Asking and Answering Why and Why Not Questions about Program Behavior

Amy J. Ko

Information School
UNIVERSITY of WASHINGTON

and

Brad Myers

Carnegie Mellon

... **identifying and correcting defects**
during the software development process
represents over **half** of development **costs** ...
and accounts for **30 to 90 percent of labor**
expended to produce a working program.”

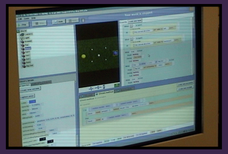
National Institute of Standards and Technology, 2002

Testing, debugging, deployment, maintenance...

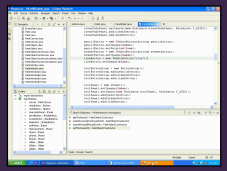
Initial development

why is debugging so **difficult**?

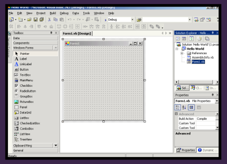
four studies to find out...



10 Alice developers in the lab and field



30 Java developers using Eclipse



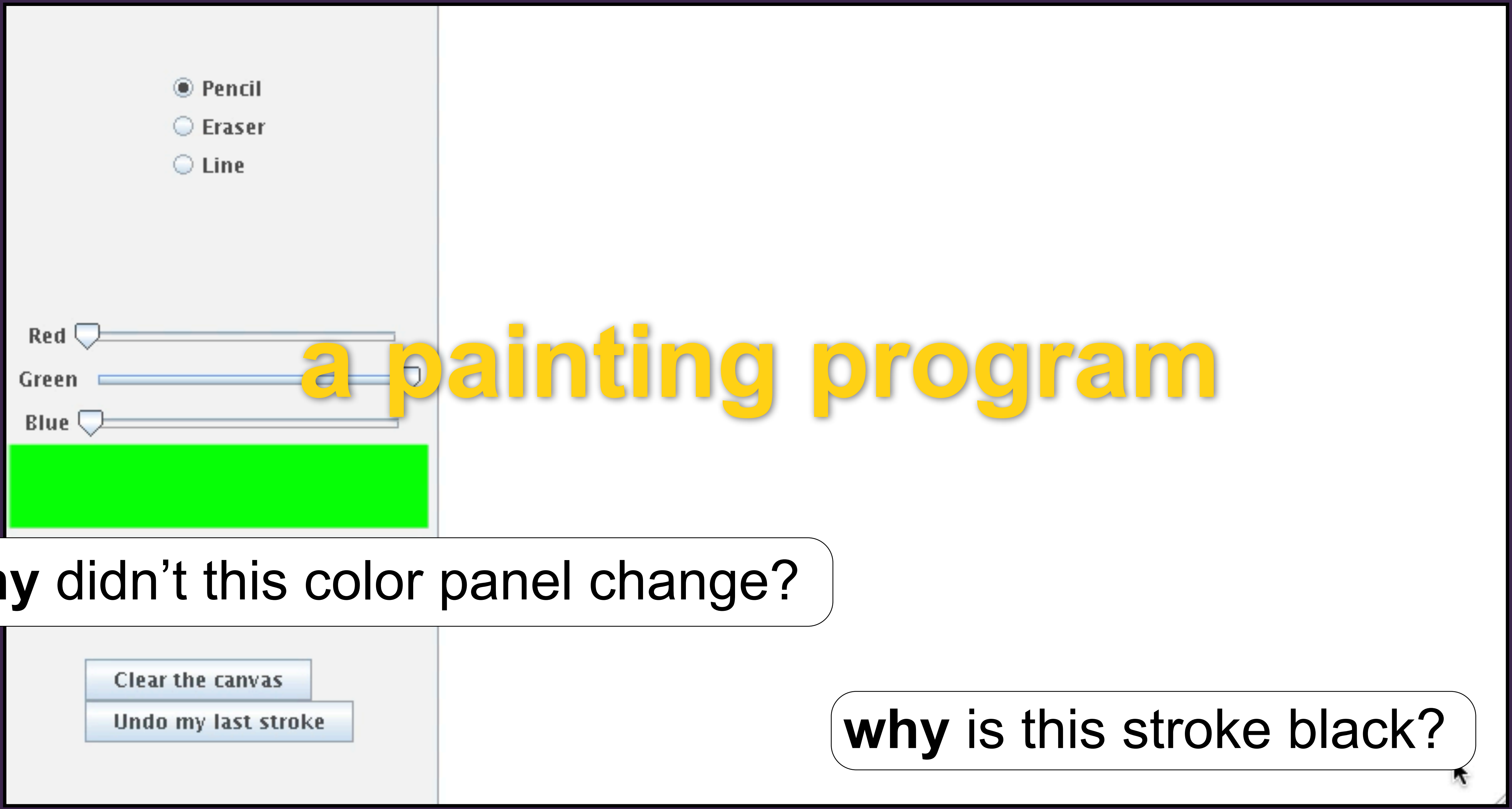
30 students learning Visual Studio

18 software teams at Microsoft

the problem

today's tools **require** people to **guess** what **code** is responsible

one bug, **two symptoms**



● Pencil
○ Eraser
○ Line

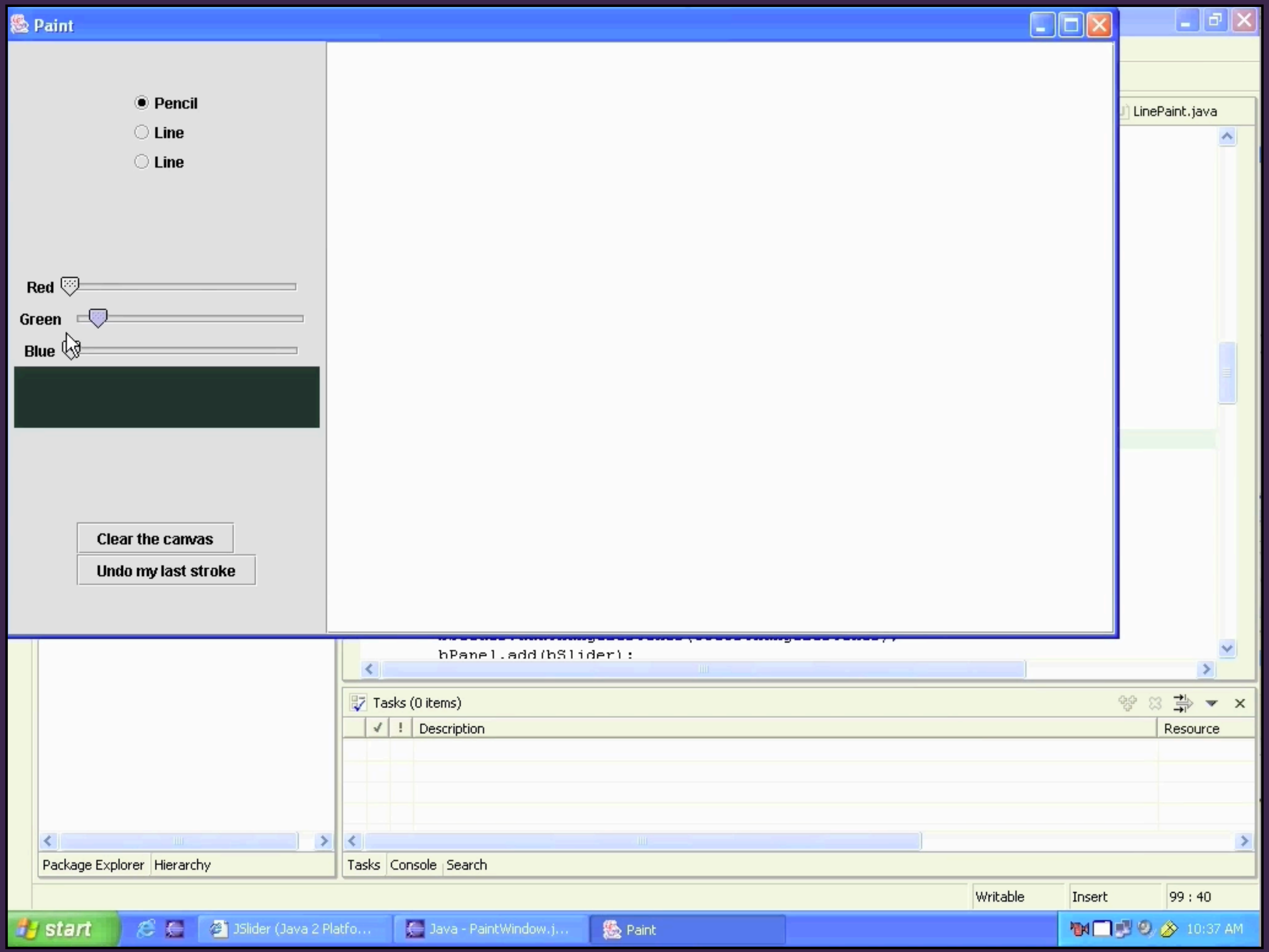
a painting program

why didn’t this color panel change?

why is this stroke black?

debugging with **current tools**

why is the stroke black?



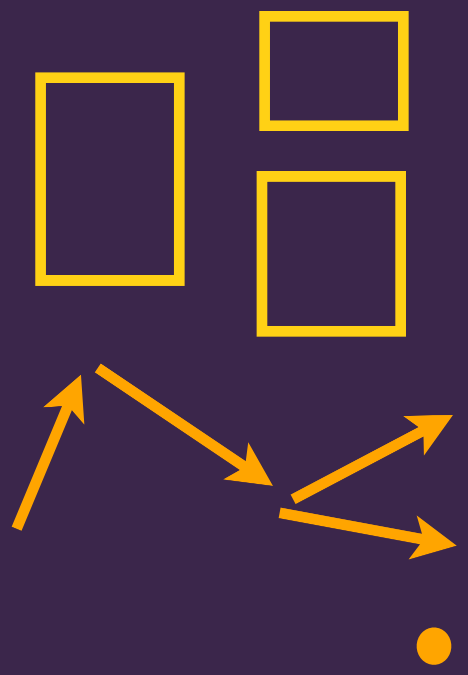
```
bPanel.add(bSlider);
```

maybe a slider initialization problem...

maybe th lider isn't connected to anything...

is the JSli argument incorrect?

maybe the olor isn't compute properly...



breakpoint

println()

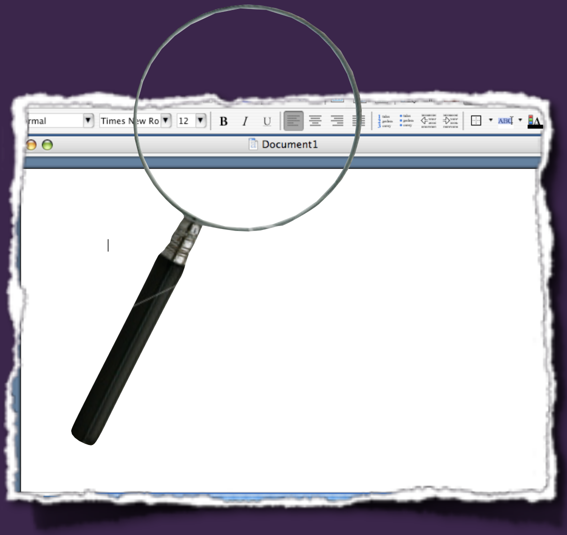
10 minutes **30× speed**

debugging with **research tools**

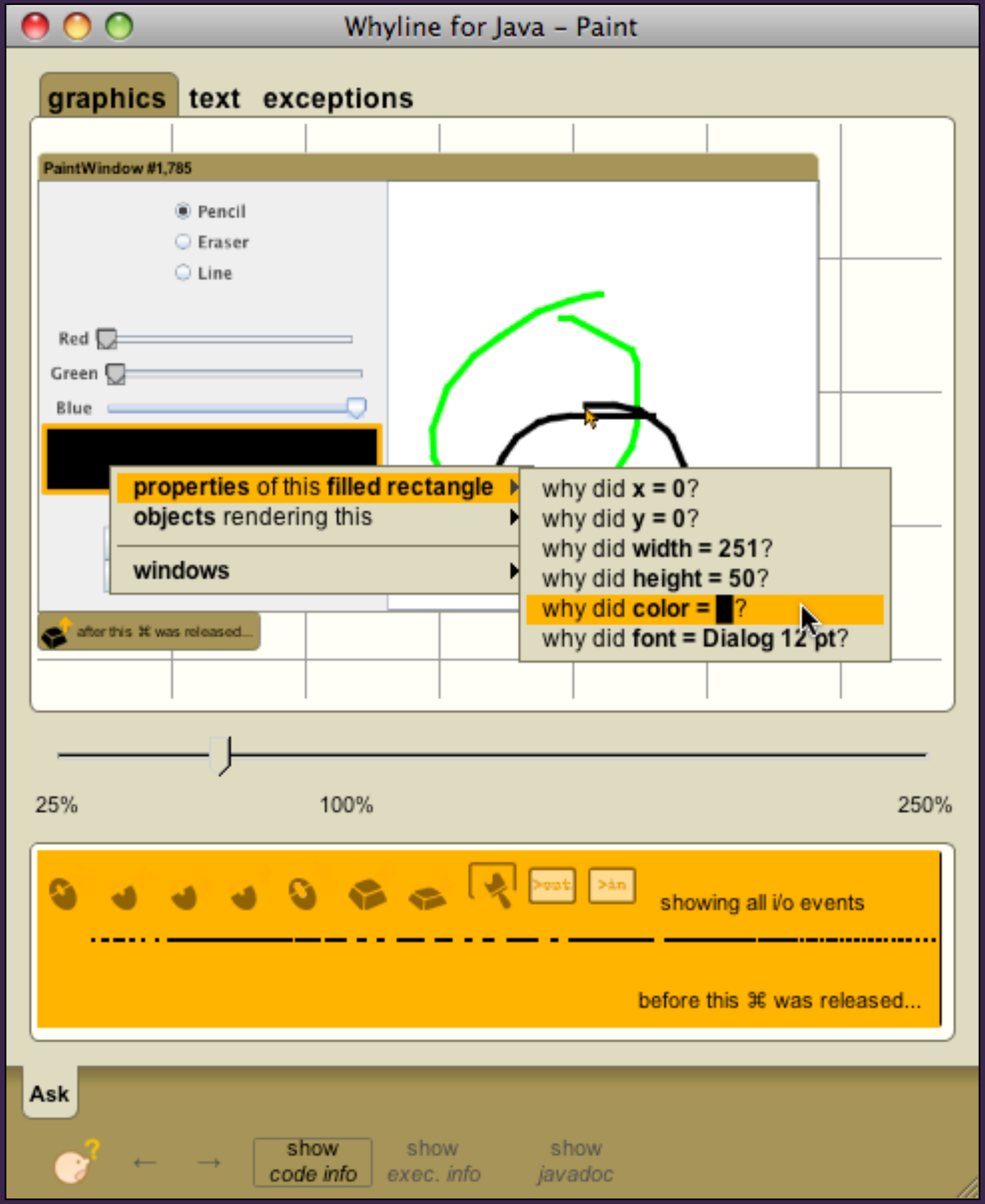
reverse execution	**guess** where to pause execution
visualizing execution	**guess** what to look for
program slicing	**guess** what code to slice on
asserting behavior	**guess** what properties won’t hold
comparing executions	**find** successful execution

the **whyline**

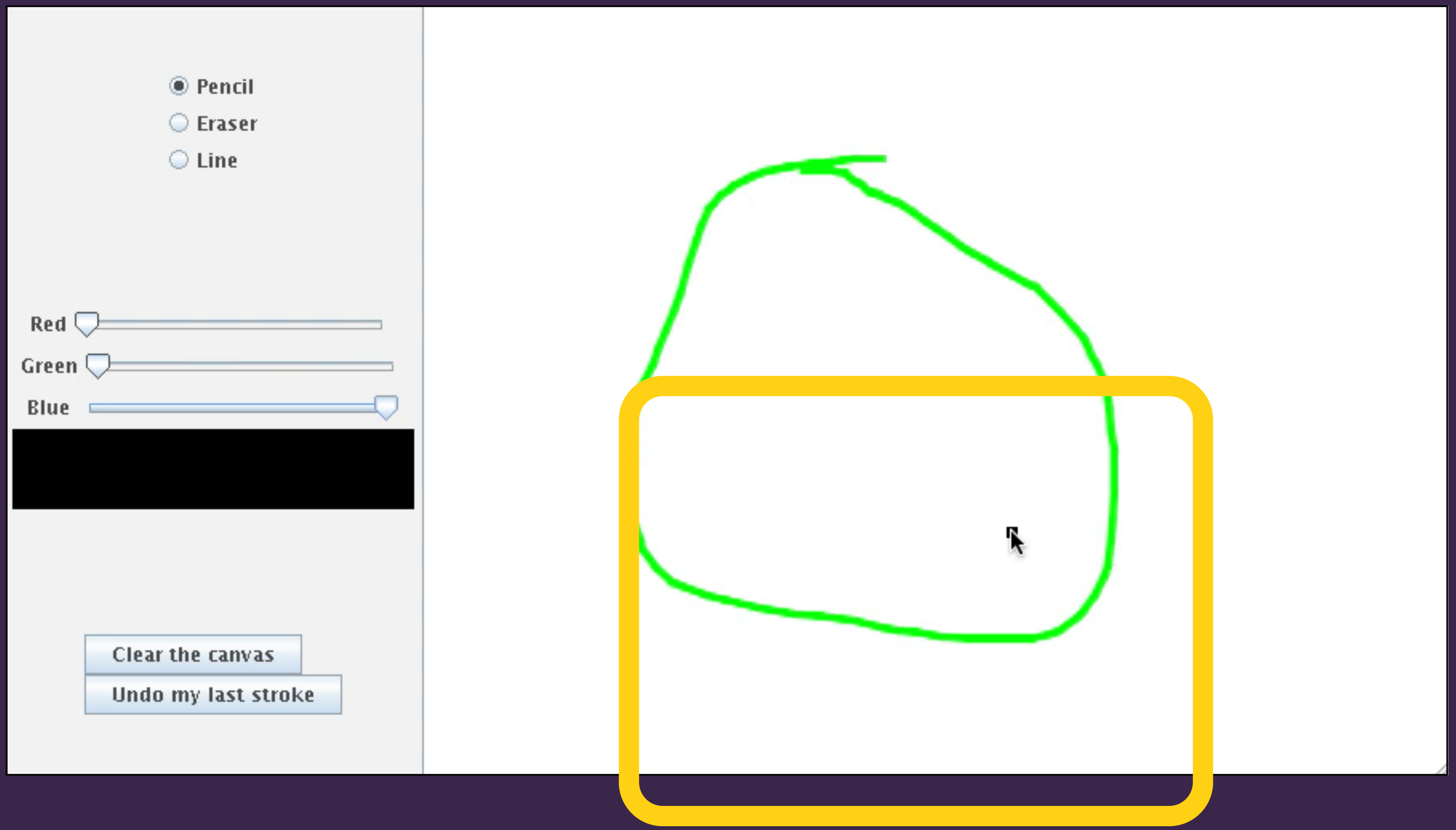
what if people could
ask about output and
see the code responsible?



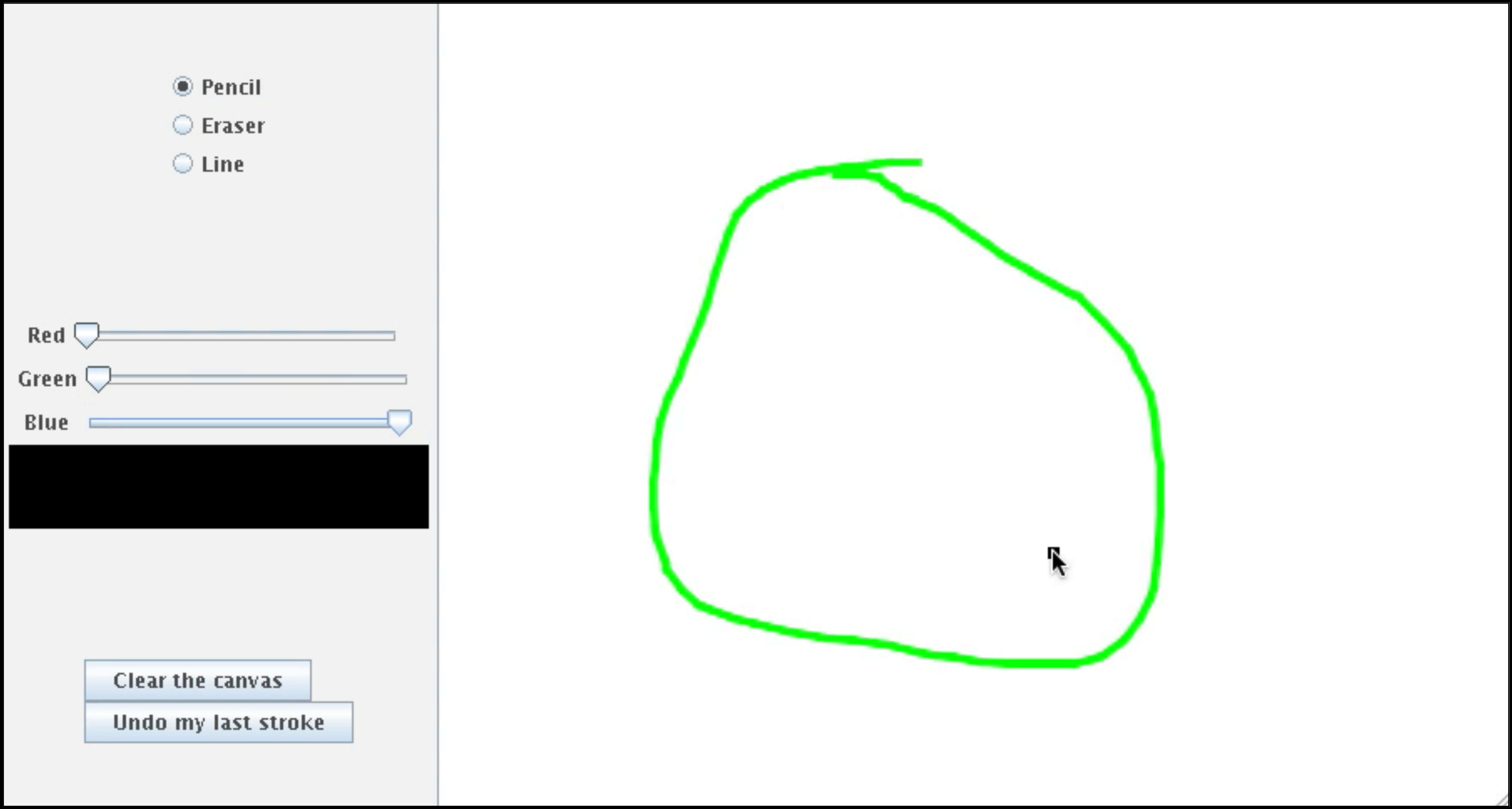
whyline for Java



why was the line **black**?

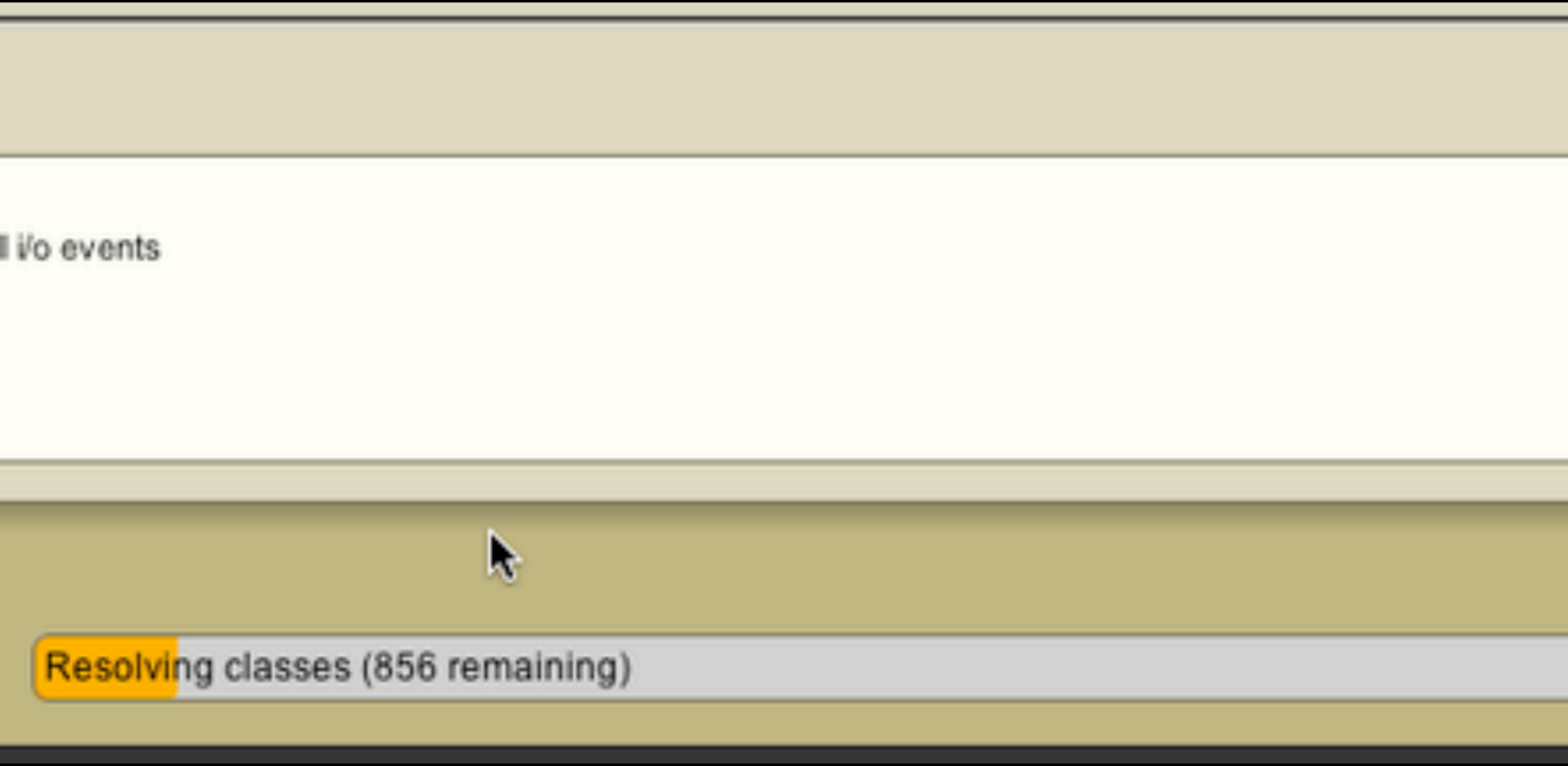


record the problem



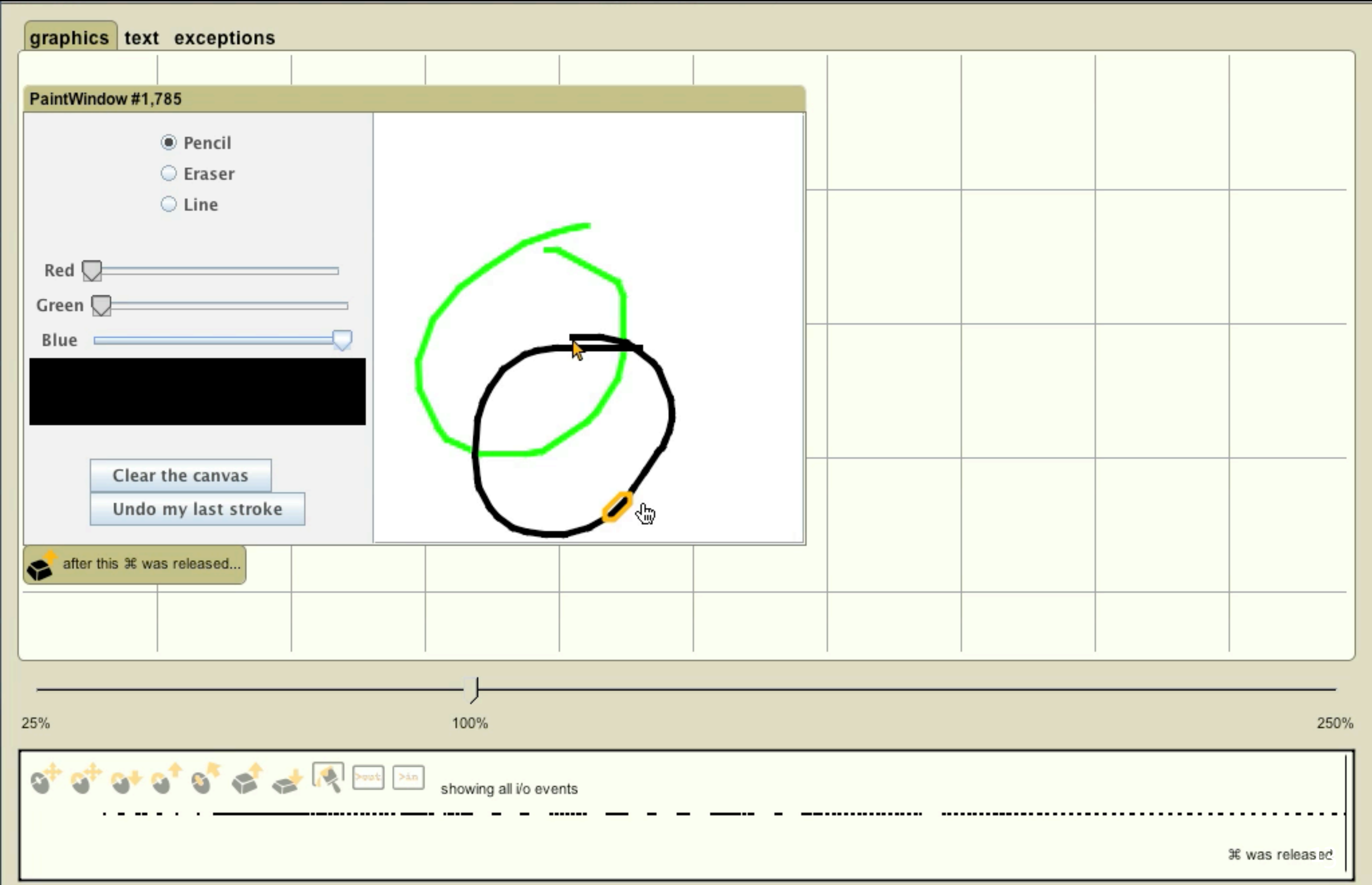
● Pencil
○ Eraser
○ Line

load the recording



Resolving classes (856 remaining)

why was the line color **black**?



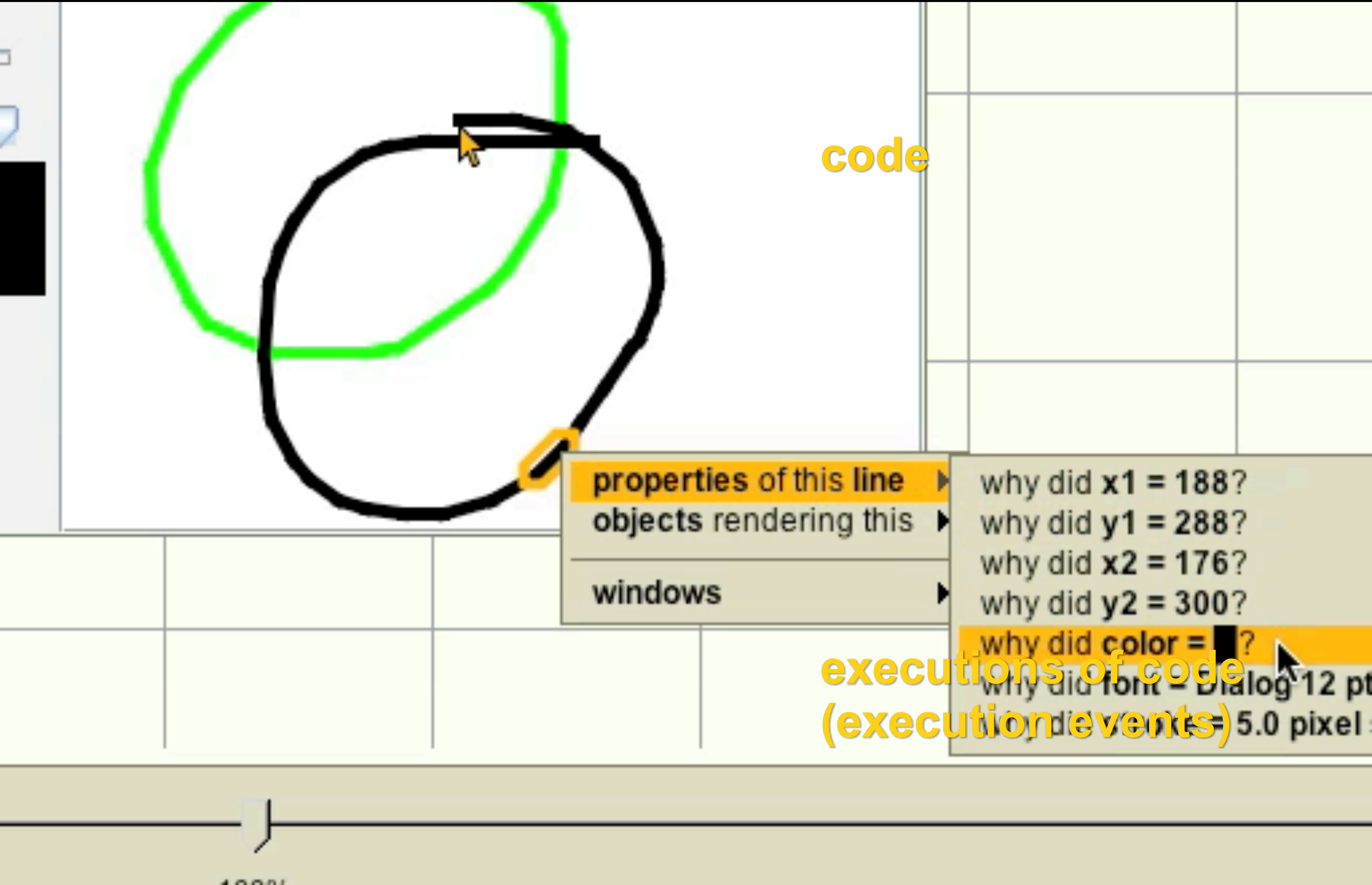
● Pencil
○ Eraser
○ Line

after this ⌘ was released...

showing all i/o events

⌘ was released

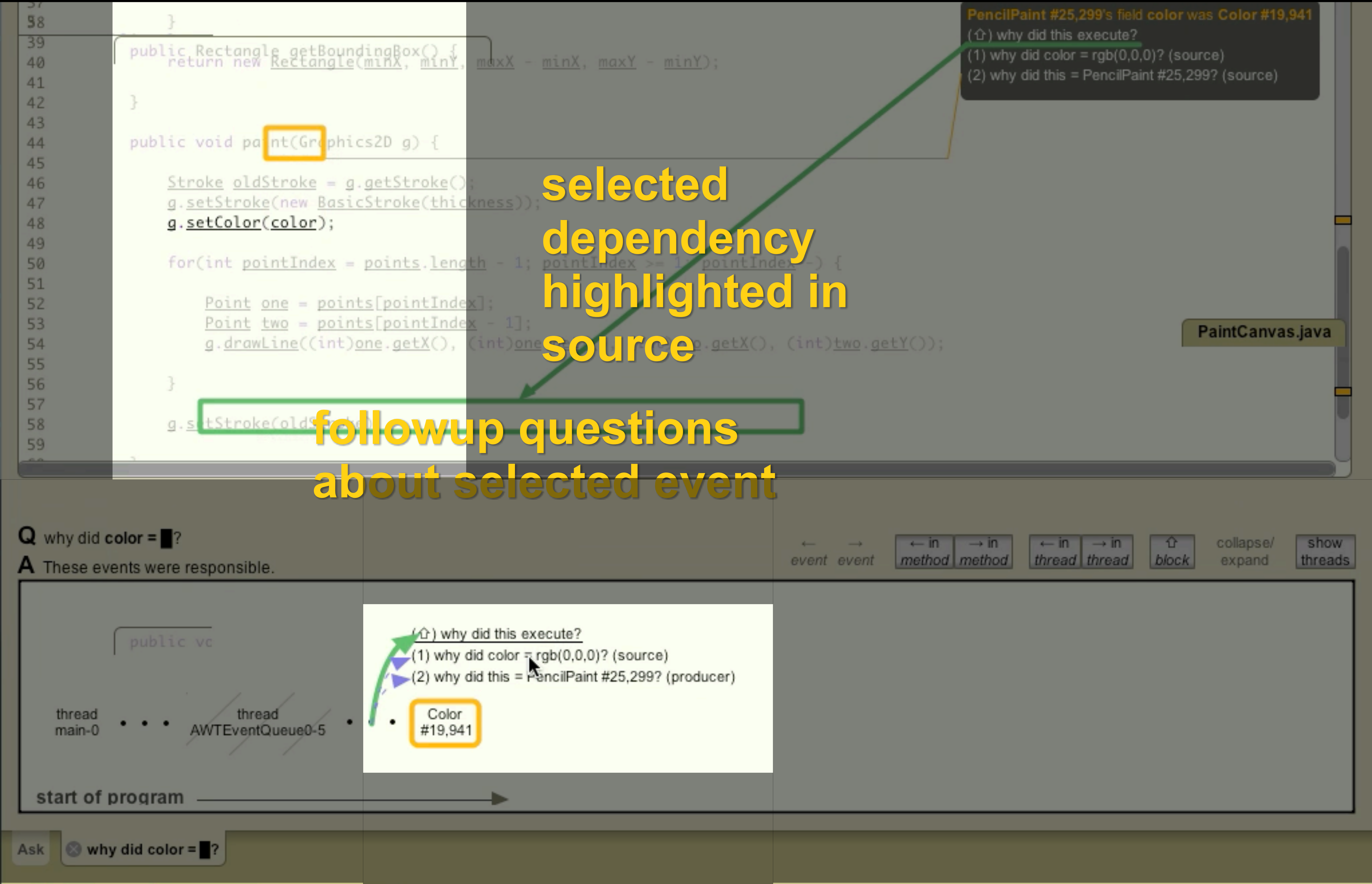
why was the line color **black**?



code

executions of code
(execution events)

why was the line color **black**?



```
    }

public Rectangle getBoundingBox() {
    return new Rectangle(minX, minY, maxX - minX, maxY - minY);

}

public void paint(Graphics2D g) {

    Stroke oldStroke = g.getStroke();
    g.setStroke(new BasicStroke(thickness));
    g.setColor(color);

    for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {

        Point one = points[pointIndex];
        Point two = points[pointIndex - 1];
        g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());

    }

    g.setStroke(oldStroke);
```

PencilPaint #25,299's field color was Color #19,941
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (source)

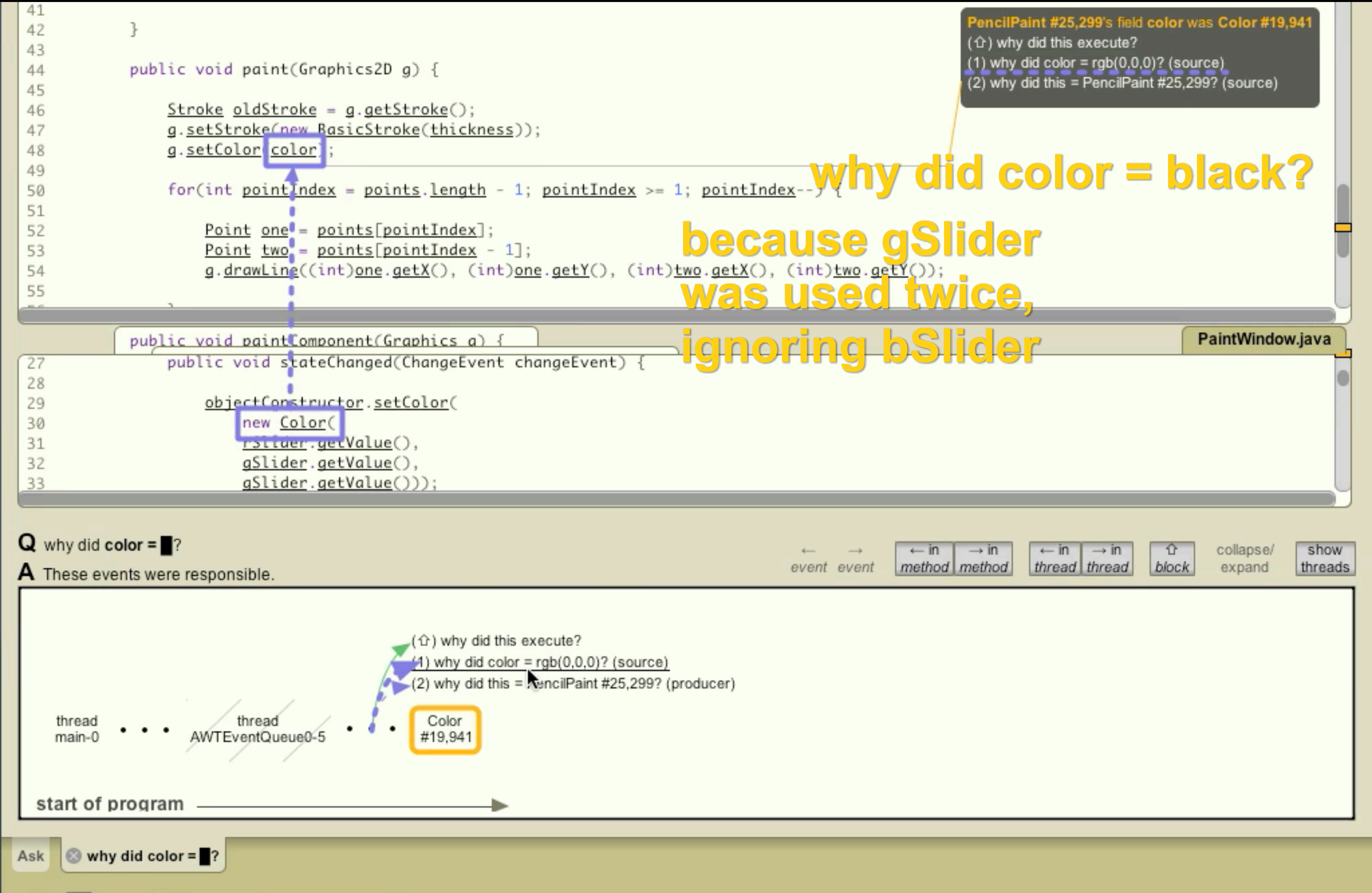
selected dependency highlighted in source

followup questions about selected event

Q why did color = █?
A These events were responsible.

(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (producer)

why was the line color **black**?



```
        }

    public void paint(Graphics2D g) {

        Stroke oldStroke = g.getStroke();
        g.setStroke(new BasicStroke(thickness));
        g.setColor(color);

        for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {

            Point one = points[pointIndex];
            Point two = points[pointIndex - 1];
            g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
```

```
    public void paintComponent(Graphics g) {
        public void stateChanged(ChangeEvent changeEvent) {

            objectConstructor.setColor(
                new Color(
                    rSlider.getValue(),
                    gSlider.getValue(),
                    gSlider.getValue()));
```

PencilPaint #25,299's field color was Color #19,941
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (source)

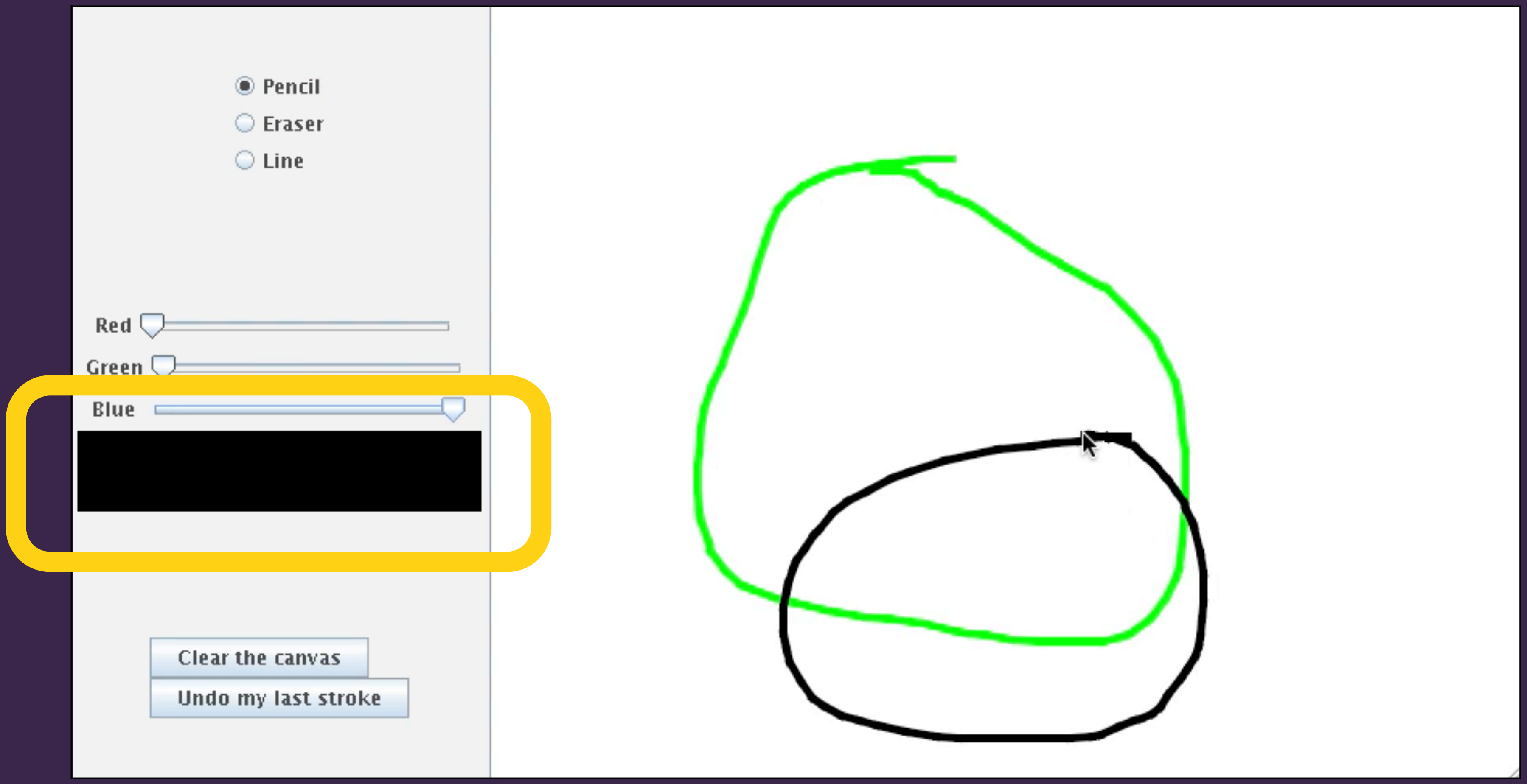
why did color = black?

because gSlider was used twice, ignoring bSlider

Q why did color = █?
A These events were responsible.

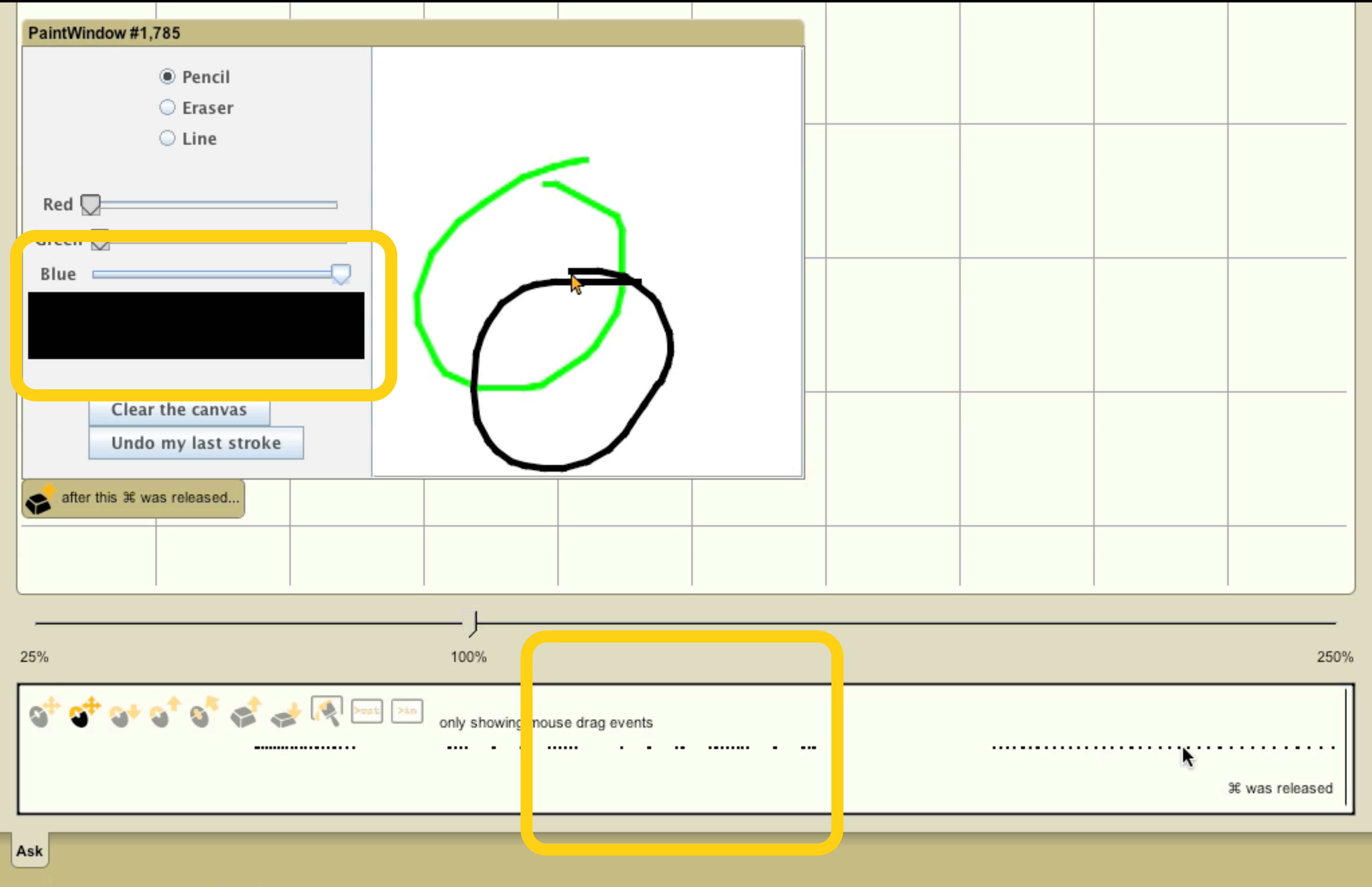
(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (source)
(2) why did this = PencilPaint #25,299? (producer)

why didn’t the panel **repaint**?



● Pencil
○ Eraser
○ Line

find the appropriate **time**



PaintWindow #1,785

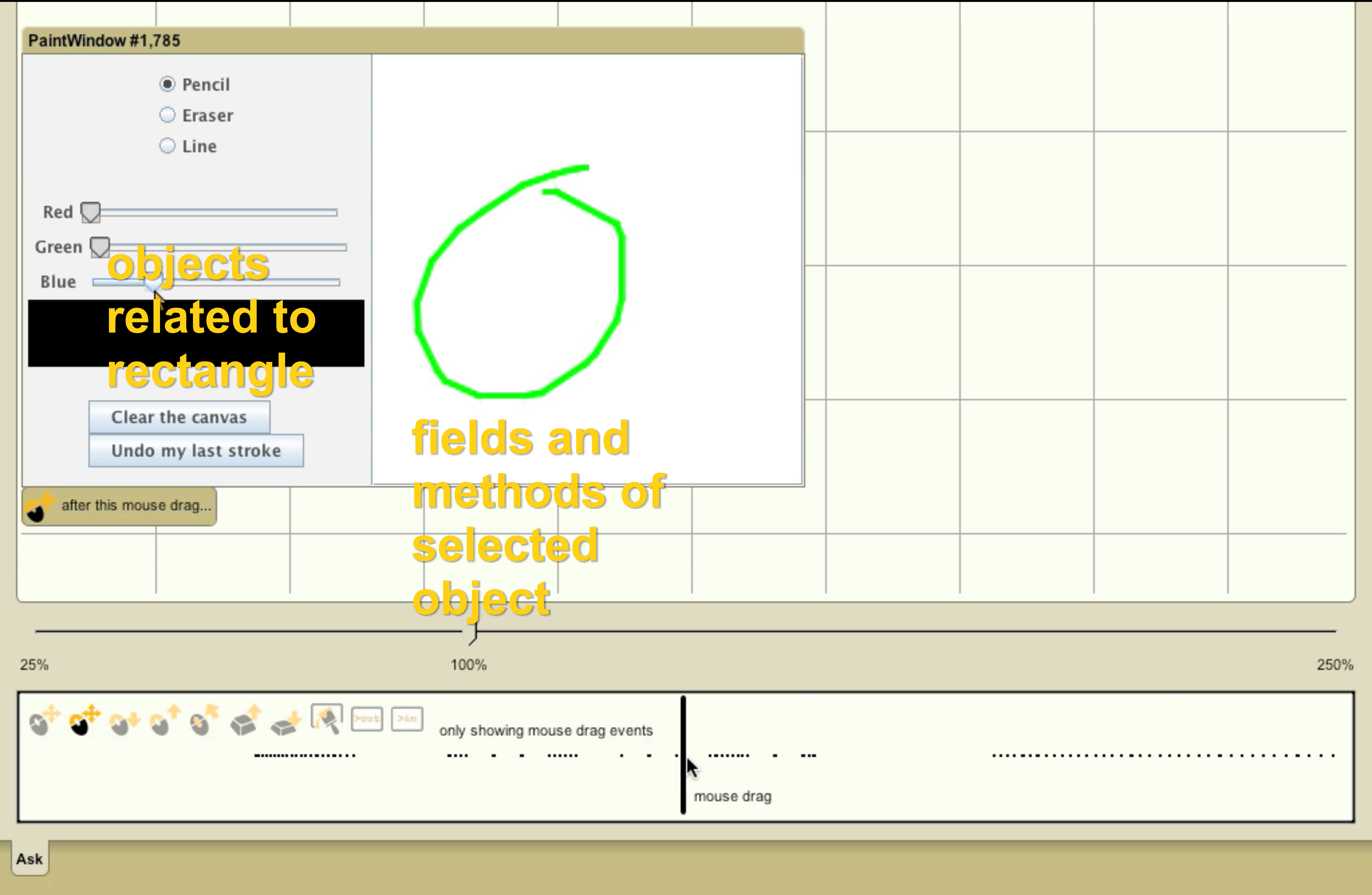
● Pencil
○ Eraser
○ Line

after this ⌘ was released...

only showing mouse drag events

⌘ was released

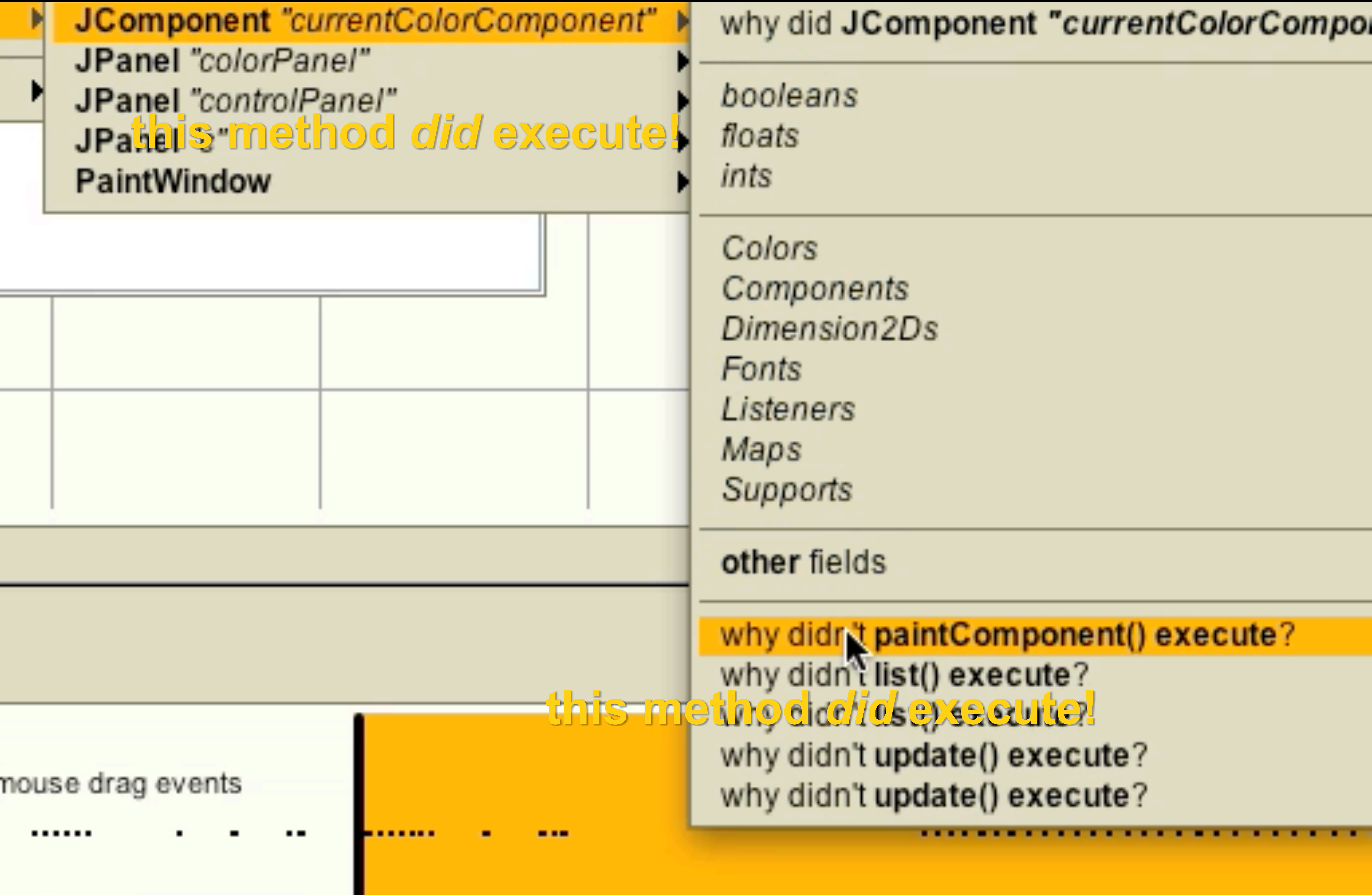
click on **relevant output**



objects related to rectangle

fields and methods of selected object

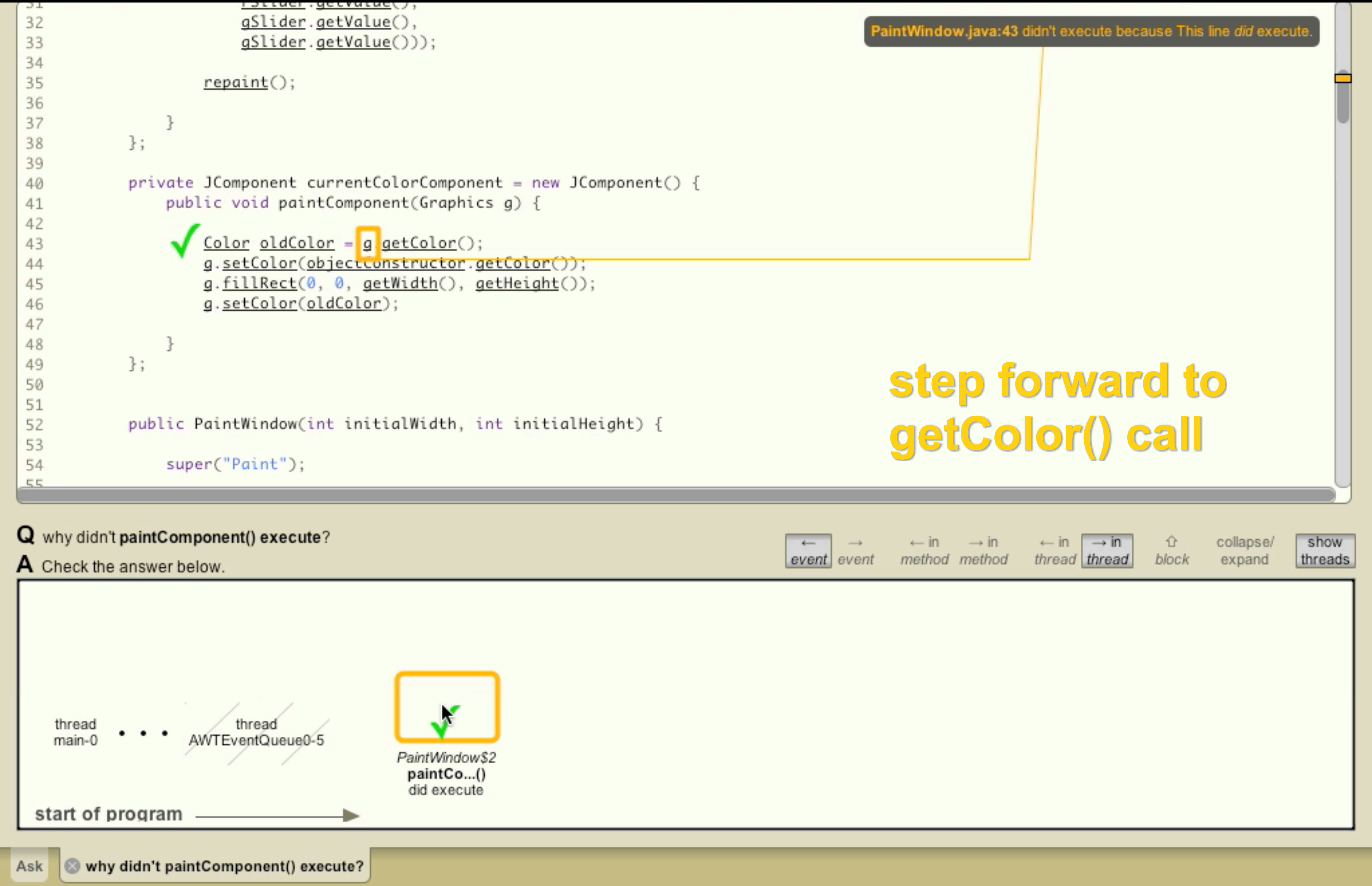
it **did** paint...


JComponent "currentColorComponent"
JPanel "colorPanel"
JPanel "controlPanel"
JPanel "
PaintWindow
why did JComponent "currentColorCompo
booleans
floats
ints
Colors
Components
Dimension2Ds
Fonts
Listeners
Maps
Supports
other fields
why didn't paintComponent() execute?
why didn't list() execute?
why didn't update() execute?
why didn't update() execute?
mouse drag events

this method *did* execute!

this method *did* execute!

where did **black** come from?



PaintWindow.java:43 didn't execute because This line *did* execute.

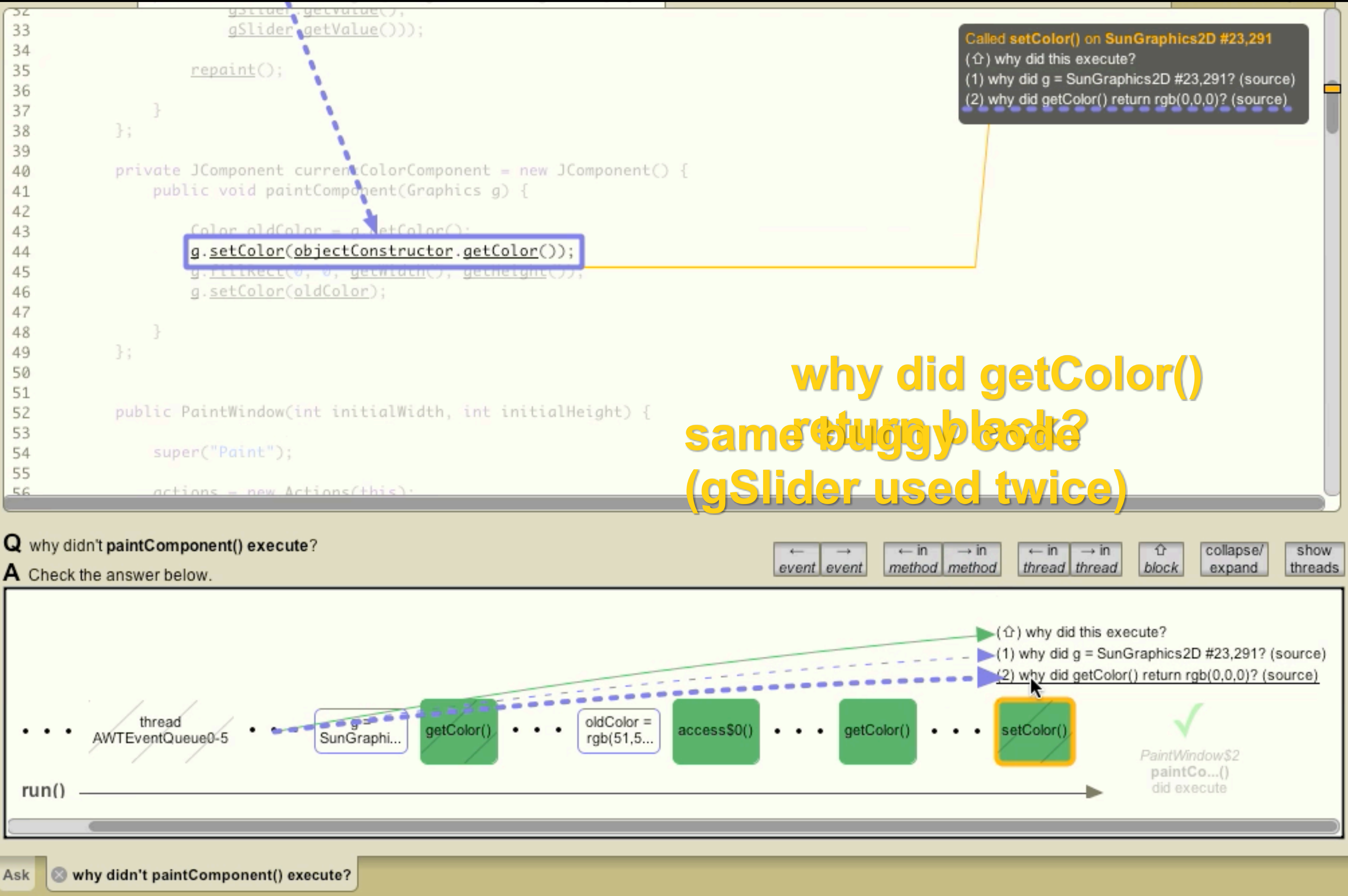
```
                gSlider.getValue(),
                gSlider.getValue()));

            repaint();

        }
    };

    private JComponent currentColorComponent = new JComponent() {
        public void paintComponent(Graphics g) {

            Color oldColor = g.getColor();
            g.setColor(objectConstructor.getColor());
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(oldColor);

        }
    };


    public PaintWindow(int initialWidth, int initialHeight) {

        super("Paint");
```

step forward to getColor() call

Q why didn't **paintComponent()** execute?

A Check the answer below.

found the bug



```
            gSlider.getValue(),
            gSlider.getValue()));

        repaint();

    }
};

private JComponent currentColorComponent = new JComponent() {
    public void paintComponent(Graphics g) {

        Color oldColor = g.getColor();
        g.setColor(objectConstructor.getColor());
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(oldColor);

    }
};


public PaintWindow(int initialWidth, int initialHeight) {

    super("Paint");

    actions = new Actions(this);
```

Called setColor() on SunGraphics2D #23,291
(⇧) why did this execute?
(1) why did g = SunGraphics2D #23,291? (source)
(2) why did getColor() return rgb(0,0,0)? (source)

why did getColor() return black?

same buggy code (gSlider used twice)

Q why didn't **paintComponent()** execute?

A Check the answer below.

how does the **Whyline** work?

the whyline cycle

developer...

edit compile debug record fix... load ask

1 2 3

system...

instruments bytecode
records thread history

converts serial history to
random access

extracts questions from code

find primitive output statements

```
drawString(x, y, string)
```

```
drawLine(x, y, width, height)
```

```
setColor(color)
```

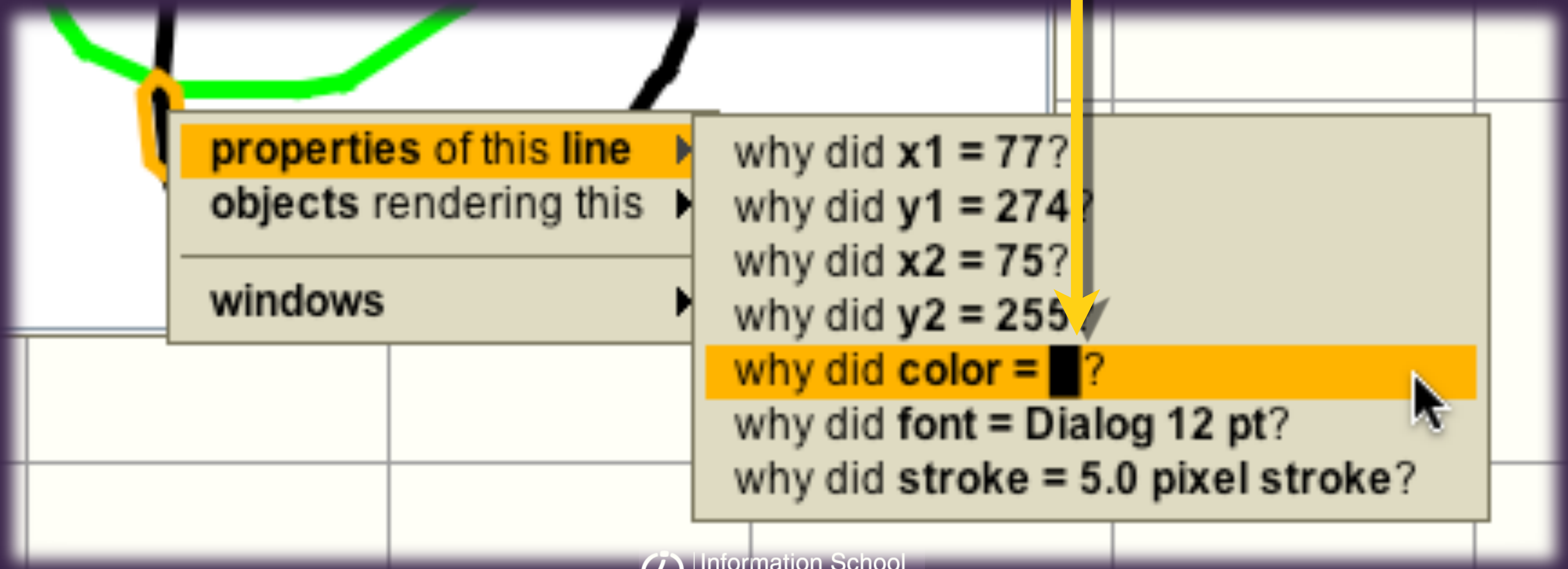
extract primitive questions

drawString(x, y, string)

drawLine(x, y, width, height)

setColor(color)

why did ***argument* = *value***?



why did **x1 = 77**?

why did **y1 = 274**?

why did **x2 = 75**?

why did **y2 = 255**?

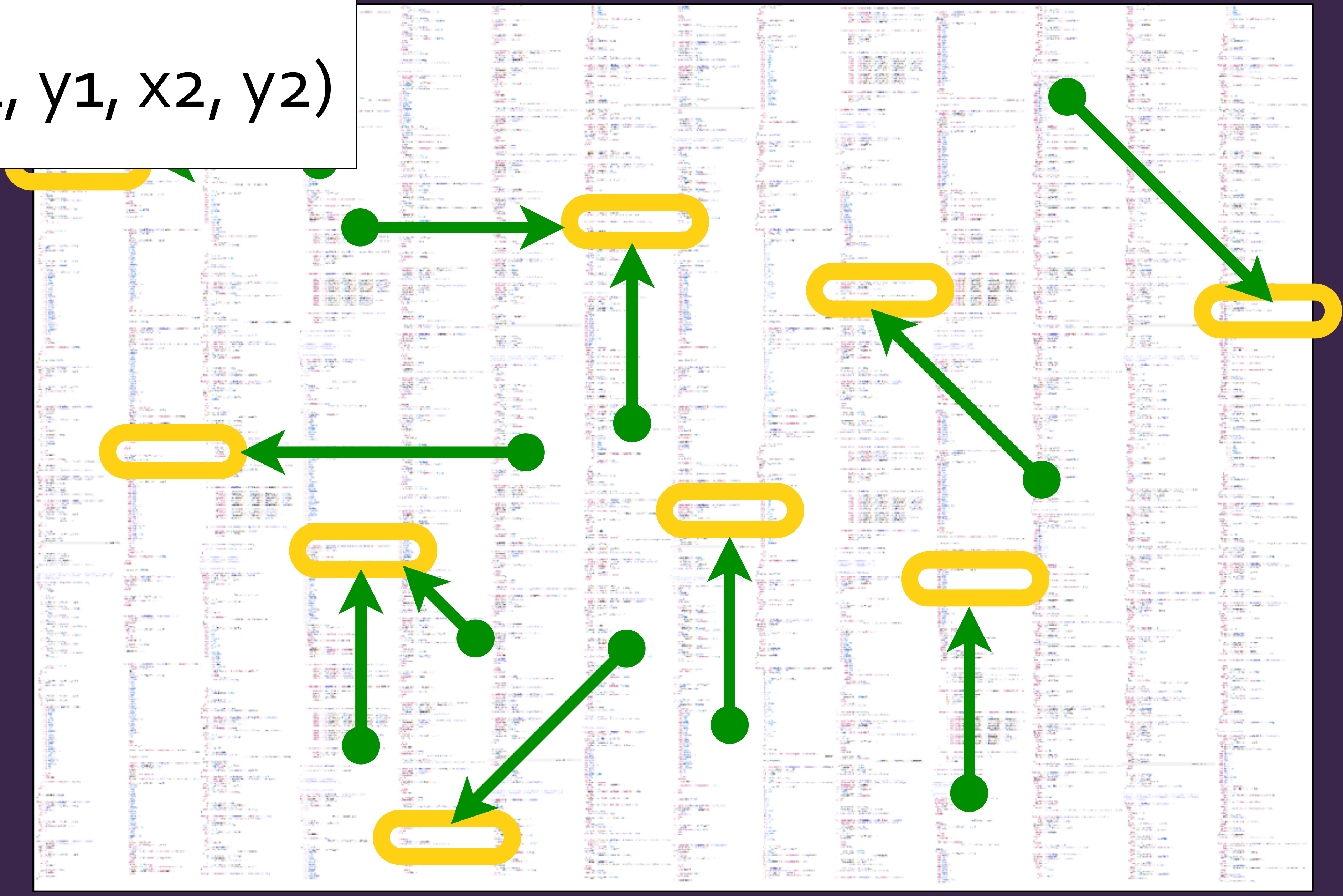
why did **color = █**?

why did **font = Dialog 12 pt**?

why did **stroke = 5.0 pixel stroke**?

find output-invoking classes

```
class PencilPaint
    draw() {
      ...
      drawLine(x1, y1, x2, y2)
```

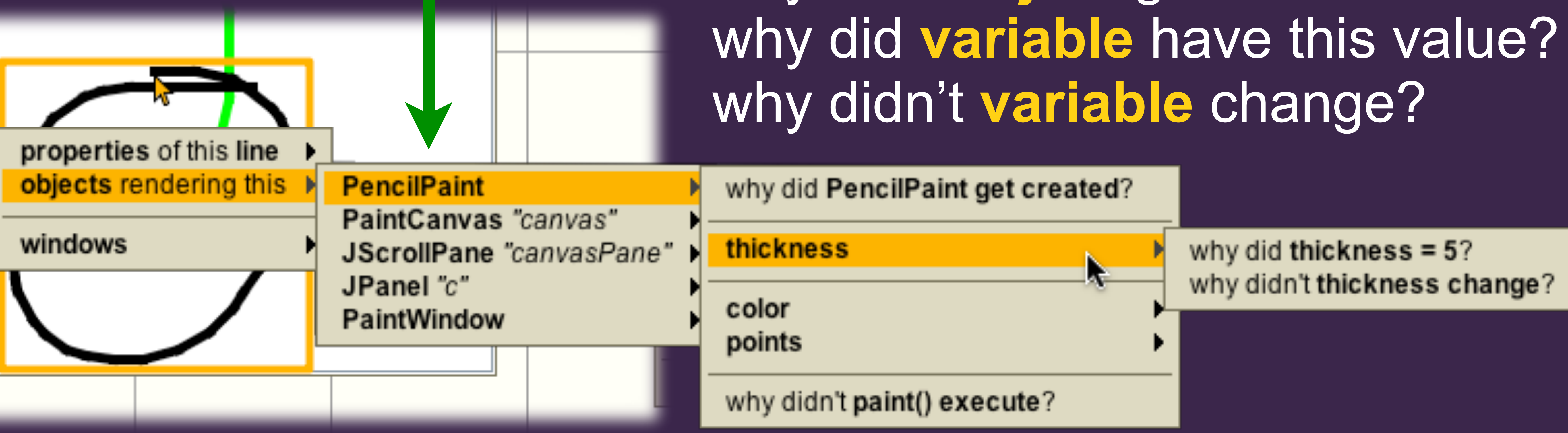


upstream
control
dependencies

extract output-invoking questions

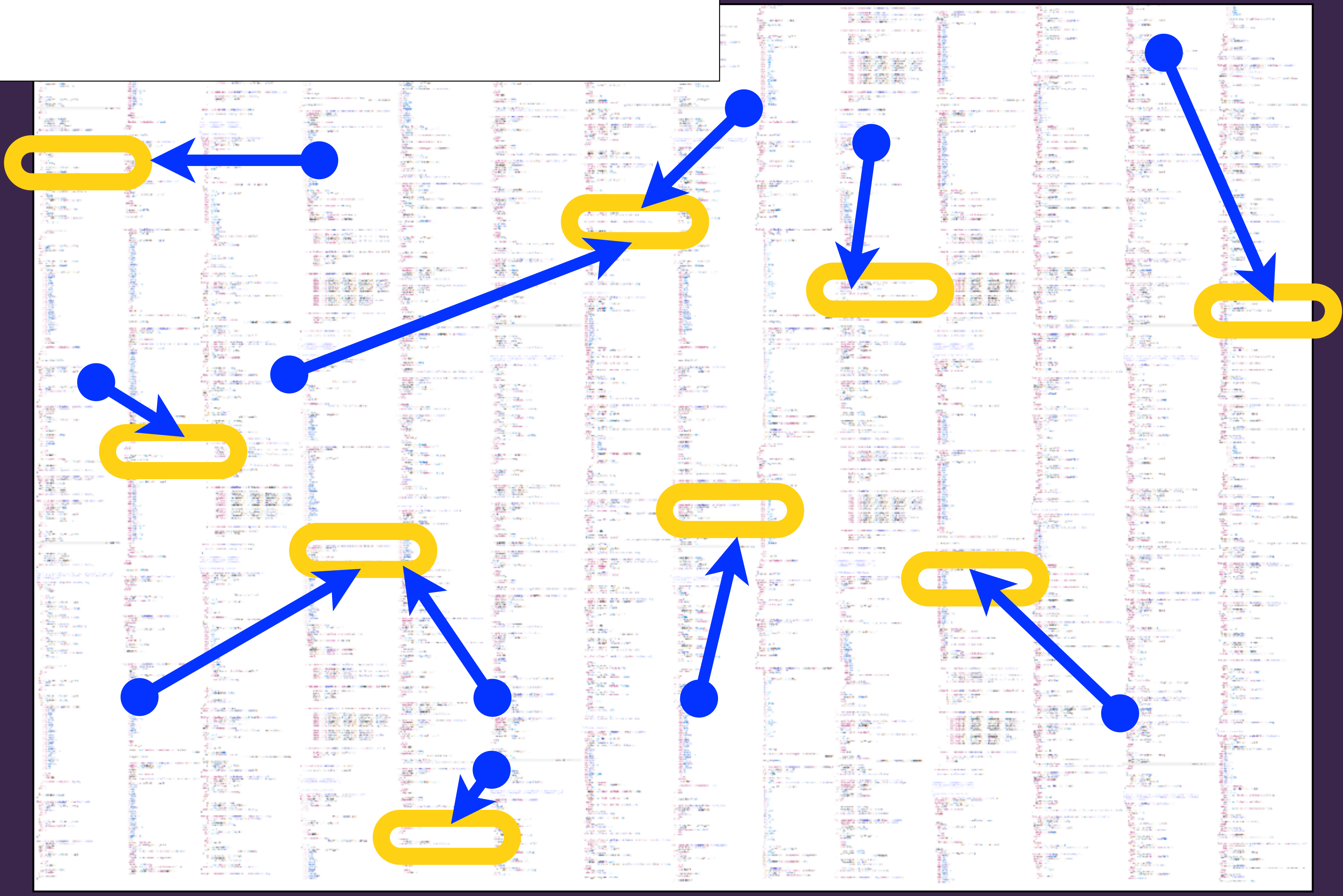
```
class PencilPaint
    draw() {
      ...
      drawLine(x1, y1, x2, y2)
```

why did **subject** get created?
why did **variable** have this value?
why didn't **variable** change?



find output-affecting fields

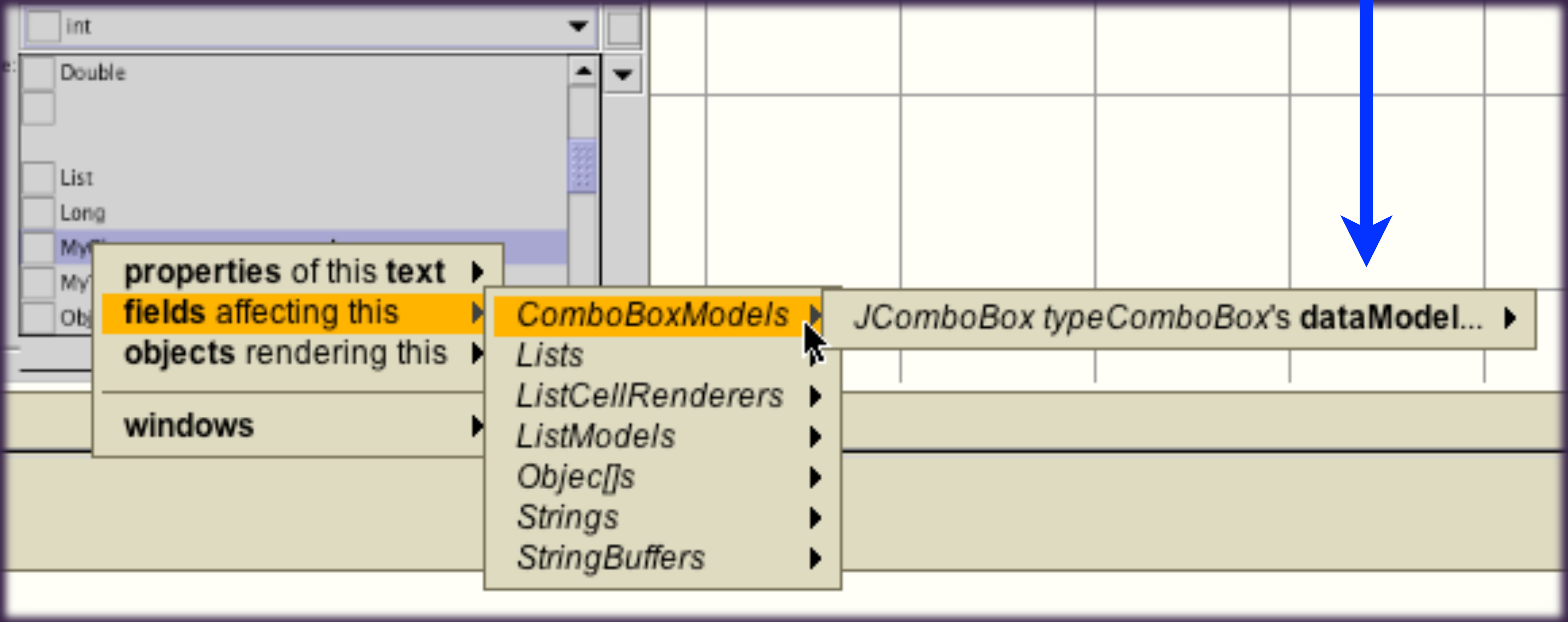
```
ComboBox combo = new
ComboBox(model)
...
```



upstream data dependencies

extract output-affecting field questions

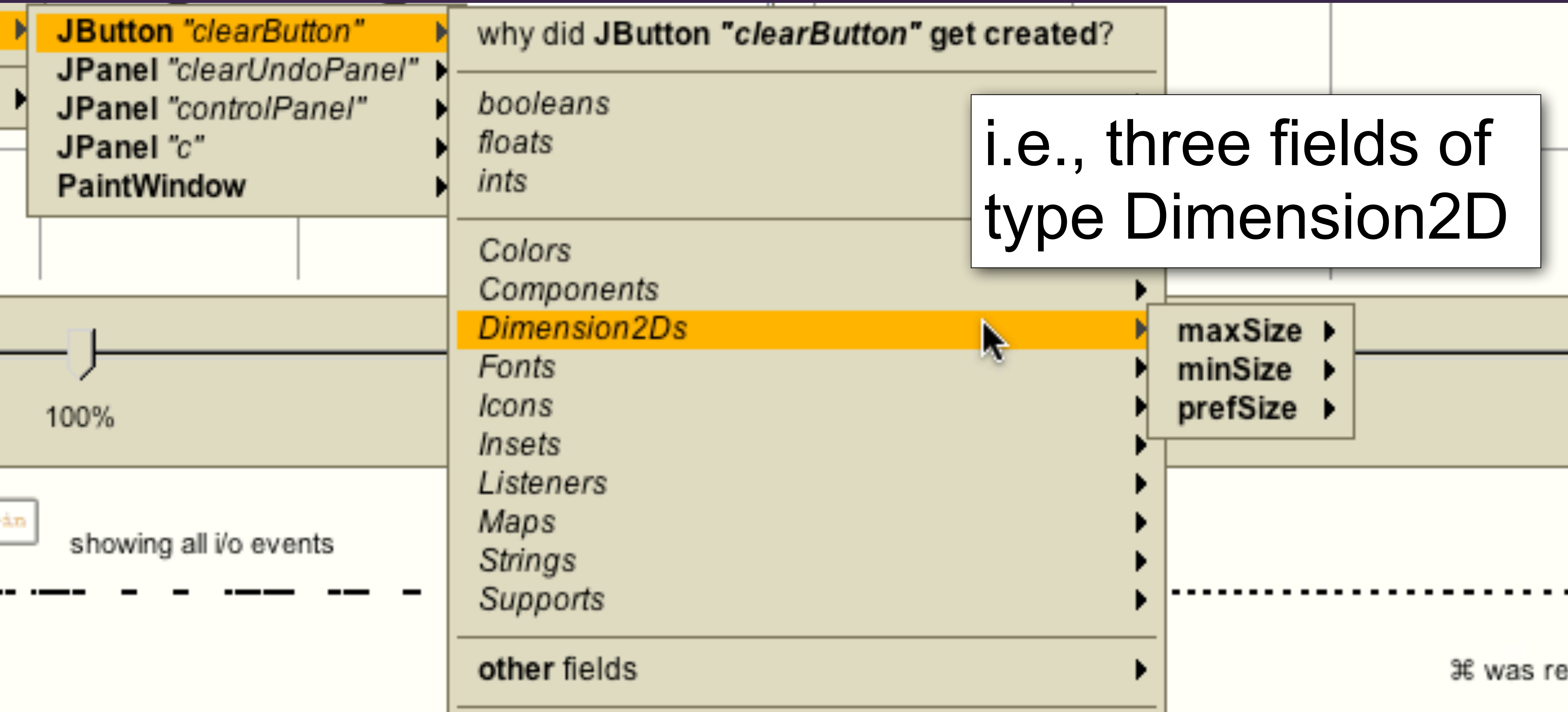
```
ComboBox combo = new
ComboBox(model)
...
paint() {
```



sorting field questions by type

“clearButton” has many fields

questions organized by primitives and superclass



i.e., three fields of type Dimension2D

filtering questions by **familiarity**

intermediaries, delegates, proxies, helpers, etc.

- may be unfamiliar

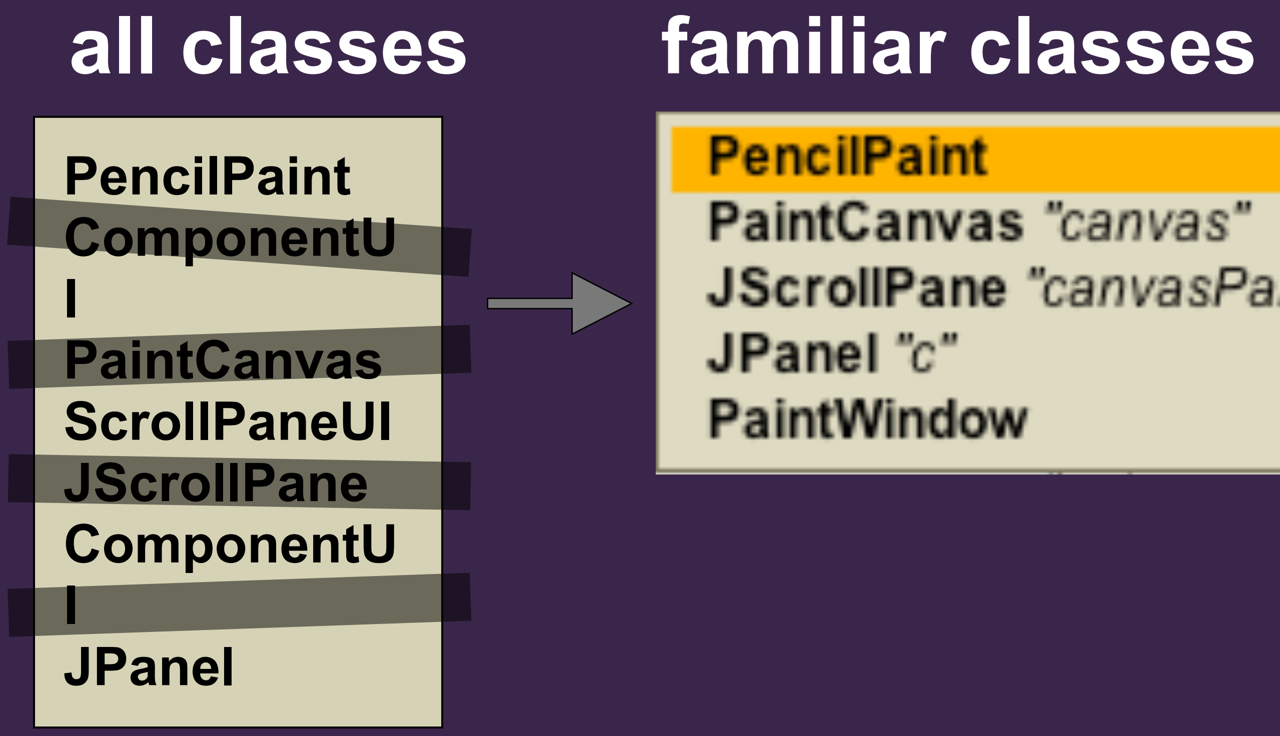
```
class Button
    paint() {
        lookandfeel.paint()
```

- **familiarity** = classes...

  declared in **editable** code

  referenced in **editable** code

- only include questions about **familiar classes**



PencilPaint
ComponentUI
PaintCanvas
ScrollPaneUI
JScrollPane
ComponentUI
JPanel

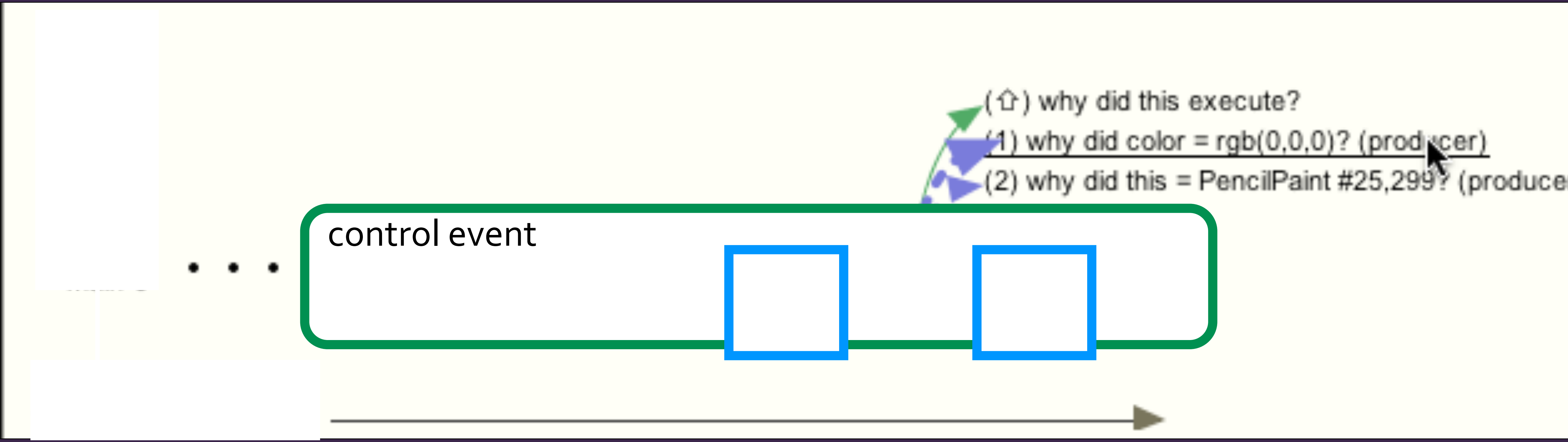
PencilPaint
PaintCanvas "canvas"
JScrollPane "canvasPa
JPanel "c"
PaintWindow

‘why did’ answers

answer derived with **precise dynamic slicing**

a timeline visualization of dependencies

- **control** dependencies as **nested blocks**
- **data** dependencies **inside** of blocks

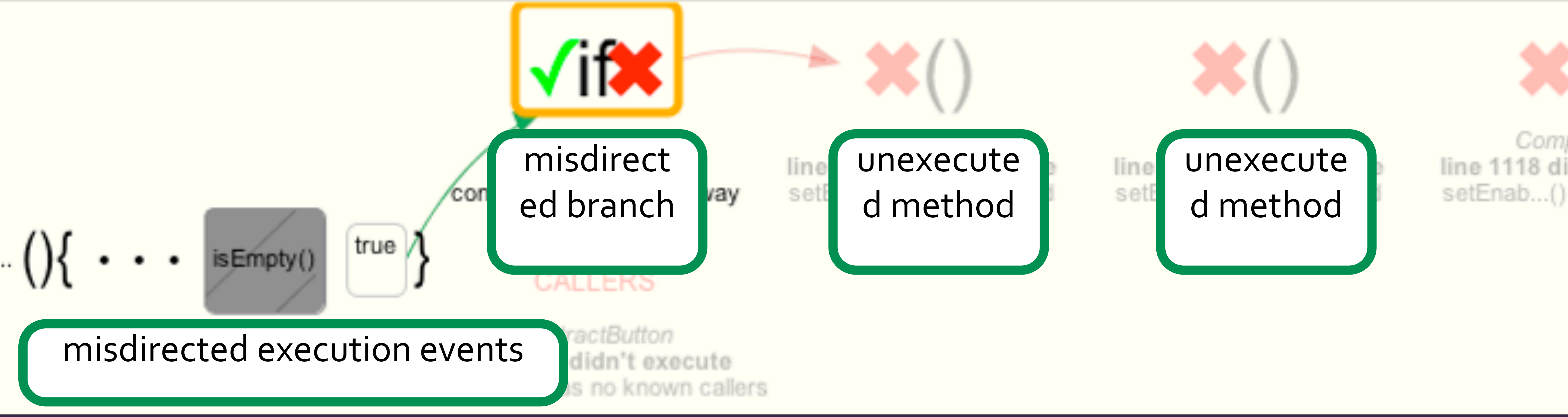

(⇧) why did this execute?
(1) why did color = rgb(0,0,0)? (producer)
(2) why did this = PencilPaint #25,299? (produce
. . .
control event

‘why didn’t’ answers

answer with **call graph reachability analysis**

a visualization of a **subgraph of the call graph**, with

- **unexecuted methods** and **branches**
- **misdirected calls** and **branches**



misdirected branch

unexecuted method

unexecuted method

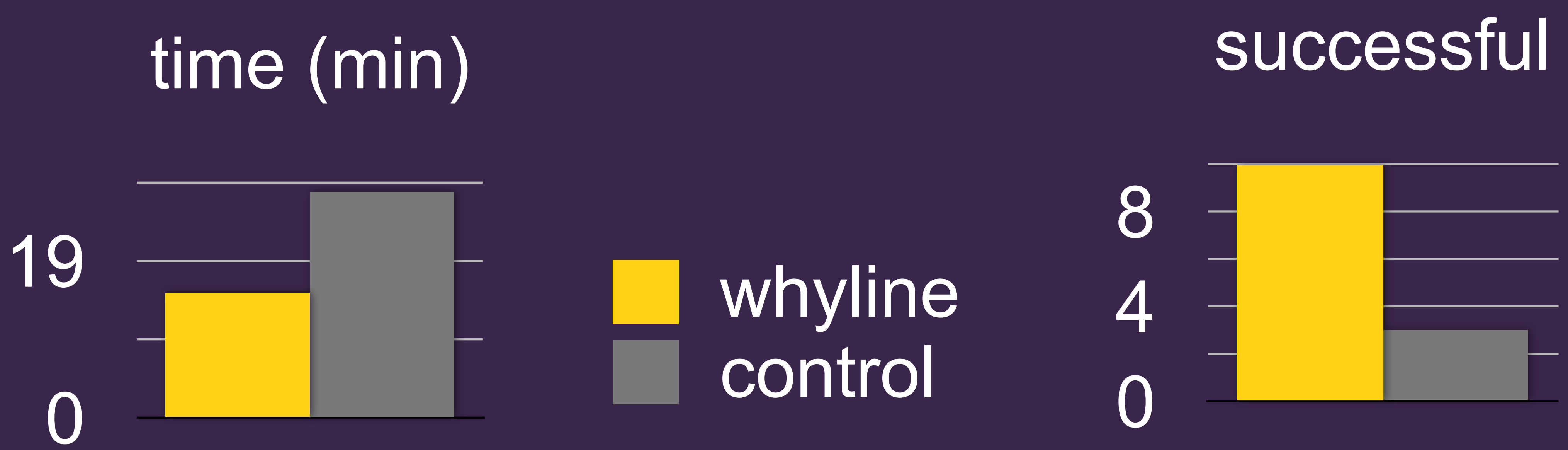
misdirected execution events

how effective is the **Whyline**?

effectiveness

in a study of two ArgoUML bugs, developers with the Whyline were ...

2x as fast **successful 3x as often**



performance

memory and **performance** (see paper)

- **slow to load** traces
- **fast to answer** questions

infeasible for **long executions**

instrumenting real time software changes behavior

limitations

quality of **question phrasing** ∝ **quality of identifiers**

question and answer **precision** ∝ **type** information

limitations

good for **causal** explanations

not change suggestions

good for ‘**where** is the buggy code’

not ‘why is the code **buggy**’

summary

today's tools require **guessing**, costing time, money and accuracy of knowledge

the whyline limits guesswork by supporting **queries on program output**

the whyline saves time, **improves success rates**

questions

download the Java whyline at

http://faculty.washington.edu/ajko

or **Google "whyline"**

This work was supported by the National Science Foundation under NSF grant IIS-0329090 and the EUSES consortium under NSF grant ITR CCR-0324770. The author is also supported by an NDSEG fellowship and by a NSF Graduate Research Fellowship.

slowdown

program	LOC	events	YourKit profiler slowdown	Whyline slowdown
Binclock	177	140K	2	2
jTidy	12K	16 million	4	15
javac	54K	35 million	2	7
jEdit	66K	9 million	2	8
ArgoUML	113 K	18 million	3	5

user interfaces are largely idle

trace size

program	**LOC**	**events**	**size (mb)**	**zipped (mb)**
Binclock	177	140K	5 mb	2 mb
jTidy	12K	16 million	118 mb	14 mb
javac	54K	35 million	284 mb	51 mb
jEdit	66K	9 million	84 mb	12 mb
ArgoUML	113K	18 million	137 mb	18 mb

# of events ∝ complexity of computation