b
UNIVERSITÄT
BERN

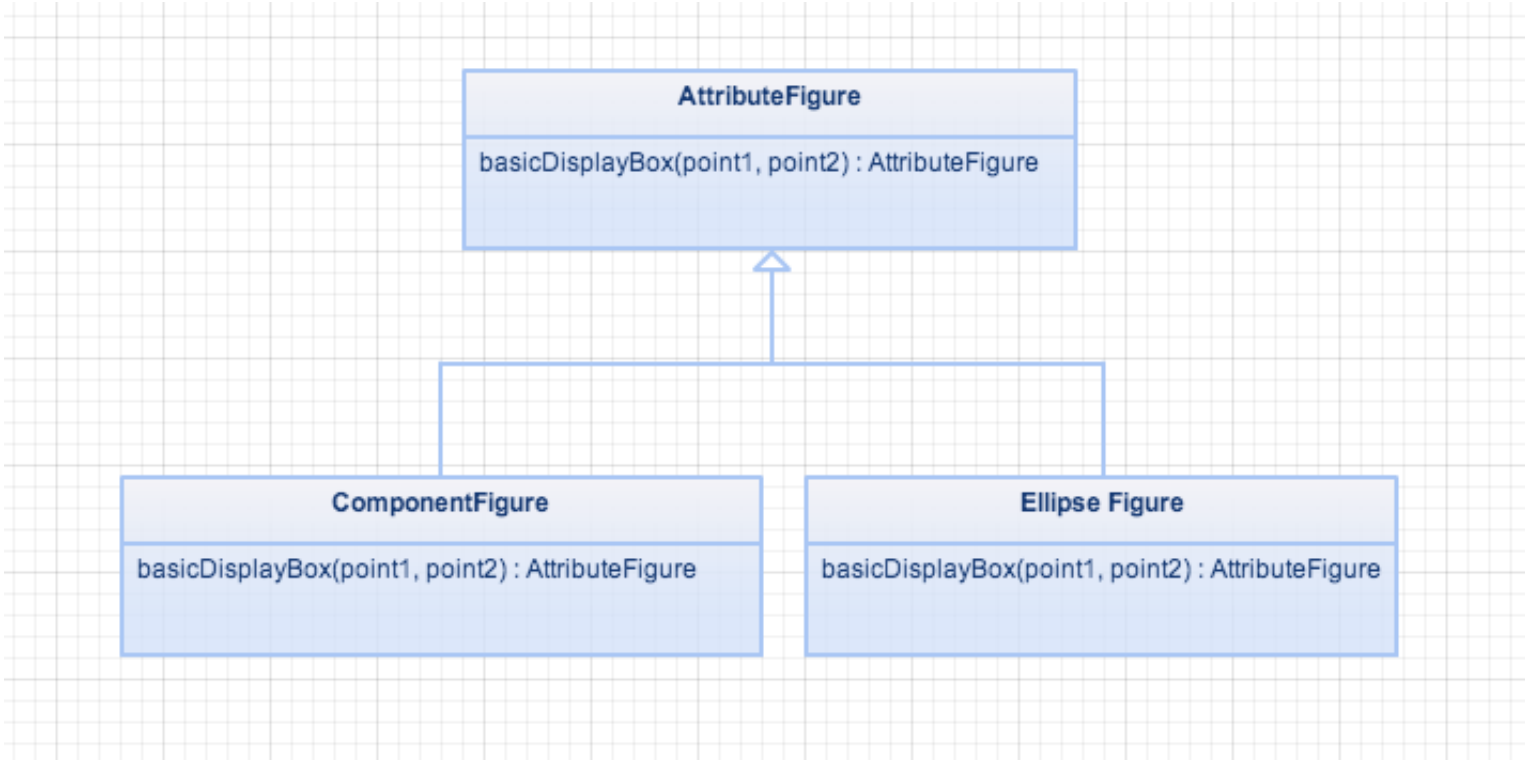
Accurate Polymorphism Detection

Nevena Milojković

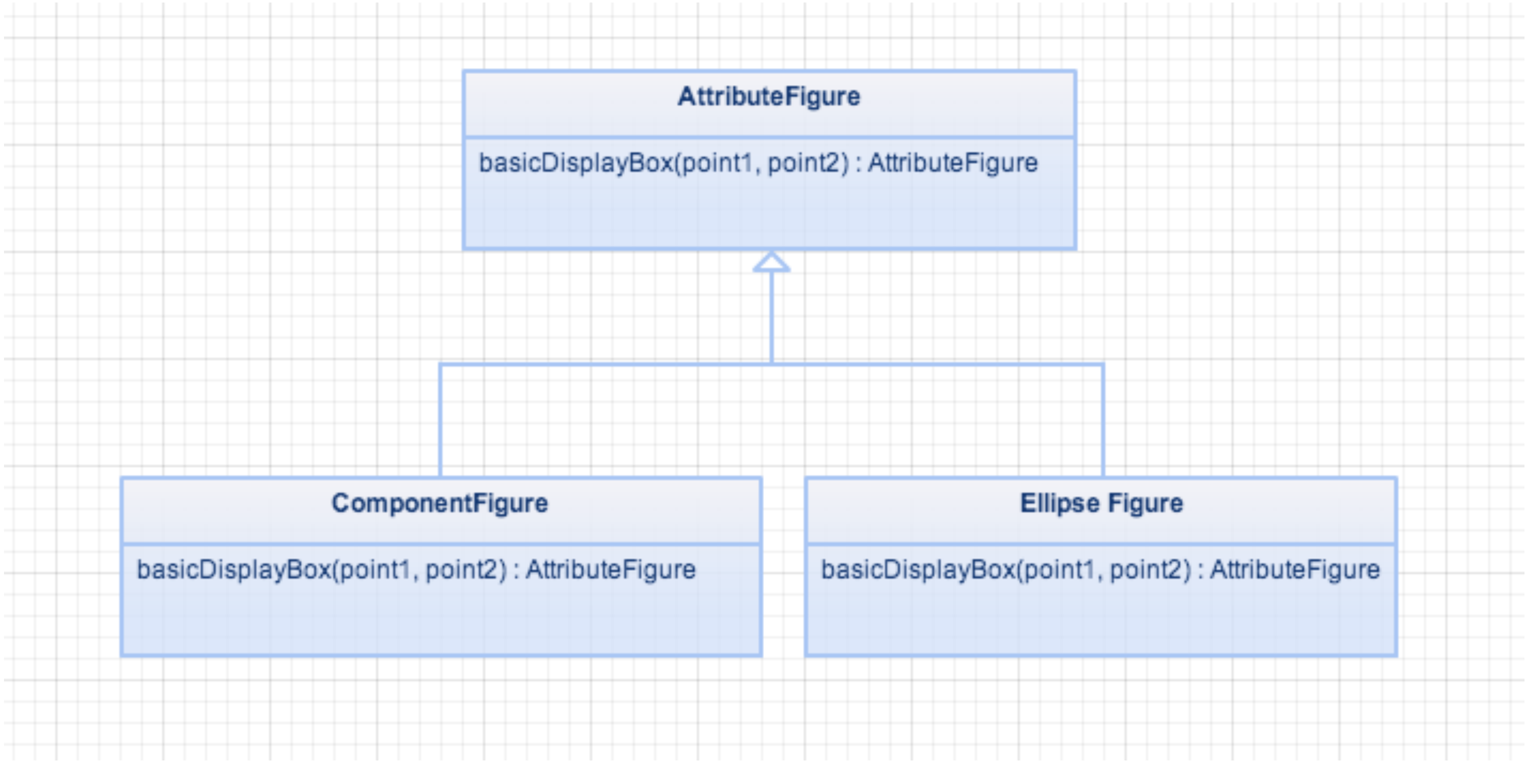
Software Composition Group

University of Bern

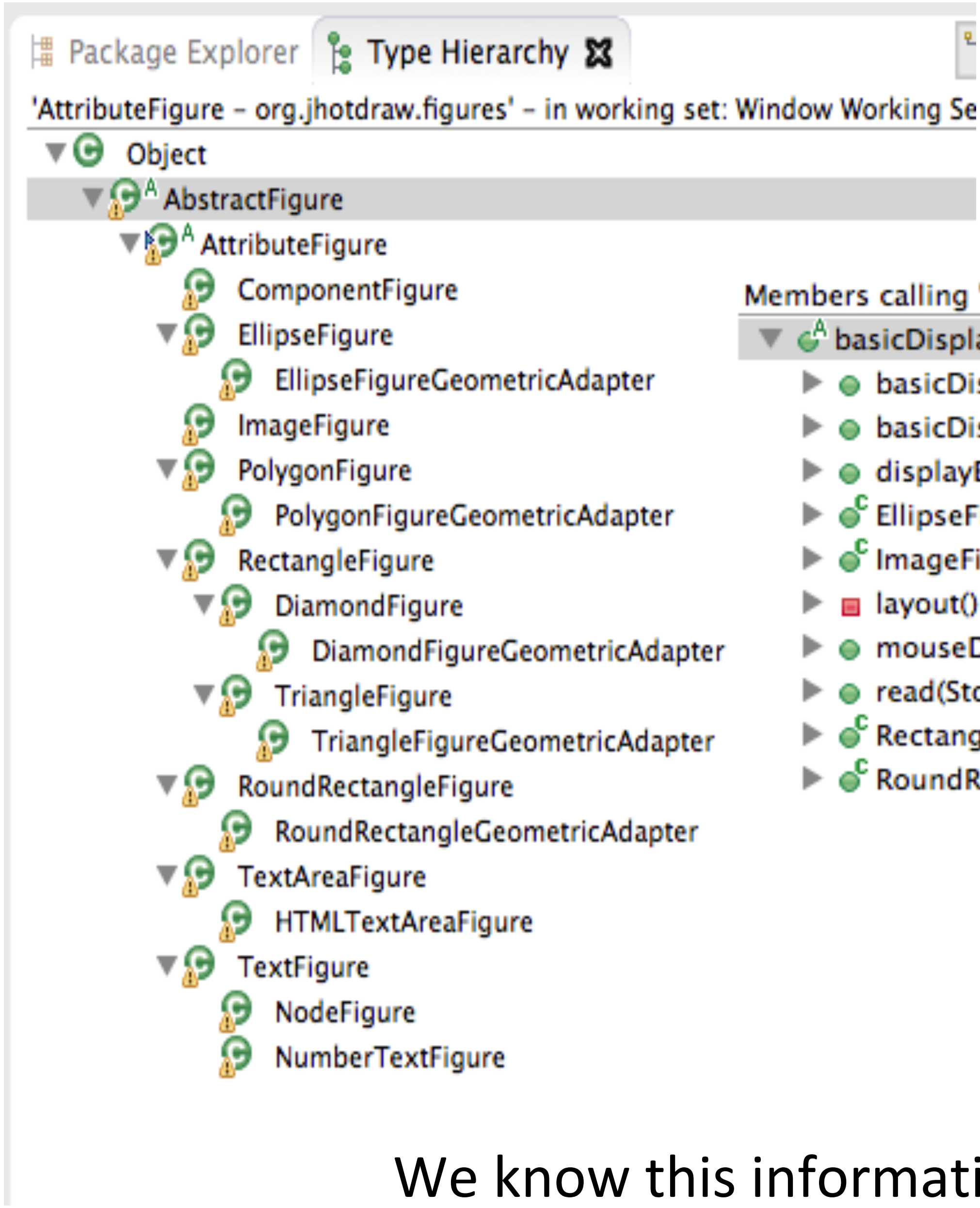
Problem


AttributeFigure
basicDisplayBox(point1, point2) : AttributeFigure
ComponentFigure
basicDisplayBox(point1, point2) : AttributeFigure
Ellipse Figure
basicDisplayBox(point1, point2) : AttributeFigure

```
public static void main(String[] args){
    AttributeFigure figure = new ComponentFigure();
    figure.basicDisplayBox(point1, point2);
}
```


AttributeFigure
basicDisplayBox(point1, point2) : AttributeFigure
ComponentFigure
basicDisplayBox(point1, point2) : AttributeFigure
Ellipse Figure
basicDisplayBox(point1, point2) : AttributeFigure

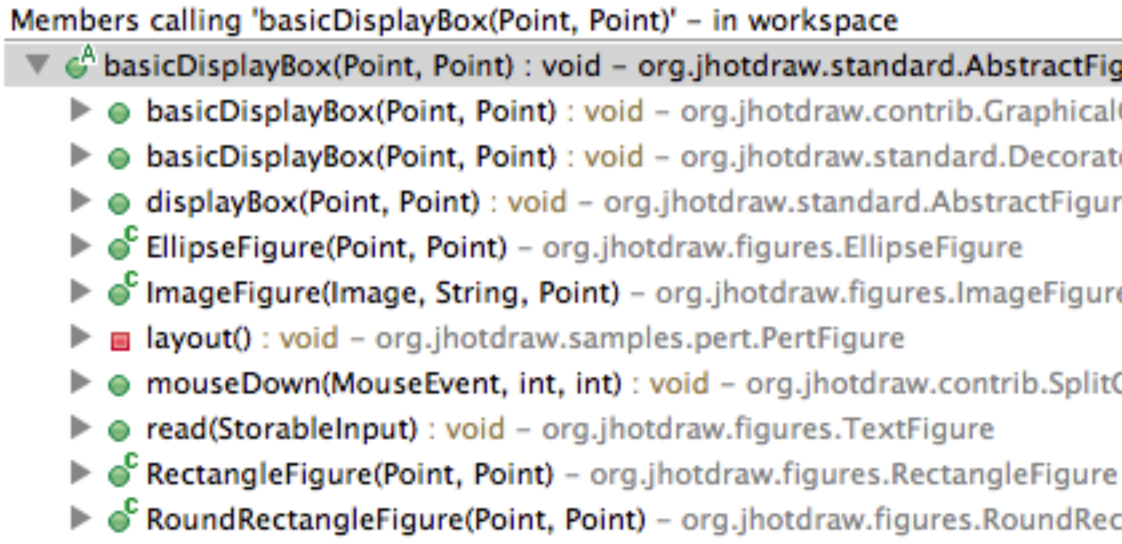
```
public static void main(String[] args){
        AttributeFigure figure = FigureFactory.getFigure();
        figure.basicDisplayBox(point1, point2);
        }
```


Package Explorer
Type Hierarchy

'AttributeFigure - org.jhotdraw.figures' - in working set: Window Working Se

```
Object
  AbstractFigure
    AttributeFigure
      ComponentFigure
      EllipseFigure
        EllipseFigureGeometricAdapter
      ImageFigure
      PolygonFigure
        PolygonFigureGeometricAdapter
      RectangleFigure
        DiamondFigure
          DiamondFigureGeometricAdapter
        TriangleFigure
          TriangleFigureGeometricAdapter
      RoundRectangleFigure
        RoundRectangleGeometricAdapter
      TextAreaFigure
        HTMLTextAreaFigure
      TextFigure
        NodeFigure
        NumberTextFigure
```

basicDisplayBox(Point origin, Point corner)


Members calling 'basicDisplayBox(Point, Point)' – in workspace

```
basicDisplayBox(Point, Point) : void – org.jhotdraw.standard.AbstractFig
  basicDisplayBox(Point, Point) : void – org.jhotdraw.contrib.Graphical
  basicDisplayBox(Point, Point) : void – org.jhotdraw.standard.Decorat
  displayBox(Point, Point) : void – org.jhotdraw.standard.AbstractFigur
  EllipseFigure(Point, Point) – org.jhotdraw.figures.EllipseFigure
  ImageFigure(Image, String, Point) – org.jhotdraw.figures.ImageFigur
  layout() : void – org.jhotdraw.samples.pert.PertFigure
  mouseDown(MouseEvent, int, int) : void – org.jhotdraw.contrib.SplitC
  read(StorableInput) : void – org.jhotdraw.figures.TextFigure
  RectangleFigure(Point, Point) – org.jhotdraw.figures.RectangleFigure
  RoundRectangleFigure(Point, Point) – org.jhotdraw.figures.RoundRec
```

We know this information at run-time.

Agenda

Problem: Program comprehension in the presence of polymorphism

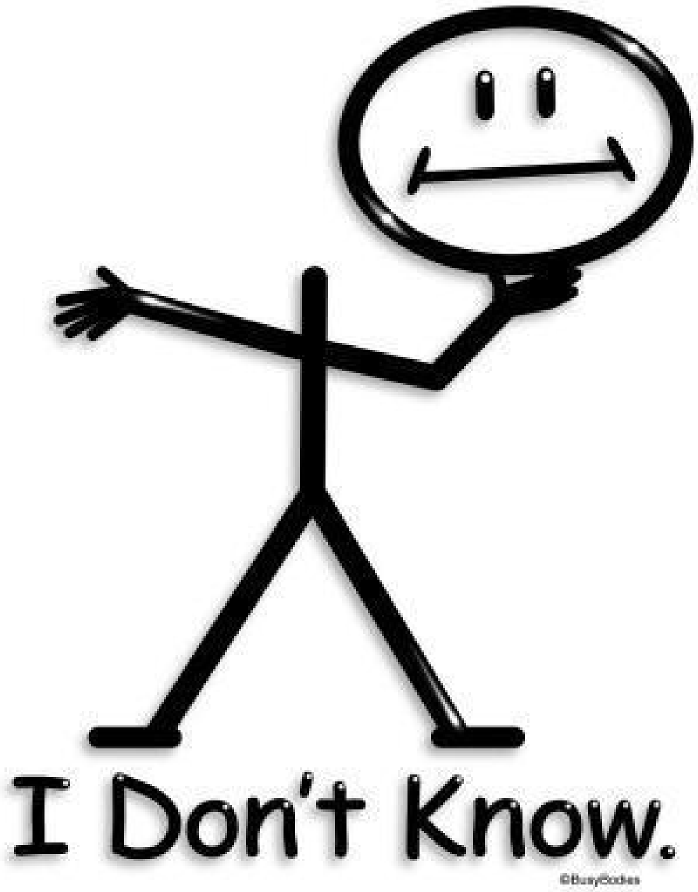
Goal: Create an accurate call-graph at code-reading-time

Idea: Compare dynamically collected results with static algorithms

Static algorithms

	Class hierarchy	Instances per project	Instances per class	Instances per field	Instances per method
1. UN					
2. CHA	✔				
3. RTA	✔	✔			
4. CTA	✔		✔		
5. MTA	✔		✔	✔	
6. FTA	✔		✔		✔
7. XTA	✔			✔	✔

What is really happening?



Collect information from a running system

Collect information about all method invocations from the project in question

Store information in a RTI (run-time information) database

Compare dynamically collected results from RTI database with static algorithms

Using Javassist to get the information

```
public void basicDisplayBox(Point origin,Point corner){
      bounds = new Rectangle(origin);
      bounds.add(corner);
}
```

```
public void basicDisplayBox(Point origin,Point corner){
      Profiler.log($0, $sig, $args);
      bounds = new Rectangle(origin);
      bounds.add(corner);
}
```

```
figure.basicDisplayBox(origin, corner);
```

```
org.jhotdraw.contrib.ComponentFigure.basicDisplayBox(Point,Point);
org.jhotdraw.contrib.PolygonFigure.basicDisplayBox(Point,Point);
org.jhotdraw.contrib.TextAreaFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.EllipseFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.ImageFigure.basicDisplayBox(Point,Point);          125
org.jhotdraw.figures.RectangleFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.RoundRectangleFigure.basicDisplayBox(Point,Point);
```

```
org.jhotdraw.contrib.ComponentFigure.basicDisplayBox(Point,Point);
org.jhotdraw.contrib.PolygonFigure.basicDisplayBox(Point,Point);        78
org.jhotdraw.contrib.TextAreaFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.EllipseFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.ImageFigure.basicDisplayBox(Point,Point);
org.jhotdraw.figures.RectangleFigure.basicDisplayBox(Point,Point);      43
org.jhotdraw.figures.RoundRectangleFigure.basicDisplayBox(Point,Point);
```

How confident are we in our results?

62% of used fields
26% of used methods
64% of used constructors
65% of used classes

- Implement more static algorithms
- Implement three-stage analysis
- Improve performance for dynamic analysis
- Run analysis on more projects
- Integrate a tool into IDE

Additional uses of the RTI database

- Usage of fields
- All methods invocations
- Study null pointer propagation

Summary

- Call graph helps source code comprehension
- Polymorphism introduces ambiguity in the call-graph
- Static algorithms give false positives
- Dynamic analysis give false negatives
- Their combination could yield more accurate results, at a reasonable cost, to support the developer