COMP 401
USER INTERFACE AND ANNOTATIONS

Instructor: Prasun Dewan
Prerequisites

- Interfaces
**Interactive App. vs. User/Prog. Interface**

```java
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);
    }
}
```

**Programming Interface**

**User Interface**

**Service**
**Programming vs. User Interface: Analogy**

**Factory**
- Manufactured by

**Operations**
- Accelerate
- Brake

**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  Sutton & Sprague ‘78
UI
Generic User Interface?

Interactive Application

- Service: 50%
- User Interface: 50%
- Library: Toolkit
- Desktop GUI: Myers ‘95

Generic UI 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);
    }
}
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);
    }
}
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);
    }
}
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);
    }
}
**Actual Trace**

```
Object: bus.uigen.ObjectEditor@5e179a
Object: state.ABMIcalculator@186c730
getHeight() Called
getWeight() Called
getBmi() Called
getWeight() Called
setWeight() Called
getWeight() Called
getHeight() Called
getWeight() Called
getBmi() Called
```

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

- Load
- Change weight
public class ABMISpreadsheetManipulatedByMainAndObjectEditor {
    public static void main (String[] args) {
        ABMISpreadsheet bmiSpreadsheet = new ABMISpreadsheet();
        ObjectEditor.edit(bmiSpreadsheet);
        bmiSpreadsheet.setHeight(1.77);
        bmiSpreadsheet.setWeight(75);
    }
}
REFRESHING OBJECTEDITOR FROM MAIN

```java
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
**DEMOING OBJECT**

```java
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

```java
public class C {
    
    public T getP() {
        ...
    }

    public void setP(T newValue) {
        ...
    }
}
```

- **Name P**
- **Type T**
- **Read-only**
- **Editable**
- **Getter method**
- **Setter method**

Bean convention: For humans and tools
public class ABMISpreadsheetNotFollowingBeanConventions {
    double height = 1.77;
    double weight = 75;
    public double getHeight() {
        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);
    }
}
**Bean Pattern Annotation**

```java
import util.annotations.StructurePattern
import util.annotations.StructurePatternNames
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
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);
    }
}
```

**Why pattern annotation?**
- **Documentation**
- **Efficiency:** OE does not need to look for pattern
- **Errors/warnings:** Can give error message if pattern not followed

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

**Available to ObjectEditor**

**Structure(<PatternName>)** before class asserts that the class is following the pattern.
Can system catch these errors?

```java
@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);
    }
}
```
import util.annotationsEditablePropertyNames;
import util.annotations.PropertyNames;
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
@PropertyNames("
  Height",
  "Weight",
  "BMI"
)
Order of Properties

```java
@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, 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);
    }
}
```
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();
}

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

Calculates BMI from height in metres and weight in kgs. Stores previously computed BMI value in a variable.

Weight is in Kilograms. Changing weight automatically recalculates BMI.
import util.annotations.Explanation;
import util.annotations.Tags;
@Tags({"Annotations", "Spreadsheet", "Properties", "Interfaces", "Object Editor"})
public class AnAnnotatedBMISpreadsheet implements AnnotatedBMISpreadsheet {
...
}
public interface Point {
    public int getX();
    public int getY();
    public double getAngle();
    public double getRadius();
}

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

(0,0) → X

(3,2) → pixels

Y → [ACartesianPoint]
**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

```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.
ADDING TREE VIEW
Tree View

```
OEFrame 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

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

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

```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);
}
```

Oval/Rectangle logical representation?

Other shapes needed in project?
**XY-based String and Image Rules**

String Shape:
- X, Y (int)
- Text
- Width (int)
- Font
- FontSize

Image Shape:
- X, Y (int)
- ImageFileName
- Height (int)
- Width (int)

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

String also has font and fontSize optional properties.
@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);
}
```java
@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; }
}
```
**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

**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 \((x, y, \text{width, height})\)

- **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
**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 (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

```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 (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

```java
@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

```java
@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