Comp 401
User Interface and Annotations

Instructor: Prasun Dewan

Prerequisites

- Interfaces

Interactive App. vs. User/Prog. Interface

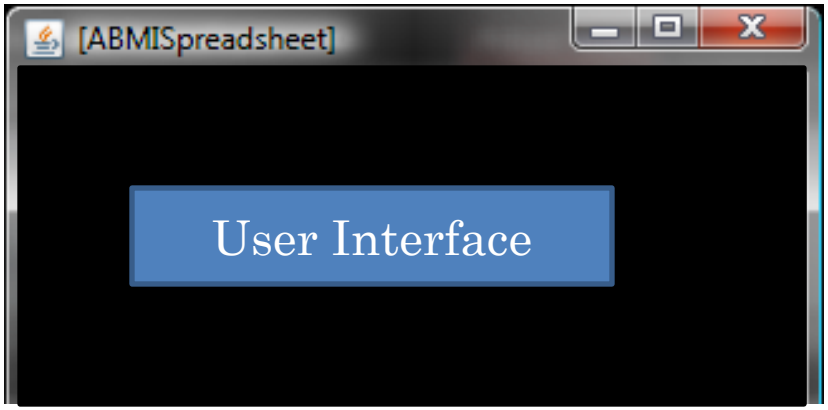
```
public class ABMISpreadsheet
implements BMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double
            newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double
            newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```

Service

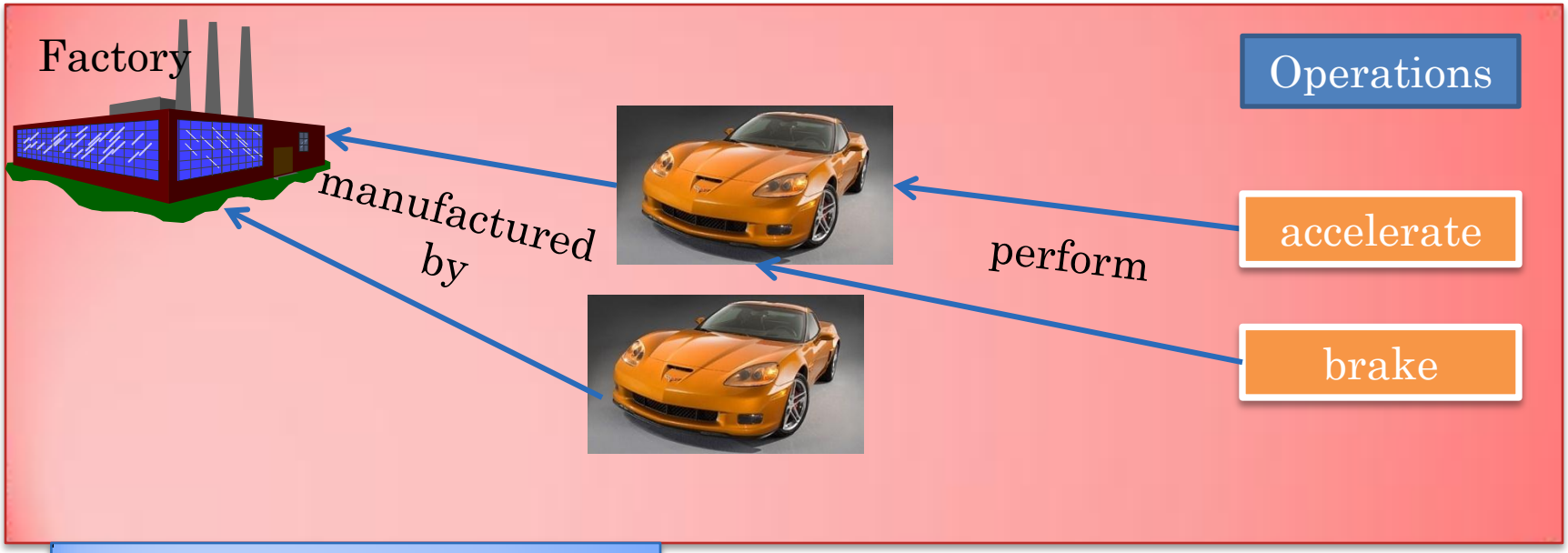
BMISpreadsheet.java

```
public interface BMISpreadsheet {

    public double getWeight();
    public void setWeight(double newVal);

    public double getHeight();
    public void setHeight(double newVal);

    public double getBMI();
}
```

Programming Interface



Programming vs. User Interface: Analogy



Programming Interface: Mechanism used to communicate with car

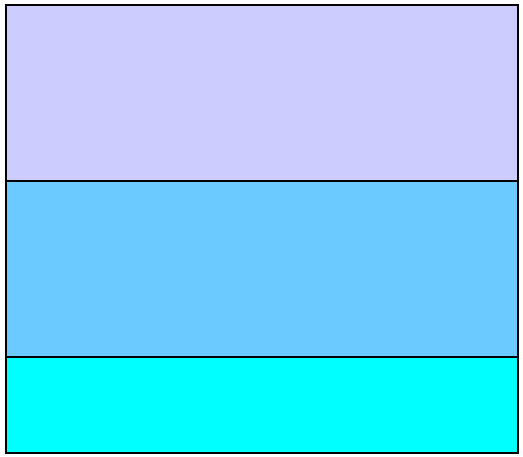
Chauffeur translates commands to car operations

Code implementing UI translated user actions to object methods

User Interface: Language used to communicate with Chauffeur

Implementing Interactive Applications

Interactive Application



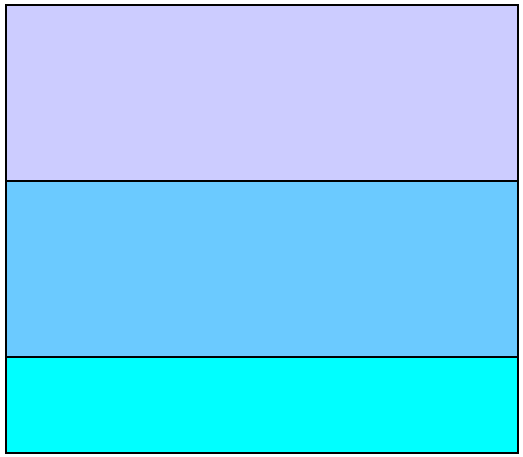
Service

User Interface

Library

Implementing Interactive Applications

Interactive Application

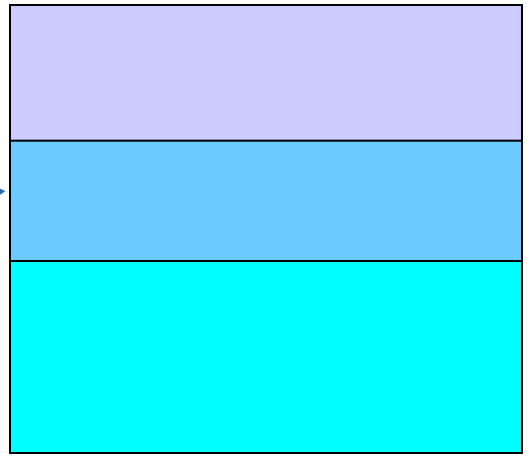


Service	50%
User Interface	50%
Library	I/O Routines

Console/ Teletype UI	Sutton & Sprague '78

Generic User Interface?

Interactive Application



Generic UI Code?

Service	50%
User Interface	50%
Library	Toolkit

Desktop GUI Myers ‘95

Generic User Interface Code

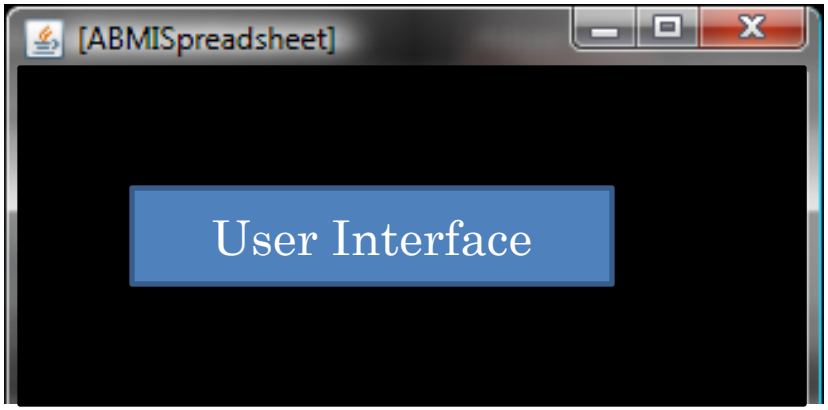
```
public class ABMISpreadsheet
implements BMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double
            newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double
            newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```

Service

BMISpreadsheet.java

```
public interface BMISpreadsheet {

    public double getWeight();
    public void setWeight(double newVal);

    public double getHeight();
    public void setHeight(double newVal);

    public double getBMI();
}
```

Programming Interface




```
ObjectEditor.edit(new ABMISpreadsheet());
```

UI for ABMISpreadsheet?

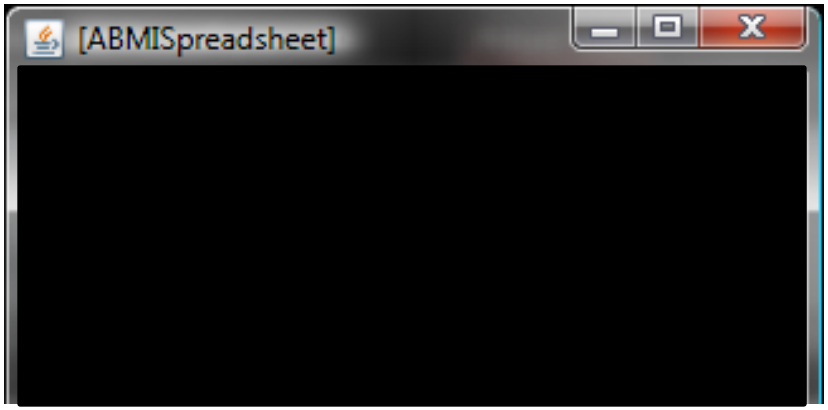
```
public class ABMISpreadsheet
implements BMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double
            newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double
            newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```

Service

BMISpreadsheet.java

```
public interface BMISpreadsheet {

    public double getWeight();
    public void setWeight(double newVal);

    public double getHeight();
    public void setHeight(double newVal);

    public double getBMI();

}
```

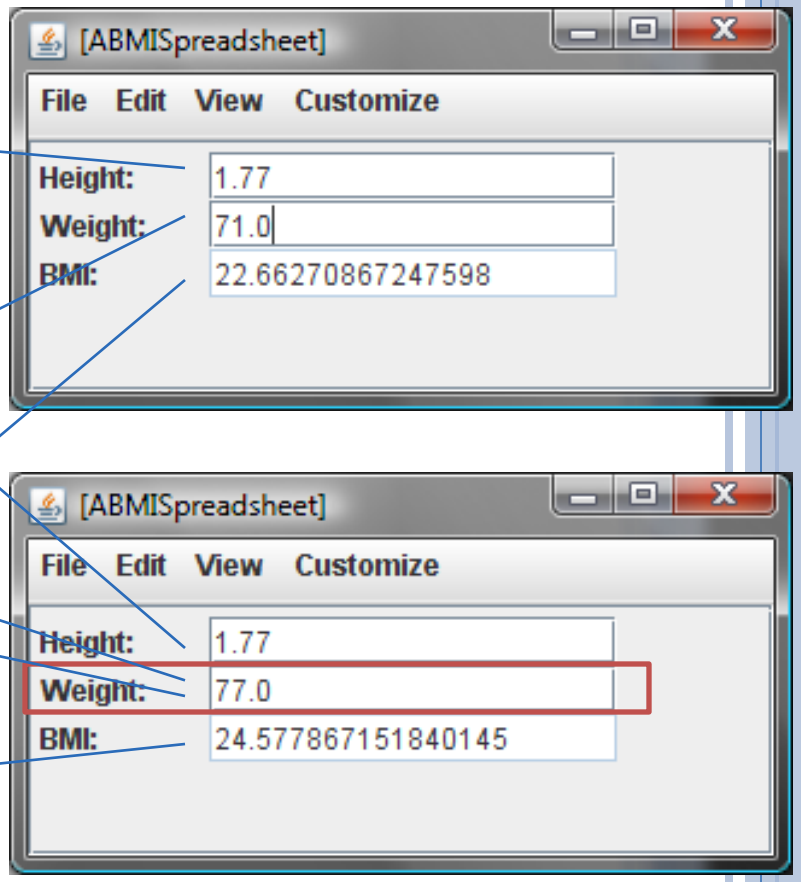
Programming Interface



ObjectEditor.edit(**new** ABMISpreadsheet());

ObjectEditor Property Manipulation

```
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        return height;
    }
    public void setHeight(double newHeight) {
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        return weight;
    }
    public void setWeight(double newWeight) {
        weight = newWeight;
    }
    public double getBMI() {
        return weight/(height*height);
    }
}
```

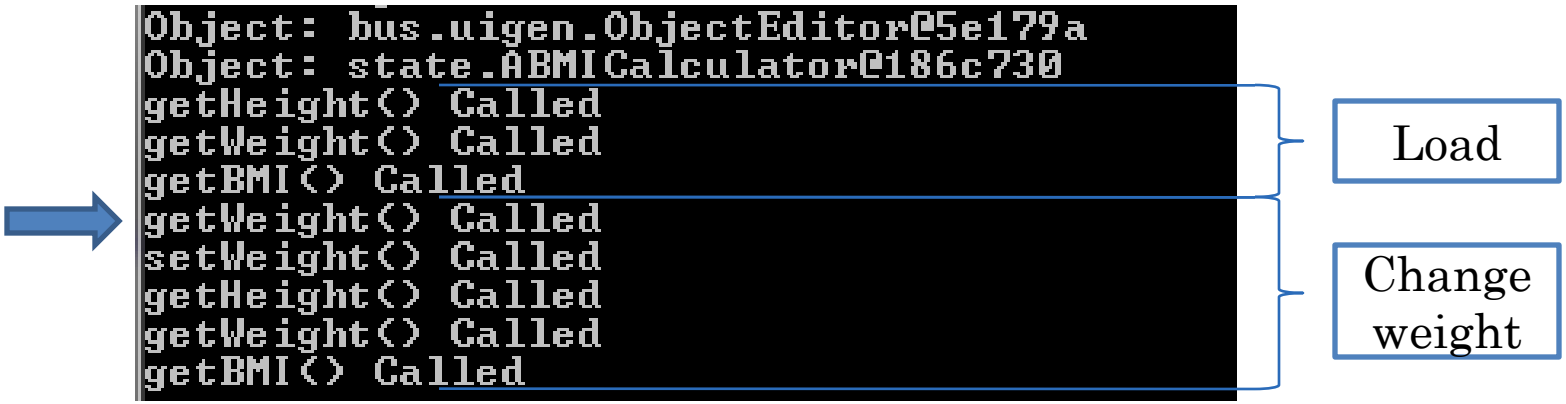


Tracing Method Calls

```
public class ABMISpreadsheet {
    double height;
    public double getHeight() {
        System.out.println("getHeight Called");
        return height;
    }
    public void setHeight(double newHeight) {
        System.out.println("setHeight Called");
        height = newHeight;
    }
    double weight;
    public double getWeight() {
        System.out.println("getWeight Called");
        return weight;
    }
    public void setWeight(double newWeight) {
        System.out.println("setWeight Called");
        weight = newWeight;
    }
    public double getBMI() {
        System.out.println("getBMI Called");
        return weight/(height*height);
    }
}
```

Debug output

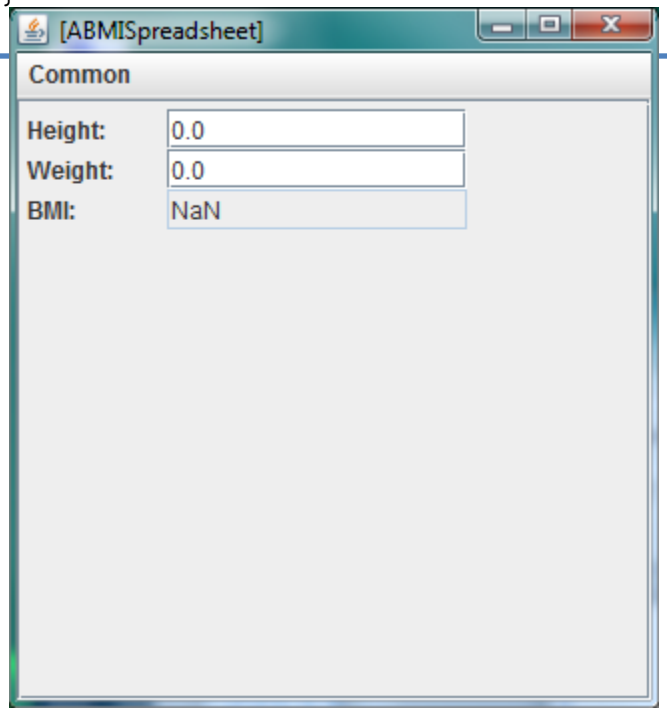
Actual Trace



Extra getWeight() call made by the undo-redo mechanism in ObjectEditor

Displaying and then Changing Object

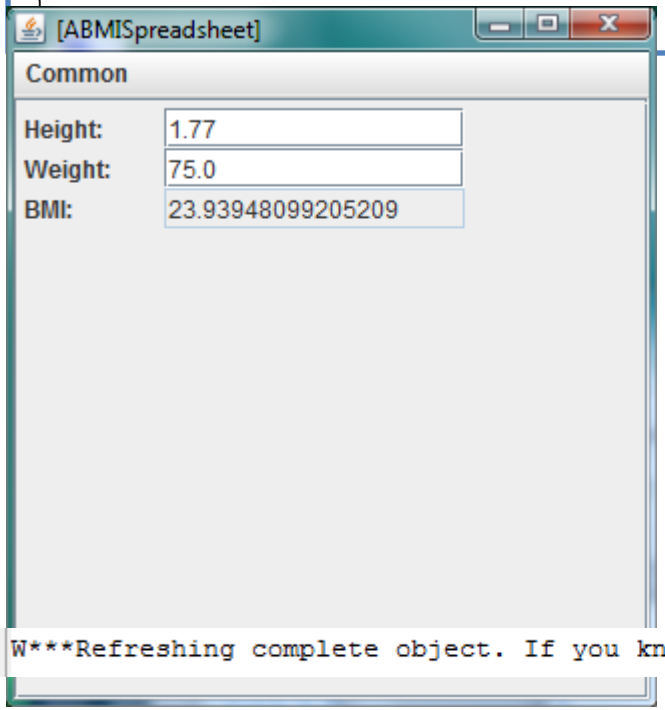
```
public class ABMISpreadsheetManipulatedByMainAndObjectEditor {
  public static void main (String[] args) {
    ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    ObjectEditor.edit(bmiSpreadsheet);
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
  }
}
```



Setters not called through ObjectEditor, so it does not know it should refresh

Refreshing ObjectEditor from Main

```
public class ABMISpreadsheetRefreshedByMain {
  public static void main (String[] args) {
    ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    OEFrame oeFrame = ObjectEditor.edit(bmiSpreadsheet);
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
    oeFrame.refresh();
  }
}
```

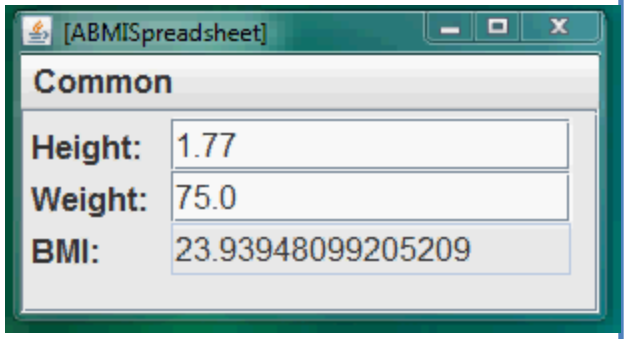


We will learn later, better ways to refresh

```
W***Refreshing complete object. If you know them, annonce property and/or list events.
```

Demoing Object

```
public class ABMISpreadsheetAnimatingDemoer {
  public static void main (String[] args) {
    ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
    bmiSpreadsheet.setHeight(1.77);
    bmiSpreadsheet.setWeight(75);
    OEFrame editor = ObjectEditor.edit(bmiSpreadsheet);
    ThreadSupport.sleep(5000);
    editor.select(bmiSpreadsheet, "Weight");
    bmiSpreadsheet.setWeight(70);
    editor.refresh();
    ThreadSupport.sleep(5000);
    editor.select(bmiSpreadsheet, "Height");
    bmiSpreadsheet.setHeight(0);
    editor.refresh();
    ThreadSupport.sleep(5000);
    editor.select(bmiSpreadsheet, "Weight");
    bmiSpreadsheet.setWeight(0);
    editor.refresh();  }
}
```



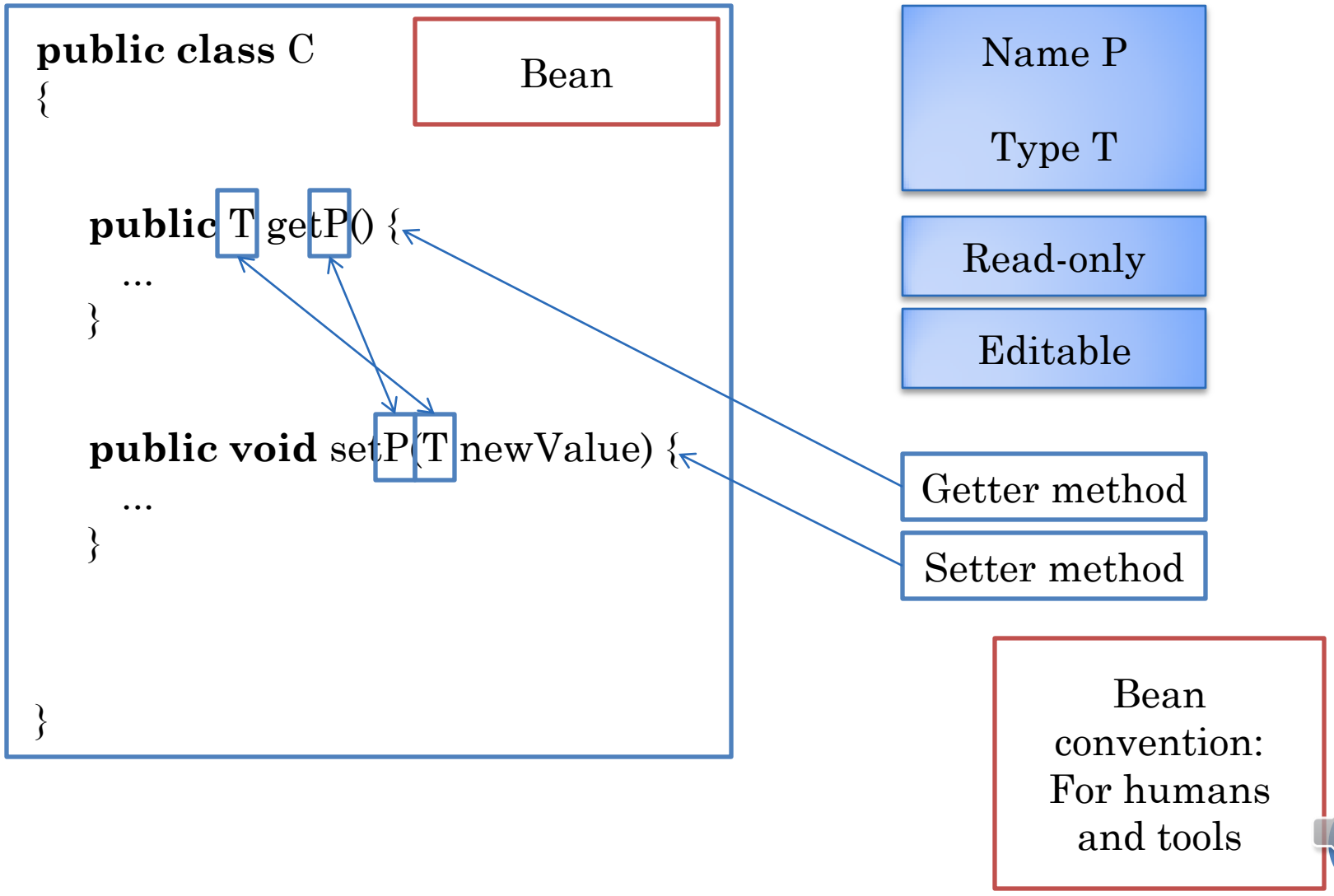
ThreadSupport() makes program wait for specified number of milliseconds

Demo

https://www.youtube.com/watch?v=dYfSuP3Io8I&feature=plcp

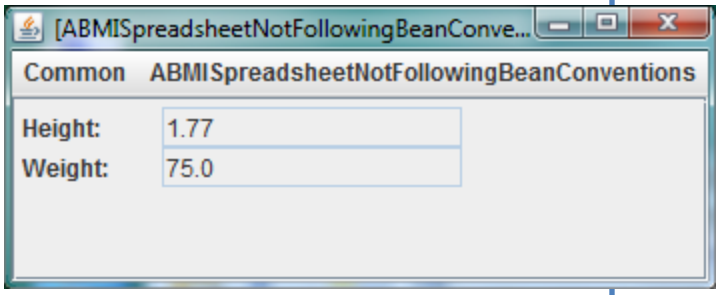
Tool Use of Read-only and Editable Properties

Typed, Named Unit of Exported Object State



Error?

```
public class ABMISpreadsheetNotFollowingBeanConventions  {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
  public void set(double newWeight, double newHeight) {
    weight = newWeight;
    height = newHeight;
  }
  public double getHeight() {
    return height;
  }
  public void setHeight(int newHeight) {
    height = newHeight;
  }
  public double BMI() {
    return weight/(height*height);
  }
}
```



Can system catch these errors?

Bean Pattern Annotation

```
import util.annotations.StructurePattern
import util.annotations.StructurePatternNames
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheet {
        double height;
        public double getHeight() {
                return height;
        }
        public void setWeight(double newWe
}
```

Why pattern annotation?

Annotation is like a comment except it is typed and available at runtime

Documentation

Available to ObjectEditor

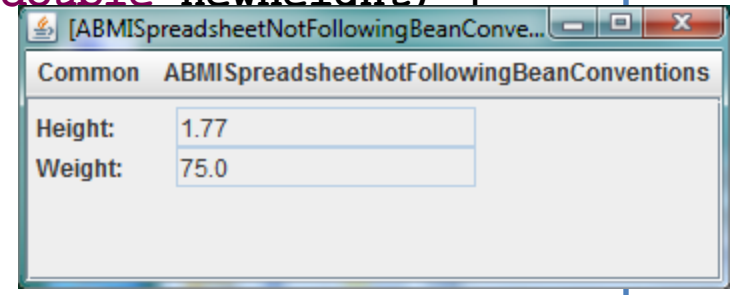
Efficiency: OE does not need to look for pattern

Structure(<PatternName>) before class asserts that the class is following the pattern.

Errors/warnings: Can give error message if pattern not followed

Error?

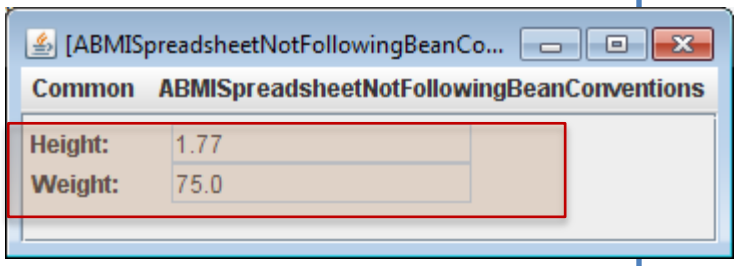
```
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
public class ABMISpreadsheetNotFollowingBeanConventions  {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
  public void set(double newWeight, double newHeight) {
    weight = newWeight;
    height = newHeight;
  }
  public double getHeight() {
    return height;
  }
  public void setHeight(int newHeight) {
    height = newHeight;
  }
  public double BMI() {
    return weight/(height*height);
  }
}
```



Can system catch these errors?

(Editable) Property Name Annotations

```
import util.annotations.EditablePropertyNames;
import util.annotations.PropertyNames;
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
@PropertyNames({ "Height", "Weight", "BMI"})
@EditablePropertyNames({"Height", "Weight"})
public class ABMISpreadsheetNotFollowingBeanConventions  {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
  public void set(double newWeight,
    weight = newWeight;
```

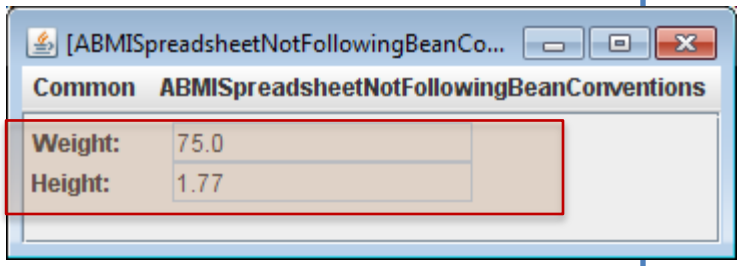



```
E***For property: height in editable property names, please define a setter with the header:
        public void setHeight(double <parameter name>)
E***For property: weight in editable property names, please define a setter with the header:
        public void setWeight(double <parameter name>)
E***For property: BMI in  property names, please define a getter with the header:
        public <T> getBMI()
```

```
    height = newHeight;
  }
  public double BMI() {
    return weight/(height*height);
  }
```

Order of Properties

```
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
@PropertyNames({ "Weight", "Height", "BMI"})
@EditablePropertyNames({"Height", "Weight"})
public class ABMISpreadsheetNotFollowingBeanConventions  {
  double height = 1.77;
  double weight = 75;
  public double getWeight() {
    return weight;
  }
  public void set(double newWeight,
    weight = newWeight;
    height = newHeight;
  }
  public double getHeight() {
    return height;
  }
  public void setHeight(int newHeight) {
    height = newHeight;
  }
  public double BMI() {
    return weight/(height*height);
  }
}
```



Explanation

```
@Explanation("Calculates BMI from height in metres and weight in
kgs.")
public interface AnnotatedBMISpreadsheet  {
  public double getHeight();
  public void setHeight(double newVal);
  @Explanation("Weight is in Kilograms. ")
  public double getWeight();
  @Explanation("Changing weight automatically recalculates BMI.")
  public void setWeight(double newVal);
  public double getBMI();
}
```

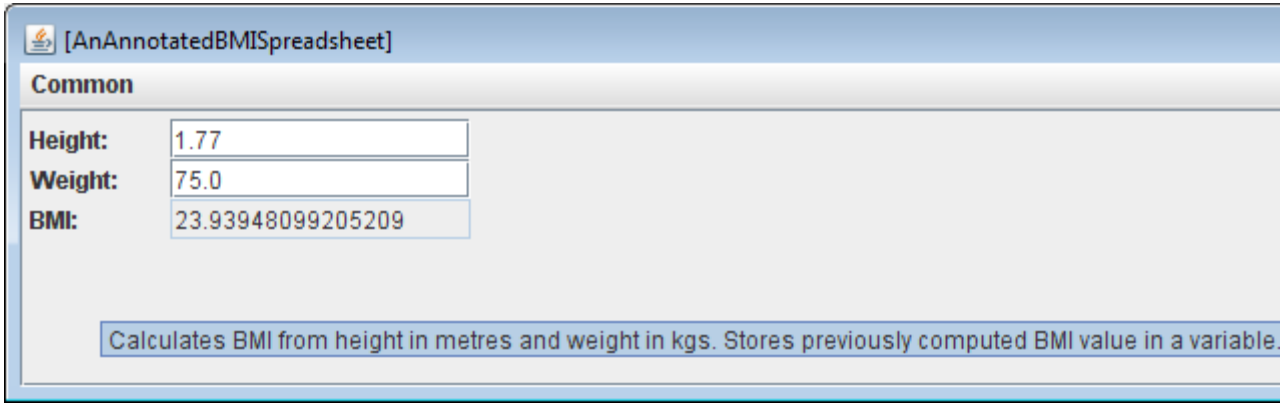
Tooltip text for all instances

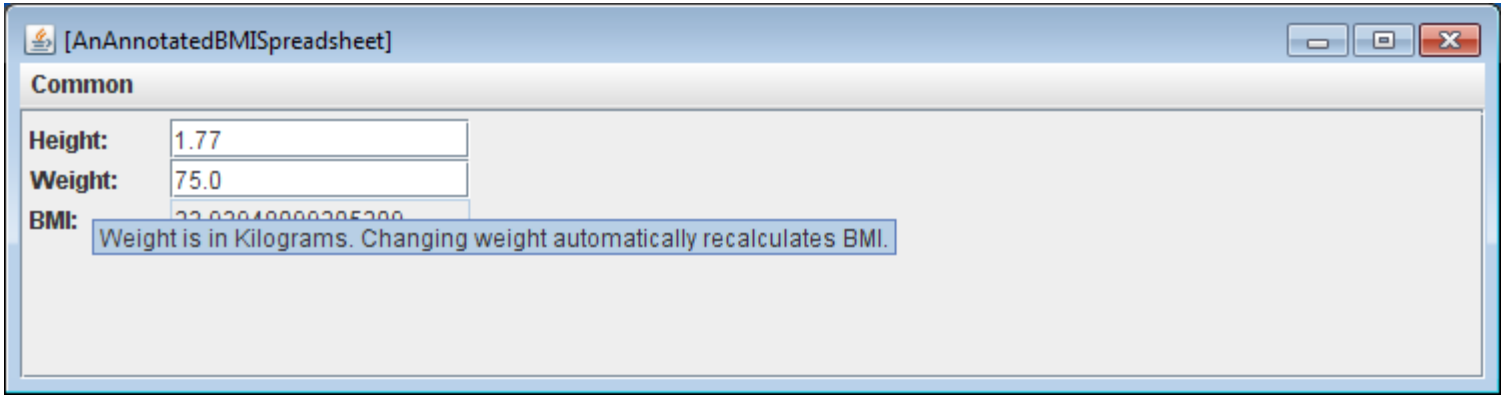
Tooltip text for property

Tooltip text for all instances

```
@Explanation("Stores previously computed BMI value in a variable.")
@Tags({"Annotations", "Spreadsheet", "Properties", "Interfaces", "Object
Editor"})
public class AnAnnotatedBMISpreadsheet implements
                                                AnnotatedBMISpreadsheet
{…}
```

Explanation Annotation





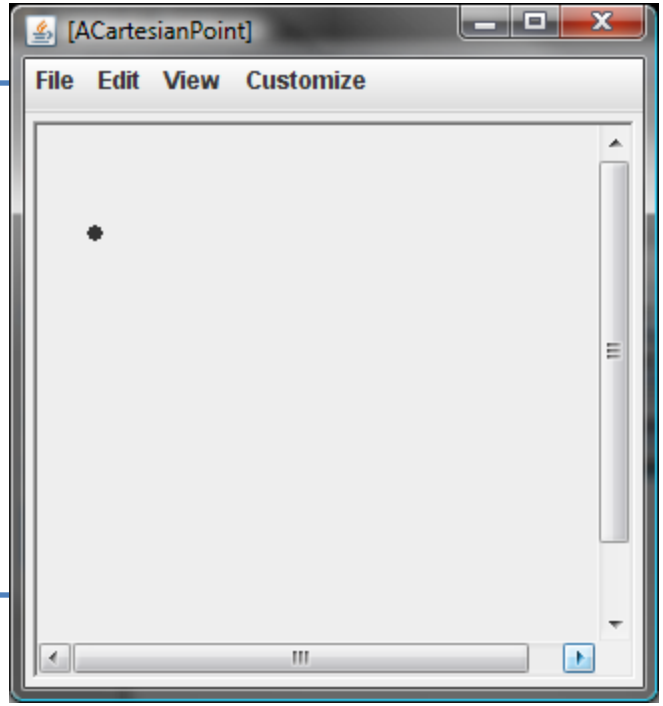
Tag Classes

```
import util.annotations.Explanation;
import util.annotations.Tags;
@Tags({"Annotations", "Spreadsheet", "Properties", "Interfaces", "Object
Editor"})
public class AnAnnotatedBMISpreadsheet implements
                                                AnnotatedBMISpreadsheet
{…}
```

Grader can find tagged classes

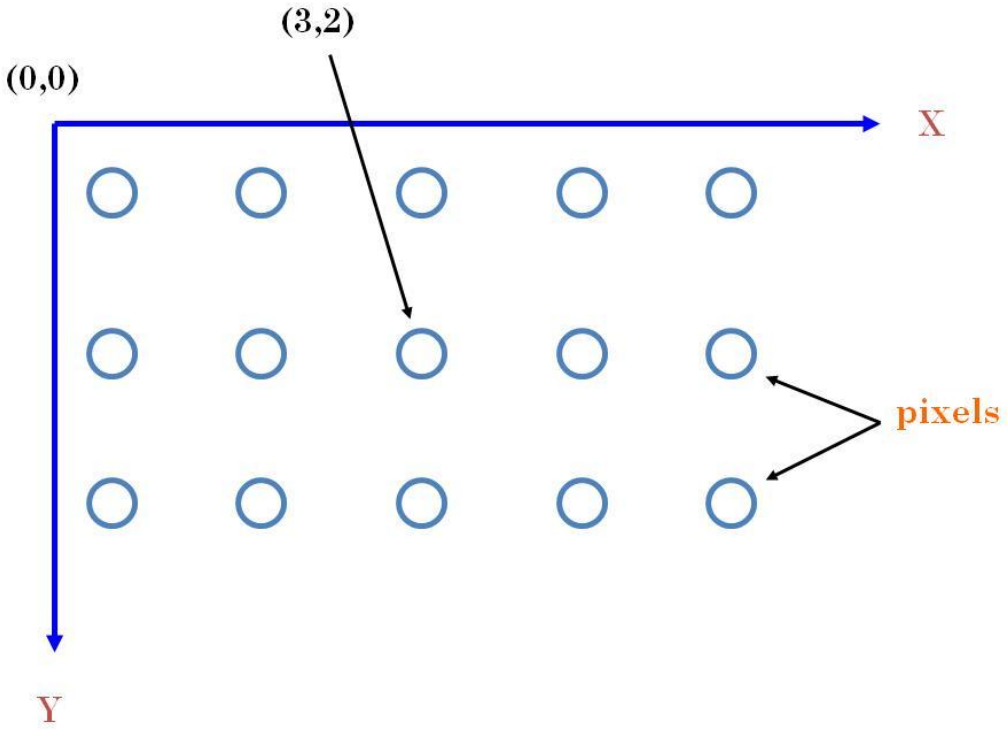
Displaying Geometric Object

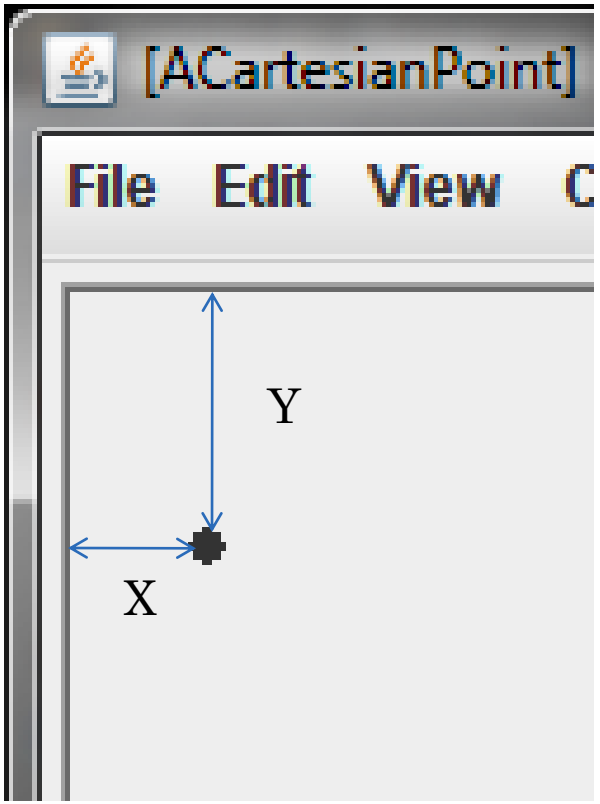
```
public interface Point {
        public int getX();
        public int getY();
        public double getAngle();
        public double getRadius();
}
```




```
ObjectEditor.edit(new ACartesianPoint (25, 50));
```

Java Graphics





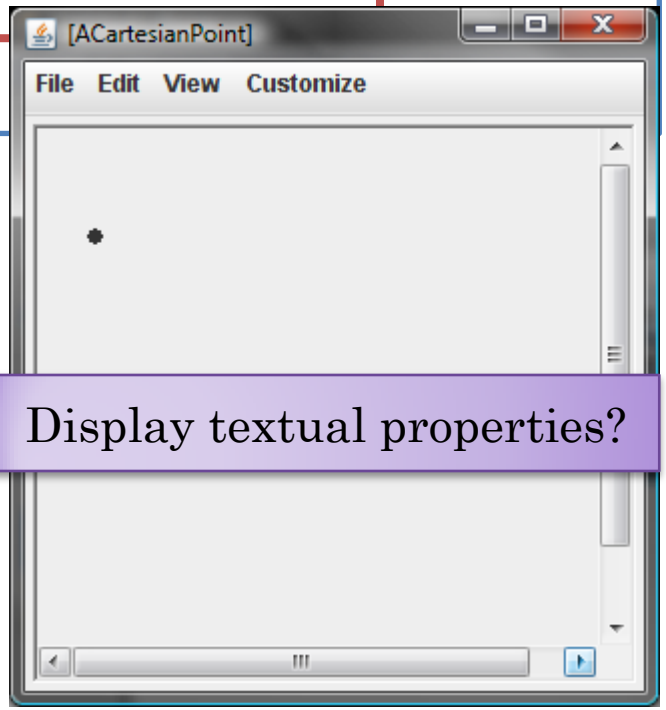
ObjectEditor Point Rules

- 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();
}
```

Point Annotation

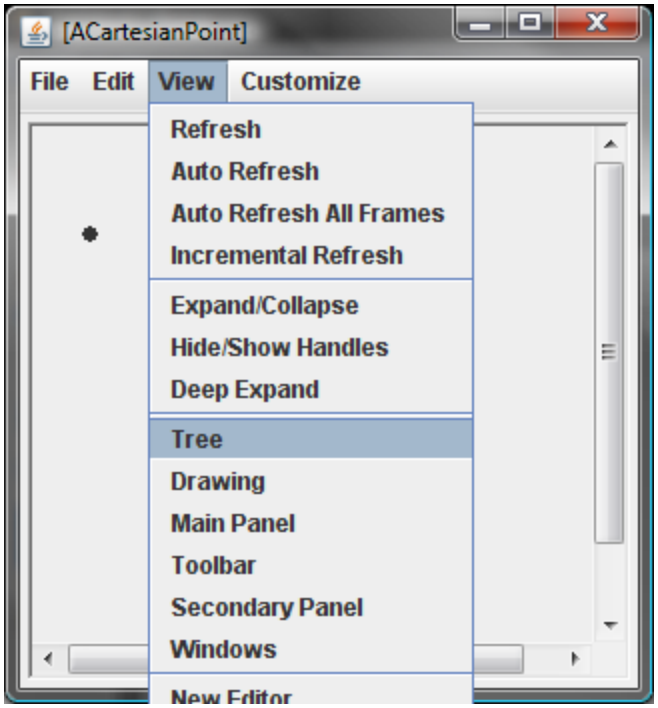
```
import util.annotations.StructurePattern;
import util.annotations.StructurePatternNames;
@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point{
    …
}
```



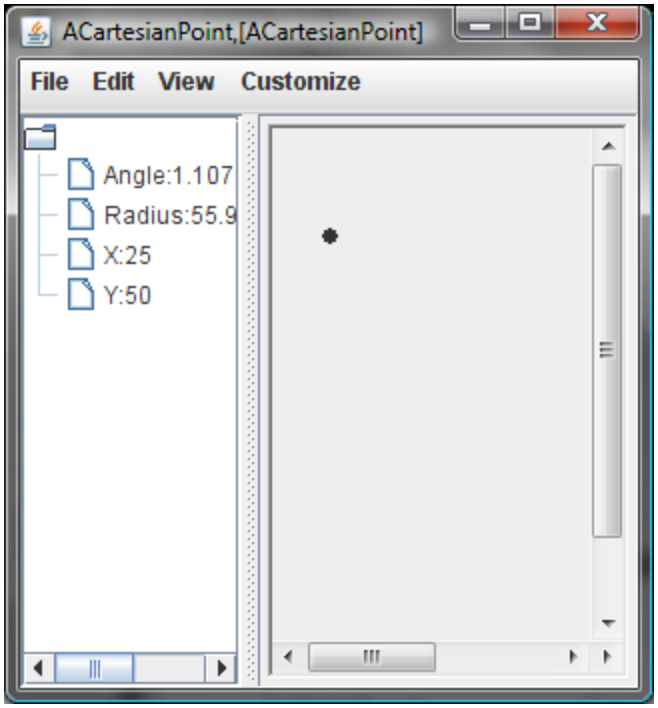
Display textual properties?

Structure(<PatternName>) before class or interface asserts that the type is following the pattern. ObjectEditor ignores class/interface name and gives warnings if methods do not follow the pattern

Adding Tree View

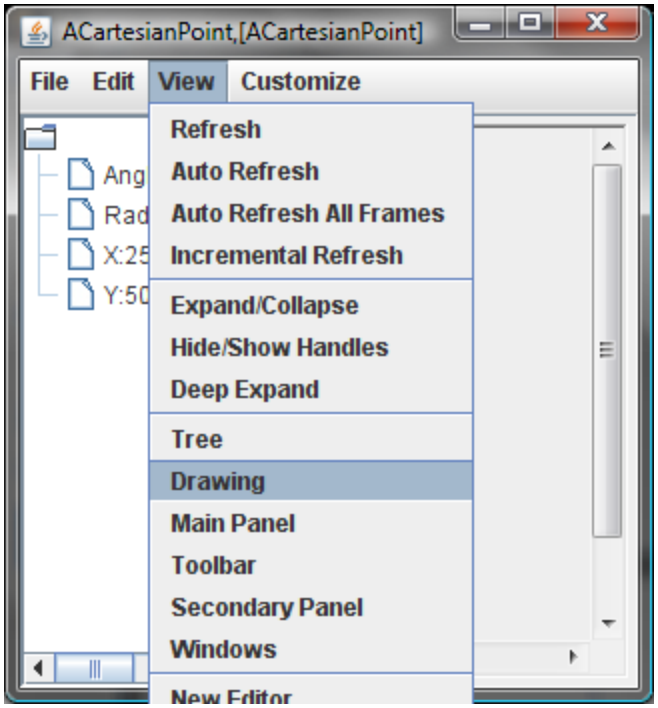


Tree View

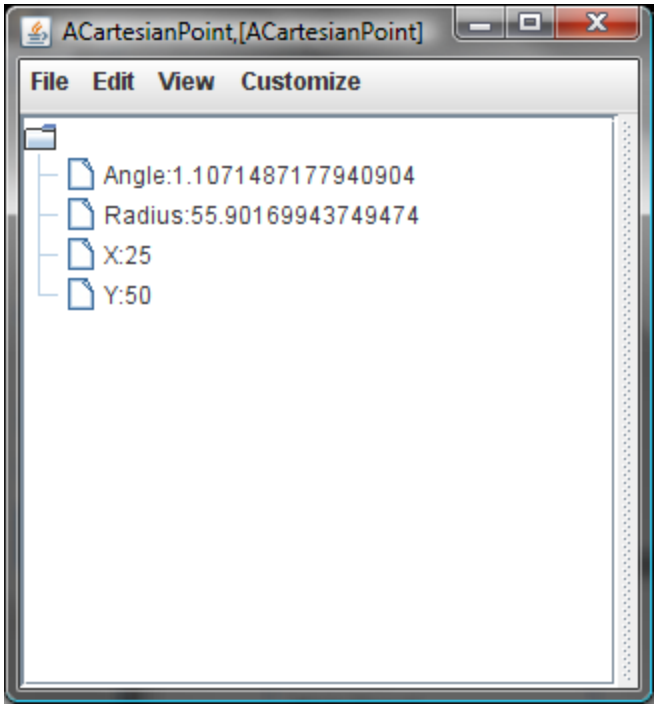



```
OEFrame frame = ObjectEditor.edit(new ACartesianPoint (25,
50));
frame.showTreePanel();
```

Removing Graphics View

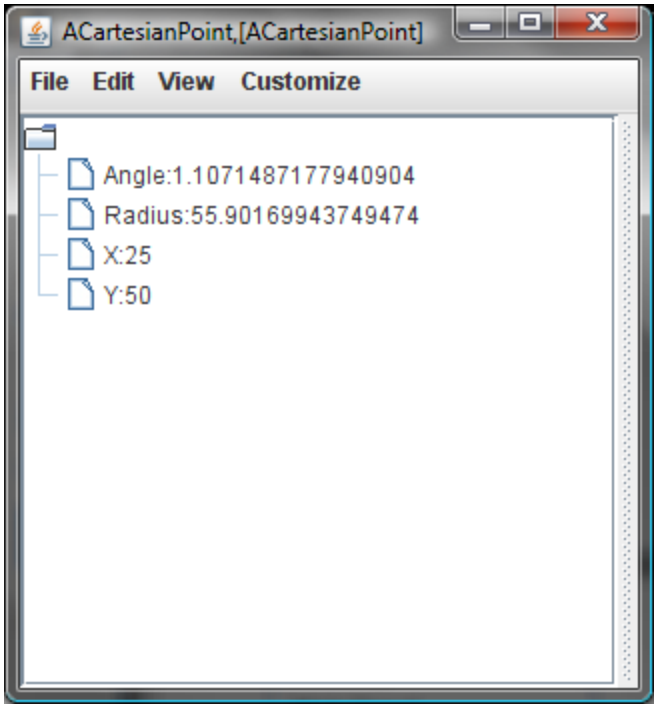


Tree-only View




```
OEFrame frame = ObjectEditor.edit(ObjectEditor.edit(new
ACartesianPoint (25, 50);
frame.hideDrawingPanel(); //to be added
frame.showTreePanel();
```

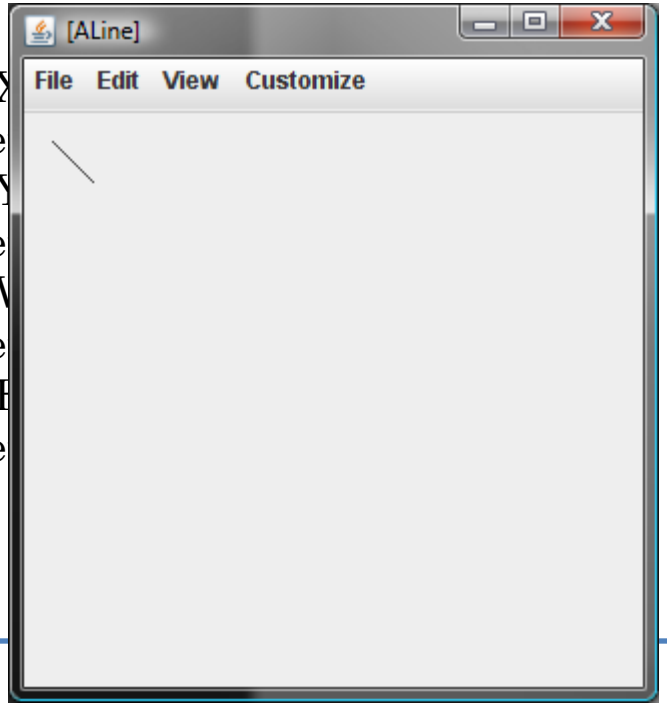
Customizing the Size of the Window



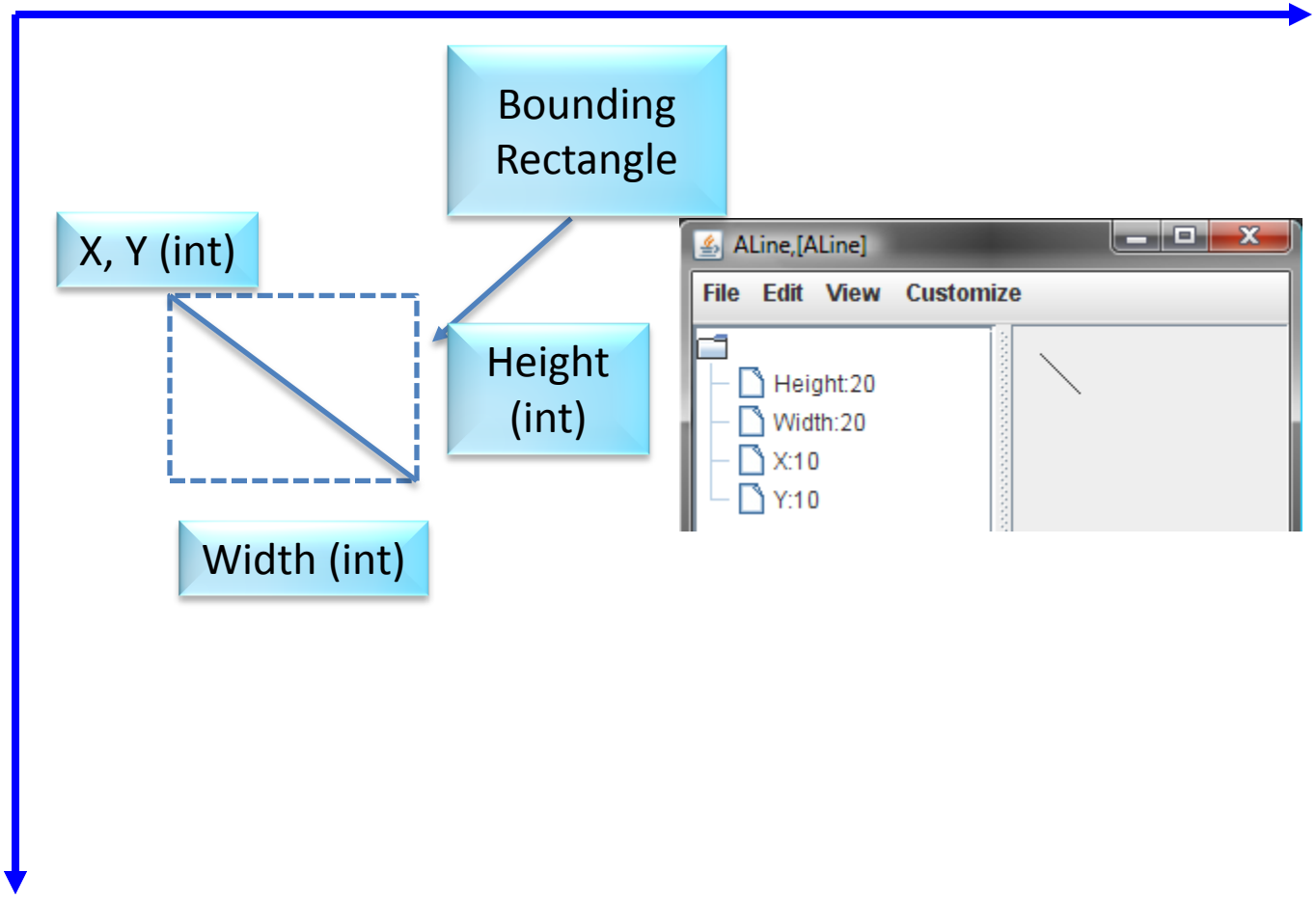

```
OEFrame frame = ObjectEditor.edit(ObjectEditor.edit(new
ACartesianPoint (25, 50);
frame.setSize(400, 300); // width and height
Frame.setLocation(500, 400);// x and y
```

Displaying a Line?

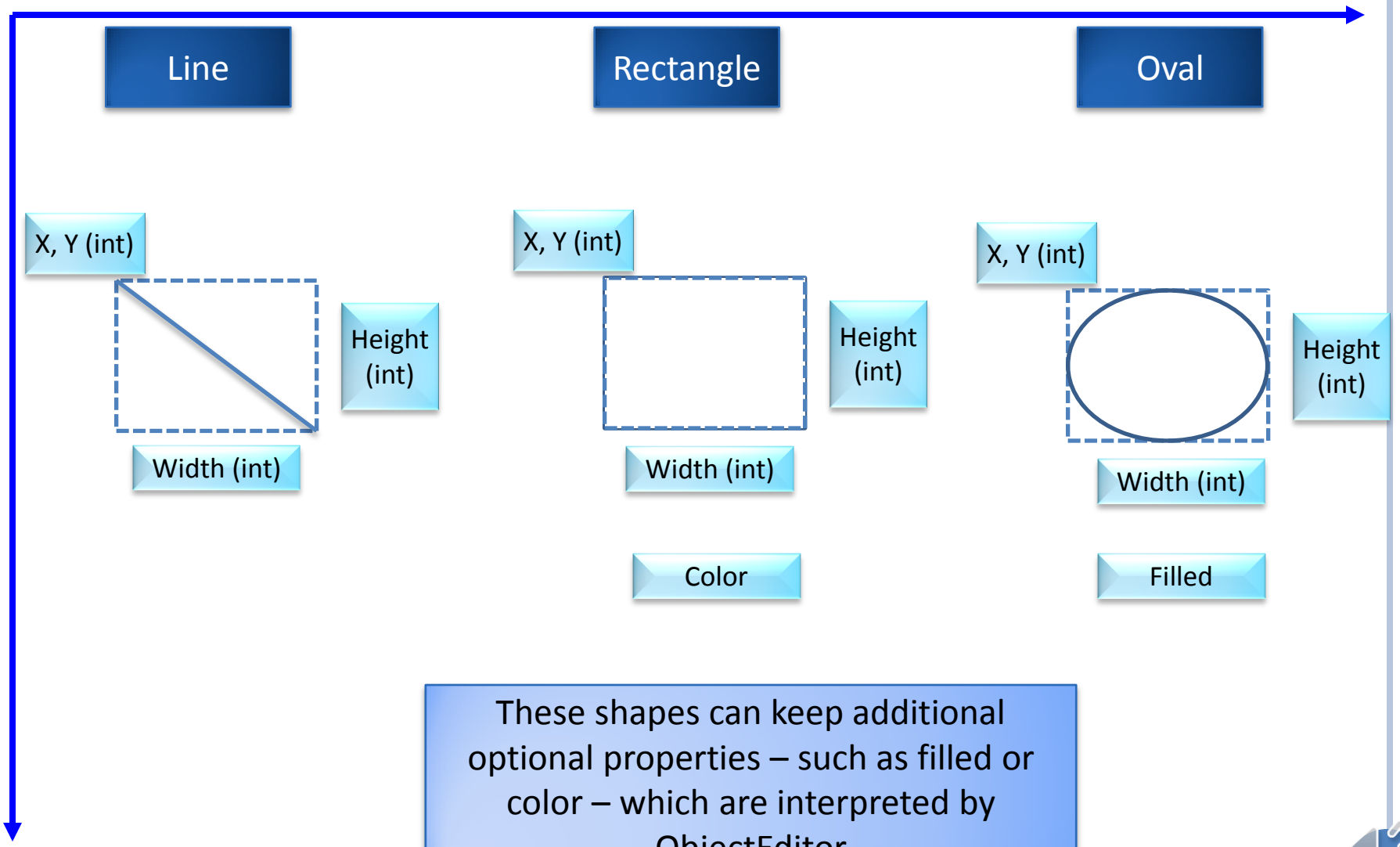
```
public interface Line {
        public int getX
        public void se
        public int getY
        public void se
        public int getW
        public void se
        public int getH
        public void se
}
```



Line Logical Representation (Properties)



ObjectEditor Shape Rules



These shapes can keep additional optional properties – such as filled or color – which are interpreted by ObjectEditor

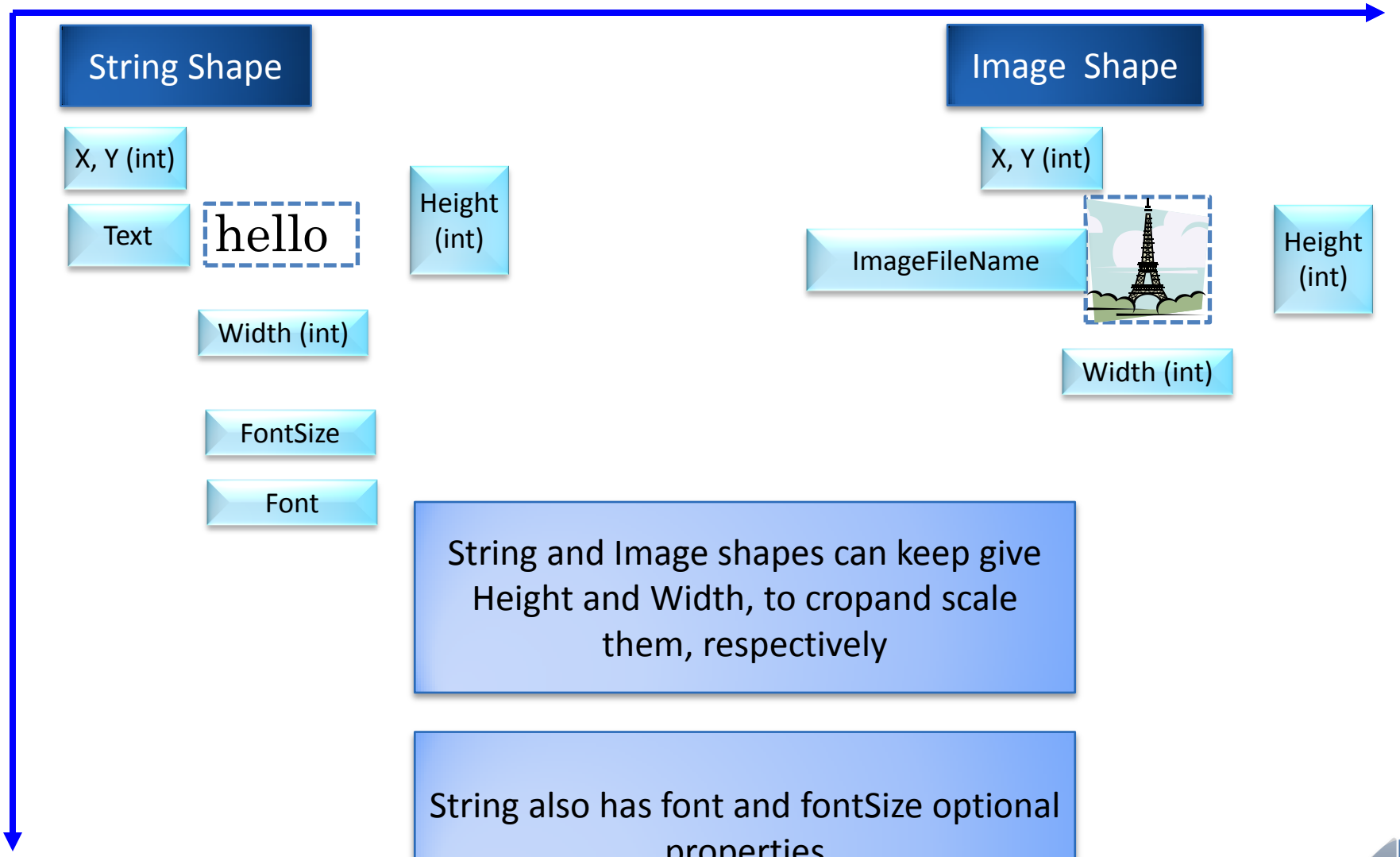
XY-Based Line/Oval/Rectangle Interface

```
public interface Rectangle
        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);
}
```

Oval/Rectangle logical representation?

Other shapes needed in project?

XY-based String and Image Rules

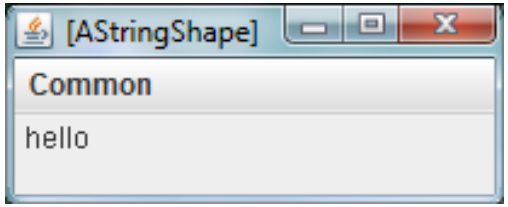


String and Image shapes can keep give Height and Width, to cropand scale them, respectively

String also has font and fontSize optional properties

AString Shape

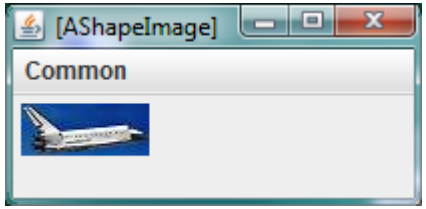
```
@StructurePattern(StructurePatternNames.STRING_PATTERN)
public class AStringShape implements StringShape {
  String text;
  int x, y;
  public AStringShape(String initText, int initX, int initY) {
    text = initText;
    x = initX;
    y = initY;
  }
  public int getX() {return x;}
  public void setX(int newX) {x = newX;}
  public int getY() { return y;}
  public void setY(int newY) {y = newY;}
  public String getText() {return text;}
  public void setText(String newVal) {text = newVal;}
}
```




```
public static void main(String args[]) {
  StringShape hello= new AStringShape("hello", 0, 0);
  ObjectEditor.edit(hello);
}
```

Image Implementation

```
@StructurePattern(StructurePatternNames.IMAGE_PATTERN)
public class AShapeImage implements ImageShape {
  String imageFileName;
  int x, y;
  public AShapeImage (String initImageFileName, int initX, int initY) {
    imageFileName = initImageFileName;
    x = initX;
    y = initY;
  }
  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 void setImageFileName(String newVal) {imageFileName = newVal ;}
}
```




```
public static void main (String args[]) {
      ImageShape shuttle = new AShapeImage("shuttle2.jpg", 0, 0);
      ObjectEditor.edit(shuttle);
}
```

Short file name (without "/") implies file in the project folder, the one containing src and bin

ObjectEditor vs. Java Graphics

```
public class ALine  implements Line{
        int x, y, width, height;
        public ALine (int initX, int initY, int in
                x = initX;
                y = initY;
                width = initWidth;
                height = initHeight;
        }
        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 int getWidth() {return width;}
        public void setWidth(int newVal) {widt
        public int getHeight() {return height;}
        public void setHeight(int newHeight) {height = newHeight;}
}
```

Encapsulates state of line in one object

Object and view are independent (can show object in tree view or graphics view)

If external state of object changes, the display is updated

```
ObjectEditor.edit(new ALine(x, y, w, h));
```

```
graphics.drawLine(x1, y1, x2, y2)
```

Requires knowledge of panel, paint events, inheritance

Graphics Types Properties

- Point
  - Location (x, y)
- Line/Rectangle/Oval
  - Bounding box (x, y, width, height)
- String/Image
  - Location
  - Contents
    - Text/image file
  - Height, Width
    - Implied by contents
    - Can scale/crop to give explicit values

ObjectEditor Graphics

- Can automatically display objects representing points, rectangles, ovals, and lines as corresponding graphics
  - Java provides libraries to manually display graphics
- Has rules for recognizing these objects
- Rules based on Java graphics standards
  - Inverted coordinate system
  - Cartesian coordinates for points
  - Bounding rectangles for lines, rectangles, ovals
- Plus naming conventions and annotations

ObjectEditor Bounding Box Rules

- A shape object describes its bounding box if it:
  - represents the size of the bounding box using int (read-only or editable) properties, “Height”, and “Width”
  - describes the location of the upper left corner of the bounding box using “X”, “Y” properties of type int

ObjectEditor Line/Shape/Oval Rules

- An object is recognized as a rectangle/line/oval if:
  - Its interface or class has (a) the string "Rectangle"/"Oval"/"Line" in its name or (b) has a Point/Oval/Line annotation
  - It has (readonly or editable) properties describing the bounding box of the shape
  - Can have additional properties such as Filled, Color

```
import util.annotations.StructurePattern;
import util.annotations.StructurePatternNames;
@StructurePattern(StructurePatternNames.LINE_PATTERN)
public interface Line{
    …
}
```

ObjectEditor String Shape Rules

- An object is recognized as a string shape if:
  - Its interface or class has (a) the string "String" in its name or a (b) String annotation
  - It has a (readonly/editable) "Text" of type String describing the string to be displayed
  - It has (readonly or editable) int "X", "Y" properties describing the location of the lower left corner of the bounding box of the shape
  - Can have additional properties such as FontSize and Font

```
@StructurePattern(StructurePatternNames.STRING_PATTERN)
public interface StringShape {
  public int getX();
  public void setX(int newX);
  public int getY();
  public void setY(int newY);
  public String getText() ;
  public void setText(String newVal);
}
```

ObjectEditor Image Shape Rules

- An object is recognized as an image shape if:
  - Its interface or class has the string "Image" in its name or the image annotation
  - It has a (readonly/editable) String "ImageFileName" property describing the name of the image file to be displayed
  - It has (readonly or editable) int X, Y properties describing the location of the upper left corner of the bounding box of the shape
  - Can have additional properties

```
@StructurePattern(StructurePatternNames.IMAGE_PATTERN)
public interface ImageShape {
public int getX();
 public void setX(int newX);
 public int getY();
 public void setY(int newY);
 public String getImageFileName() ;
 public void setImageFileName(String newVal);
}
```

Documentation

- Required to put shape annotations
  - Should get rid of ObjectEditor warnings