

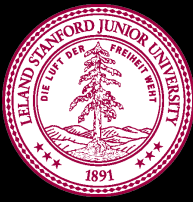
# Events

Chris Piech

CS106A, Stanford University

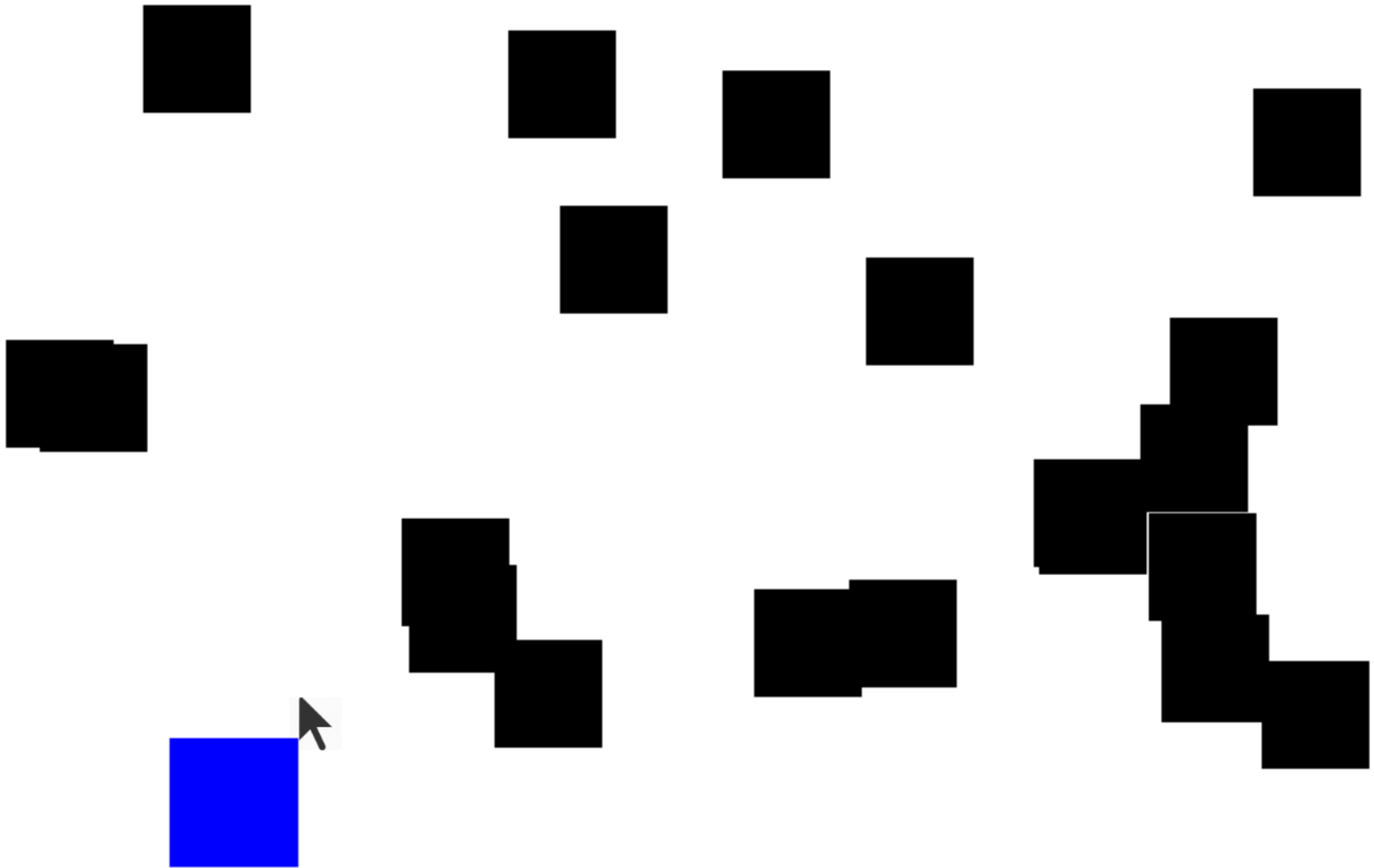
Warmup

# Making Vegas 2.0



End Warmup

# Catch Me If You Can



# We've Gotten Ahead of Ourselves



# Start at the Beginning



Source: The Hobbit

# Learning Goals

1. Write a program that can respond to mouse events
2. Use an instance variable in your program





# Listener Model

- When users interact with computer they generate events (e.g., moving/clicking the mouse)
- Can respond to events by having listener for events  
**addMouseListeners ( )**
- Listeners get control of the program when an event happens.



# Responding to Mouse Events

1. The `run` method should call `addMouseListeners`
2. Write definitions of any listener methods needed

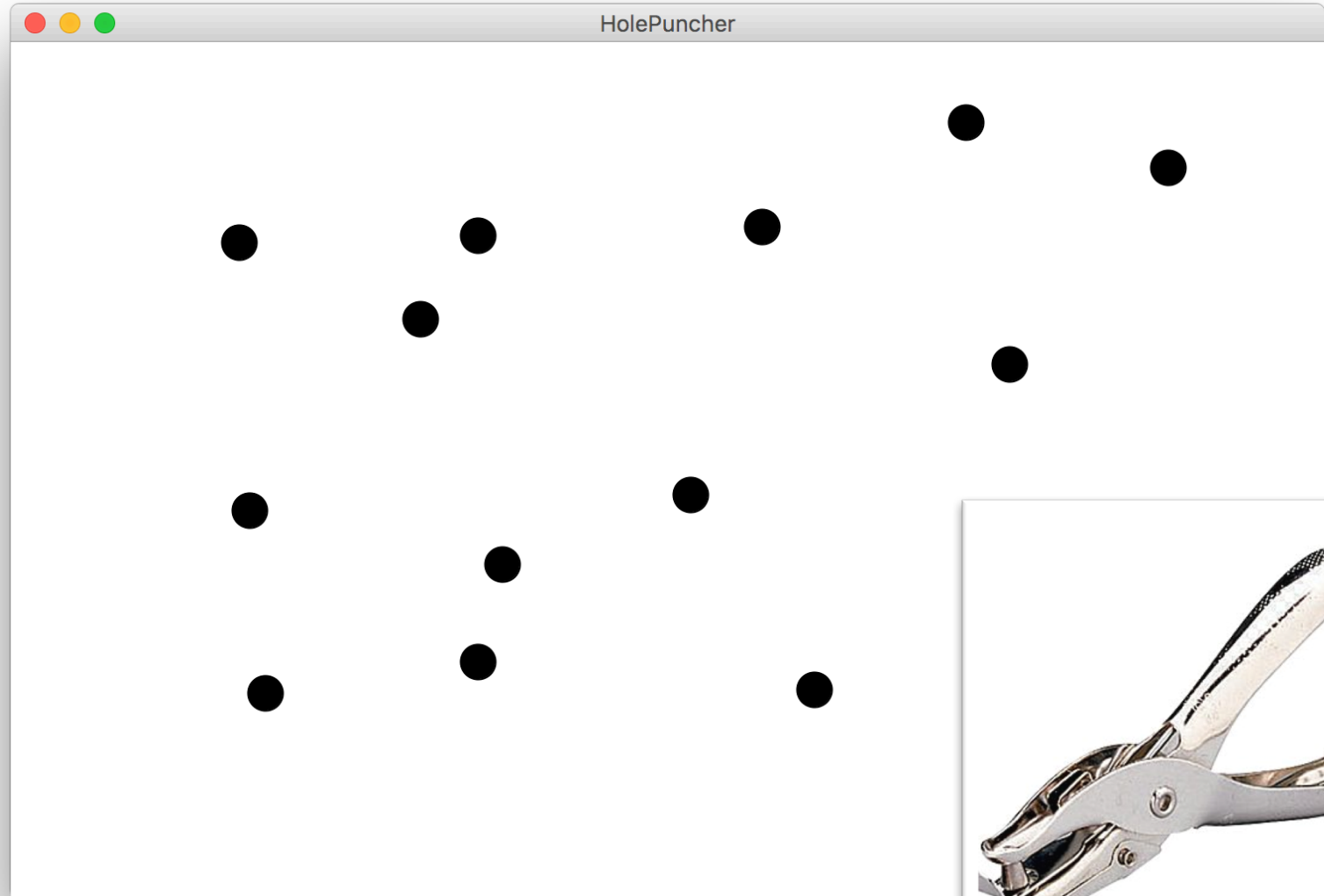
<code>mouseClicked(<i>e</i>)</code>	Called when the user clicks the mouse
<code>mousePressed(<i>e</i>)</code>	Called when the mouse button is pressed
<code>mouseReleased(<i>e</i>)</code>	Called when the mouse button is released
<code>mouseMoved(<i>e</i>)</code>	Called when the user moves the mouse
<code>mouseDragged(<i>e</i>)</code>	Called when the mouse is dragged with the button down

The parameter *e* is **MouseEvent** object, which provides more data about event, such as the location of mouse.



Example

# Hole Puncher



Now With Dancing Children

# Normal Program

## Run Method

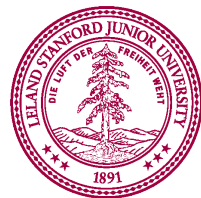


# Normal Program

## Run Method



```
public void run() {  
    for(int i = 0; i < N_DRIBBLES; i++) {  
        dropOneDribble();  
    }  
}
```

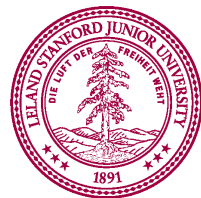


# Normal Program

## Run Method



```
public void run() {  
    for(int i = 0; i < N_DRIBBLES; i++) {  
        dropOneDribble();  
    }  
}
```



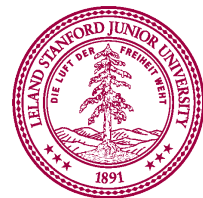


# Normal Program

## Run Method



```
public void run() {  
    for(int i = 0; i < N_DRIBBLES; i++) {  
        dropOneDribble();  
    }  
}
```



# Normal Program

## Run Method



```
public void run() {  
    for(int i = 0; i < N_DRIBBLES; i++) {  
        dropOneDribble();  
    }  
}
```

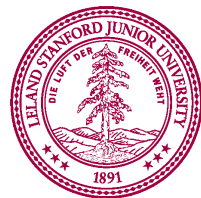


# Normal Program

## Run Method



```
public void run() {  
    for(int i = 0; i < N DRIBBLES; i++) {  
        dropOneDribble();  
    }  
}
```



# Normal Program

## Run Method

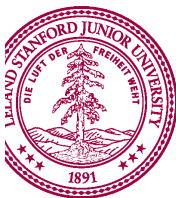


# New Listener Characters

Mouse Listener



Mouse Moved Method



# Program with a Mouse Method

Run Method

Mouse Moved Method



# Program Starts Running

Run Method

Mouse Moved Method



# Add Mouse Listener

Run Method

Mouse Moved Method

Mouse Listener



```
addMouseListeners ( );
```

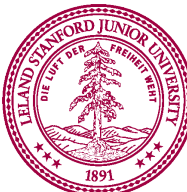


# Program Runs as Usual

Run Method

Mouse Moved Method

Mouse Listener



# Mouse Moved!

Run Method



Mouse Moved Method



Mouse Listener



# Calls Mouse Moved Method

Run Method

Mouse Moved Method

Mouse Listener



# When done, Run continues.

Run Method

Mouse Moved Method

Mouse Listener



# Keeps Doing Its Thing...

Run Method



Mouse Moved Method



Mouse Listener



# Mouse Moved!

Run Method



Mouse Moved Method



Mouse Listener



# Calls Mouse Moved Method

Run Method

Mouse Moved Method

Mouse Listener



# When done, Run continues.

Run Method

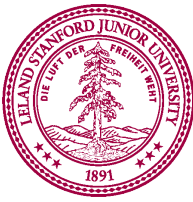
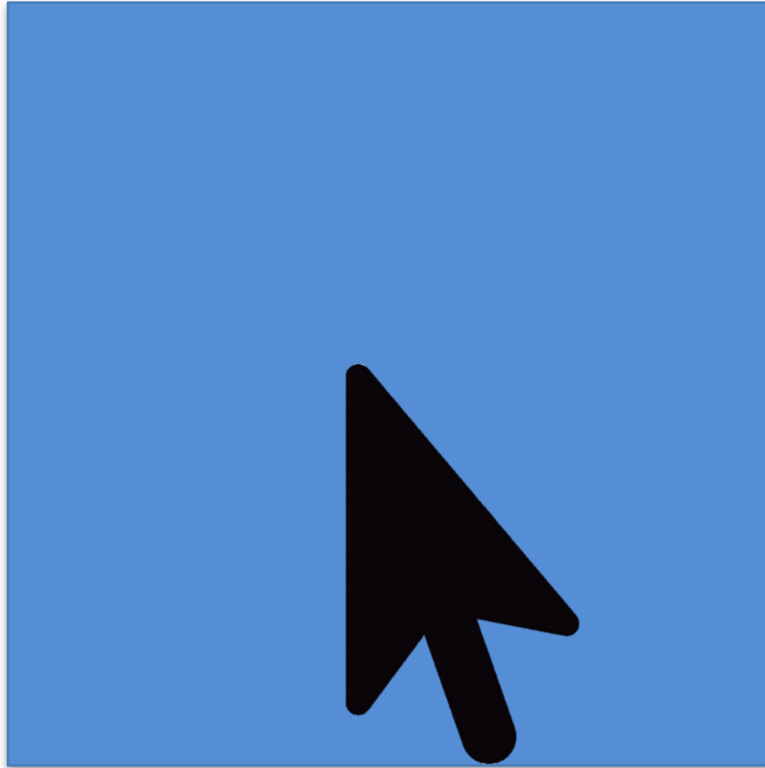
Mouse Moved Method

Mouse Listener

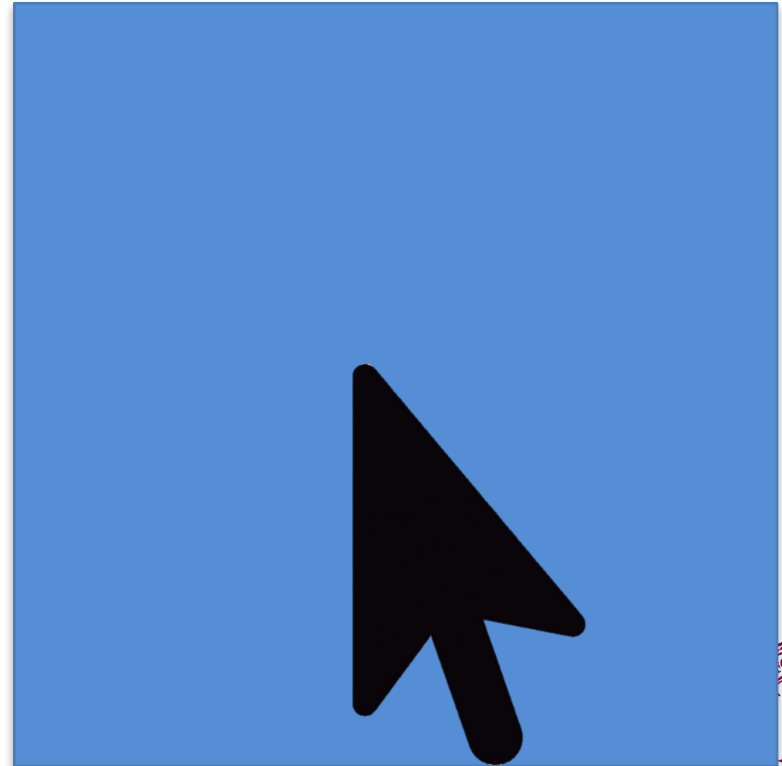




# Mouse Tracker



# Mouse Tracker



# Instance Variables

1. Variables exist until their inner-most control block ends.
2. If a variable is defined outside all methods, its inner-most control block is the entire program!
3. We call these variables **instance variables**

```
public class MouseTrackerSoln extends GraphicsProgram {  
  
    /* Instance variable for the square to be tracked */  
    GRect square = null;  
  
    public void run() {  
        addSquare();  
        addMouseListeners();  
    }  
}
```

\* Instance variables have special meanings in programs with multiple files. For now you need to know that all methods can see them and that their initialization line is executed before run.



# Instance Variables + Events

Often you need instance variables to pass information between the run method and the mouse event methods!

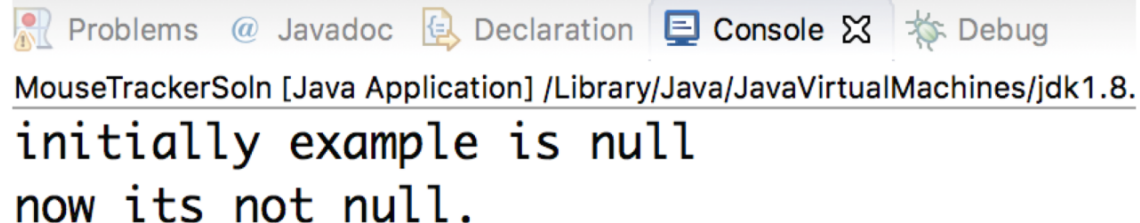
```
public class MouseTrackerSoln extends GraphicsProgram {  
  
    /* Instance variable for the square to be tracked */  
    GRect square = null;  
  
    public void run() {  
        square = makeSquare();  
        addMouseListeners();  
    }  
  
    public void mouseMoved(MouseEvent e) {  
        int x = e.getX() - SQUARE_SIZE/2;  
        int y = e.getY() - SQUARE_SIZE/2;  
        square.setLocation(x, y);  
    }  
}
```



# Null

Objects have a special value called **null** which means this variable is not associated with a value yet.

```
public void run() {  
    G0val example = null;  
    if(example == null) {  
        println("initially example is null");  
    }  
    example = new G0val(5, 5);  
    if(example != null) {  
        println("now its not null.");  
    }  
}
```

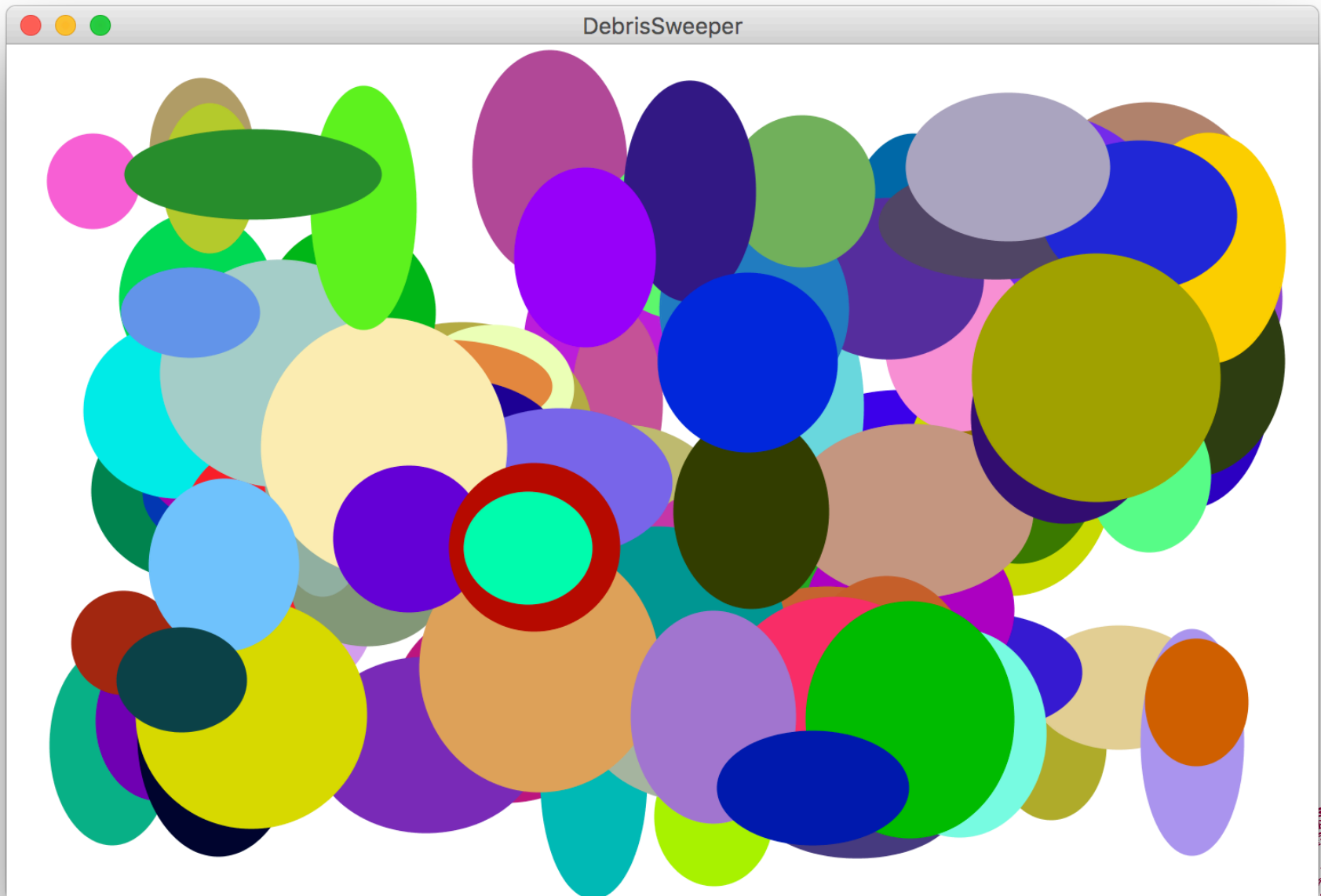


The screenshot shows an IDE console window with a toolbar at the top containing icons for Problems, Javadoc, Declaration, Console, and Debug. The console output for the application 'MouseTrackerSoln' is as follows:

```
MouseTrackerSoln [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.  
initially example is null  
now its not null.
```



# Debris Sweeper



# New Concepts

## New Commands

- `addMouseListeners()`;
- `getElementAt(x, y)`;
- `remove(obj)`;

## New Ideas

- The Listener Model
- Instance Variables
- **`null`**



# Responding to Mouse Events

1. The `run` method should call `addMouseListeners`
2. Write definitions of any listener methods needed

<code>mouseClicked(<i>e</i>)</code>	Called when the user clicks the mouse
<code>mousePressed(<i>e</i>)</code>	Called when the mouse button is pressed
<code>mouseReleased(<i>e</i>)</code>	Called when the mouse button is released
<code>mouseMoved(<i>e</i>)</code>	Called when the user moves the mouse
<code>mouseDragged(<i>e</i>)</code>	Called when the mouse is dragged with the button down

The parameter *e* is **MouseEvent** object, which provides more data about event, such as the location of mouse.





# Responding to Keyboard Events

1. The `run` method should call `addKeyListeners`
2. Write definitions of any listener methods needed

<code>keyPressed(<i>e</i>)</code>	Called when the user presses a key
<code>keyReleased(<i>e</i>)</code>	Called when the key comes back up
<code>keyTyped(<i>e</i>)</code>	Called when the user types (presses and releases) a key

The parameter *e* is a **KeyEvent** object, which indicates which key is involved.



# And Here We Are...



# Catch Me If You Can?

