Instructor: Prasun Dewan
Prerequisite

- Interfaces
MORE ON OBJECTS

- Graphics types
- Test-first approach
- Stubs
- Physical vs. logical representation
- Representations with errors
MATHEMATICAL POINT

X
Y
R
\theta
Java Coordinate System

X and Y coordinates must be int values

Radius and Angle can be double

Angle is a decimal value between 0 and 2π

Each window has its own coordinate system
Why not Mathematical Coordinate System

Computer coordinate system modeled after paper documents

In LTR (left to write languages) coordinate origing top left, and x, goes left to right, and y top to bottom

A window typically shows an underlying electronic document
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}

Point Interface (Review)

Read-only properties defining immutable object!

Top-down, test-first?

Stubs?
public class ACartesianPointTester {
    public void test () {
        test (10, 0, 10.0, 0); // 0 degree angle
        test (0, 10, 10.0, Math.PI / 2); // 90 degree angle
        .. //more tests
    }
    public void test (int theX, int theY, double theCorrectRadius, double theCorrectAngle) {
        Point point = new ACartesianPoint (theX, theY);
        double computedRadius = point.getRadius();
        double computedAngle = point.getAngle();
        System.out.println("-------------");
        System.out.println("X:" + theX);
        System.out.println("Y:" + theY);
        System.out.println("Expected Radius:" + theCorrectRadius);
        System.out.println("Computed Radius:" + computedRadius);
        System.out.println("Radius Error:" + (theCorrectRadius - computedRadius));
        System.out.println("Expected Angle:" + theCorrectAngle);
        System.out.println("Computed Angle:" + computedAngle);
        System.out.println("Angle Error:" + (theCorrectAngle - computedAngle));
        System.out.println("-------------");
    }
}
STUBS FOR A CartesianPoint

```java
public class ACartesianPoint implements Point {
    public ACartesianPoint(int theX, int theY) {
    }
    public ACartesianPoint(double theRadius, double theAngle) {
    }
    public int getX() {
        return 0;
    }
    public int getY() {
        return 0;
    }
    public double getAngle() {
        return 0;
    }
    public double getRadius() {
        return 0;
    }
}
```
**Test-First Approach**

1. Write the interface
2. Create a class with stubs for each interface method and constructor
   1. If method is procedure method does nothing
   2. If method is function, it returns 0 or null value
   3. No variables need be declared as this point!
3. Write a tester for it
4. Write/rewrite in one or more stub methods
5. Use tester
6. If tester results not correct, go back to 4

Steps may be combined for simple classes!
POINT VARIABLE REPRESENTATIONS

- X, Y (Cartesian Representation)
- Radius, Angle (Polar Representation)
- X, Radius
- X, Y, Radius, Angle
- ...
**Algorithms**

**Cartesian Representation**

\[
R = \sqrt{X^2 \times Y^2} \\
\theta = \arctan \left( \frac{Y}{X} \right)
\]

**Polar Representation**

\[
X = R \times \cos(\theta) \\
Y = R \times \sin(\theta)
\]
public class ACartesianPoint implements Point {
    int x, y;
    public ACartesianPoint(int theX, int theY) {
        x = theX;
        y = theY;
    }
    public ACartesianPoint(double theRadius, double theAngle) {
        x = (int) (theRadius*Math.cos(theAngle));
        y = (int) (theRadius*Math.sin(theAngle));
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public double getAngle() { return Math.atan2(y, x); }
    public double getRadius() { return Math.sqrt(x*x + y*y); }
}
public class APolarPoint implements Point {
    double radius, angle;
    public APolarPoint(double theRadius, double theAngle) {
        radius = theRadius;
        angle = theAngle;
    }
    public APolarPoint(int theX, int theY) {
        radius = Math.sqrt(theX*theX + theY*theY);
        angle = Math.atan((double) theY/theX);
    }
    public int getX() { return (int) (radius*Math.cos(angle)); }
    public int getY() { return (int) (radius*Math.sin(angle)); }
    public double getAngle() { return angle; }
    public double getRadius() { return radius; }
}
Using the Interface and Its Implementations

Point point1 = new A CartesianPoint (50, 50);

Point point2 = new A PolarPoint (70.5, Math.pi()/4);

point1 = point2;

Constructor chooses implementation

Constructor cannot be in interface
Representing Geometric Objects

- Geometric example to show multiple useful implementations of an interface
- Most geometric objects have multiple representations
WHAT IS A REPRESENTATION

- Logical representation
  - Defined by its interface
  - Specifies properties

- Physical representation
  - Defined by its instance variables

- ACartesianPoint and APolarPoint have the same logical representation but different physical representation
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}

ObjectEditor.edit(new A CartesianPoint (25, 50));
JAVA GRAPHICS

(0,0) → (3,2) → pixels

X

Y
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

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

@StructurePattern(StructurePatternNames.POINT_PATTERN)
public interface Point{
    ...
}

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.

Display textual properties?
Adding Tree View
OEOFrame frame = ObjectEditor.edit(new ACartesianPoint (25, 50));
frame.showTreePanel();
Removing Graphics View
TREE-ONLY VIEW

```java
OEFrame frame = ObjectEditor.edit(ObjectEditor.edit(new ACartesianPoint (25, 50));
frame.hideDrawingPanel(); //to be added
frame.showTreePanel();
```
CUSTOMIZING THE SIZE OF THE WINDOW

OxEFramen frame = ObjectEditor.edit(ObjectEditor.edit(new ACartesianPoint(25, 50);
frame.setSize(400, 300); // width and height
Frame.setLocation(500, 400); // x and y
LINE?
WHAT IS A REPRESENTATION

- Logical representation
  - Defined by its interface
  - Specifies properties

- Physical representation
  - Defined by its instance variables
**Line Logical Representation (Properties)**

- **Bounding Rectangle**
  - **X, Y (int)**
  - **Width (int)**
  - **Height (int)**

![Diagram showing the properties of a line logical representation](image)
LINE IMPLEMENTATION

@StructurePattern(StructurePatternNames.LINE_PATTERN)
public class ALine implements Line {
    int x, y, width, height;
    public ALine (int initX, int initY, int initWidth, int initHeight) {
        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) {width = newVal;}
    public int getHeight() {return height;}
    public void setHeight(int newHeight) {height = newHeight;}
}

Logical representation == physical representation
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

- Location is the lower right corner in the case of a String (based on Java draw primitives)!
- String and Image shapes can keep additional properties – in particular height and width.
@StructurePattern(StructurePatternNames.STRING_PATTERN)

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, 10);
    ObjectEditor.edit(hello);
}
@StructurePattern(StructurePatternNames.IMAGE_PATTERN)

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**

```java
public class ALine implements Line{
    int x, y, width, height;
    public ALine(int initX, int initY, int initWidth, int initHeight) {
        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) {width = newVal;}
    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**

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

- **Requires knowledge of panel, paint events, inheritance**

```java
graphics.drawLine(x1, y1, x2, y2)
```
GRAPHICS TYPES

- **Point**
  - No shape
  - Just a location

- **Line/Rectangle/Oval**
  - A shape
  - A location plus an area on the screen determined by object
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
**Object Editor 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 the string “Rectangle”/”Oval”/”Line” in its name or has a Point/Oval/Line annotation
- It has (readonly or editable) properties describing the bounding box of the shape
- Can have additional properties

```java
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 the string “String” in its name or a 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

```java
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
  - 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

```java
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);
}
```
Typing Point Objects

ACartesianPoint

APolarPoint

Point
Equivalent Representations

ObjectEditor.edit(new APolarPoint (195.05, 0.87));

ObjectEditor.edit(new ACartesianPoint (125, 149));

Cartesian coordinates the same but not the polar coordinates!
Calculated X and Y values are truncated when converted to int
CONSEQUENCE OF PHYSICAL REPRESENTATION CHOICE?

- Physical representation determines
  - Space efficiency
  - Time efficiency
  - Precision of various properties
EXTRA SLIDES
**ObjectEditor Shape Rules**

- An object is recognized as a rectangle/line/oval if:
  - Its interface or class has the string “Rectangle”/”Oval”/”Line” in its name
  - It has properties describing the bounding box of the shape
  - Can have additional properties

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

```java
public interface AnotherLine {
    public Point getLocation();
    public void setLocation(Point newVal);
    public int getWidth();
    public void setWidth(int newVal);
    public int getHeight();
    public void setHeight(int newVal);
}
```
ASCALABLENESTER

Properties
NestedRectangle (Rectangle)
EnclosingRectangle (Rectangle)

Methods (non getter/setter)
public void scale (int percentage)
ACartesianPlane Stubs

public class ACartesianPlane implements CartesianPlane {
    int axesLength;
    Line xAxis;
    Line yAxis;
    Label xLabel;
    Label yLabel;
    final int LABEL_BOUNDS_SIDE_LENGTH = 10;
    public ACartesianPlane (int theAxesLength, int theOriginX, int theOriginY) {
    }
    public Line getXAxis() {return xAxis;}
    public Line getYAxis() {return yAxis;}
    public int getAxesLength() {return axesLength;}
    public void setAxesLength(int newVal) {
    }
    public Label getXLabel() {return xLabel;}
    public Label getYLabel() {return yLabel;}
}
**ObjectEditor vs. Java Graphics**

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

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

Requires knowledge of panel, paint events, inheritance

- 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
Graphics Types Seen So Far

- **Point**
  - No shape
  - Just a location

- **Line**
  - A shape
  - A location plus an area on the screen determined by object

- **Other shapes and rules?**
  - Line rules with slight adaptations
**LINE, RECTANGLE OVAL, LABEL PATTERNS**

```java
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);
}
```

```java
public interface Oval {
    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);
}
```

```java
public interface Label {
    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);
    public String getText();
    public void setText(String newString);
    public String getImageFileName();
    public void setImageFileName(String newVal);
}
```

```java
public interface ImmutableIcon {
    public int getX();
    public int getY();
    public int getWidth();
    public int getHeight();
    public String getText();
    public String getImageFileName();
    public String getImageFileName(String newVal);
}
```
XY-based Object Editor Shape Rules

- **Rectangle**
  - X, Y
  - Width
  - Height

- **Oval**
  - X, Y
  - Width
  - Height

- **Line**
  - X, Y
  - Width
  - Height

- **TextBox**
  - X, Y
  - Height
  - Width

- **Label**
  - X, Y
  - Height
  - Width

- **Eiffel Tower**
  - X, Y
  - Height
  - Width
**Object Editor 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
**Object Editor Shape Rules**

- An object is recognized as a rectangle/line/oval if:
  - Its interface or class has the string “Rectangle”/”Oval”/”Line” in its name
  - It has properties describing the bounding box of the shape
  - Can have additional properties

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

```java
public interface Oval extends Line {
    // Additional properties for Oval
}
```
**ObjectEditor TextBox Rules**

- An object is recognized as a textbox if:
  - Its interface or class has the string “Text” in its name
  - It has properties described the bounding box of the text-box
  - It has String property Text representing contents of the text-box
  - Can have additional properties

```java
public interface TextBox {
    public int getX();
    public void setX(int newVal);
    public int getY();
    public void setY(int newVal);
    public int getWidth();
    public void setWidth(int newVal);
    public int getHeight();
    public void setHeight(int newVal);
    public String getText();
    public void setText(String newVal);
}
```
An object is recognized as a label if:

- Its interface or class has the string “Label” or “Icon” in its name
- It has properties described the bounding box of the label.
- It either has String property
  - Text giving the label text.
  - ImageFileName describing the name of a graphics file describing the label icon.
- Can have additional properties

```java
public interface SimpleLabel {
    public int getX();
    public int getY();
    public int getWidth();
    public int getHeight();
    public String getImageFileName();
}
```
**Example Label Implementation**

```java
public class AnImmutableDownLabel implements ImmutableLabel {
    int x, y;
    int width = 100;
    int height = 100;
    String imageFileName = "dn.gif";
    String text = "Down Button";
    public AnImmutableDownLabel(int theX, int theY) {
        x = theX;
        y = theY;
    }
    public int getX() { return x; }
    public int getY() { return y; }
    public int getWidth() { return width; }
    public int getHeight() { return height; }
    public String getText() { return text; }
    public String getImageFileName() { return imageFileName; }
}
```
public class ALabel implements Label {
    int width, height;
    String text, imageFile;
    Point location;
    public ALabel(
        int initX, int initY, int initWidth, int initHeight,
        String initText, String theImageFile) {
        location = new ACartesianPoint(initX, initY);
        width = initWidth;    height = initHeight;
        text = initText;    imageFile = theImageFile;
    }
    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;}
    public String getText() {return text;}
    public void setText(String newVal) {text = newVal;}
    public String getImageFileName() {return imageFile;}
    public void setImageFileName(String newVal) {imageFile = newVal;}
}
Using ALabel

```java
ObjectEditor.edit(
    new AnImmutableDownLabel(0, 0);
}
```

The label icon and image are offset 0, height/2 from upper left corner

Text follows the label in a left-to-right order.

File should be in the project folder (directory) containing bin and src