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

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

        // pause
        pause(milliseconds);
    }
}
```java
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
The method:

```java
GObject getElementAt(double x, double y);
```
returns which object is at the given location on the canvas.
getElementAt

- The method:

  ```java
  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:
  
  ```java
  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.
/*
  * 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;
}
/*  
* 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;  
}
/*
 * 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;
}
null is a special variable value that objects can have that means “nothing”. Primitives cannot be 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 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;`)

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

G0val circle; // initially null
You can check if something is null using `==` and `!=`

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

```c
// 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!
}
```
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.
Common events include:

- *Mouse motion / clicking.*
- Keyboard buttons pressed.
- Timers expiring.
- Network data available.
public void run() {
    // Java runs this when program launches
}

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

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

```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
}
```
public void mouseClicked(MouseEvent e) {
}

public void mouseClicked(MouseEvent e) {

}
public void mouseClicked(MouseEvent e) {

}
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) {

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

Get information about the event
Anatomy of a Mouse Method

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

    }

}
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:

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouseMoved</td>
<td>mouse cursor moves</td>
</tr>
<tr>
<td>mouseDragged</td>
<td>mouse cursor moves while button is held down</td>
</tr>
<tr>
<td>mousePressed</td>
<td>mouse button is pressed down</td>
</tr>
<tr>
<td>mouseReleased</td>
<td>mouse button is lifted up</td>
</tr>
<tr>
<td>mouseClicked</td>
<td>mouse button is pressed and then released</td>
</tr>
<tr>
<td>mouseEntered</td>
<td>mouse cursor enters your program's window</td>
</tr>
<tr>
<td>mouseExited</td>
<td>mouse cursor leaves your program's window</td>
</tr>
</tbody>
</table>
Example: Doodler

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

private static final int \texttt{SIZE} = 10;
...

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

```java
public void mouseClicked(...) {
    ...
}
```
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

```java
public void mouseClicked(...) {
    ...
}
```
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

```java
public void mouseClicked(...) {
    ...
}
```
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);
}
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
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

```java
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?

```java
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.
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.

```java
private type name; // declared outside any method!
```
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*.

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

private GRect square;

public void run() {
    square = new GRect(...);
    GRect localSquare = new GRect(...);
}
```
Example: MouseTracker
Use instance variables if you need to pass information between the run method and the mouse event methods.

```java
/* 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

Score: 2
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?

```console
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
  at acm.graphics.GObjectList.remove(GObjectList.java:58)
  at acm.graphics.GCanvas.remove(GCanvas.java:857)
  at acm.program.GraphicsProgram.remove(GraphicsProgram.java:558)
  at WhackAMole.mouseClicked(WhackAMole.java:74)
  at java.desktop/java.awt.AWTEventMulticaster.mouseClicked(AWTEventMulticaster.java:278)
  at java.desktop/java.awt.Component.processMouseEvent(Component.java:6635)
  at java.desktop/javafx.swing.JComponent.processMouseEvent(JComponent.java:3342)
```
Plan for Today

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

Next Time: Tracing & Memory