# CS 106A, Lecture 13
# More Events; Animation

reading:

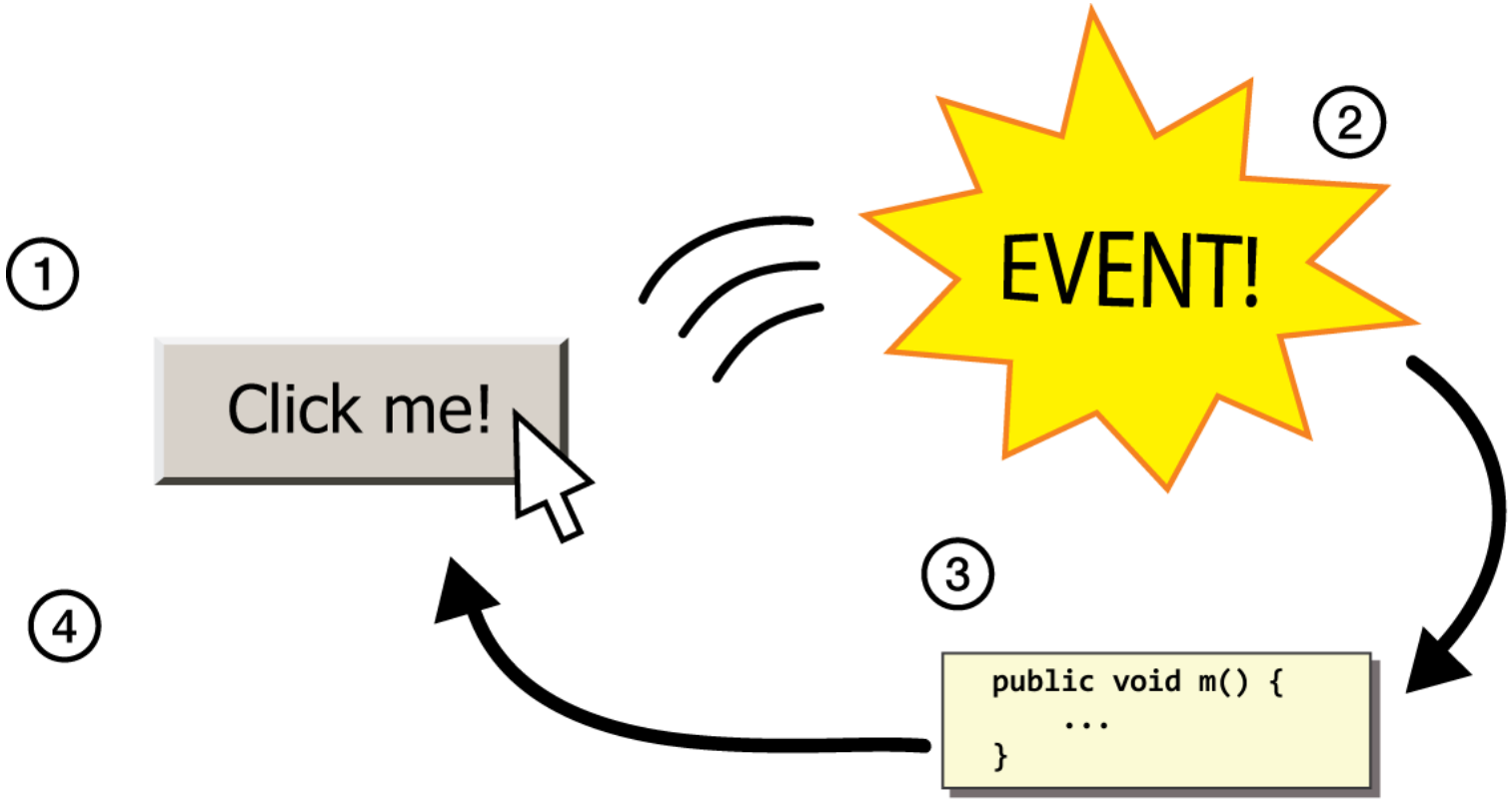
*Art & Science of Java*, 10.1 - 10.3;  6.3 - 6.4

# Whack-a-mole exercise

- Write a **`WhackAMole`** graphical program where images of a mole are randomly added to the screen every 1 second.
  - Place each mole at a randomly chosen x/y position.
  - If the user clicks on a mole image, remove it from the screen.

# The event cycle

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.



# Mouse events

```
import acm.program.*;
import acm.graphics.*;   // Stanford graphical objects
import java.awt.*;       // for Java graphical objects
import java.awt.event.*; // for Java events

public class Name extends GraphicsProgram {
    public void run() {
        addMouseListeners();   // enable mouse events
        statements;
    }

    // the code to run when the event occurs
    public void eventMethodName(MouseEvent event) {
        ...
    }
}
```

# Event methods

- You can write any of the following methods in your program.
  - 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 |

# Event objects

- The `MouseEvent` passed to your event method contains useful information about the event that just occurred:

| Method | Description |
|---|---|
| ***e*.getButton()** | which mouse button was pressed, if any |
| ***e*.getX()** | the *x*-coordinate of mouse cursor in the window |
| ***e*.getY()** | the *y*-coordinate of mouse cursor in the window |

```
// create a small circle wherever the user clicks
public void mousePressed(MouseEvent event) {
    int x = event.getX();
    int y = event.getY();
    println("You clicked x=" + x + ", y=" + y);
}
```

# getElementAt

- The `GraphicsProgram` contains a method **`getElementAt`** that accepts x/y parameters and returns the graphical object found at that position in the window.
  - If multiple objects are there, it returns the "top" one in the Z-order.
  - It could return any type of graphic object.

```
GRect obj = getElementAt(x, y);
...
```

# getElementAt details

- If you aren't sure what type of shape might be returned, specify your variable as type `GObject`, which can be any graphic object.

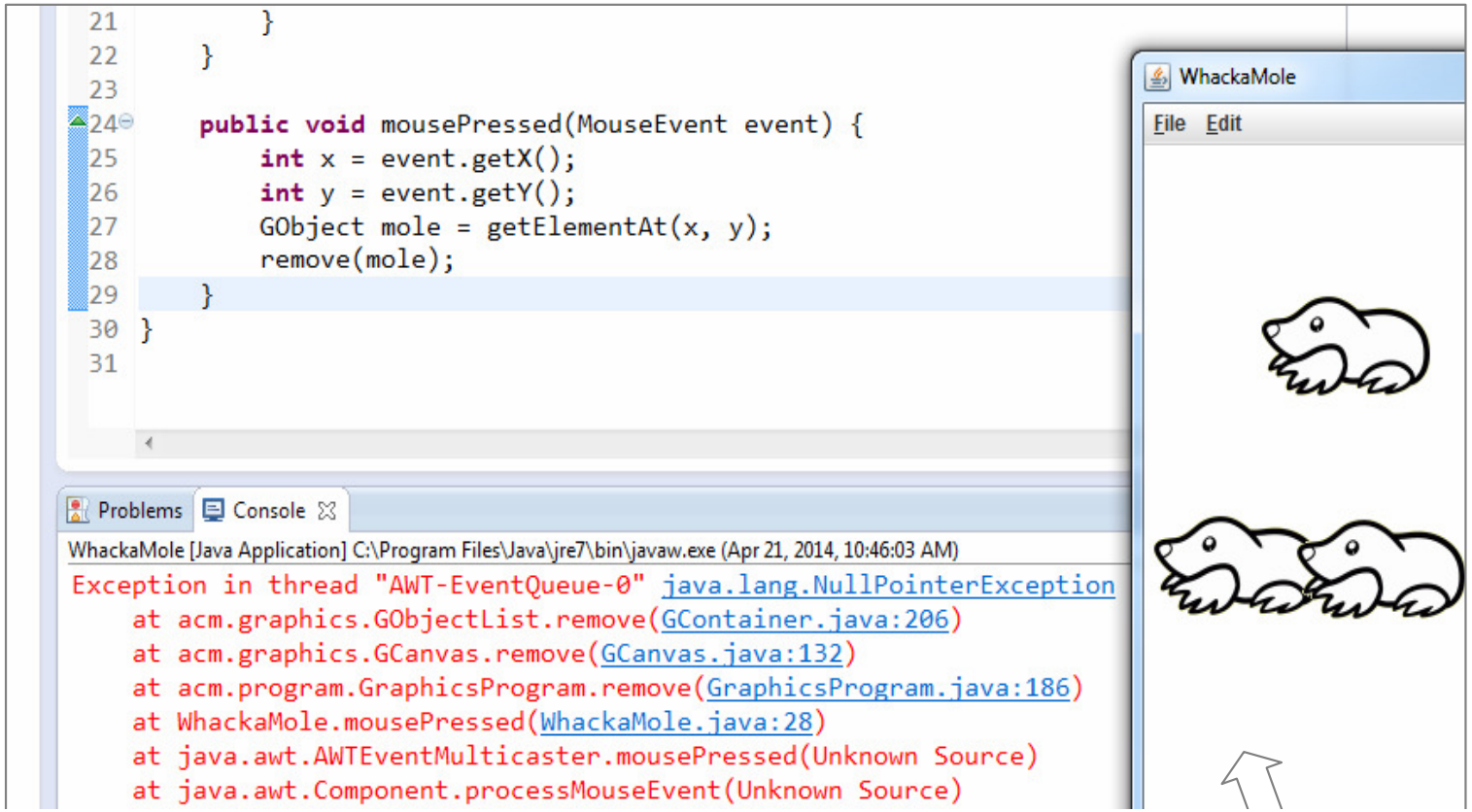
```
// could return any type of graphical object
GObject obj = getElementAt(x, y);
...
```

- To search multiple x/y locations, pass additional coordinates.

```
// search (x1,y1), then (x2,y2), then (x3,y3), ...
GObject obj = getElementAt(x1, y1, x2, y2, x3, y3);
```

# 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.



# Null

- **`null`**: A special constant value meaning, "no object."
  - `getElementAt` returns `null` if no object is at that position.
  - You can check for `null` using the == and != operators.

```
GObject mole = getElementAt(x, y);
if (mole != null) {
    remove(mole);
}

// alternate syntax: use hasElementAt to avoid null
if (hasElementAt(x, y)) {
    GObject mole = getElementAt(x, y);
    remove(mole);
}
```

# Animation

# Simple animation

- A Graphics program can be made to animate with a loop such as:

```
public void run() {
    ...
    while (test) {
        update the position of shapes;
        pause(milliseconds);
    }

}
```

- The best number of ms to pause depends on the program.
  - most video games ~= 50 frames/sec = 25ms pause

# Graphical methods

- These methods in graphical objects can be useful for animation:

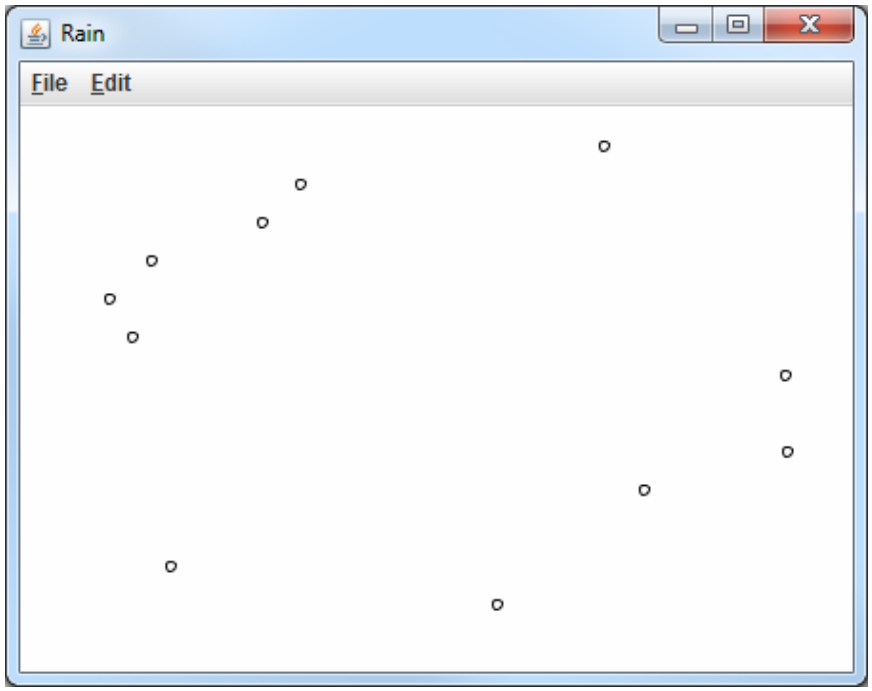
| Method | Description |
|---|---|
| `obj.getX()` | the left *x*-coordinate of the shape |
| `obj.getY()` | the top *y*-coordinate of the shape |
| `obj.getWidth()` | number of pixels wide the shape is |
| `obj.getHeight()` | number of pixels tall the shape is |
| `obj.move(dx, dy);` | adjusts location by the given amount |
| `obj.setLocation(x, y);` | change the object's x/y position |
| `obj.setSize(w, h);` | change the object's width*height size |

- The `GraphicsProgram` itself has these methods, too:

| | |
|---|---|
| `getWidth()` | number of pixels wide the window is |
| `getHeight()` | number of pixels tall the window is |
| `setSize(w, h)` | change the window's width*height size |

# Raindrop animation

- Write a graphics program **`Rain`** that simulates "rainfall".
  - Every half-second, a new raindrop (5x5 black oval) is added to the top of the window at a random horizontal location.
  - The raindrops fall down the screen at a rate of 2px every 50ms.
  - *Variation:* Make it also add a new raindrop wherever the user clicks.



# Process all shapes

- You can loop over all the shapes in your graphical program:

```
// a "for each" loop over all shapes on screen
for (GObject obj : this) {
    do something with obj;
}
```

  - This is a loop where each shape is returned one at a time and temporarily called `obj`. You can move or modify that shape.

Limitations/drawbacks:

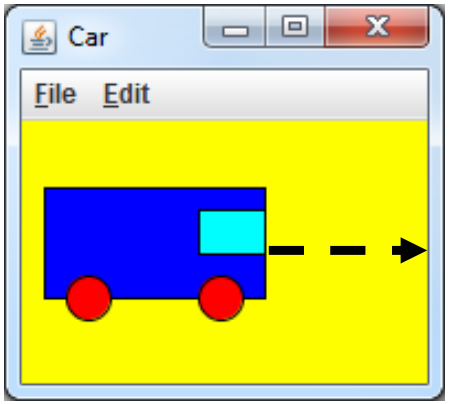
- Cannot **remove** a shape from the screen inside the loop (crashes!)
  - But can move the shape, change its color, etc.
- Returns *all* shapes; not possible to **exclude** particular ones.

# Raindrop solution

```
import acm.graphics.*;
import acm.program.*;
import acm.util.*;

public class Rain extends GraphicsProgram {
    public void run() {
        RandomGenerator rg = RandomGenerator.getInstance();
        int time = 0;
        while (true) {
            // make all the raindrops fall by 2px
            for (GObject drop : this) {
                drop.move(0, 2);
            }

            if (time % 500 == 0) {   // add a new raindrop
                int rainX = rg.nextInt(0, getWidth() - 5);
                add(new GOval(rainX, 0, 5, 5));
            }
            pause(50);   // sleep for animation
            time += 50;
        }
    }
}
```

# Animated car

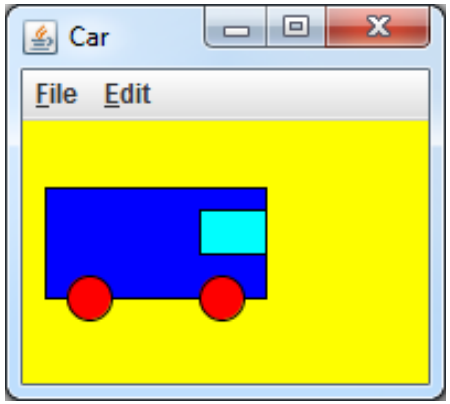
- Write a variation of the **Car** program where the car drives forward with acceleration until it hits the "wall" (edge of the window).
  - Can you make the car turn around when it hits the right wall?
  - Can you make the car "accelerate" as it drives further?



# GCompound

- Sometimes it is useful to create a *compound shape* that represents several shapes grouped into one.
  - Makes it easy to perform a bulk operation on all shapes together.

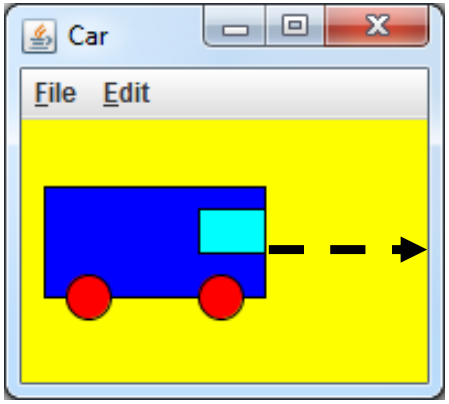
```
GCompound compound = new GCompound();
compound.add(shape);
compound.add(shape);
...
compound.add(shape);

add(compound);
```




  - Make a GCompound to represent a car.

# Animated car crash

- Modify our program so that the car drives forward with acceleration until it hits the "wall" (edge of the window).
  - Once it hits either edge, it should turn around and move in the other direction.
  - Can you make the car "accelerate" as it drives further?



# Animated solution

```
public class CarAnimated extends GraphicsProgram {
    public void run() {
        setBackground(Color.YELLOW);
        GCompound car = new GCompound();

        GRect body = new GRect(10, 30, 100, 50);
        body.setFilled(true);
        body.setFillColor(Color.BLUE);
        car.add(body);

        GOval wheel1 = new GOval(20, 70, 20, 20);
        wheel1.setFilled(true);
        wheel1.setFillColor(Color.RED);
        car.add(wheel1);

        GOval wheel2 = new GOval(80, 70, 20, 20);
        wheel2.setFilled(true);
        wheel2.setFillColor(Color.RED);
        car.add(wheel2);

        GRect windshield = new GRect(80, 40, 30, 20);
        windshield.setFilled(true);
        windshield.setFillColor(Color.CYAN);
        car.add(windshield);
        add(car);
        ...
```

# Animated solution, cont'd.

```
        ...

        // animation loop
        double velocityX = 0.0;
        double acceleration = 0.01;
        while (true) {
            car.move(velocityX, 0);
            velocityX += acceleration;

            if ((car.getX() + car.getWidth() >= getWidth()
                    && velocityX > 0) ||
                    (car.getX() <= 0 && velocityX < 0)) {
                velocityX = -velocityX;
                acceleration = -acceleration;
            }
            pause(10);
        }
    }
}
```