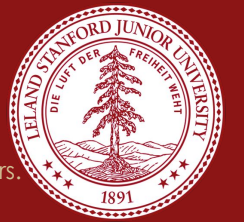
Mouse Events and Instance Variables

Lecture 10

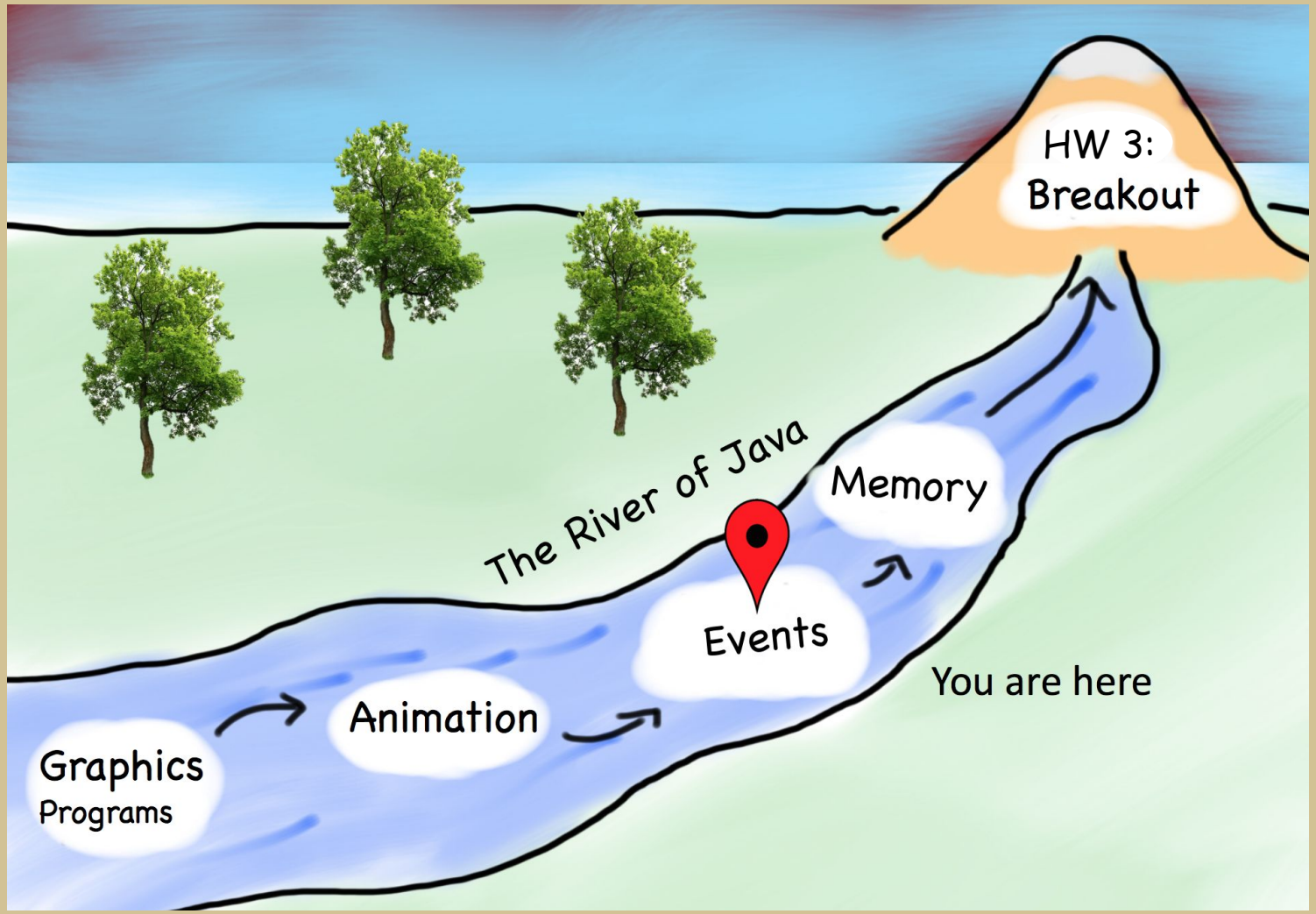
CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech, Brahm Capoor and others.



Announcements

- HW2 was due at 10AM today
- HW 3 goes out today after lecture
  - Due Thursday July 18 at 10AM
  - Can optionally be done in pairs; check out the Pair Programming link on website


HW 3:
Breakout
The River of Java
Memory
Events
You are here
Animation
Graphics
Programs

Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Review: Animation Loop

```
public void run() {
    // setup
    Make variables. Add graphics to canvas.

    while (condition) {
        // update world
        Update graphics.

        // pause
        pause(milliseconds);
    }
}
```

Review: Animation Loop

```
public void run() {
    // setup
    GRect square = makeSquare();

    while (true) {
        // update world
        square.move(1, 0);

        // pause
        pause(PAUSE_TIME);
    }
}
```

Review: RandomGenerator

```
// this variable can generate random values
RandomGenerator rgen = RandomGenerator.getInstance();

// make a random number between 1 and 6 inclusive
int diceRoll = rgen.nextInt(1, 6);

// also: nextDouble, nextBoolean, nextColor, etc
```

Plan for Today

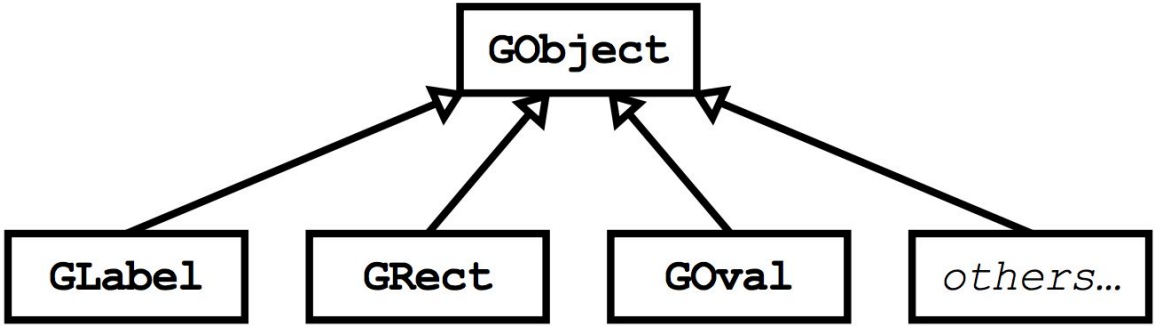
- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

getElementAt

- The method:

```
GObject getElementAt(double x, double y);
```

returns which object is at the given location on the canvas.



getElementAt

- The method:

```
GObject getElementAt(double x, double y);
```

  returns which object is at the given location on the canvas.

- The return type is `GObject`, since we don't know what specific type (`GRect`, `GOval`, etc.) is really there.

getElementAt

- The method:

```
GObject getElementAt(double x, double y);
```

  returns which object is at the given location on the canvas.

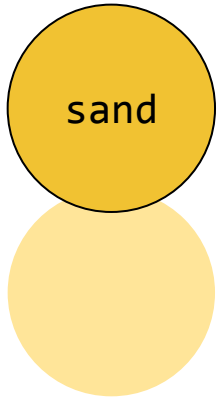
- The return type is `GObject`, since we don't know what specific type (`GRect`, `GOval`, etc.) is really there.
- If no object is present, the special value `null` is returned.

Sand Art Revisited

```
/*
 * Given a grain of sand, returns whether that sand has
 * collided with any other objects on screen.
 */
private boolean hasHitSomethingElse(GOval sand) {
  double checkX = sand.getX() + sand.getWidth() / 2.0;
  double checkY = sand.getY() + sand.getHeight();

  GObject collidingObject = getElementAt(checkX, checkY);

  return collidingObject != null;
}
```



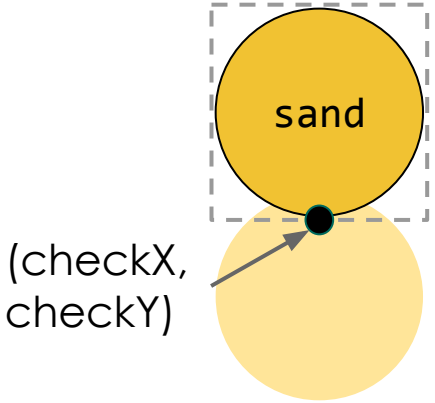
Has sand hit something else?

getElementAt

```
/*
 * Given a grain of sand, returns whether that sand has
 * collided with any other objects on screen.
 */
private boolean hasHitSomethingElse(GOval sand) {
  double checkX = sand.getX() + sand.getWidth() / 2.0;
  double checkY = sand.getY() + sand.getHeight();

  GObject collidingObject = getElementAt(checkX, checkY);

  return collidingObject != null;
}
```



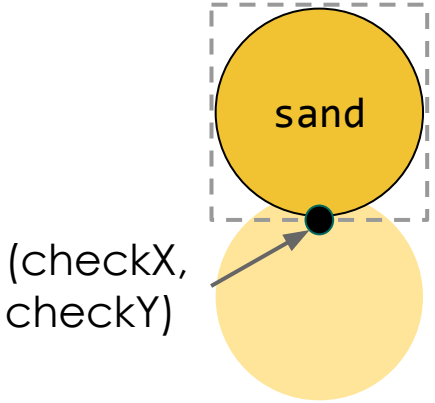
Has sand hit something else?

getElementAt

```
/*
 * Given a grain of sand, returns whether that sand has
 * collided with any other objects on screen.
 */
private boolean hasHitSomethingElse(GOval sand) {
  double checkX = sand.getX() + sand.getWidth() / 2.0;
  double checkY = sand.getY() + sand.getHeight();

  GObject collidingObject = getElementAt(checkX, checkY);

  return collidingObject != null;
}
```



Has sand hit something else?

Null

`null` is a special variable value that objects can have that means "nothing". Primitives cannot be `null`.

Null

`null` is a special variable value that objects can have that means "nothing". Primitives cannot be `null`.

If a method returns an object, it can return `null` to signify "nothing". (just say `return null;`)

Null

`null` is a special variable value that objects can have that means "nothing". Primitives cannot be null.

If a method returns an object, it can return `null` to signify "nothing". (just say `return null;`)

```
// may be a GObject, or null if nothing at (x, y)
GObject maybeAnObject = getElementAt(x, y);
```

Null

`null` is a special variable value that objects can have that means "nothing". Primitives cannot be null.

If a method returns an object, it can return `null` to signify "nothing". (just say `return null;`)

```
// may be a GObject, or null if nothing at (x, y)
GObject maybeAnObject = getElementAt(x, y);
```

Objects have the value `null` before being initialized.

```
GOval circle;   // initially null
```

Null

You can check if something is `null` using `==` and `!=`

```
// may be a GObject, or null if nothing at (x, y)
GObject maybeAnObject = getElementAt(x, y);
if (maybeAnObject != null) {
    // do something with maybeAnObject
} else {
    // null – nothing at that location
}
```

Null

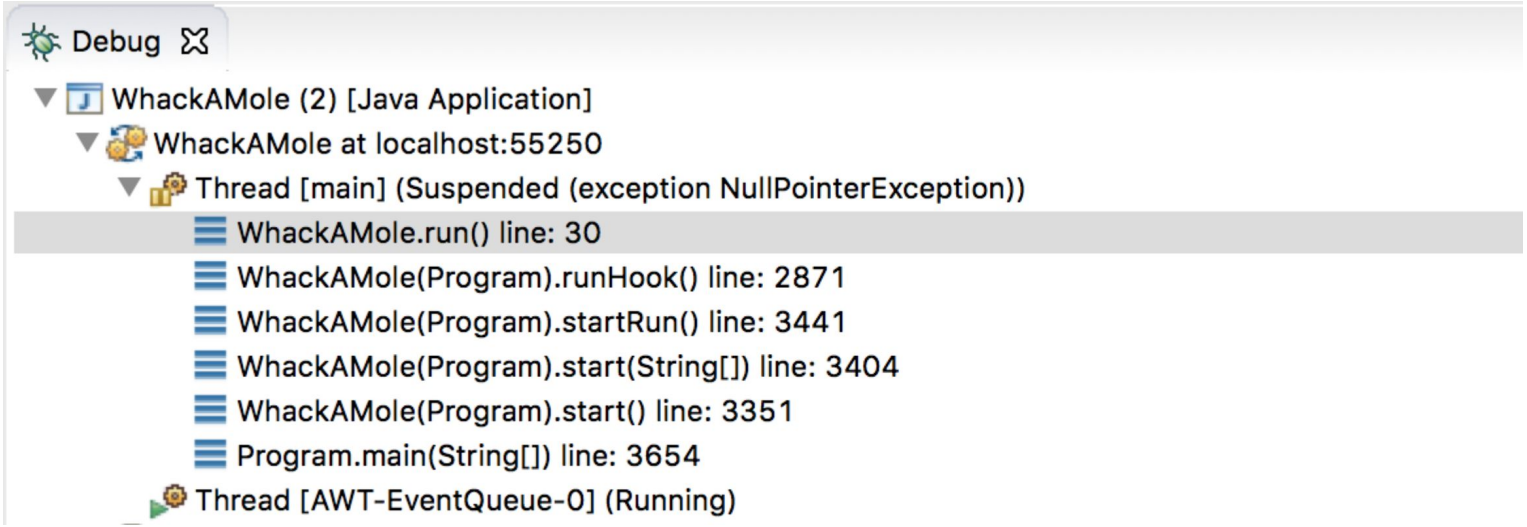
Calling methods on an object that is `null` will crash your program!

```
// may be a GObject, or null if nothing at (x, y)
GObject maybeAnObject = getElementAt(x, y);
if (maybeAnObject != null) {
    int x = maybeAnObject.getX();  // OK
} else {
    int x = maybeAnObject.getX();  // CRASH!
}
```

Null

Calling methods on an object that is `null` will crash your program!

⇒ Throws a `NullPointerException`



Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Events

- An ***event*** is some external stimulus that your program can respond to.

- ***event-driven programming***: A programming paradigm (common in graphical programs) where your code is executed in response to user events.

Events

- Common events include:
  - Mouse motion / clicking.
  - Keyboard buttons pressed.
  - Timers expiring.
  - Network data available.

Events

- Common events include:
  - **Mouse motion / clicking.**
  - Keyboard buttons pressed.
  - Timers expiring.
  - Network data available.

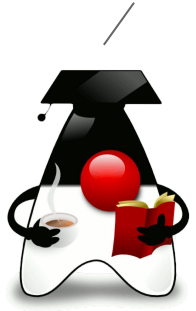
Events

```
public void run() {
    // Java runs this when program launches
}
```

Events

```
public void run() {
    // Java runs this when program launches
}
```

To **respond** to events, your program must write methods to **handle** those events.

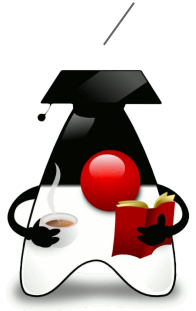


Events

```
public void run() {
    // Java runs this when program launches
}

public void mouseClicked(MouseEvent event) {
    // Java runs this when mouse is clicked
}
```

To **respond** to events, your program must write methods to **handle** those events.



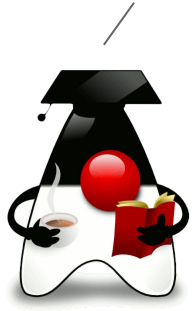
Events

```
public void run() {
    // Java runs this when program launches
}

public void mouseClicked(MouseEvent event) {
    // Java runs this when mouse is clicked
}

public void mouseMoved(MouseEvent event) {
    // Java runs this when mouse is moved
}
```

To **respond** to events, your program must write methods to **handle** those events.



Anatomy of a Mouse Method

```
public void mouseClicked(MouseEvent e) {



}
```

Anatomy of a Mouse Method

Public so other programs can call it


```
public void mouseClicked(MouseEvent e) {



}
```

Anatomy of a Mouse Method

Doesn't return anything


```
public void mouseClicked(MouseEvent e) {



}
```

Anatomy of a Mouse Method

It *must* have one of the mouse event names


```
public void mouseClicked(MouseEvent e) {



}
```

Anatomy of a Mouse Method

A collection of information about the mouse event that just occurred


```
public void mouseClicked(MouseEvent e) {



}
```

Anatomy of a Mouse Method

```
public void mouseClicked(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();

}
```


Get information about the event

Anatomy of a Mouse Method

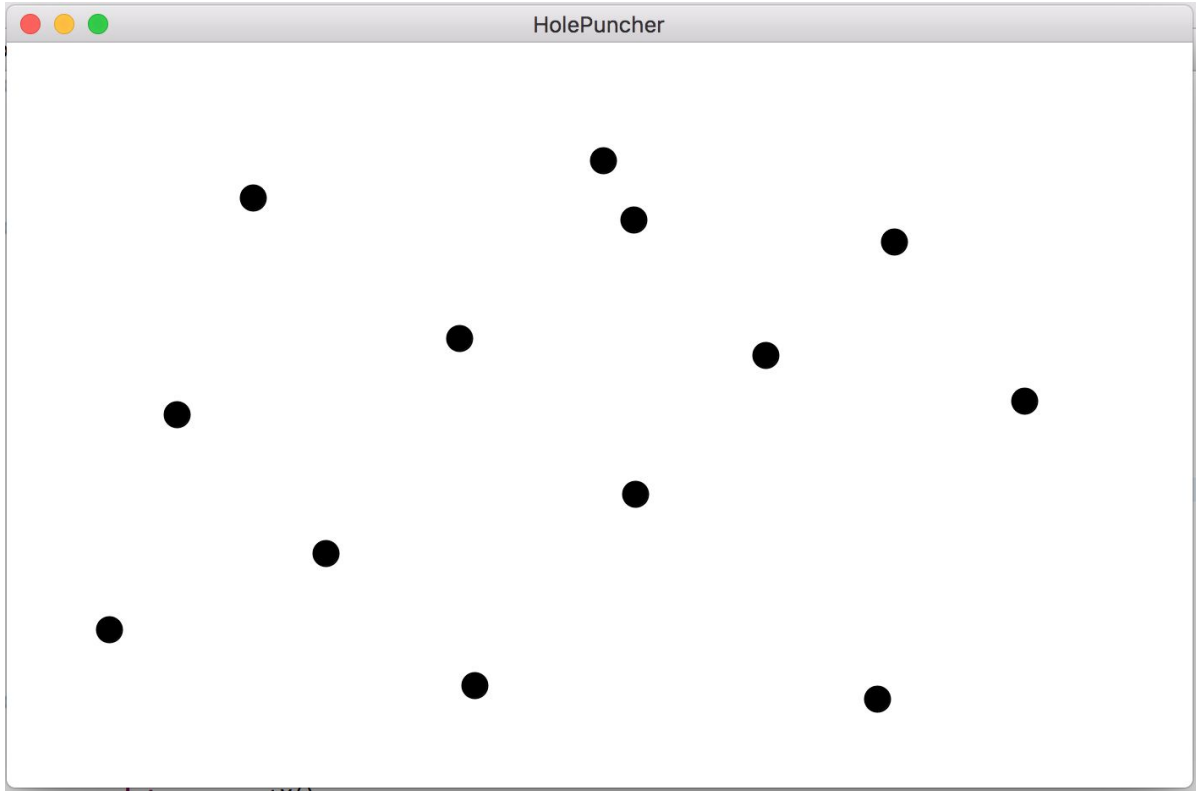
```
public void mouseClicked(MouseEvent e) {
    double mouseX = e.getX();  // mouse X-coord
    double mouseY = e.getY();  // mouse Y-coord

}
```

Anatomy of a Mouse Method

```
public void mouseClicked(MouseEvent e) {
    double mouseX = e.getX();  // mouse X-coord
    double mouseY = e.getY();  // mouse Y-coord
    // more code ...
}
```

Example: Hole Puncher



Example: Hole Puncher

```
. . .
import java.awt.event.*; // NEW

public class HolePuncher extends GraphicsProgram {

    // Adds a “hole punch” where the user clicks
    public void mouseClicked(MouseEvent e) {









    }


}
```

Example: Hole Puncher

```
. . .
import java.awt.event.*; // NEW

public class HolePuncher extends GraphicsProgram {

    // Adds a “hole punch” where the user clicks
    public void mouseClicked(MouseEvent e) {
        // Get information about the event
        double x = e.getX();
        double y = e.getY();



    }


}
```

Example: Hole Puncher

```
. . .
import java.awt.event.*; // NEW

public class HolePuncher extends GraphicsProgram {

    // Adds a "hole punch" where the user clicks
    public void mouseClicked(MouseEvent e) {
        // Get information about the event
        double x = e.getX();
        double y = e.getY();

        // Add hole punch (GOval) at the mouse location
        addHole(x, y);
    }

    private void addHole(double centerX, double centerY) { … }
}
```

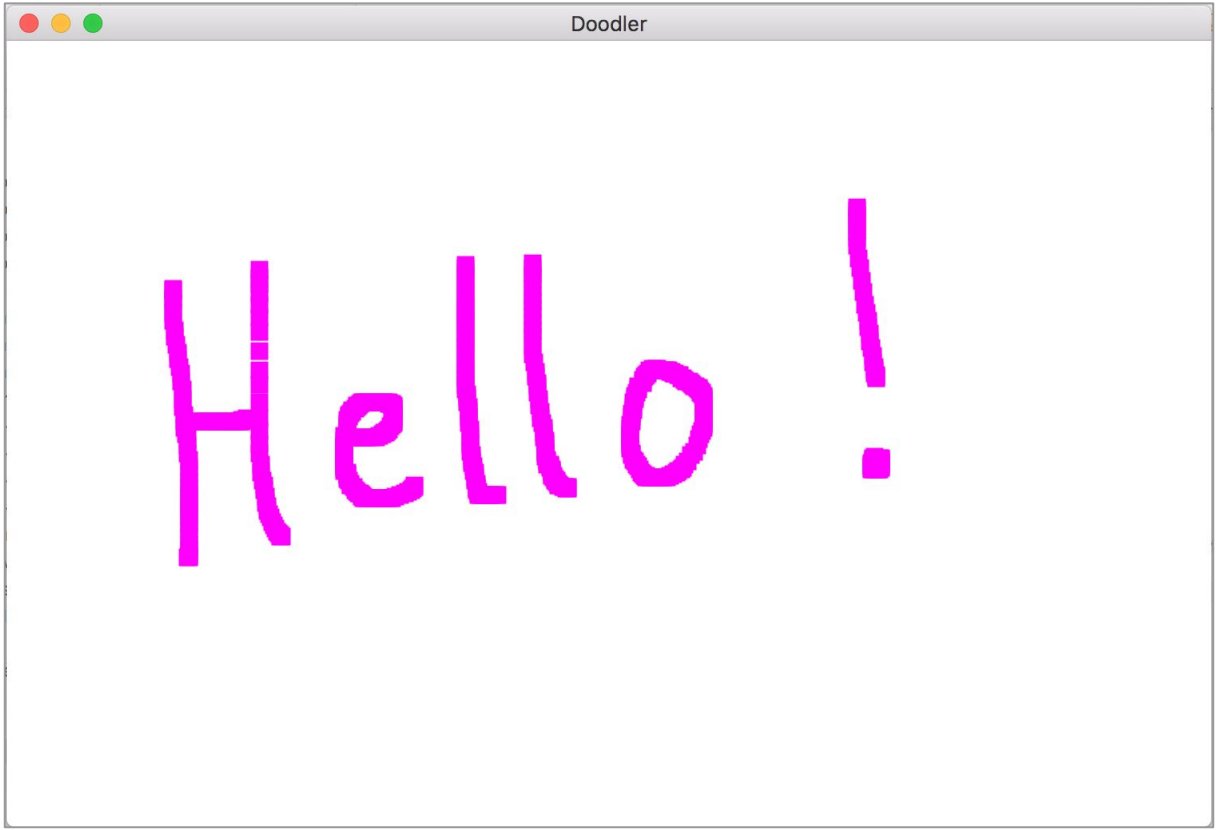
Types of Mouse Events

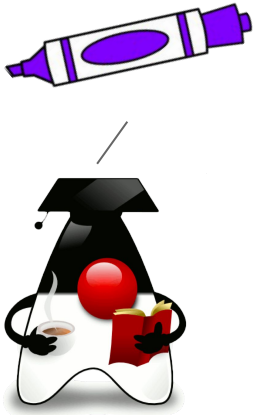
- There are many different types of mouse events!
- Each takes the form:

```
public void eventMethodName(MouseEvent e) { ...
```

Method	Description
mouseMoved	mouse cursor moves
mouseDragged	mouse cursor moves while button is held down
mousePressed	mouse button is pressed down
mouseReleased	mouse button is lifted up
mouseClicked	mouse button is pressed and then released
mouseEntered	mouse cursor enters your program's window
mouseExited	mouse cursor leaves your program's window

Example: Doodler





Example: Doodler

```
private static final int SIZE = 10;
...
public void mouseDragged(MouseEvent event) {









}
```

Example: Doodler

```
private static final int SIZE = 10;
...
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();









}
```

Example: Doodler

```
private static final int SIZE = 10;
...
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    double rectX = mouseX - SIZE / 2.0;
    double rectY = mouseY - SIZE / 2.0;




}
```

Example: Doodler

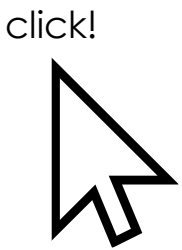
```
private static final int SIZE = 10;
...
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    double rectX = mouseX - SIZE / 2.0;
    double rectY = mouseY - SIZE / 2.0;
    GRect rect = new GRect(rectX, rectY, SIZE, SIZE);
    rect.setFilled(true);
    rect.setColor(Color.MAGENTA);
    add(rect);
}
```

Recap: Events

1. User performs some action, like moving / clicking the mouse.

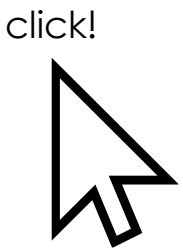
Recap: Events

1. User performs some action, like moving / clicking the mouse.



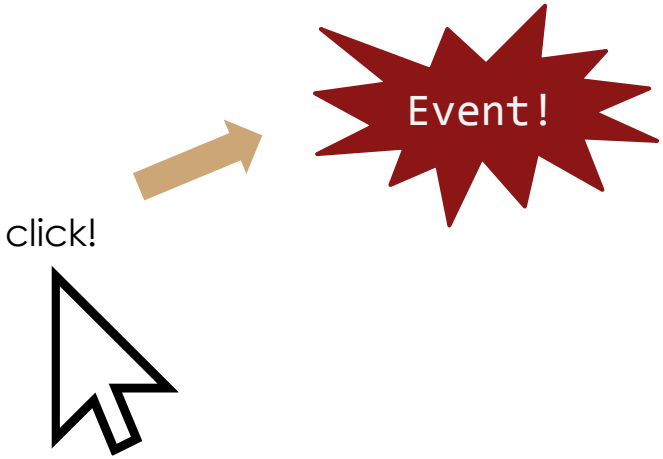
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!



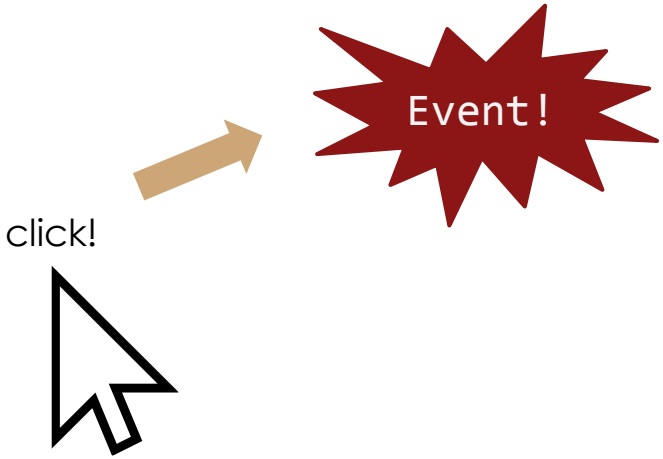
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!



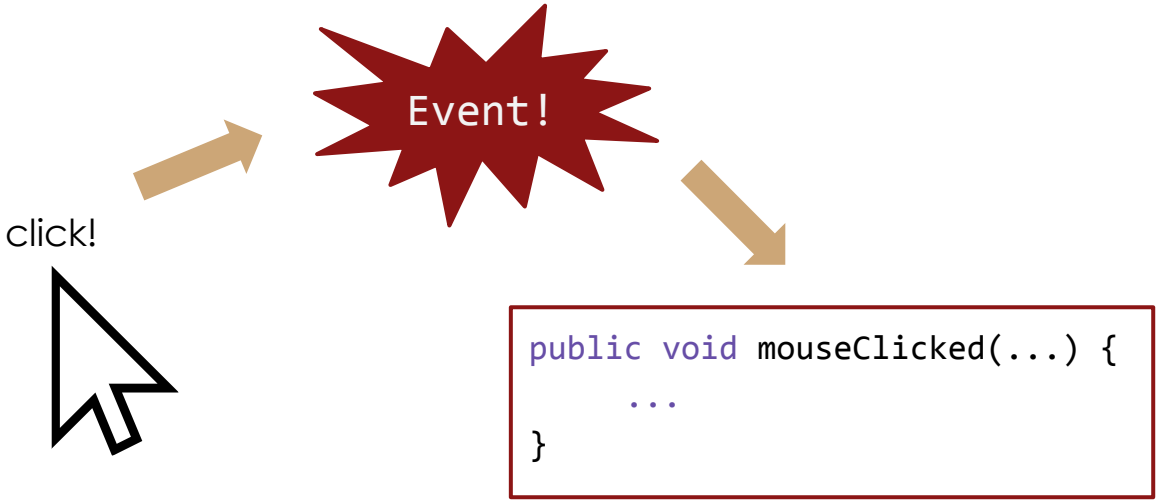
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!
3. Java executes a particular method to handle the event.



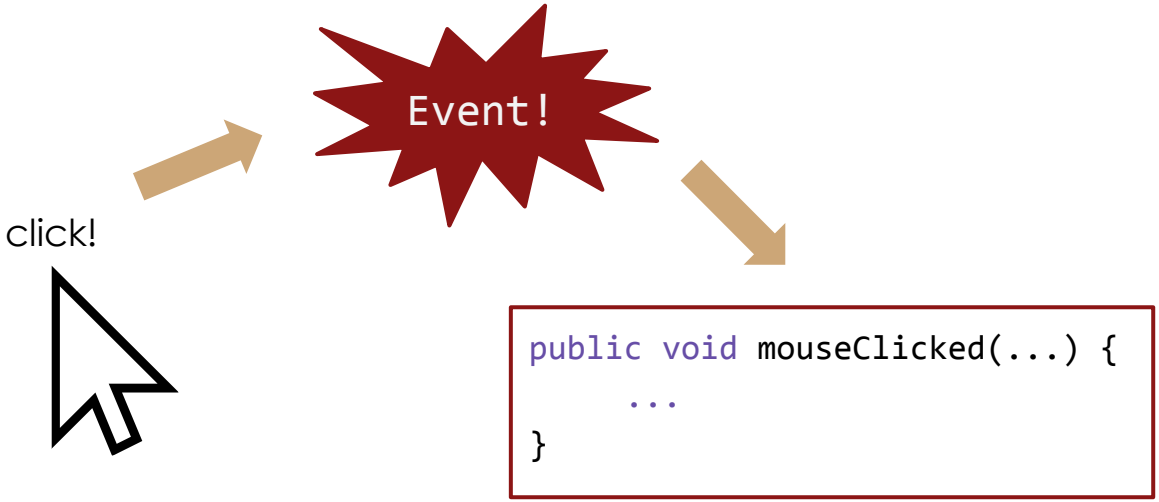
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!
3. Java executes a particular method to handle the event.



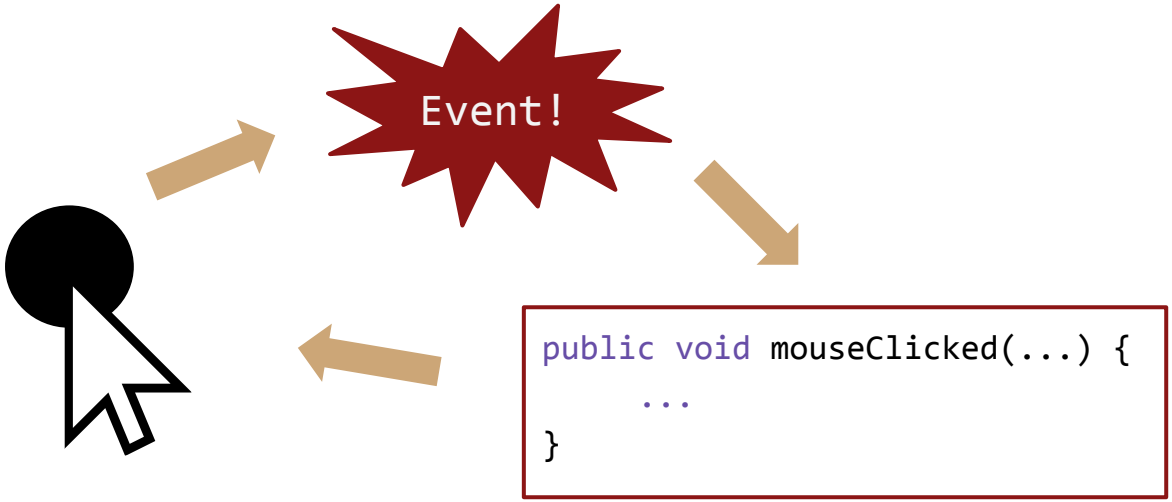
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!
3. Java executes a particular method to handle the event.
4. That method's code updates the screen appearance in some way



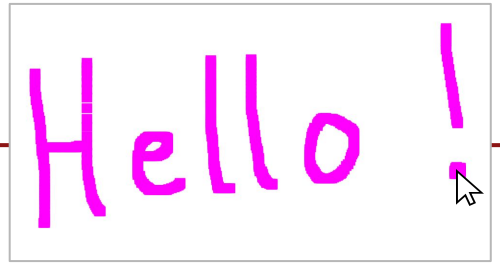
Recap: Events

1. User performs some action, like moving / clicking the mouse.
2. This causes an event to occur!
3. Java executes a particular method to handle the event.
4. That method's code updates the screen appearance in some way



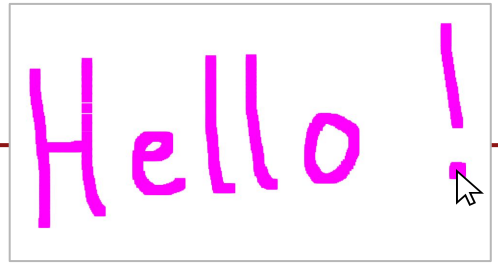
Revisiting Doodler

```
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    double rectX = mouseX - SIZE / 2.0;
    double rectY = mouseY - SIZE / 2.0;
    GRect rect = new GRect(rectX, rectY, SIZE, SIZE);
    rect.setFilled(true);
    rect.setColor(Color.MAGENTA);
    add(rect);
}
```



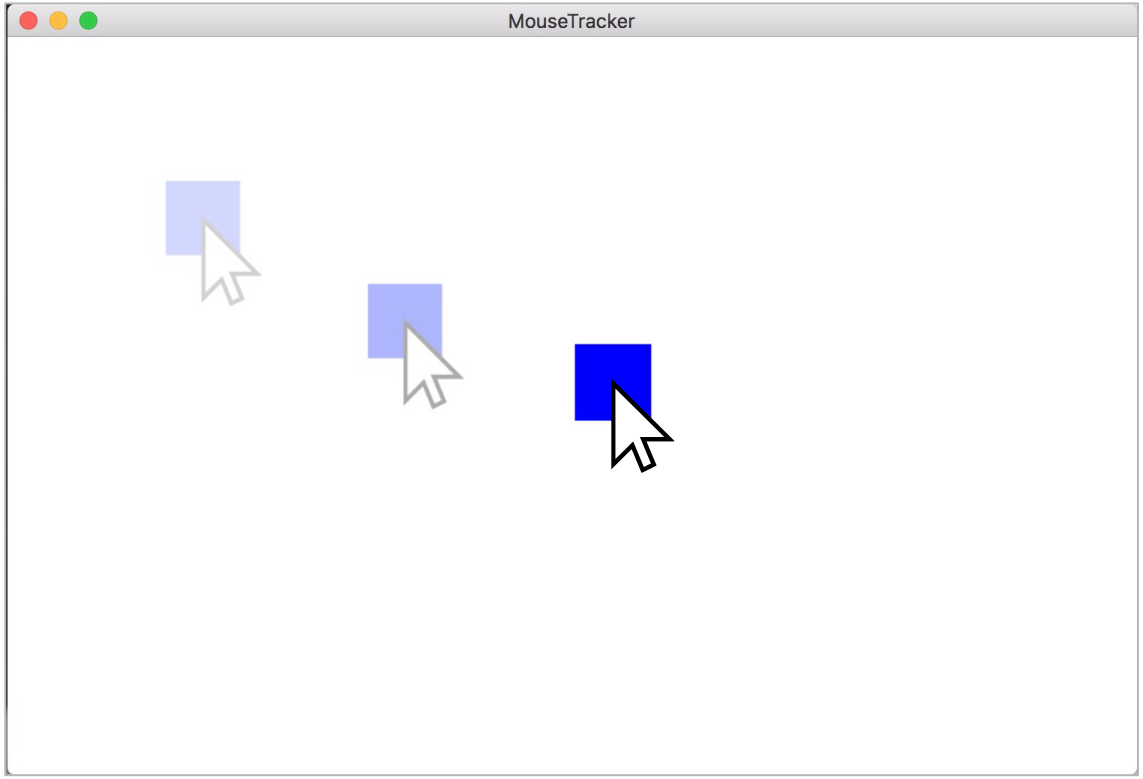
Revisiting Doodler

```
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    double rectX = mouseX - SIZE / 2.0;
    double rectY = mouseY - SIZE / 2.0;
    GRect rect = new GRect(rectX, rectY, SIZE, SIZE);
    rect.setFilled(true);
    rect.setColor(Color.MAGENTA);
    add(rect);
}
```



What if we wanted the ***same*** GRect to track the mouse, instead of making a new one each time?

MouseTracker



A Problem...

```
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more code ...
}
```

A Problem...

You don't call this method, so you can't specify its parameters


```
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more code ...
}
```

A Problem...

You don't call this method, so you can't specify its parameters

So, how can we give mouseMoved access to a single GRect we want to track?

```
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more code ...
}
```

Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Instance Variables

1. Variables exist until their inner-most control block ends.

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!

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***.

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***.

```
private type name;     // declared outside any method!
```

Instance Variables

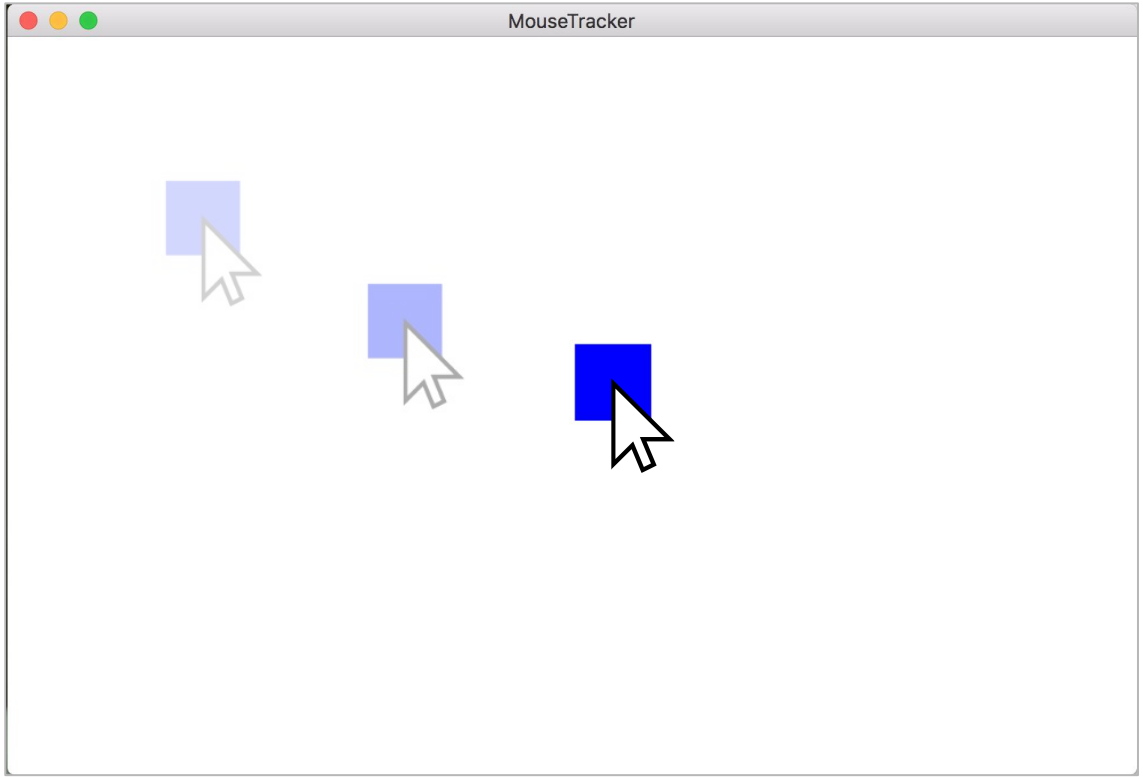
1. Variables exist until their inner-most control block ends.
2. If a variable is *defined outside all methods*, its inner-most control block is the entire program!
3. We call these variables ***instance variables***.

```
private type name;     // declared outside any method!
```

```
private GRect square;

public void run() {
    square = new GRect(...);
    GRect localSquare = new GRect(...);
}
```

Example: MouseTracker



Instance Variables + Events

Use instance variables if you need to pass information between the **run method** and the **mouse event methods**.

```
/* Instance variable for the square to be tracked */
private GRect square;

public void run() {
     square = makeSquare();
     add(square);
}

public void mouseMoved(MouseEvent e) {
     double x = e.getX() - SQUARE_SIZE / 2.0;
     double y = e.getY() - SQUARE_SIZE / 2.0;
     square.setLocation(x, y);
}
```

The Importance of Style

- It is considered extremely poor style to use instance variables unnecessarily:

Do not use instance variables where local variables, parameters, and return values suffice.

The Importance of Style

- It is considered extremely poor style to use instance variables unnecessarily:

  Do not use instance variables where local variables, parameters, and return values suffice.

- Use *local variables* for temporary information.

The Importance of Style

- It is considered extremely poor style to use instance variables unnecessarily:

Do not use instance variables where local variables, parameters, and return values suffice.

- Use *local variables* for temporary information.
- Use *parameters* to communicate data into a method.

The Importance of Style

- It is considered extremely poor style to use instance variables unnecessarily:

Do not use instance variables where local variables, parameters, and return values suffice.

- Use *local variables* for temporary information.
- Use *parameters* to communicate data into a method.
- Use *return values* to communicate data out of a method.

Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Putting it all together

Whack-a-Mole

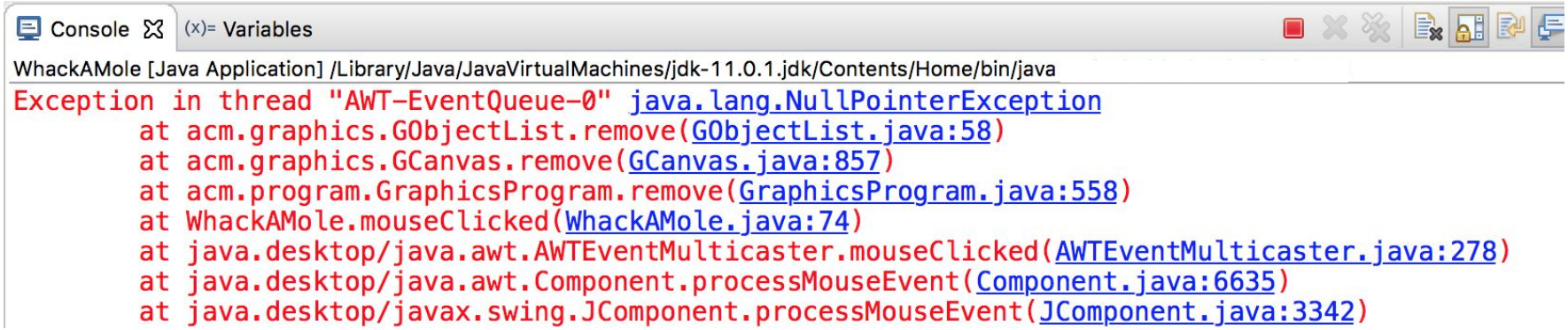
Let's use instance variables and events to make Whack-A-Mole!

- A mole should appear every second at a random location, and stop once the user has gotten at least 10 points.
- If the user clicks a mole, remove it and increase their score by 1
- There should be a GLabel in the left corner showing their score

Let’s Code It!

Exception

- If the user clicks an area with no mole, the program crashes
  - A program crash in Java is called an **exception**
  - When you get an exception, Eclipse shows red error text
  - The error text shows the line number where the error occurred
  - Why did this error happen?
  - How can we avoid it?



Plan for Today

- Review: Animation & Randomness
- getElementAt & Null
- Event-driven Programming
- Instance Variables
- Whack-a-Mole

Next Time: Tracing & Memory