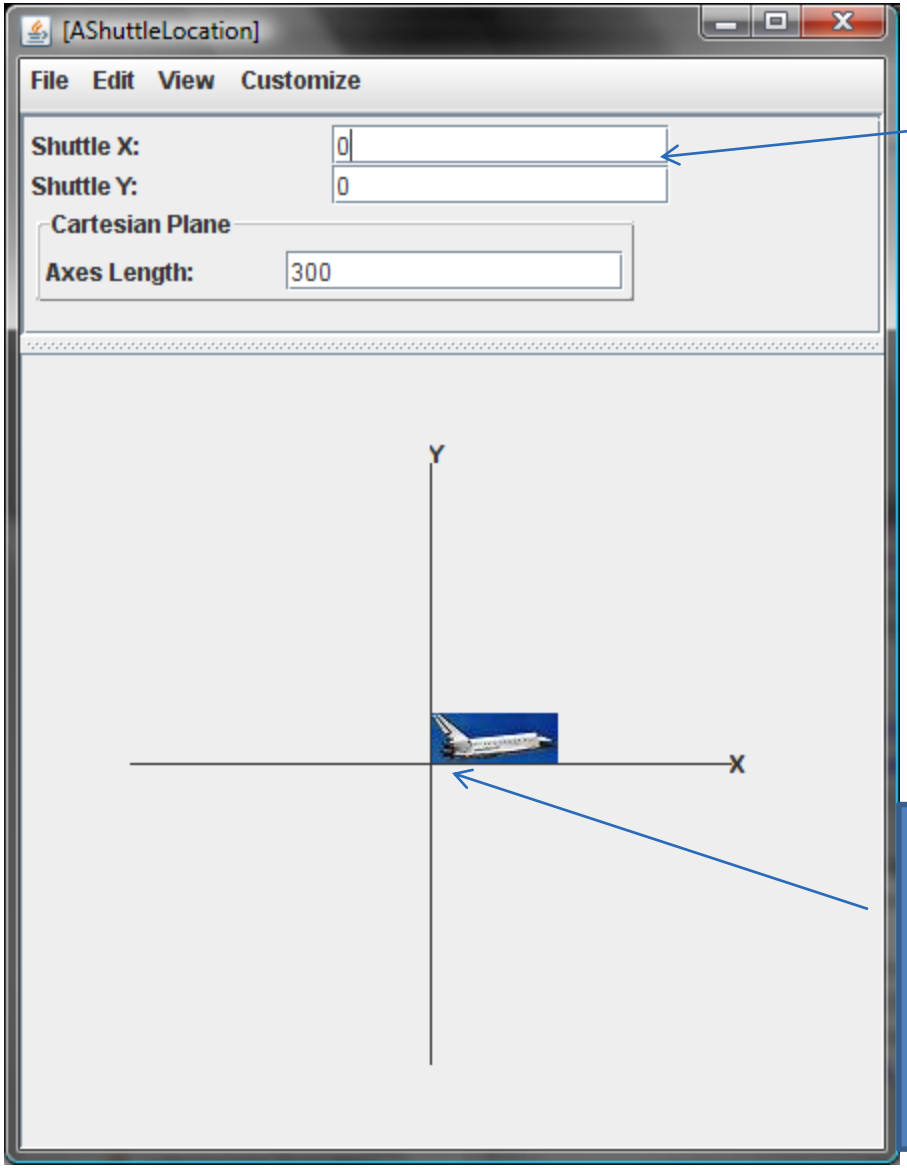
XAxis (Line)

YAxis (Line)

XLabel (StringShape)

YLabel (StringShape

Plotted Shuttle (Reuse)



Cartesian, not Java coordinates, require translation

ShuttleX (int)

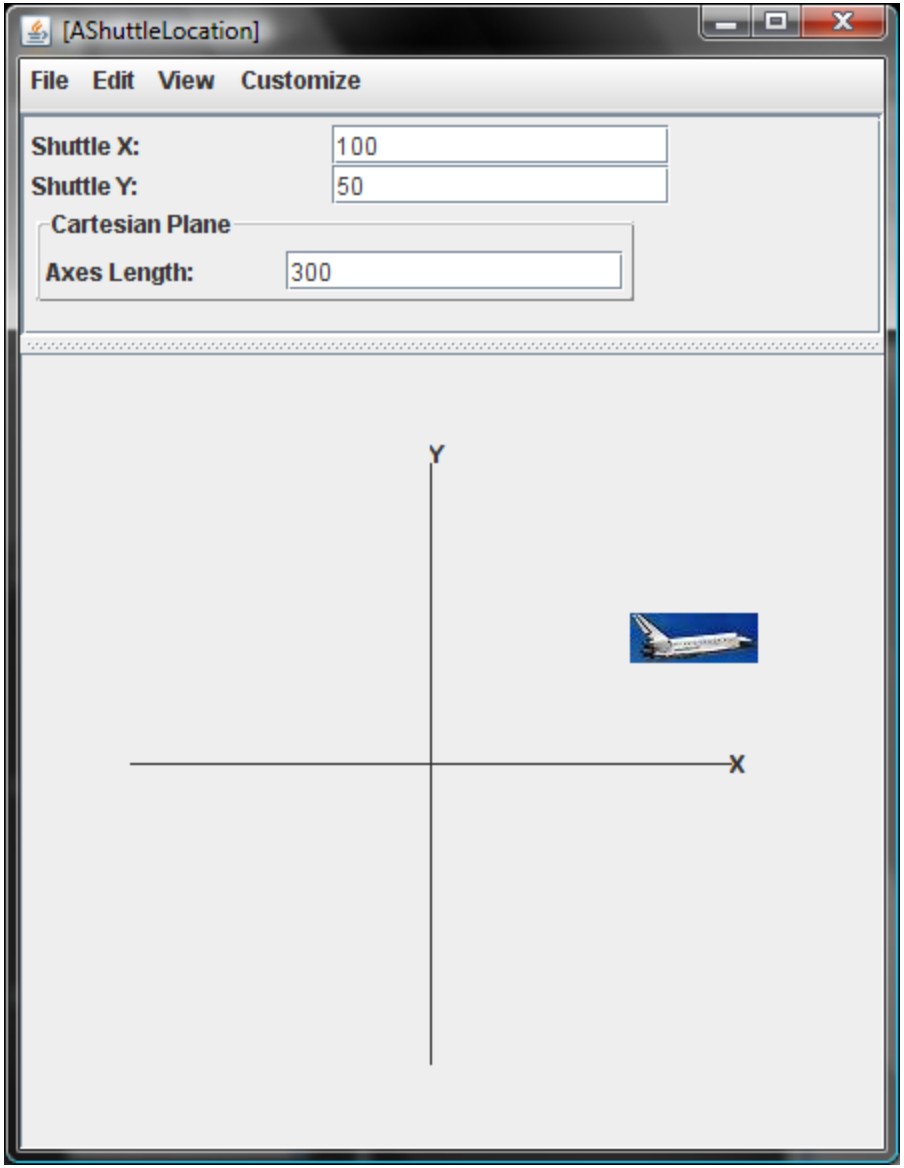
ShuttleY (int)

ShuttleImage (ImageWithHeight)

CartesianPlane (CartesianPlane)

Lower left rather than upper left corner is at (0, 0) so shuttle object needs to know the height of shuttle image

Plotted Shuttle



ShuttleX (int)

ShuttleY (int)

ShuttleImage (ImageWithHeight)

CartesianPlane (CartesianPlane)

ShuttleLabel depends on ShuttleX and ShuttleY

PLOTTEDSHUTTLE

```
public interface ShuttleLocation {
        public CartesianPlane getCartesianPlane();
        public ShuttleImage getShuttleImage();
        public int getShuttleX();
        public void setShuttleX(int newVal);
        public int getShuttleY();
        public void setShuttleY(int newVal);
}
```

APlottedShuttle: Constructor

```
public class APlottedShuttle implements PlottedShuttle {
  static final String SHUTTLE_IMAGE_FILE_NAME = "shuttle2.jpg";
  static final int ORIGIN_X = 200, ORIGIN_Y = 200;
  static final int AXES_LENGTH = 300;
  int shuttleX = 0, shuttleY = 0;
  CartesianPlane cartesianPlane;
  ImageWithHeight shuttleImage;
  public APlottedShuttle(int anX, int aY) {
    cartesianPlane = new ACartesianPlane (AXES_LENGTH,
                                          ORIGIN_X, ORIGIN_Y);
   shuttleImage = new AnImageWithHeight(SHUTTLE_IMAGE_FILE_NAME);
   setShuttleX(anX);
   setShuttleY(aY);
  }
```

APLOTTEDSHUTTLE: PROPERTIES

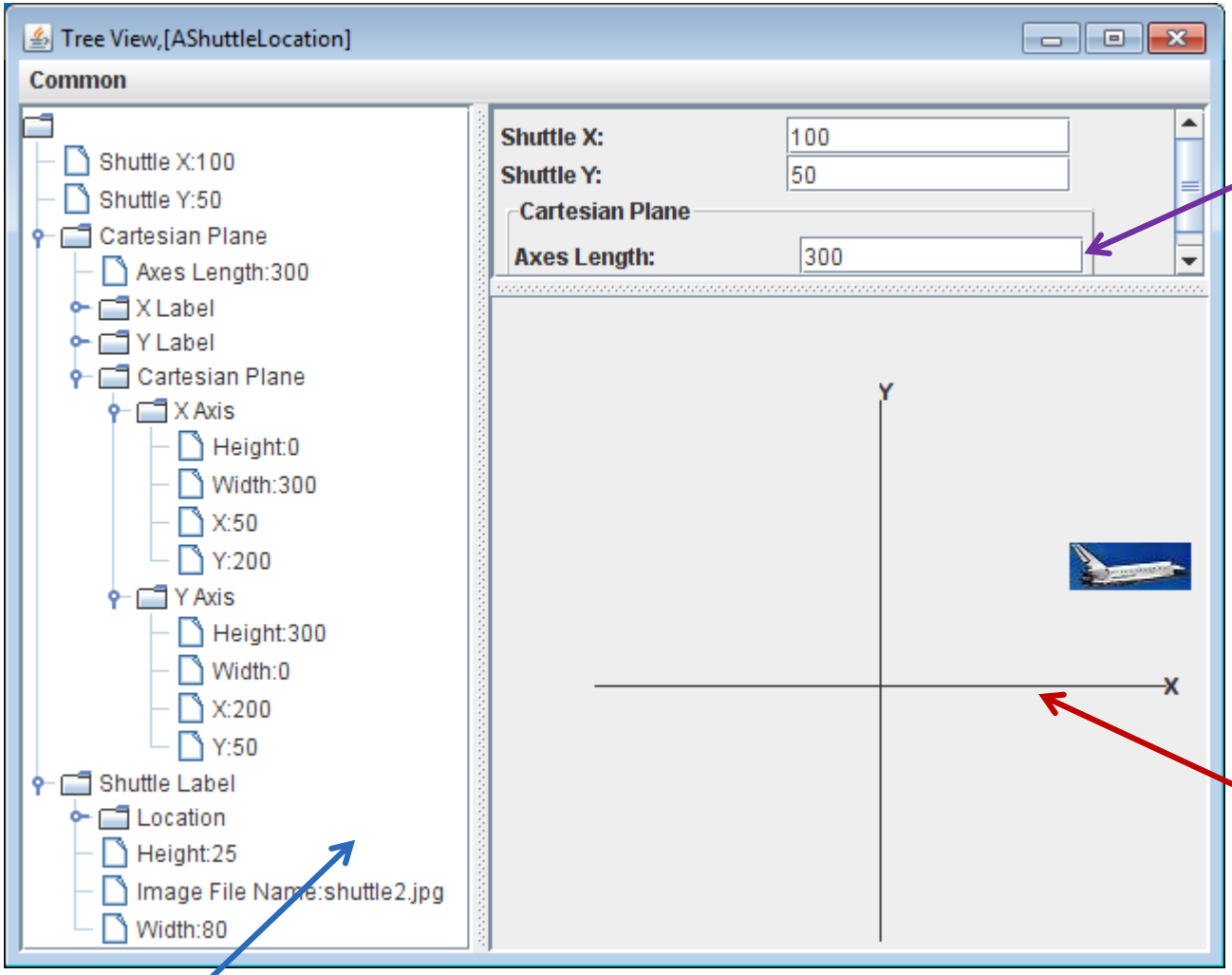
```
public CartesianPlane getCartesianPlane() {return cartesianPlane;}
public ShuttleImage getShuttleImage() {return shuttleImage;}
public int getShuttleX() {return shuttleX;}
public void setShuttleX(int newVal) {
  shuttleX = newVal;
  shuttleImage.setX(toWindowX());
}
public int getShuttleY() {return shuttleY;}
public void setShuttleY(int newVal) {
  shuttleY = newVal;
  shuttleImage.setY(toWindowY());
}
int toWindowX() {
  return ORIGIN_X + shuttleX;
}
int toWindowY() {
  return ORIGIN_Y - shuttleY -                    ;
}
```

Extra optional height property

AShuttleImage

```
public class AnImageWithHeight implements ImageWithHeight {
  int x, y;
  String imageFileName;
  int imageHeight;
  public AnImageWithHeight(String anImageFileName) {
    imageFileName = anImageFileName;
    Icon icon = new ImageIcon(imageFileName);
    imageHeight = icon.getIconHeight();
  }
  public int getX() {return x;}
  public void setX(int newX) {x = newX;}
  public int getY() { return y; }
  public void setY(int newY) {y = newY;}
  public String getImageFileName() {return imageFileName;}
  public int getHeight() { return  imageHeight;}
}
```

Alternate Views of Logical Structure



Logical Structure Without Graphical Properties

Graphical Properties

Complete Logical Structure