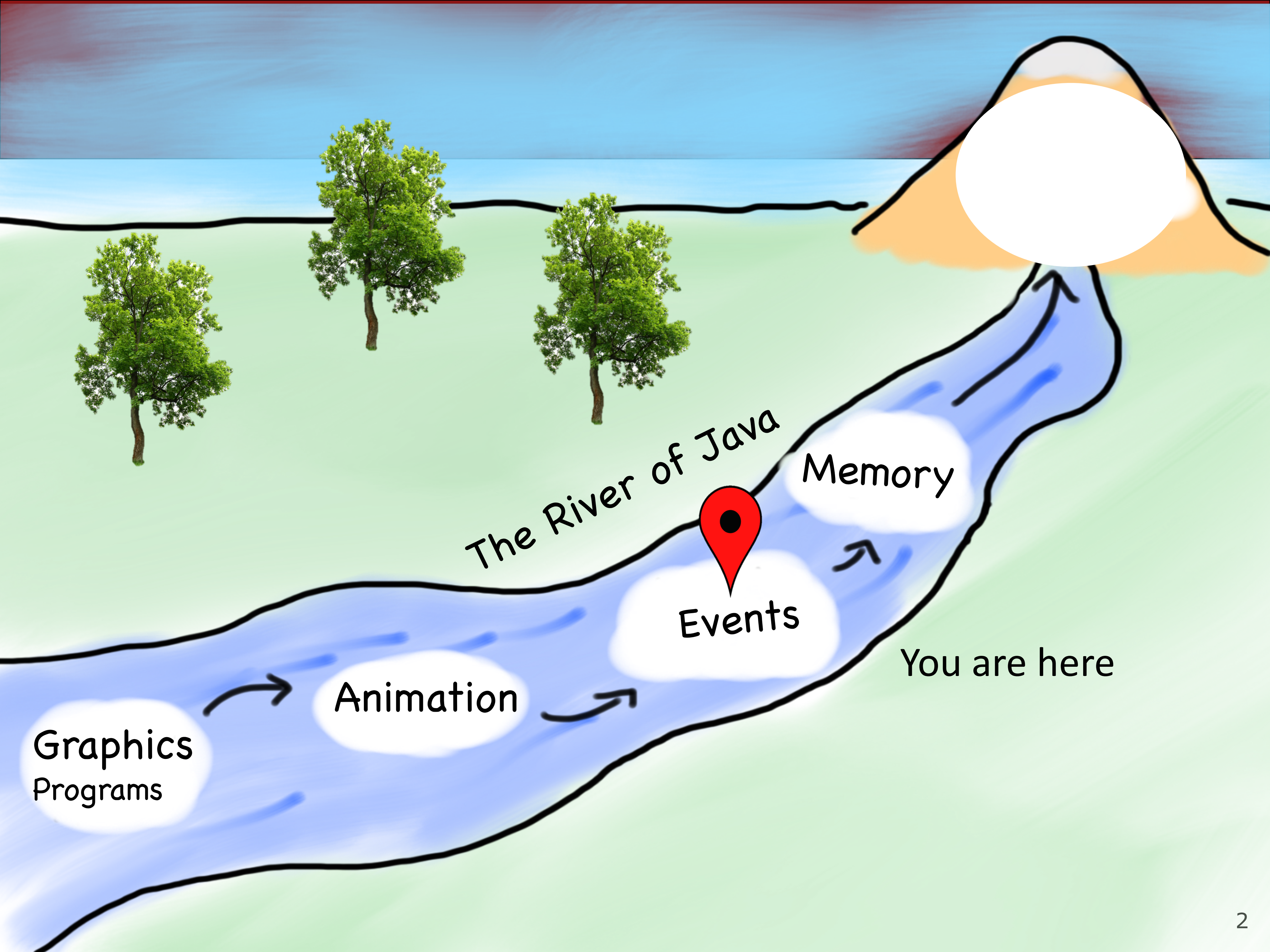
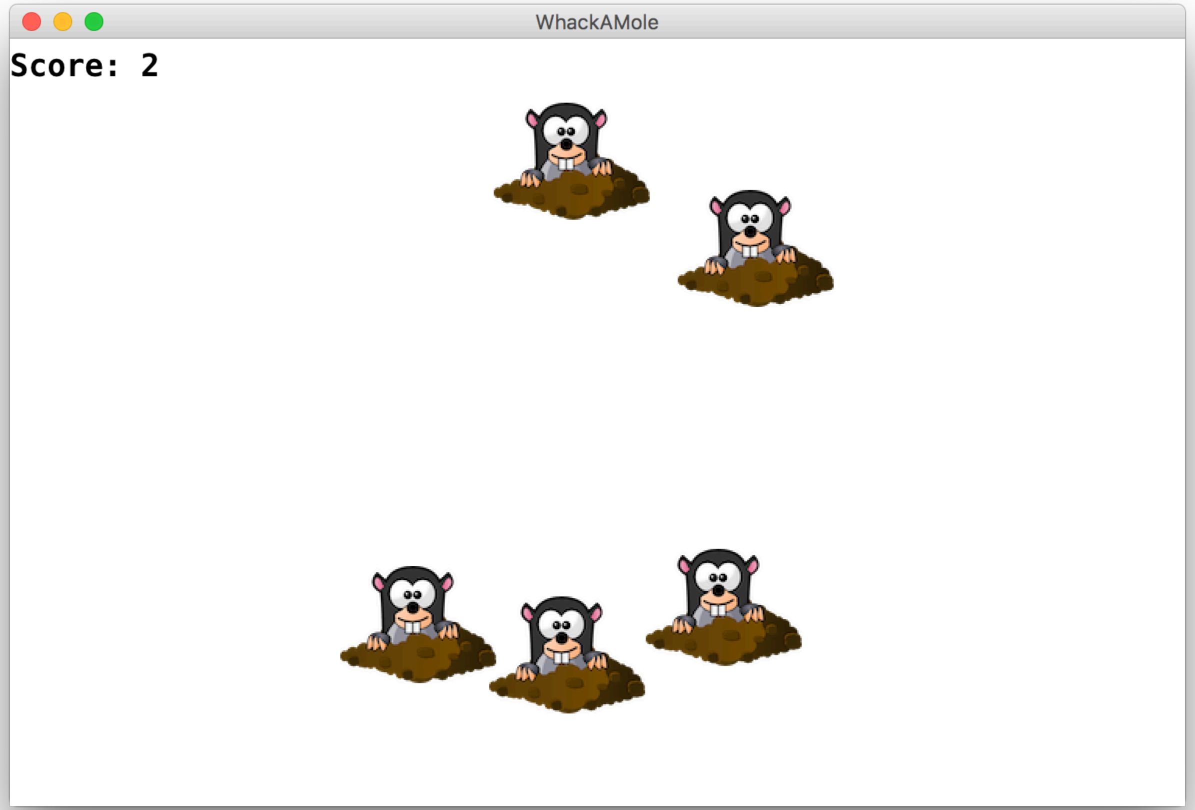

The River of Java
Graphics
Programs
Animation
Events
Memory
You are here

Learning Goals

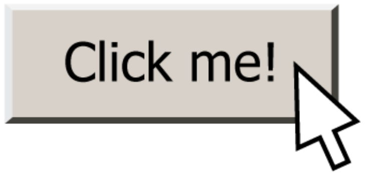
- Learn to respond to mouse events in **`GraphicsPrograms`**
- Learn to use *instance variables* to store information outside of methods



```
Score: 2
```

Events

- **event**: Some external stimulus that your program can respond to.



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

Events

- Program launches

Events

- Program launches
- Mouse motion
- Mouse clicking
- Keyboard keys pressed
- Device rotated
- Device moved
- GPS location changed
- and more…

Events

- Program launches
- Mouse motion
- Mouse clicking
- Keyboard keys pressed
- Device rotated
- Device moved
- GPS location changed
- and more…

Events

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

Events

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

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

Events

```java
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
}
```

Example: ClickForDaisy

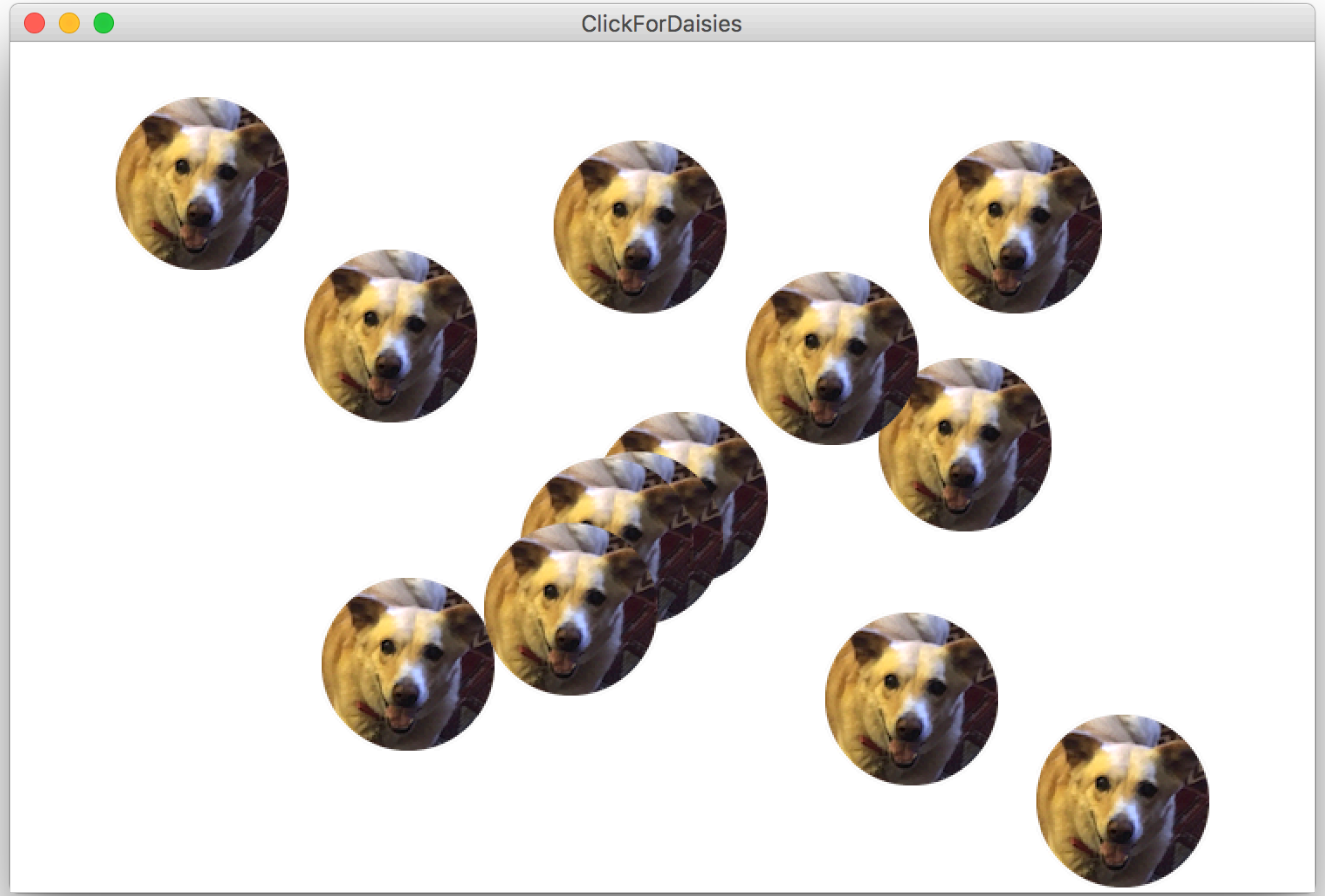
```java
import acm.program.*;
import acm.graphics.*;
import java.awt.*;
import java.awt.event.*;      // NEW

public class ClickForDaisy extends GraphicsProgram {

    // Add a Daisy image at 50, 50 on mouse click
    public void mouseClicked(MouseEvent event) {
       GImage daisy = new GImage("res/daisy.png", 50, 50);
       add(daisy);
    }
}
```

MouseEvent Objects

- A `MouseEvent` contains information about the event that just occurred:

Method	**Description**
`e.getX()`	the *x*-coordinate of mouse cursor in the window
`e.getY()`	the *y*-coordinate of mouse cursor in the window

Example: ClickForDaisies



Example: ClickForDaisies

```
public class ClickForDaisies extends GraphicsProgram {

   // Add a Daisy image where the user clicks
   public void mouseClicked(MouseEvent event) {
     // Get information about the event
     double mouseX = event.getX();
     double mouseY = event.getY();

     // Add Daisy at the mouse location
     GImage daisy = new GImage("res/daisy.png", mouseX, mouseY);
     add(daisy);
   }
}
```

Example: ClickForDaisies

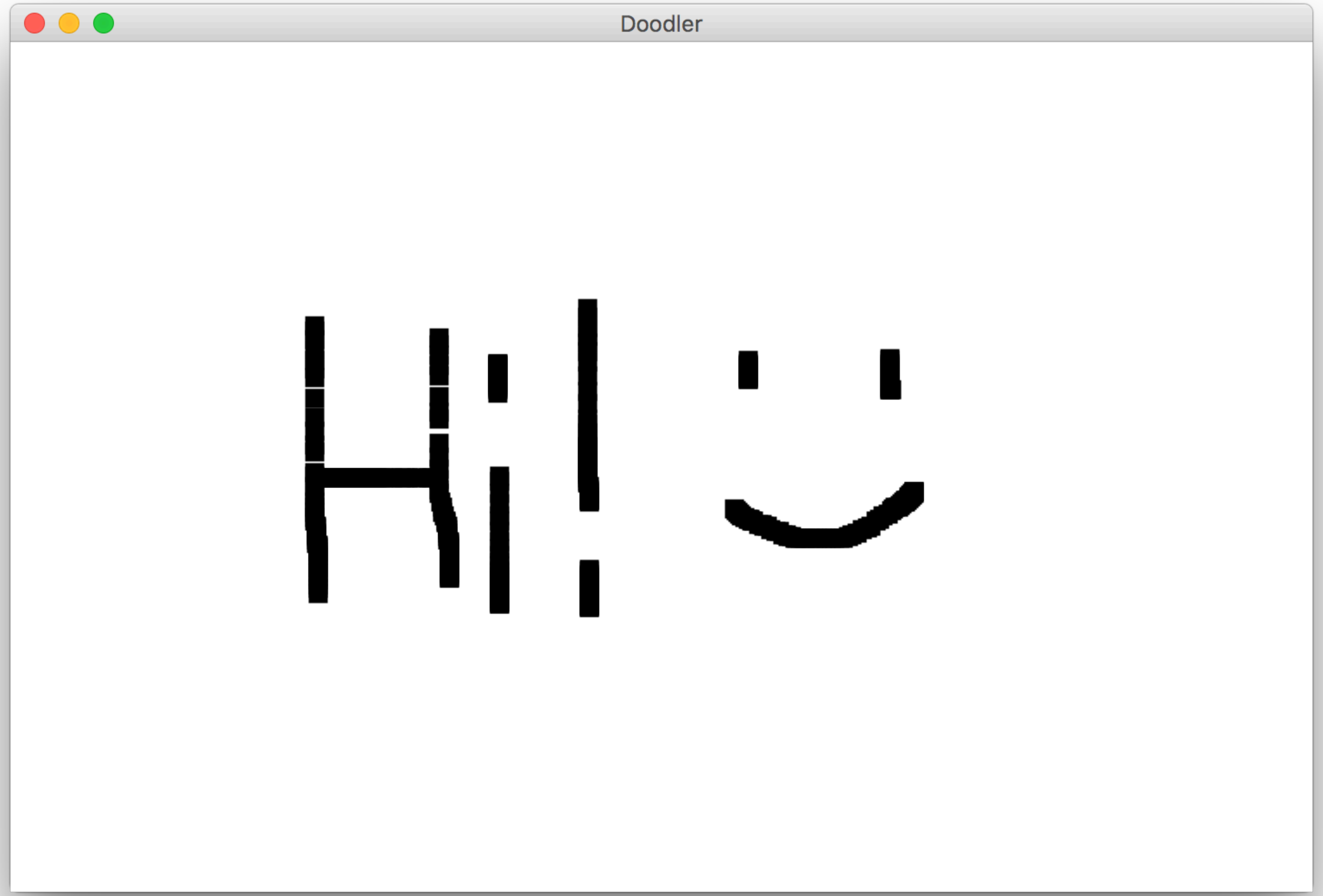
```
public class ClickForDaisies extends GraphicsProgram {

   // Add a Daisy image where the user clicks
   public void mouseClicked(MouseEvent event) {
      // Get information about the event
      double mouseX = event.getX();
      double mouseY = event.getY();

      // Add Daisy at the mouse location
      GImage daisy = new GImage("res/daisy.png", mouseX, mouseY);
      add(daisy);
   }
}
```

Types of Mouse Events

- There are many different types of mouse events.
  - Each takes the form:
    `public void` ***`eventMethodName`***`(MouseEvent event) { ...`

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



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);
    add(rect);
}
```

Doodler

```java
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);
    add(rect);
}
```

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);
    add(rect);
}
```

Doodler

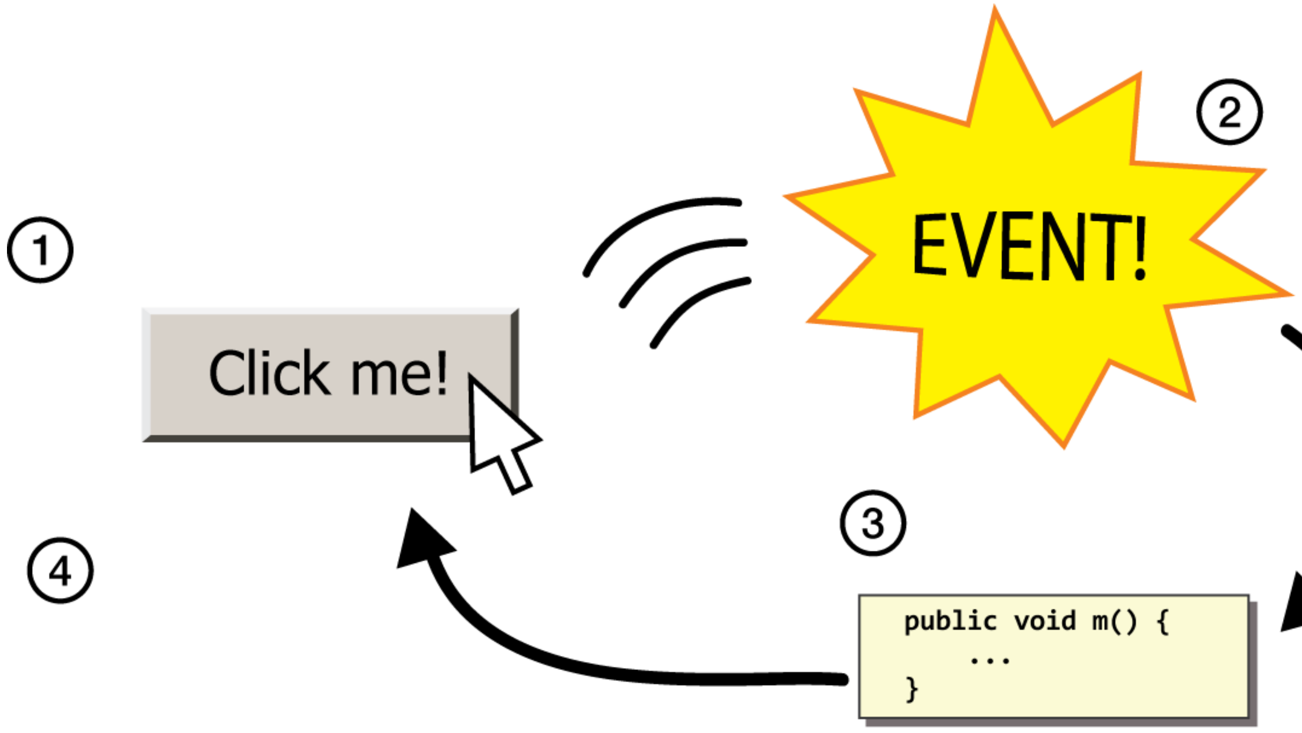
```
public void mouseDragged(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    double rectX = mouseX - SIZE / 2.0;
    double rectY = mouseY - SIZE / 2.0;
    GRect rect = new GRect(rectX, rectY, SIZE, SIZE);
    rect.setFilled(true);
    add(rect);
}
```

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);
    add(rect);
}
```

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 that event.
4) The method's code updates the screen appearance in some way.



```
public void m() {
    ...
}
```

Revisiting Doodler

```java
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);
    add(rect);
}
```

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