

# Do We Really Need to Extend Syntax for Advanced Modularity?

Shigeru Chiba<sup>1</sup>, Michihiro Horie<sup>2</sup>, Kei Kanazawa,  
Fuminobu Takeyama, Yuuki Teramoto

Tokyo Institute of Technology, Japan

<sup>1</sup>Also, The University of Tokyo and JST CREST

<sup>2</sup>Currently, IBM Research Tokyo

# Advanced modularity

---

## ▶ Objectives

- Code reuse
  - Avoid code clones (duplicated code)
- Composability
  - Build software by combining modules **as is**  
(without modifications)
- Grouping related code
  - for **spatial locality** per concern
- Information hiding
  - Give higher-level abstraction

# Observer pattern in Java

```
class Line extends Shape {
    int angle, len;

    void setPos(int nx, int ny) {
        x = nx; y = ny;
        DisplayUpdate.after(this);
    }

    void setLength(int nlen) {
        len = nlen;
        DisplayUpdate.after(this);
    }

    int getLength() { return len; }
    /* (snip) */
}
```

Crosscutting concern

```
class Rectangle {
    void setWidth(int nw) {
        width = nw;
        DisplayUpdate.after(this);
    }
}
```

```
class DisplayUpdate {
    static void after(Shape s) {
        s.display().repaint();
    }
}
```



# Modularization in AspectJ

- ▶ The display-update concern is in an aspect.
  - Good composability *i.e.* easily removable

```
class Line extends Shape {  
    int angle, len;  
    void setPos(int nx, int ny) {  
        x = nx; y = ny;  
    }  
    // (snip)  
}
```

```
aspect DisplayUpdate {  
    pointcut change(Shape s):  
        execution(void Shape+.set*(..)) && this(s);  
    after(Shape s): change(s) {  
        s.display().repaint(s);  
    }  
}
```

define when the advice is invoked

# Limitation of AspectJ: grouping related code

- ▶ A code snippet belongs to multiple concerns.

Original

```
class Line extends Shape {  
    void setPos(int nx, int ny) {  
        x = nx; y = ny;  
        DisplayUpdate.after(this);  
    }  
  
    void setLength(int nlen) {  
        len = nlen;  
        DisplayUpdate.after(this);  
    }  
    // (snip)  
}
```

Class + aspect

```
class Line extends Shape {  
    void setPos(int nx, int ny) {  
        x = nx; y = ny;  
        }  
    }  
  
    void setLength(int nlen) {  
        len = nlen;  
        }  
    }  
    // (snip)  
}
```

Cannot see the entire  
Line concern!!

```
aspect DisplayUpdate {  
    // (snip)  
    after(Shape s): change(s) {  
        s.display().repaint(s);  
    }}  
}
```

# Another Limitation of AspectJ: Code reuse

- ▶ Cannot avoid code duplication

DisplayUpdate

```
void around(Line line, int x, int y):  
    execution(void Shape+.setPos(int, int)) && this(line) && args(x, y) {  
    if (line.x != x || line.y != y) {  
        proceed(line, x, y);  
        line.display().repaint(line);  
    }  
}  
  
void around(Line line, int len):  
    execution(void Line.setLength(int)) && this(line) && args(len) {  
    if (line.len != len) {  
        proceed(line, len);  
        line.display.repaint(line);  
    }  
}  
// (snip)
```

only if a filed value is changed

# Meta-helical contextual predicate dispatch

- ▶ An extension to contextual predicate dispatch

[Chiba, et al, OOPSLA'10]

```
void around anySetter(Line line, Method setter, Object[] args):  
    execution(void Shape+.setPoint(int,int)  
        || void Shape+.setPoint.setLength(int))  
    && this(line) => $setter($args)  
  
boolean around posSetter(int x, int y): within anySetter  
    execution(void Shape+.setPos(int,int)) && args(x, y)  
{ line.x != x || line.y != y } => $conditionExpr  
  
boolean around lenSetter(int len): within anySetter  
    execution(void Shape+.setLength(int)) && args(len)  
{ line.len != len } => $conditionExpr  
  
void setter(args) within anySetter && (posSetter || lenSetter) {  
    super.setter(args);  
    if (conditionExpr)  
        line.display().repaint(line);  
}
```

# End of Skit

An extension to contextual predicate dispatch

[Chiba, et al, OOPSLA'10]

## Developing a complex language.

This is our typical research, but  
is this a right approach?

**(The last slide is fake)**



# Do We Really Need to Extend Syntax for Advanced Modularity?

Shigeru Chiba<sup>1</sup>, Michihiro Horie<sup>2</sup>, Kei Kanazawa,  
Fuminobu Takeyama, Yuuki Teramoto

Tokyo Institute of Technology, Japan

<sup>1</sup>Also, The University of Tokyo and JST CREST

<sup>2</sup>Currently, IBM Research Tokyo

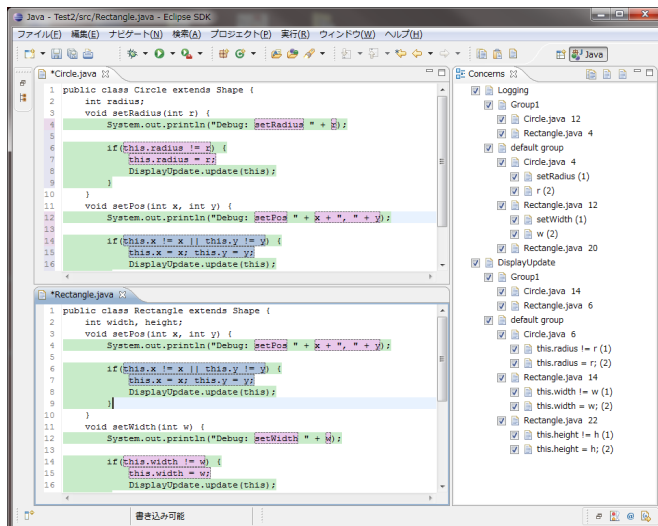
# Programs expressed in dynamic text

Our vision

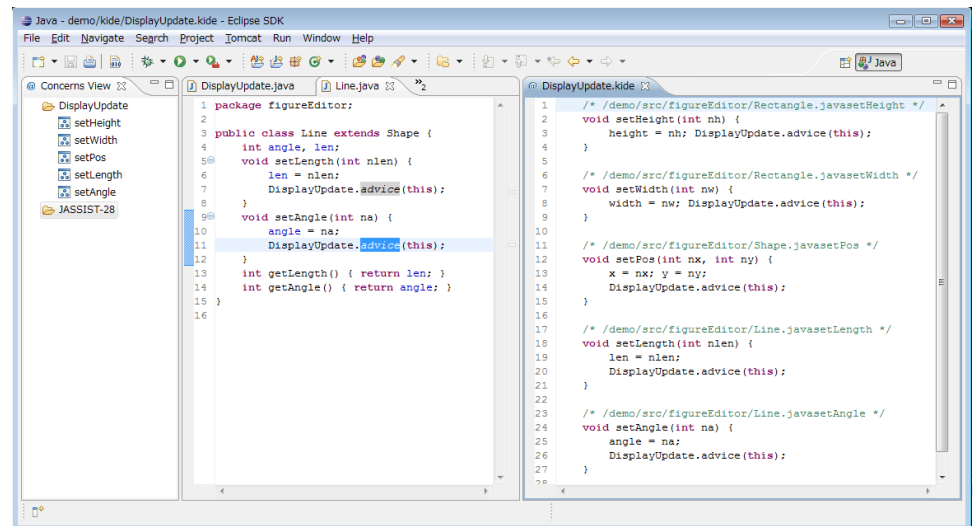
## ► Dynamic text

- changes its appearance while editing and browsing
  - like web page written in HTML + JavaScript
- Traditional languages uses only **static text**.

## ► Two preliminary systems for implementing this idea.



```
1 public class Circle extends Shape {
2     int radius;
3     void setRadius(int r) {
4         system.out.println("Debug: setRadius = " + r);
5
6         if (this.radius != r) {
7             this.radius = r;
8             DisplayUpdate.update(this);
9         }
10    }
11    void setPos(int x, int y) {
12        system.out.println("Debug: setPos = " + x + ", " + y);
13
14        if (this.x != x || this.y != y) {
15            this.x = x; this.y = y;
16            DisplayUpdate.update(this);
17        }
18    }
19 }
20
21 *Rectangle.java
22 1 public class Rectangle extends Shape {
23     int width, height;
24     void setPos(int x, int y) {
25         system.out.println("Debug: setPos = " + x + ", " + y);
26
27         if (this.x != x || this.y != y) {
28             this.x = x; this.y = y;
29             DisplayUpdate.update(this);
30         }
31     }
32     void setWidth(int w) {
33         system.out.println("Debug: setWidth = " + w);
34
35         if (this.width != w) {
36             this.width = w;
37             DisplayUpdate.update(this);
38         }
39     }
40     void setHeight(int h) {
41         system.out.println("Debug: setHeight = " + h);
42
43         if (this.height != h) {
44             this.height = h;
45             DisplayUpdate.update(this);
46         }
47     }
48 }
```



```
1 package figureEditor;
2
3 public class Line extends Shape {
4     int angle, len;
5     void setLength(int nlen) {
6         len = nlen;
7         DisplayUpdate.advice(this);
8     }
9     void setAngle(int na) {
10        angle = na;
11        DisplayUpdate.advice(this);
12    }
13    int getLength() { return len; }
14    int getAngle() { return angle; }
15 }
16
17 *DisplayUpdate.kide
18 1 /* /demo/src/figureEditor/Rectangle.javasetHeight */
19 2 void setHeight(int nh) {
20 3     height = nh; DisplayUpdate.advice(this);
21 4 }
22
23 /* /demo/src/figureEditor/Rectangle.javasetWidth */
24 5 void setWidth(int nw) {
25 6     width = nw; DisplayUpdate.advice(this);
26 7 }
27
28 /* /demo/src/figureEditor/Shape.javasetPos */
29 8 void setPos(int nx, int ny) {
30 9     x = nx; y = ny;
31 10    DisplayUpdate.advice(this);
32 11 }
33
34 /* /demo/src/figureEditor/Line.javasetLength */
35 12 void setLength(int nlen) {
36 13     len = nlen;
37 14    DisplayUpdate.advice(this);
38 15 }
39
40 /* /demo/src/figureEditor/Line.javasetAngle */
41 16 void setAngle(int na) {
42 17     angle = na;
43 18    DisplayUpdate.advice(this);
44 19 }
45
46 20
47 21
48 22
49 23
50 24
51 25
52 26
53 27
54 28
55 29
56 30
```

# Programs expressed in dynamic text

Our vision

- ▶ Live text
  - changes its appearance while editing and browsing
    - like web page written in HTML + JavaScript
  - Traditional languages uses only **static text**.
- ▶ Two preliminary systems for implementing this idea.

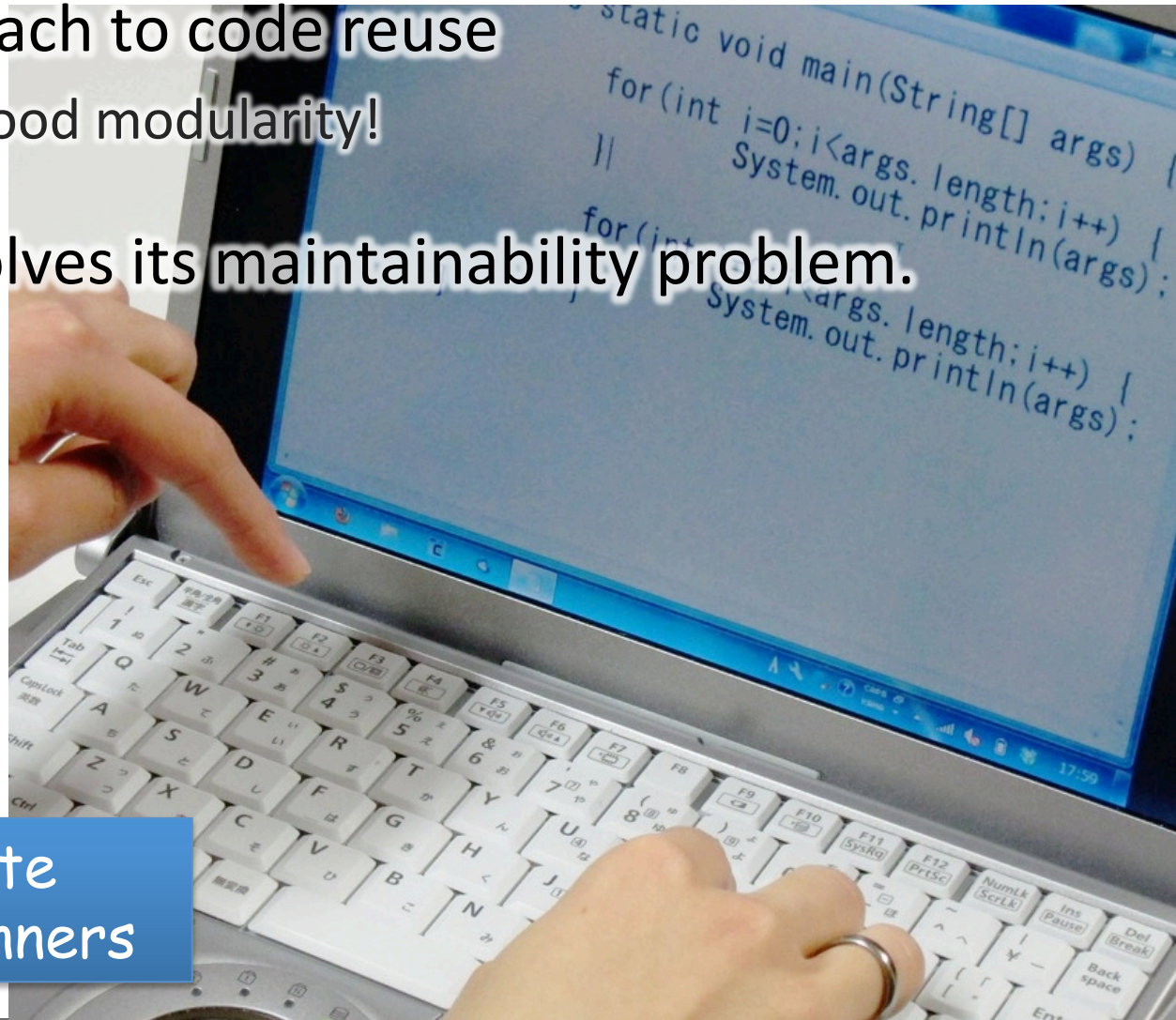
```
1 public class Circle extends Shape {
2     int radius;
3     void setRadius(int r) {
4         system.out.println("Debug: setRadius = " + r);
5
6         if (this.radius != r) {
7             this.radius = r;
8             DisplayUpdate.update(this);
9         }
10    }
11    void setPos(int x, int y) {
12        System.out.println("Debug: setPos = " + x + ", " + y);
13
14        if (this.x != x || this.y != y) {
15            this.x = x; this.y = y;
16            DisplayUpdate.update(this);
17        }
18    }
19 }
20
21 *Rectangle.java
22 1 public class Rectangle extends Shape {
23     int width, height;
24     void setPos(int x, int y) {
25         System.out.println("Debug: setPos = " + x + ", " + y);
26
27         if (this.x != x || this.y != y) {
28             this.x = x; this.y = y;
29             DisplayUpdate.update(this);
30         }
31     }
32     void setWidth(int w) {
33         System.out.println("Debug: setWidth = " + w);
34
35         if (this.width != w) {
36             this.width = w;
37             DisplayUpdate.update(this);
38         }
39     }
40     void setHeight(int h) {
41         System.out.println("Debug: setHeight = " + h);
42
43         if (this.height != h) {
44             this.height = h;
45             DisplayUpdate.update(this);
46         }
47     }
48 }
```

```
1 package figureEditor;
2
3 public class Line extends Shape {
4     int angle, len;
5     void setLength(int nlen) {
6         len = nlen;
7         DisplayUpdate.advice(this);
8     }
9     void setAngle(int na) {
10        angle = na;
11        DisplayUpdate.advice(this);
12    }
13    int getLength() { return len; }
14    int getAngle() { return angle; }
15 }
16
17 *DisplayUpdate.kide
18 1 /* /demo/src/figureEditor/Rectangle.javasetHeight */
19 2 void setHeight(int nh) {
20 3     height = nh; DisplayUpdate.advice(this);
21 4 }
22
23 /* /demo/src/figureEditor/Rectangle.javasetWidth */
24 5 void setWidth(int nw) {
25 6     width = nw; DisplayUpdate.advice(this);
26 7 }
27
28 /* /demo/src/figureEditor/Shape.javasetPos */
29 8 void setPos(int nx, int ny) {
30 9     x = nx; y = ny;
31 10    DisplayUpdate.advice(this);
32 11 }
33
34 /* /demo/src/figureEditor/Line.javasetLength */
35 12 void setLength(int nlen) {
36 13     len = nlen;
37 14    DisplayUpdate.advice(this);
38 15 }
39
40 /* /demo/src/figureEditor/Line.javasetAngle */
41 16 void setAngle(int na) {
42 17     angle = na;
43 18    DisplayUpdate.advice(this);
44 19 }
45
46 20
47 21
48 22
49 23
50 24
51 25
52 26
53 27
54 28
55 29
56 30
```

# Copy & paste

---

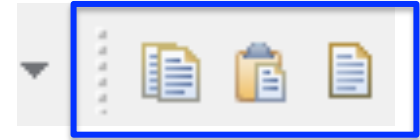
- ▶ The easiest approach to code reuse
  - actually provides good modularity!
- ▶ Dynamic text resolves its maintainability problem.



We love copy & paste  
when we were beginners

# Synchronous copy & paste

- ▶ Synchronizes code clones
- ▶ Automatically updates synchronized clones when one of them is updated



```
*Line.java X
1
2
3
4
5     void setPos(int nx, int ny) {
6         x = nx; y = ny;
7         this.display().repaint(this);
8     }
9
10    void setLength(int nlen) {
11        len = nlen;
12        this.display().repaint(this);
13    }
14 }
15
```

# Holes

- ▶ The text in the holes is not synchronized
  - Can synchronize somewhat different code clones

```
ne.java X
public class Line extends Shape {
    int angle, len;
    void setPos(int x, int y) {
        if (this.x != x || this.y != y) {
            this.x = x; this.y = y;
            this.display().repaint(this);
        }
    }
    void setLength(int len) {
        if (this.len != len) {
            this.len = len;
            this.display().repaint(this);
        }
    }
    void setAngle(int angle) {
        if (this.angle != angle) {
            this.angle = angle;
            this.display().repaint(this);
        }
    }
    int getLength() { return len; }
}
```

The image shows a screenshot of a Java IDE with a file named 'ne.java'. The code defines a 'Line' class that extends 'Shape'. It has three setter methods: 'setPos', 'setLength', and 'setAngle', and one getter method 'getLength'. Each setter method contains an 'if' statement that checks for a change in a field and then updates it and calls 'display().repaint(this)'. The code is highlighted in green. A blue callout bubble labeled 'Hole' points to the 'if' condition in the 'setPos' method. A green callout bubble labeled 'Synchronized' points to the 'if' condition in the 'setAngle' method.

# It's like a procedure or an advice!

- ▶ Synchronized clones look like a procedure
  - Holes are parameters
- ▶ Better reusability
  - resolves the problem of AspectJ

```
void updateDisplay(cond, do) {  
  if (cond) {  
    do;  
    this.display().repaint(this);  
  }  
}
```

```
ne.java X  
public class Line extends Shape {  
    int angle, len;  
    void setPos(int x, int y) {  
        if (this.x != x || this.y != y) {  
            this.x = x; this.y = y;  
            this.display().repaint(this);  
        }  
    }  
    void setLength(int len) {  
        if (this.len != len) {  
            this.len = len;  
            this.display().repaint(this);  
        }  
    }  
    void setAngle(int angle) {  
        if (this.angle != angle) {  
            this.angle = angle;  
            this.display().repaint(this);  
        }  
    }  
    int getLength() { return len; }  
}
```

call

# Concerns view

- ▶ Shows a group of code synchronized with each other.
  - provides the same information that AspectJ pointcuts do.
  - actual parameters are displayed

where

parameters

```
Line.java 4  
this.x != x || this.y != y (1)  
this.x = x; this.y = y; (2)
```

```
execution(void Shape+.setPos(int, int)) &&  
this(line) && args(x, y)
```

Concerns

- DisplayUpdate
  - Default Group
    - Line.java 4
      - this.x != x || this.y != y (1)
      - this.x = x; this.y = y; (2)
    - Line.java 10
      - this.len != len (1)
      - this.len = len; (2)
    - Line.java 16
    - Circle.java 4
      - this.radius != radius (1)

Tokyo Tech



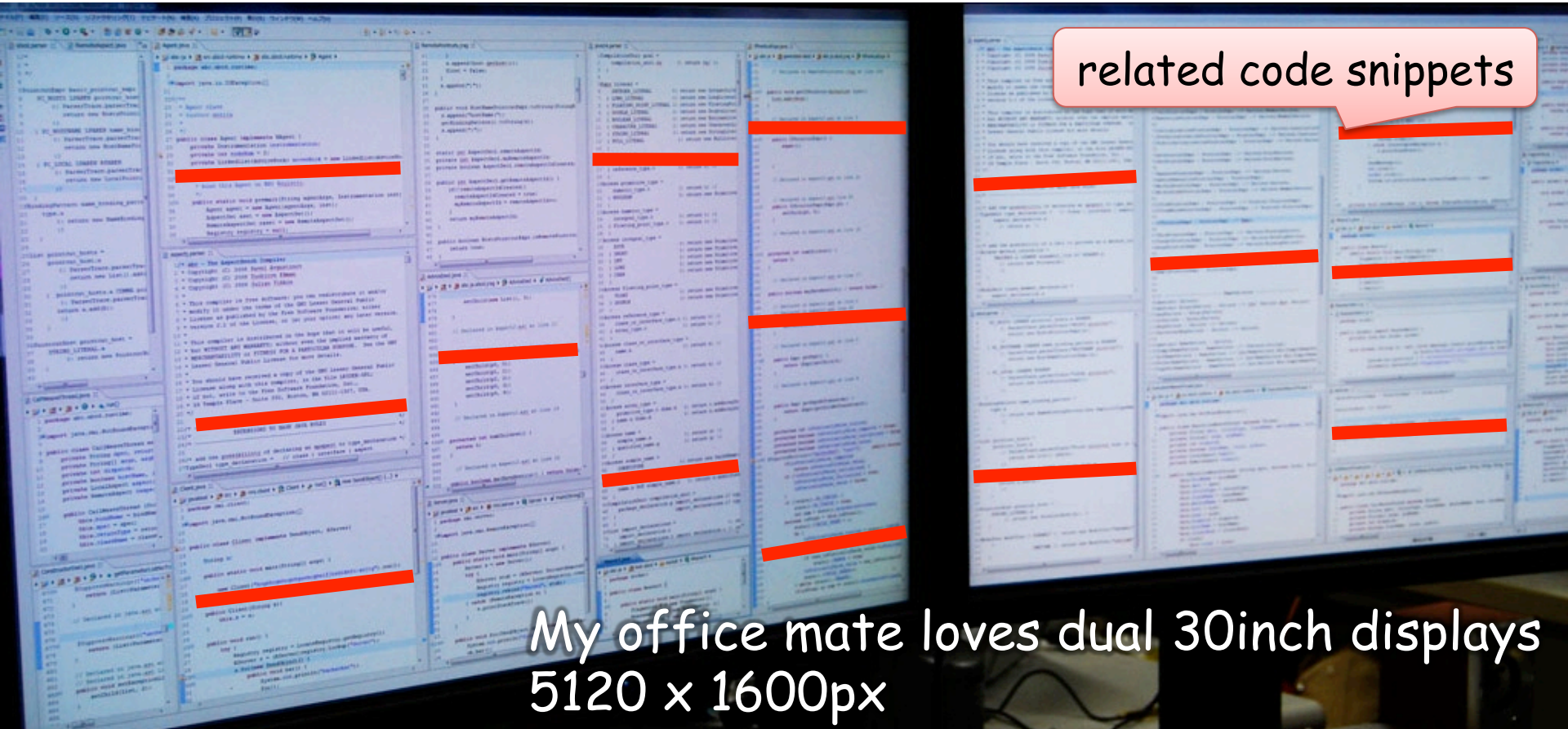
# Browsing related code at a glance

---

- ▶ Developers have to open each file

# Browsing related code at a glance

- ▶ Developers have to open each file

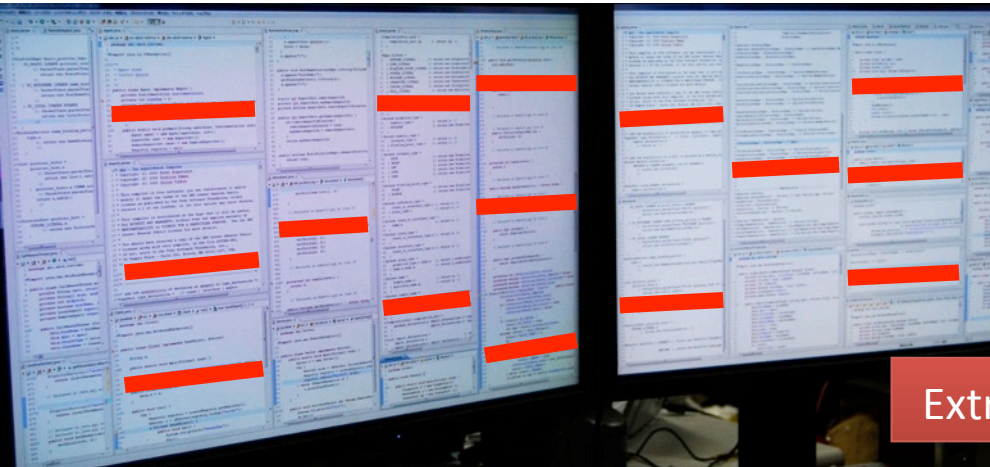


related code snippets

My office mate loves dual 30inch displays  
5120 x 1600px

# Grouping related code with Kide

- ▶ A virtual source file
  - collects related snippets from many source files



Physical source files

Extract

A virtual source file

DisplayUpdate.kide

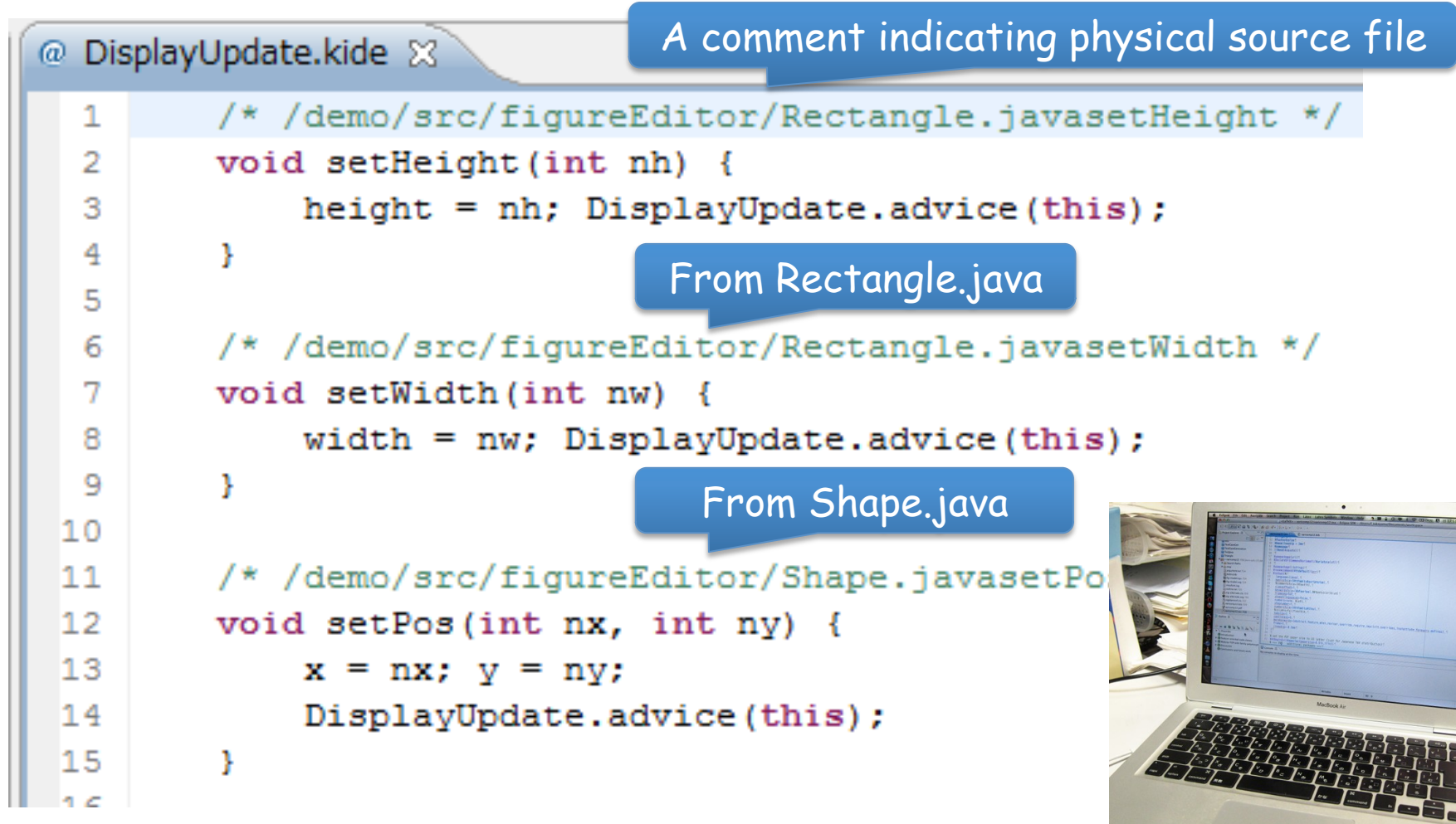
```
/* src/Line.java */  
void setPos(int nx, int ny) {  
    x = nx; y = ny;  
    DisplayUpdate.advice(this);  
}
```

```
/* src/Line.java */  
void setLength(int nlen) {  
    len = nlen;  
    DisplayUpdate.advice(this);  
}
```

```
/* src/Rectangle.java */  
void setWidth(int nlen) {  
    len = nlen;  
    DisplayUpdate.advice(this);  
}
```

# A virtual source file

- ▶ Developers can edit as a normal source file

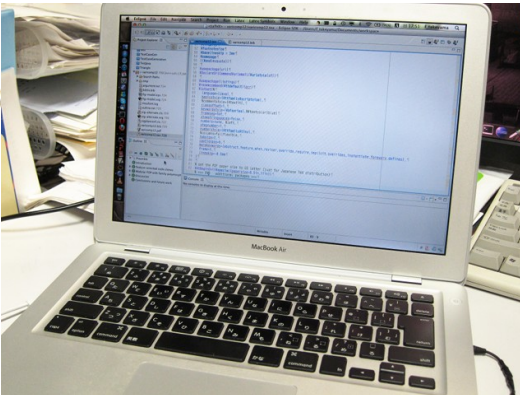


The screenshot shows an IDE window titled "@ DisplayUpdate.kide x". The code is as follows:

```
1  /* /demo/src/figureEditor/Rectangle.javasetHeight */
2  void setHeight(int nh) {
3      height = nh; DisplayUpdate.advice(this);
4  }
5
6  /* /demo/src/figureEditor/Rectangle.javasetWidth */
7  void setWidth(int nw) {
8      width = nw; DisplayUpdate.advice(this);
9  }
10
11 /* /demo/src/figureEditor/Shape.javasetPo
12 void setPos(int nx, int ny) {
13     x = nx; y = ny;
14     DisplayUpdate.advice(this);
15 }
```

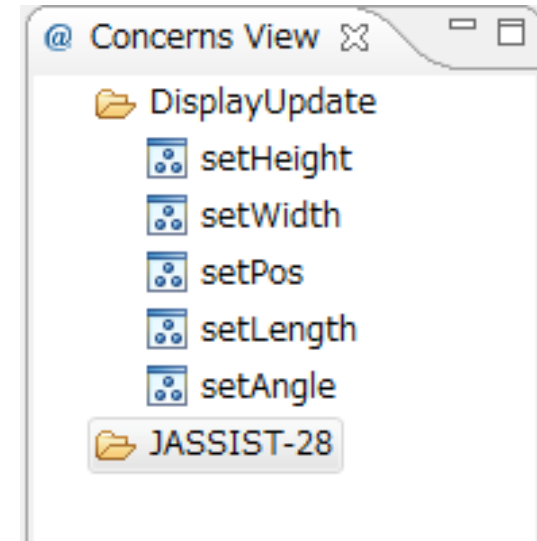
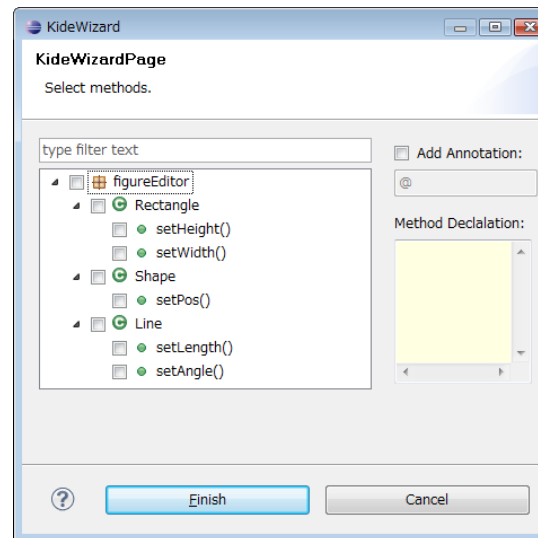
Annotations in blue callouts:

- "A comment indicating physical source file" points to the first comment line.
- "From Rectangle.java" points to the `setHeight` method.
- "From Shape.java" points to the `setPos` method.



# Selecting code snippets for a virtual source

- ▶ by Wizard-like UI
  - no pointcut language
  - supports to select all methods calling a specific method
    - like AspectJ's call pointcut
- ▶ A virtual source files is saved as part of a project
  - listed in concerns view



# Better grouping than AspectJ

- ▶ From the viewpoint of the Line concern
  - the same presentation as in pure Java

```
class Line extends Shape {
    void setPos(int nx, int ny) {
        x = nx; y = ny;
        DisplayUpdate.after(this);
    }

    void setLength(int nlen) {
        len = nlen;
        DisplayUpdate.after(this);
    }

    int getLength() {
        return len;
    }
    // (snip)
}
```

# Better grouping than AspectJ (cont.)

## ► From other viewpoints

```
class Rectangle extends Shape {  
    void setWidth(int nw) {  
        width = nw;  
        DisplayUpdate.after(this)  
    }  
    // (snip)  
}
```

A physical source file

DisplayUpdate.kide

```
/* src/Line.java */  
void setLength(int nlen) {  
    len = nlen;  
    DisplayUpdate.after(this);  
}  
  
/* src/Rectangle.java */  
void setWidth(int nlen) {  
    len = nlen;  
    DisplayUpdate.after(this);  
}
```

Virtual source files

SetWidthHeight.kide

```
/* src/Circle.java */  
void setWidth(int nw) {  
    this.width = nw; /*(snip)*/  
    DisplayUpdate.after(this);  
}  
  
/* src/Rectangle.java */  
void setWidth(int nlen) {  
    len = nlen;  
    DisplayUpdate.after(this);  
}
```

# Crosscutting over documentation

- ▶ A virtual source can contain plain text from any text file
  - Literate programming
- ▶ Bug report
  - Kide helps to quote method declarations from source files

description about this bug

Related declarations of method

jassist#28.kide

```
at sun.reflect.DelegatingMethodAccessorImpl.in
at java.lang.reflect.Method.invoke (Method.java
at javassist.util.proxy.FactoryHelper.toClass(
... 46 more
```

The performance is also very bad in comparison

Solution:

The folloing are the methods I changed for fix

```
/* /javassist/src/javassist/CtClass.java/a
public void addConstructor(CtConstructor c
    throws CannotCompileException
{
    checkModify();
    if (c.getDeclaringClass() != this)
        throw new CannotCompileException("

    getConstructorsCache();
    constructorsCache = (CtConstructor)CtM
    getClassFile2().addMethod(c.getMethodI
```



# Related work: code clone

---

- ▶ Tools for managing code clones
  - Simultaneous editing [R. C. Miller, et al., USENIX 2002]
  - Linked Editing [M. Toomim, et al., VLHCC 2004]
    - supports parameters
  - CReN [D. Hou, et al., CASCON 2009]
- ▶ These tools do not provide the concerns view
  - Sync. copy & paste does since its focus is on modularity.

# Related work: aspect orientation

---

- ▶ Mylar [M. Kersten, et al., AOSD 2005]
  - monitors developers' action and detects a group of related code
    - Temporary concerns
  - does not change the expression of source code
- ▶ Fluid AOP [T. Hon, et al., OOPSLA 2006]
  - can show a program differently from the representation in the actual source file
  - requires AspectJ-like syntax
    - Sync. Copy & paste and Kide does not

# Related work: other systems

---

- ▶ CIDE [C. Kästner, et al., ICSE 2008]
  - Developers can mark a code snippet with a color of its feature
    - They can include/exclude all the code snippets related to a given feature
    - by a single action
    - without syntactic directive `#ifdef`
  - Its navigation panel is similar to our concerns view
  - It does not provide a virtual source file
- ▶ Code Bubbles [A. Bragon, et al. ICSE 2010]
  - A small editor dedicated for every method can be freely placed on the screen
    - can be used as an editor of a virtual source file
  - does not provide a wizard-based support tool

# Approaches to modularity

- ▶ A language based on ...

## static text

What is it suitable for?

## Dynamic text

Synchronous copy & paste  
Kide

## 2D figures and diagrams

Executable UML

