

Asking and Answering **Why and Why Not Questions** about Program Behavior

Amy J. Ko



Information School
UNIVERSITY of WASHINGTON

and

Brad Myers

Carnegie Mellon

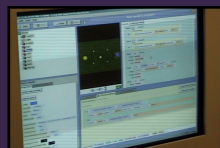
... **identifying and correcting defects** during the software development process represents over **half** of development **costs** ... and accounts for **30 to 90 percent** of labor expended to produce a working program.”

National Institute of Standards and Technology, 2002

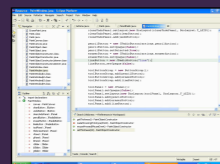


why is debugging so **difficult**?

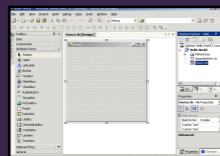
four studies to find out...



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**

The screenshot shows a painting application interface. On the left, there is a toolbar with three radio buttons: 'Pencil' (selected), 'Eraser', and 'Line'. Below these are three color sliders labeled 'Red', 'Green', and 'Blue'. The 'Red' slider is at the far left, 'Green' is in the middle, and 'Blue' is at the far right. A bright red rectangular color panel is visible below the sliders. On the right, a large white canvas contains a single black horizontal stroke. At the bottom left of the interface are two buttons: 'Clear the canvas' and 'Undo my last stroke'. The text 'a painting program' is overlaid in large yellow font across the center of the canvas.

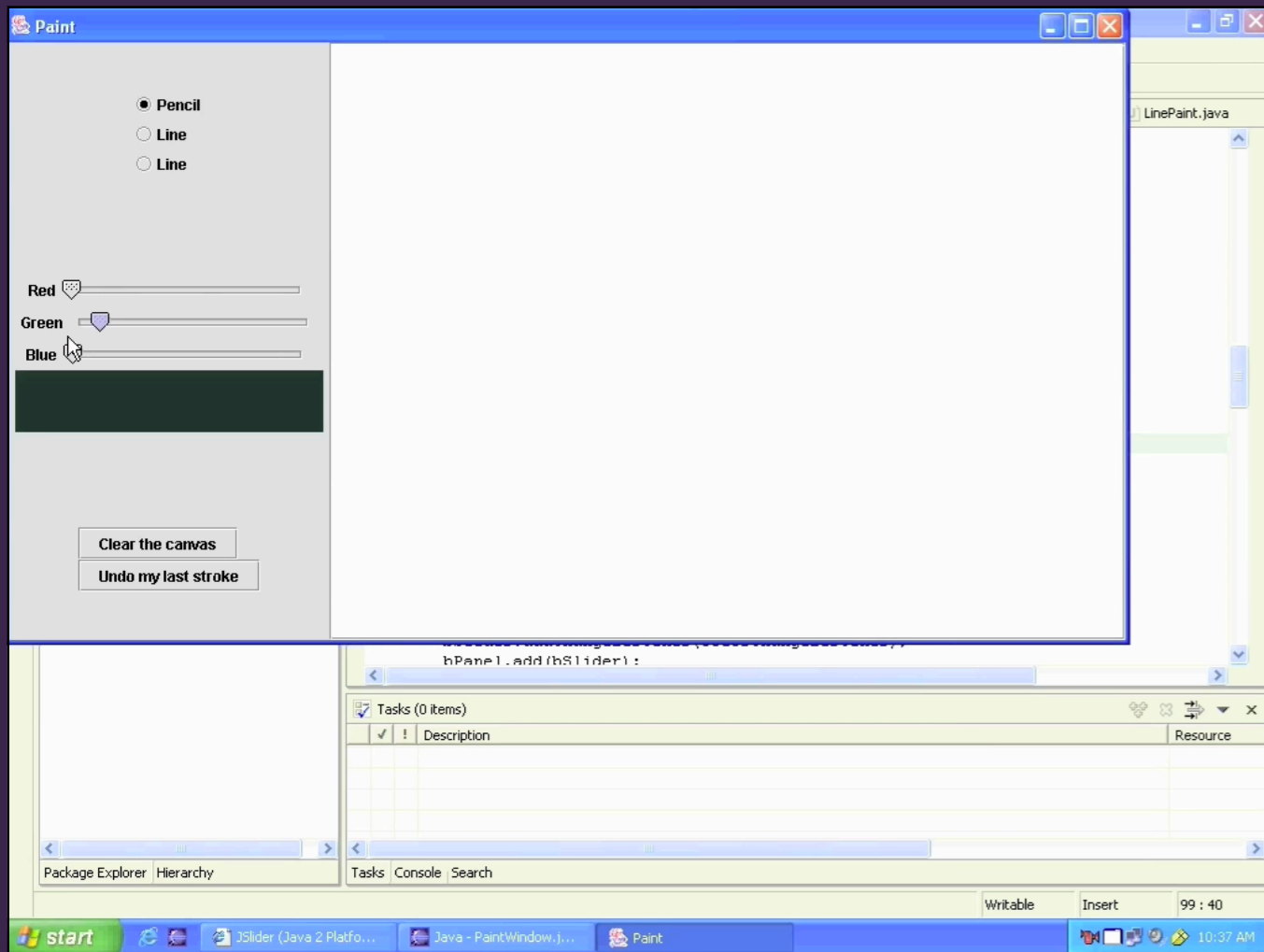
why didn't this color panel change?

why is this stroke black?

debugging with **current tools**



why is the stroke black?

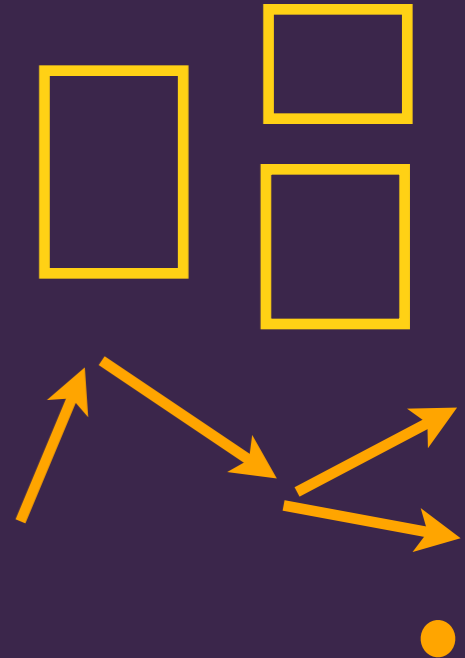


maybe ~~the~~ a slider initialization problem...

maybe the ~~slider~~ slider isn't connected to anything...

is the JSli~~der~~ argument incorrect?

maybe the ~~color~~ color isn't computed  properly...



breakpoint

println()

10 minutes **30x speed**

debugging with **research tools**

reverse execution **guess** where to pause execution

visualizing execution **guess** what to look for

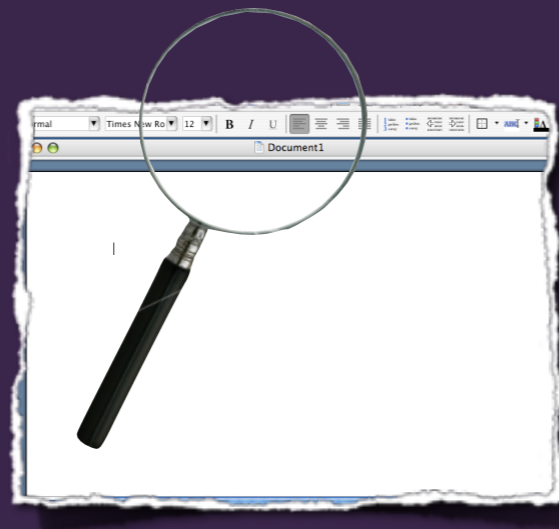
program slicing **guess** what code to slice on

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



Whyline for Java - Paint

graphics text exceptions

PaintWindow #1,785

- Pencil
- Eraser
- Line

Red

Green

Blue

properties of this filled rectangle

- objects rendering this
- windows

- why did x = 0?
- why did y = 0?
- why did width = 251?
- why did height = 50?
- why did color = [black]?
- why did font = Dialog 12 pt?

after this [symbol] was released...

25% 100% 250%

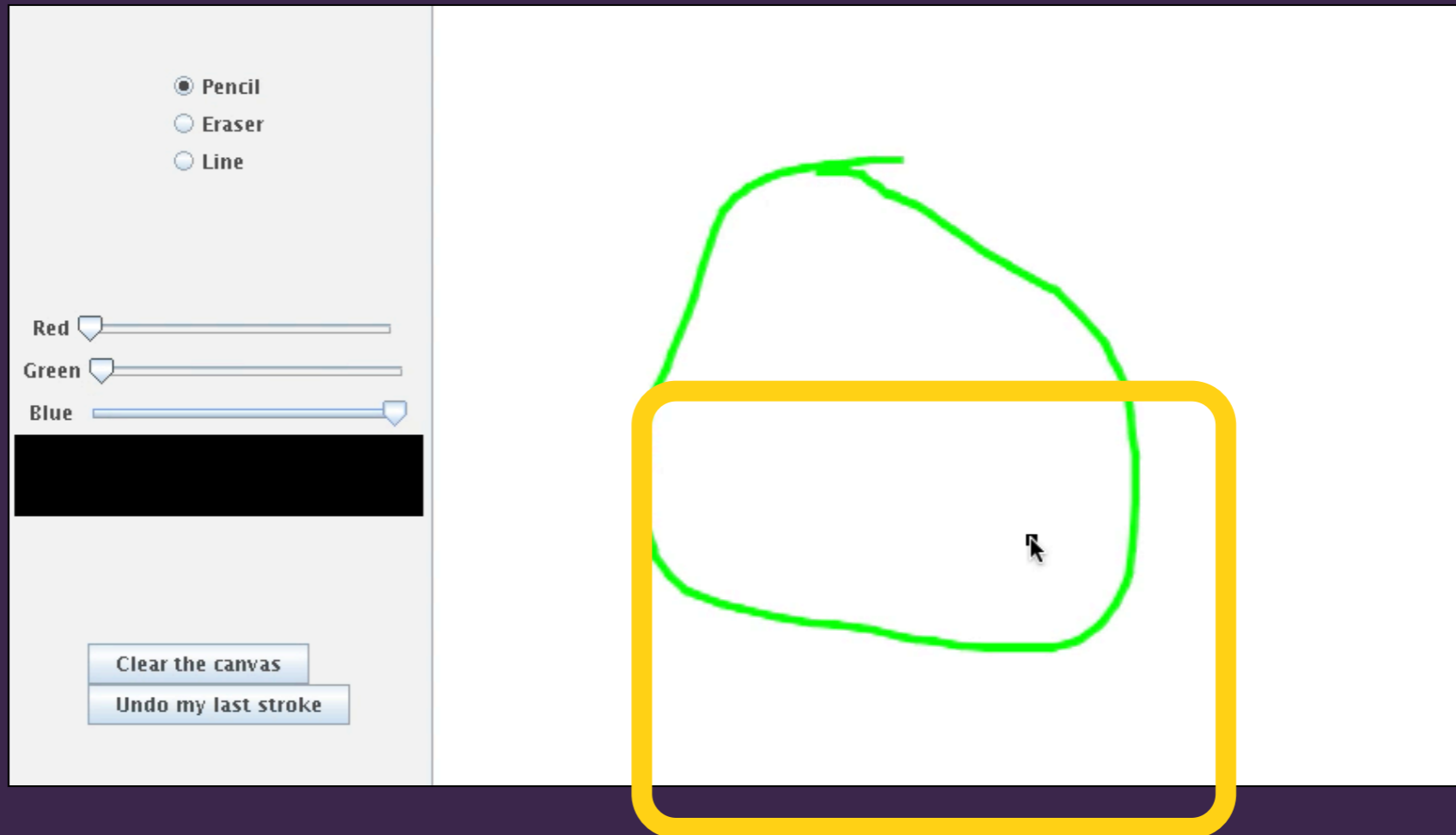
showing all i/o events

before this [symbol] was released...

Ask

show code info show exec. info show javadoc

why was the line **black**?



record the problem

The image shows a digital drawing application interface. On the left side, there is a control panel with the following elements:

- Tool selection: Three radio buttons are visible, with "Pencil" selected. The other options are "Eraser" and "Line".
- Color selection: Three horizontal sliders are present for "Red", "Green", and "Blue". The "Blue" slider is currently set to its maximum value, while "Red" and "Green" are at zero.
- Buttons: Two buttons are located at the bottom of the control panel: "Clear the canvas" and "Undo my last stroke".

The main canvas area on the right is white and contains a single, thick, bright green hand-drawn shape. The shape is irregular and roughly triangular, with a mouse cursor arrow pointing to its right side.

load the recording

no events

Resolving classes (856 remaining)

why was the line color **black**?

graphics text exceptions

PaintWindow #1,785

- Pencil
- Eraser
- Line

Red

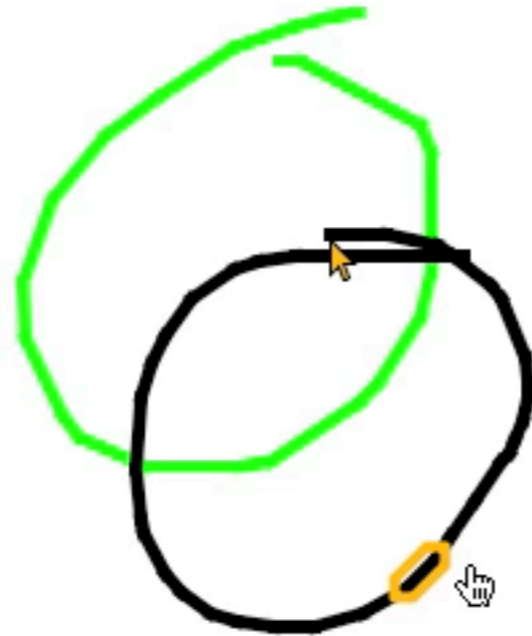
Green

Blue



Clear the canvas

Undo my last stroke



after this ⌘ was released...

25%

100%

250%



showing all i/o events

⌘ was released

why was the line color **black**?

code

properties of this line

- objects rendering this
- windows

why did color = black?

- why did x1 = 188?
- why did y1 = 288?
- why did x2 = 176?
- why did y2 = 300?
- why did font = Dialog 12 pt
- why did stroke = 5.0 pixel

executions of code (execution events)

why was the line color **black**?

```
57 }
58 }
39 public Rectangle getBoundingBox() {
40     return new Rectangle(minX, minY, maxX - minX, maxY - minY);
41 }
42 }
43
44 public void paint(Graphics2D g) {
45     Stroke oldStroke = g.getStroke();
46     g.setStroke(new BasicStroke(thickness));
47     g.setColor(color);
48
49     for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {
50         Point one = points[pointIndex];
51         Point two = points[pointIndex - 1];
52         g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
53     }
54
55     g.setStroke(oldStroke);
56 }
57
58
59
```

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

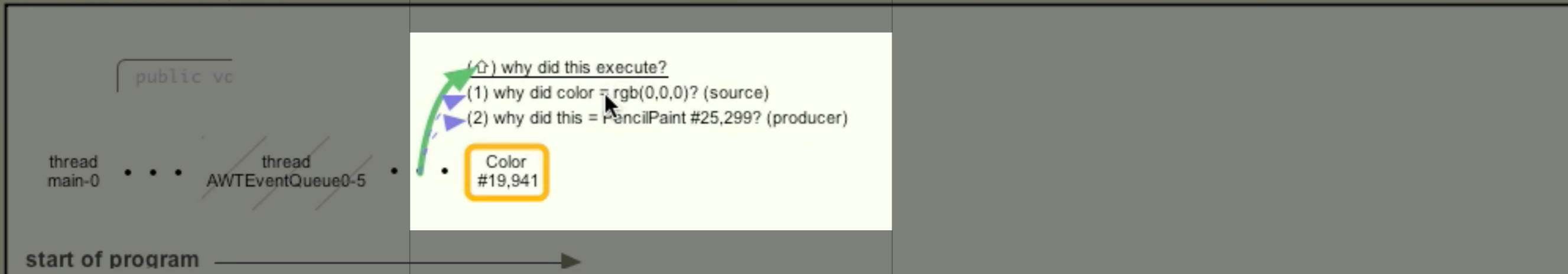
followup questions
about selected event

PaintCanvas.java

Q why did color = ■?

A These events were responsible.

← event → event ← in method → in method ← in thread → in thread ↑ block collapse/expand show threads



Ask why did color = ■?

why was the line color **black**?

```
41 }
42
43
44 public void paint(Graphics2D g) {
45
46     Stroke oldStroke = g.getStroke();
47     g.setStroke(new BasicStroke(thickness));
48     g.setColor(color);
49
50     for(int pointIndex = points.length - 1; pointIndex >= 1; pointIndex--) {
51
52         Point one = points[pointIndex];
53         Point two = points[pointIndex - 1];
54         g.drawLine((int)one.getX(), (int)one.getY(), (int)two.getX(), (int)two.getY());
55
56     }
```

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

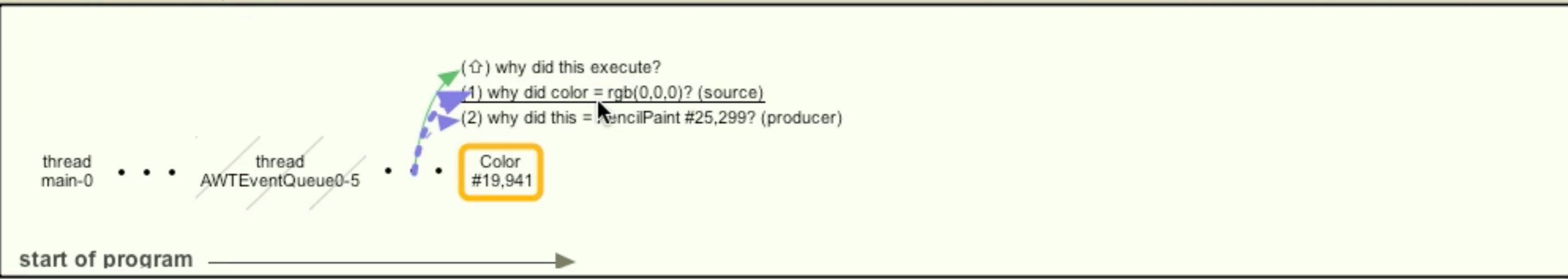
```
public void paintComponent(Graphics g) {
    public void stateChanged(ChangeEvent changeEvent) {
        objectConstructor.setColor(
            new Color(
                rSlider.getValue(),
                gSlider.getValue(),
                gSlider.getValue()));
    }
}
```

PaintWindow.java

Q why did color = ■?

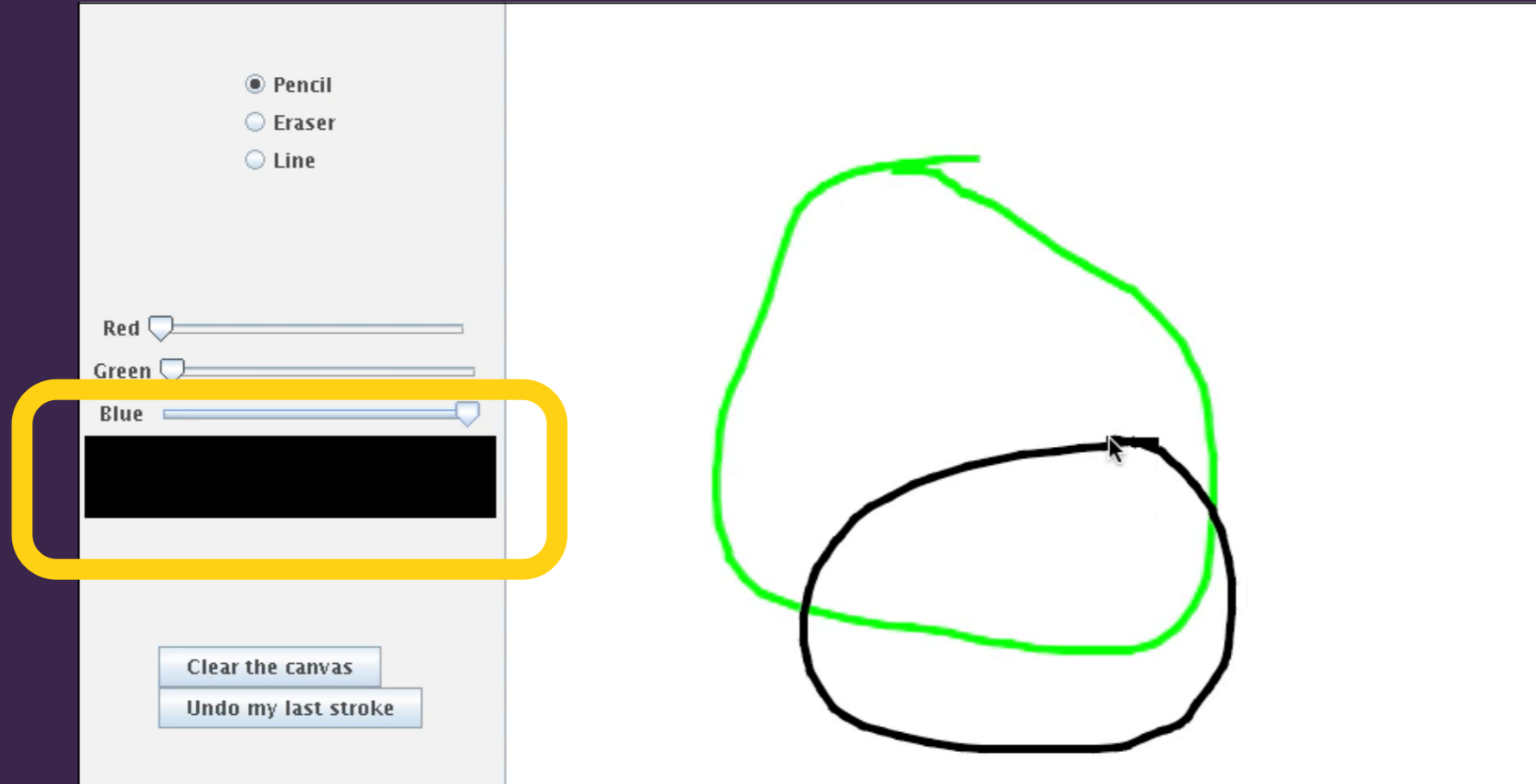
A These events were responsible.

← event → event ← in method → in method ← in thread → in thread ⤴ block collapse/expand show threads

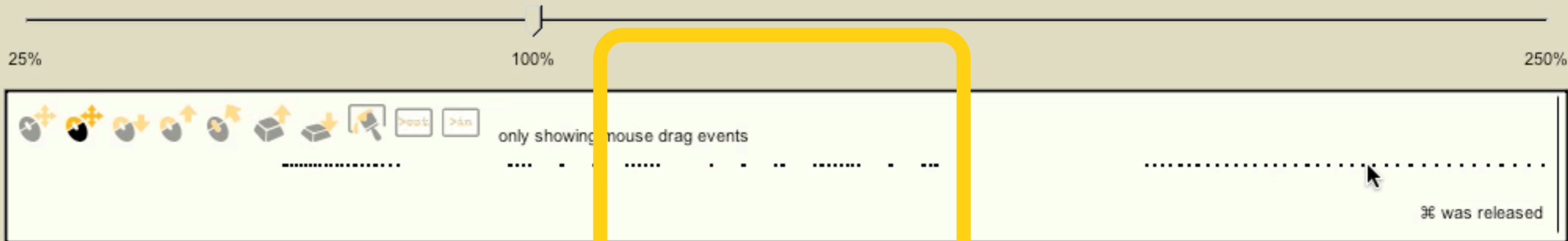
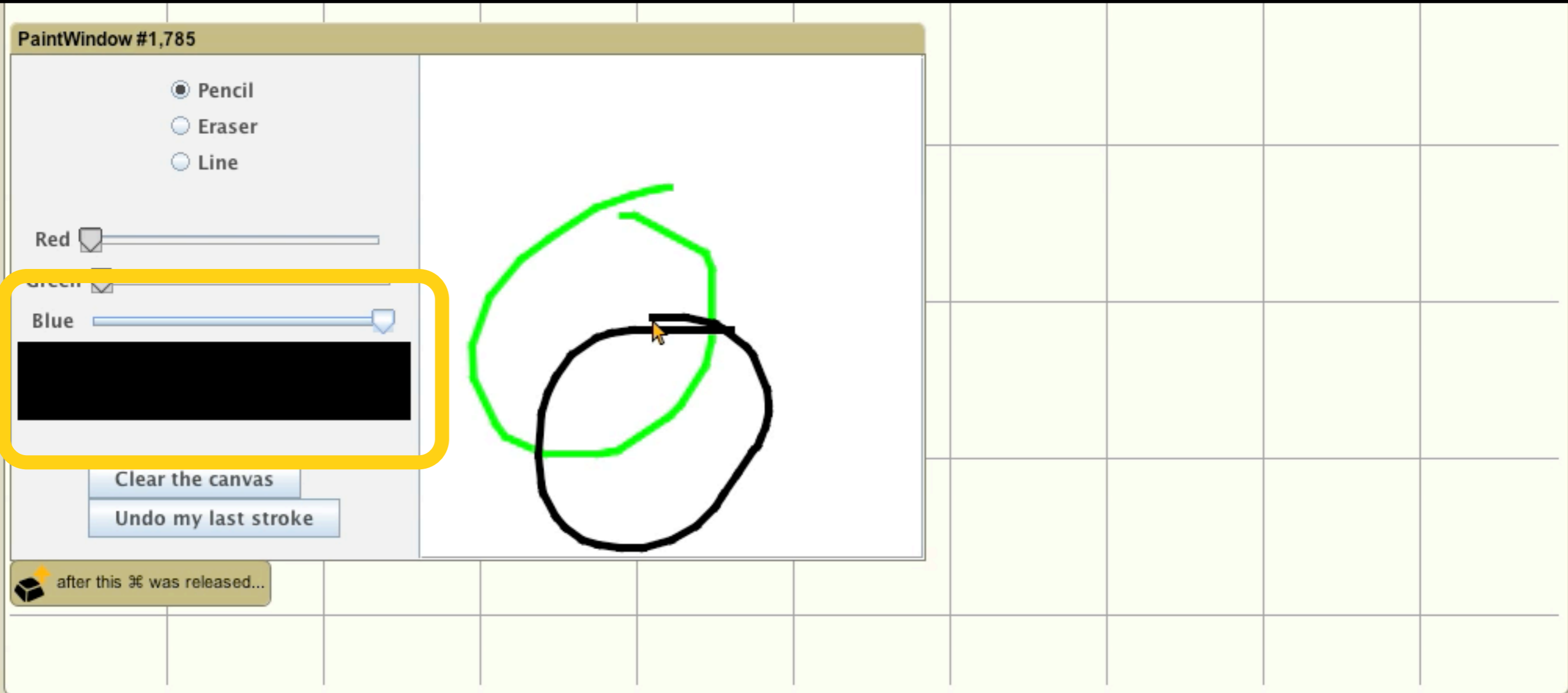


Ask why did color = ■?

why didn't the panel **repaint**?



find the appropriate **time**



click on relevant output

PaintWindow #1,785

- Pencil
- Eraser
- Line

Red

Green

Blue

objects related to rectangle

Clear the canvas

Undo my last stroke

fields and methods of selected object

after this mouse drag...

25% 100% 250%

only showing mouse drag events

mouse drag

it **did** paint...

JComponent "currentColorComponent" ▶ why did JComponent "currentColorComponent" execute?

JPanel "colorPanel" ▶

JPanel "controlPanel" ▶

JPanel "5" ▶ **this method did execute!**

PaintWindow ▶

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 not execute!

where did **black** come from?

```
31         gSlider.getValue(),
32         gSlider.getValue(),
33         gSlider.getValue());
34
35         repaint();
36
37     }
38 };
39
40 private JComponent currentColorComponent = new JComponent() {
41     public void paintComponent(Graphics g) {
42
43         ✓ Color oldColor = g.getColor();
44         g.setColor(objectConstructor.getColor());
45         g.fillRect(0, 0, getWidth(), getHeight());
46         g.setColor(oldColor);
47
48     }
49 };
50
51 public PaintWindow(int initialWidth, int initialHeight) {
52
53     super("Paint");
54
55 }
```

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

step forward to
getColor() call

Q why didn't paintComponent() execute?

A Check the answer below.

← event → event ← in method → in method ← in thread → in thread ↑ block collapse/expand show threads

thread main-0 • • • thread AWTEventQueue0-5



PaintWindow\$2
paintCo...()
did execute

start of program →

Ask

why didn't paintComponent() execute?

found the **bug**

```
32     g.setColor(getValue(),  
33     gSlider.getValue());  
34  
35     repaint();  
36  
37 }  
38 };  
39  
40 private JComponent currentColorComponent = new JComponent() {  
41     public void paintComponent(Graphics g) {  
42  
43         Color oldColor = g.getColor();  
44         g.setColor(objectConstructor.getColor());  
45         g.fillRect(0, 0, getWidth(), getHeight());  
46         g.setColor(oldColor);  
47  
48     }  
49 };  
50  
51 public PaintWindow(int initialWidth, int initialHeight) {  
52     super("Paint");  
53  
54     actions = new Actions(this);  
55  
56
```

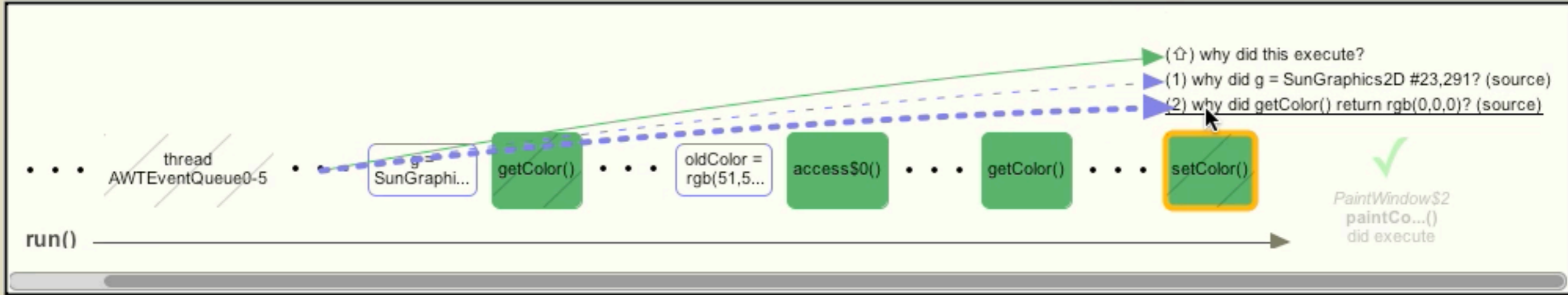
Called setColor() on SunGraphics2D #23,291
(⤴) why did this execute?
(1) why did g = SunGraphics2D #23,291? (source)
(2) why did getColor() return rgb(0,0,0)? (source)

why did getColor() return black?
same buggy code (gSlider used twice)

Q why didn't paintComponent() execute?

A Check the answer below.

← event → event ← in method → in method ← in thread → in thread ⤴ block collapse/expand show threads



Ask why didn't paintComponent() execute?

how does the **Whyline** work?

the whyline cycle

developer...

edit compile debug **record** **load..** **ask**

1 2 3

system...

instruments bytecode
records thread history

converts serial history to
random access

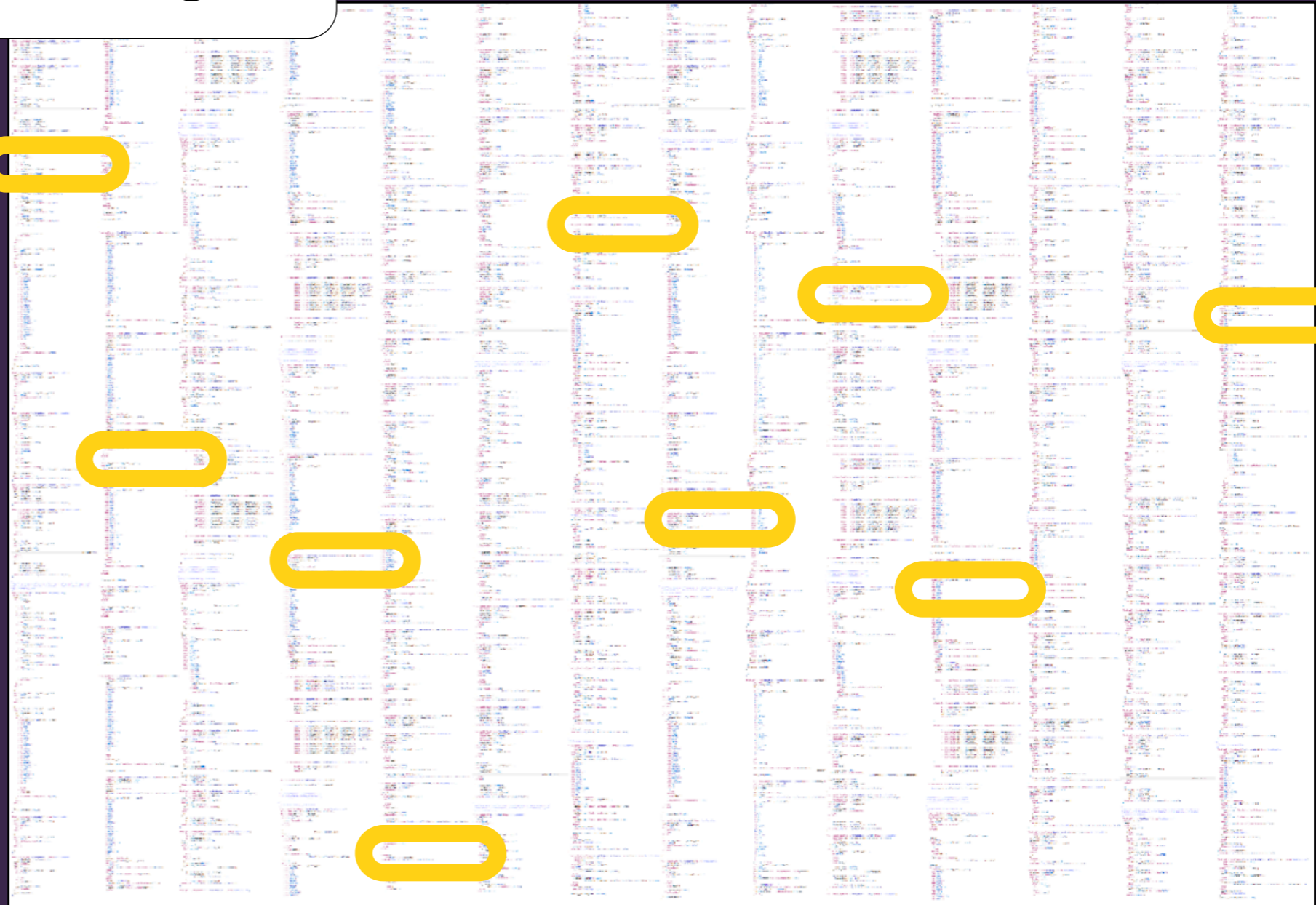
extracts questions from code

find primitive output statements

`drawString(x, y, string)`

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

`setColor(color)`



The image shows a screenshot of a code editor with a dark background and light-colored text. The code is written in a programming language that uses a mix of colors (blue, green, red, yellow) for syntax highlighting. Several lines of code are highlighted with yellow callout boxes, indicating the locations of primitive output statements. The highlighted lines include:

- `drawString(100, 100, "Hello World")`
- `drawLine(100, 100, 200, 20)`
- `setColor(255, 0, 0)`

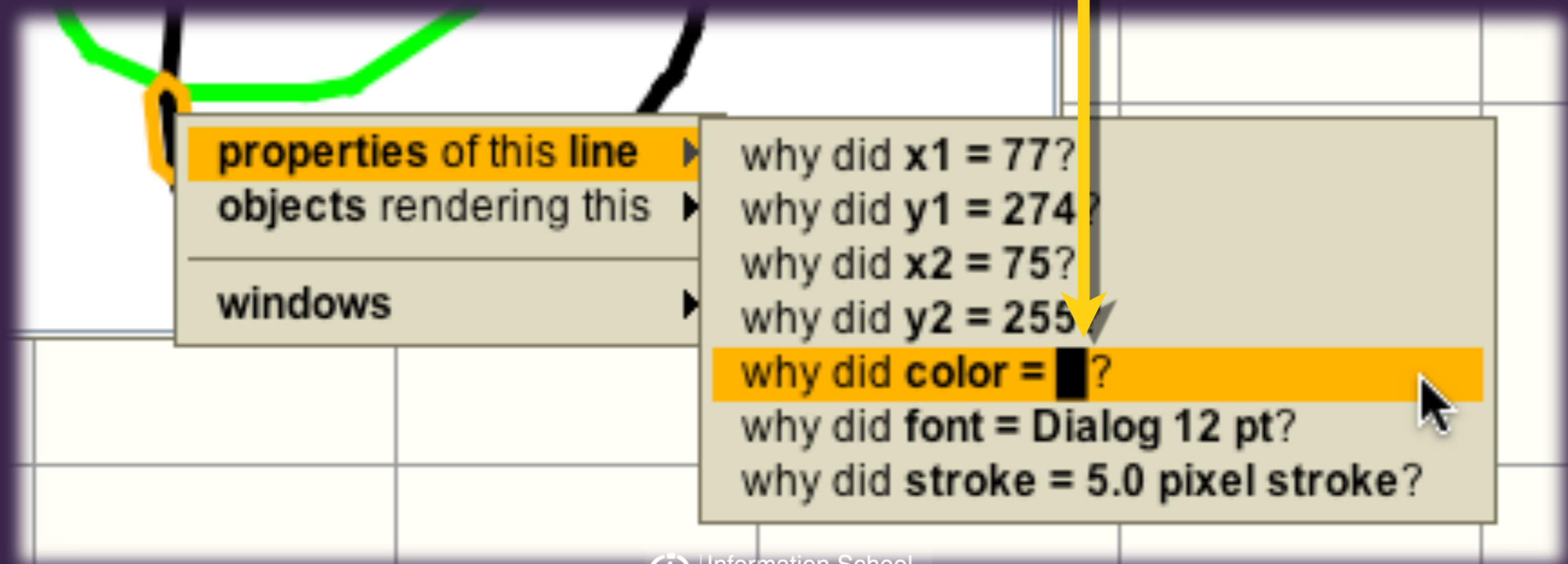
extract primitive questions

`drawString(x, y, string)`

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

`setColor(color)`

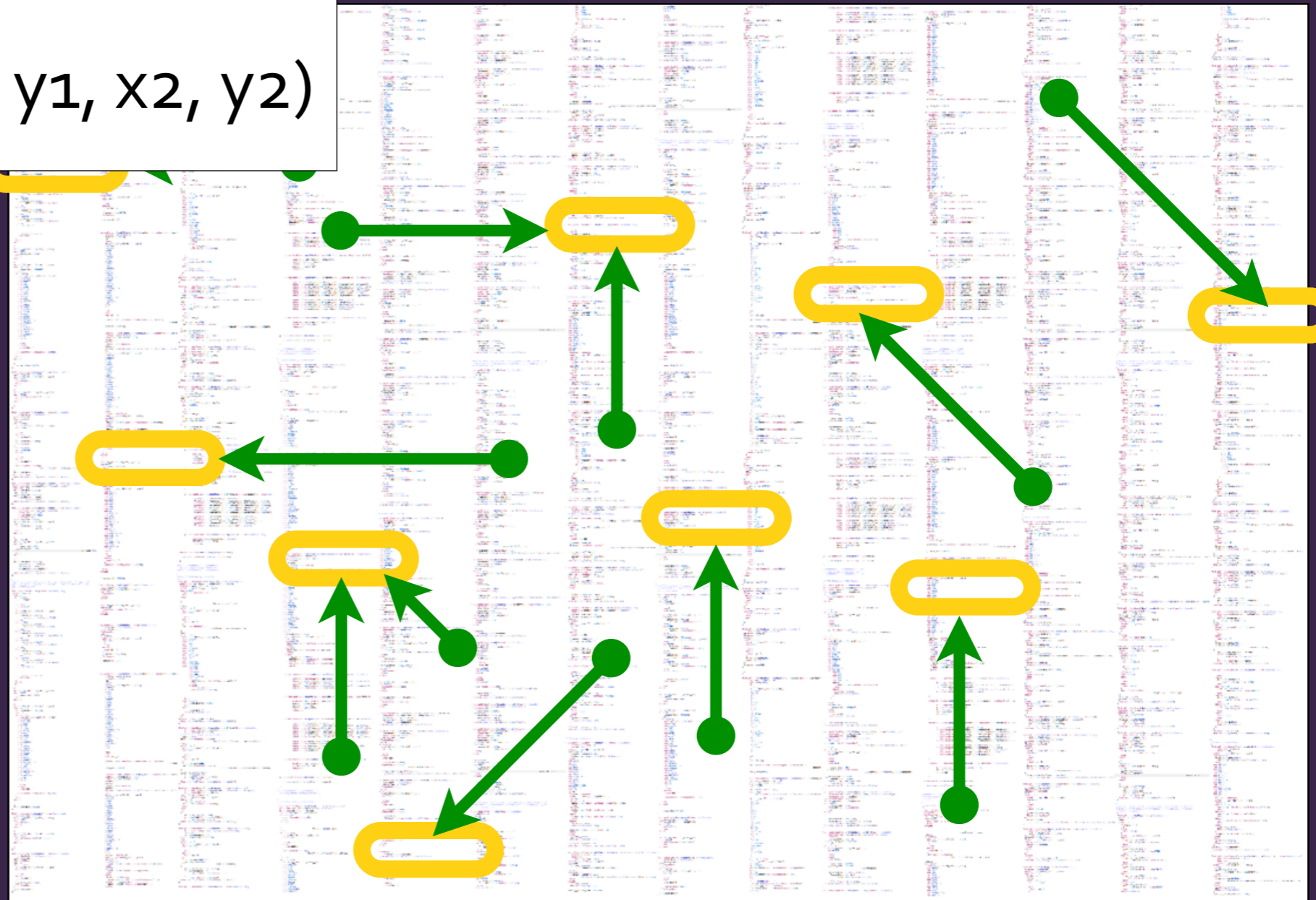
why did *argument = value*?



find output-invoking classes

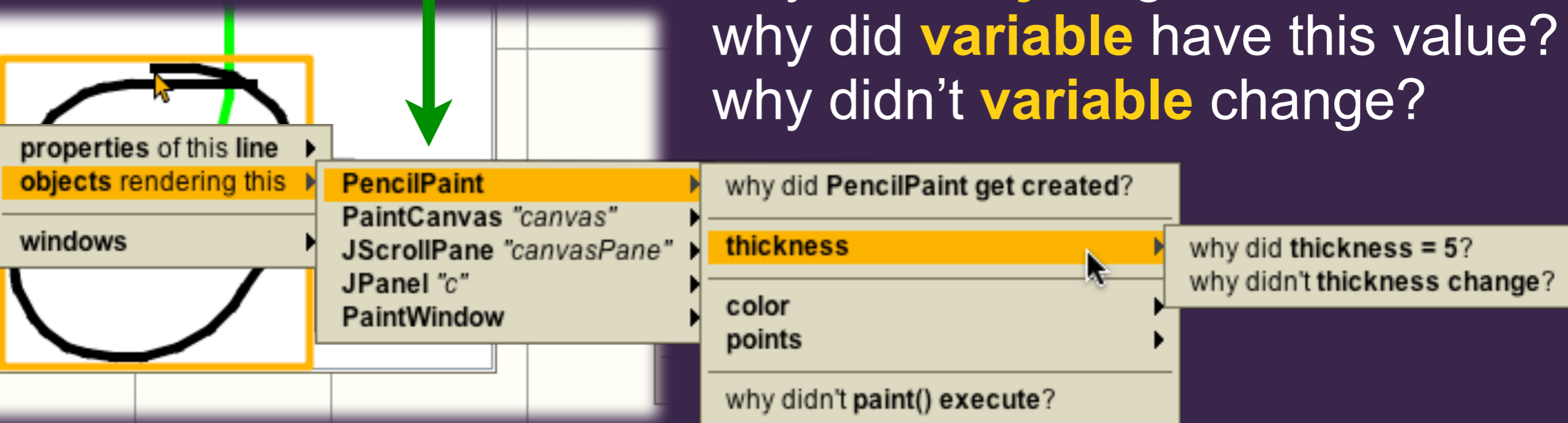
```
class PencilPaint  
  draw() {  
    ...  
    drawLine(x1, y1, x2, y2)
```

upstream
control
dependencies



extract output-invoking questions

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



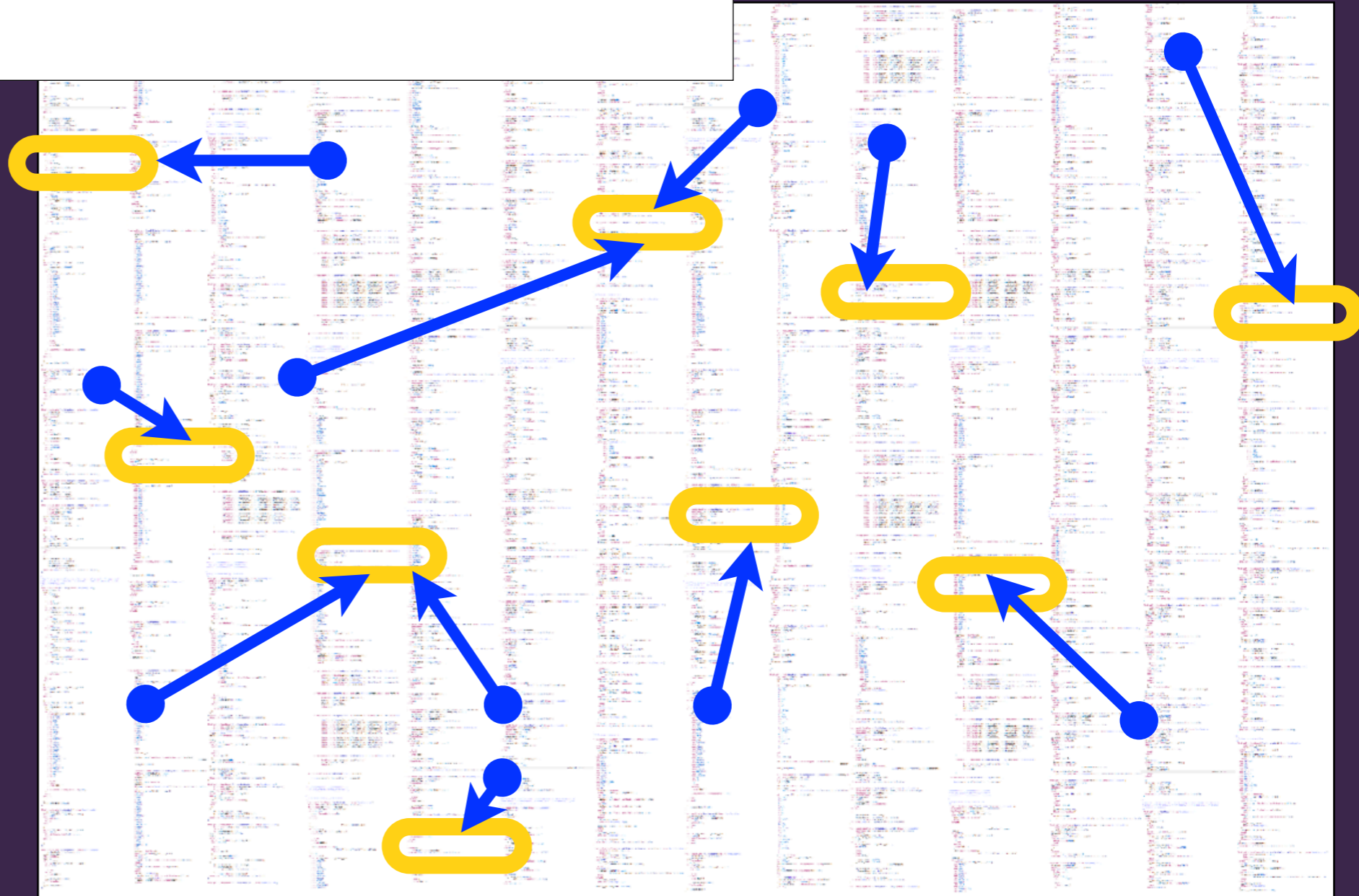
why did **subject** get created?
why did **variable** have this value?
why didn't **variable** change?

find output-affecting fields

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

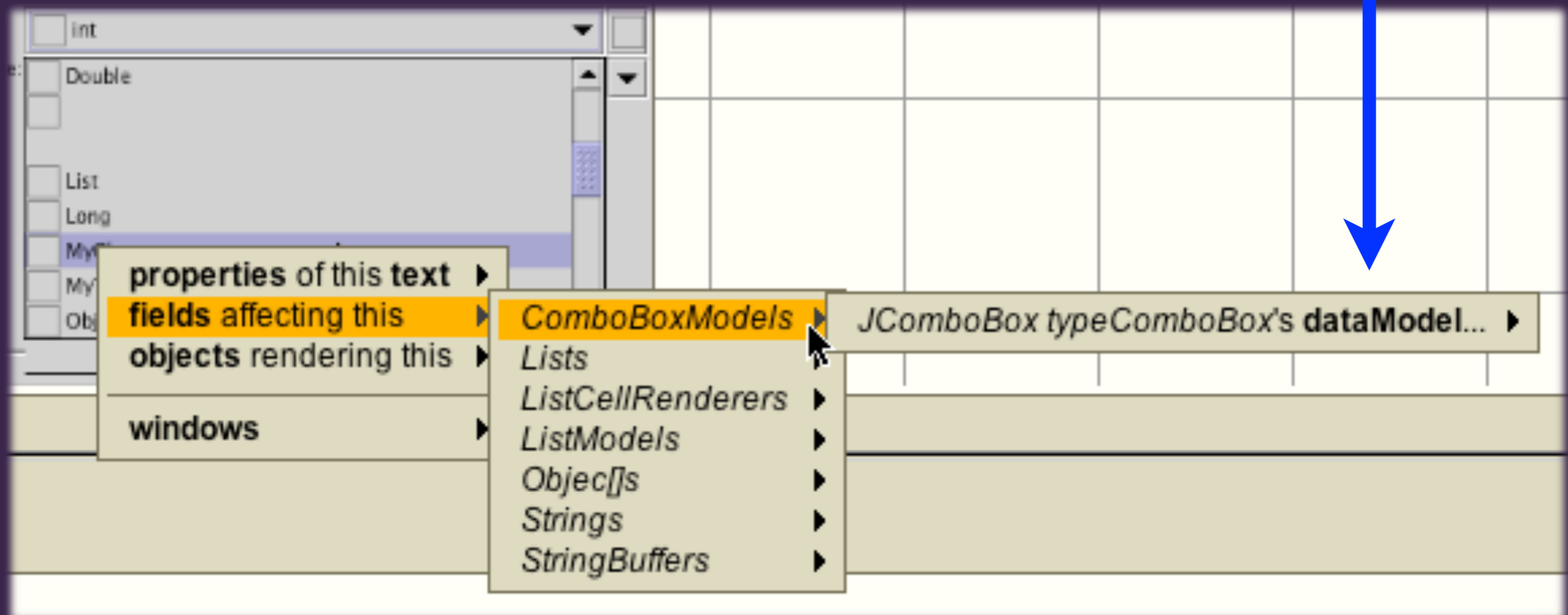
...

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

The screenshot shows an IDE's field explorer for the class `JButton "clearButton"`. The tree view is organized by type, with categories like `booleans`, `floats`, `ints`, `Colors`, `Components`, `Dimension2Ds`, `Fonts`, `Icons`, `Insets`, `Listeners`, `Maps`, `Strings`, and `Supports`. The `Dimension2Ds` category is highlighted, and its sub-fields `maxSize`, `minSize`, and `prefSize` are visible. A callout box points to these fields with the text: "i.e., three fields of type Dimension2D".

why did JButton "clearButton" get created?

JButton "clearButton"
JPanel "clearUndoPanel"
JPanel "controlPanel"
JPanel "c"
PaintWindow

booleans
floats
ints

Colors
Components
Dimension2Ds
Fonts
Icons
Insets
Listeners
Maps
Strings
Supports

maxSize
minSize
prefSize

100%

showing all i/o events

other fields

⌘ was re

filtering questions by **familiarity**

intermediaries,
delegates, proxies,
helpers, etc.

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

- may be unfamiliar

- **familiarity** = classes...
 - declared** in editable code
 - referenced** in editable code
- only include questions about **familiar classes**

all classes

PencilPaint
ComponentU
I
PaintCanvas
ScrollPaneUI
JScrollPane
ComponentU
I
JPanel

familiar classes

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

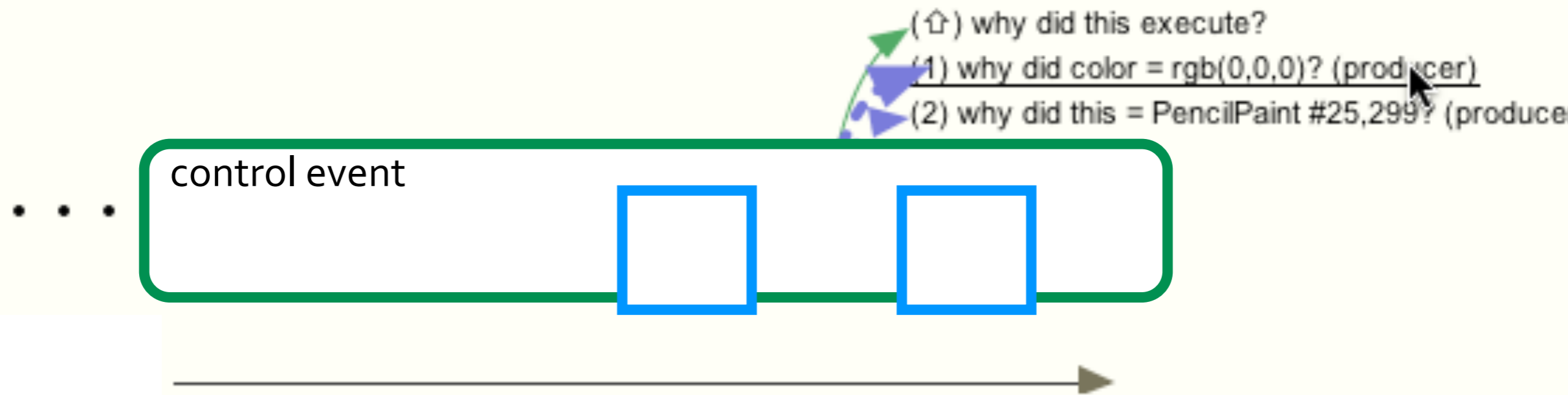
'why did' answers

answer derived with **precise dynamic slicing**

a timeline visualization of dependencies

control dependencies as **nested blocks**

data dependencies **inside** of blocks



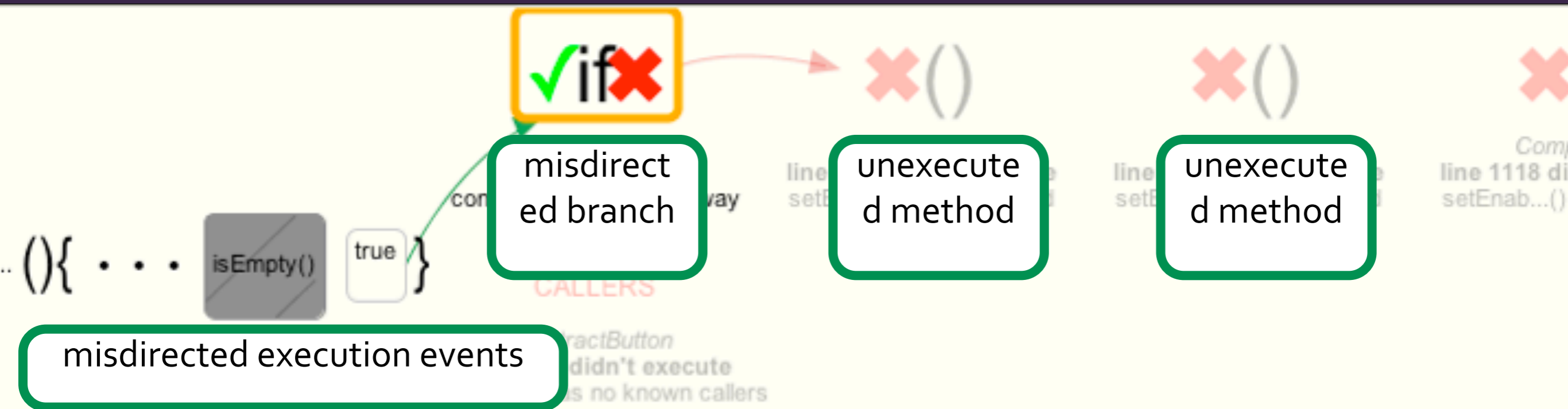
'why didn't' answers

answer with **call graph reachability** analysis

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

unexecuted methods and **branches**

misdirected calls and **branches**



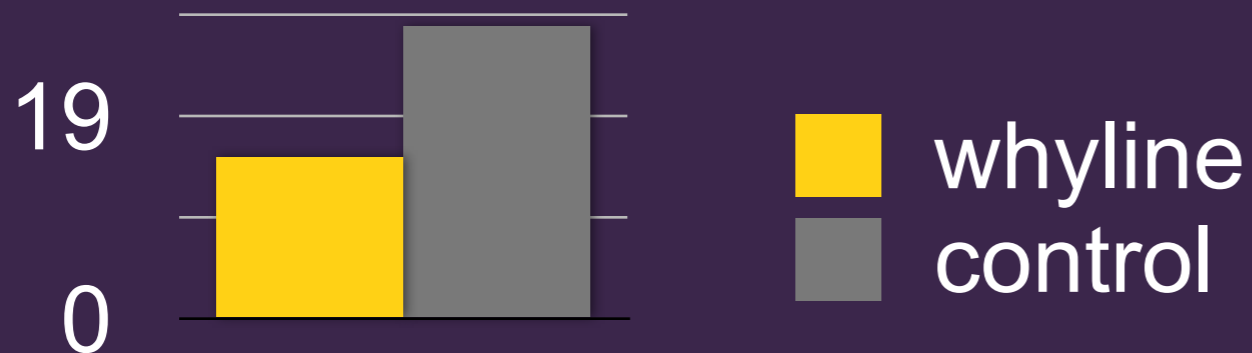
how effective is the **Whyline**?

effectiveness

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

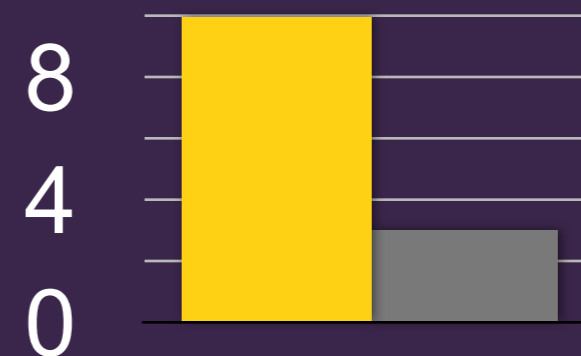
2x as fast

time (min)



successful 3x as often

successful



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** \propto

quality of identifiers

question and answer **precision** \propto

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 |
| ArgoUM L | 113K | 18 million | 137 mb | 18 mb |

of events \propto complexity of computation