Announcements

- Honor Code Reminder
Honor Code

- Do not look at assignment solutions that are not your own.
  - Online or another student’s.
- Do not share solutions with other students.
- If you discuss strategies (NOT SOLUTIONS), you must indicate assistance you have received.
  - You do not need to do this if the person is a CS106 staff member.
- You can only reuse work in certain, limited situations.
- If you have questions about honor code, please ask the instructors.
Plan for Today

- Review: Null, Events, Instance Variables
- Pass by Reference vs. Pass by Value
- Types of Errors
- Eclipse Debugger
- Practice!
It is possible to determine what, if anything, is at the canvas at a particular point.

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.
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
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
}
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`
● An **event** is some external stimulus that your program can respond to.

● **Common events include:**
  ○ Mouse motion / clicking.
  ○ Keyboard buttons pressed.
  ○ Timers expiring.
  ○ Network data available.
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.
Review: Types of Mouse Events

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

```java
public void eventName(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>
public void mouseClicked(MouseEvent e) {

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.getX()</td>
<td>the x-coordinate of mouse cursor in the window</td>
</tr>
<tr>
<td>e.getY()</td>
<td>the y-coordinate of mouse cursor in the window</td>
</tr>
</tbody>
</table>
Review: 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(...) {
    ...
}
```
Review: 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!

private GRect square = null;

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

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

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

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);
}
```
Review: 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.
Speaking of Parameters...

Pass by Reference vs. Pass by Value
Pass by Reference vs Value

**Pass by Reference**

Objects are passed by reference.

A few examples:
- GRect
- GOval
- GImage

**Pass by Value**

Primitives are passed by value.

A few examples:
- int
- char
- double
- boolean
Pass by Reference vs Value

What does this mean?
Let’s Look at a Program!
What does this mean?

If something is passed by reference, it can be altered simply by passing it into a method.

If something is passed by value, it cannot be altered simply by passing it into a method.
What Happened?
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}

What Happened: Primitive
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}
What Happened: Primitive

```java
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);

    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}
```

![Diagram](26)
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    
    private void changeInt(int primitiveInt){
        primitiveInt += 10;
    }
}

These are not the same variable.
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}

These are not the same variable.
These are two variables with the same name.
What Happened: Primitive

public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: "+ primitiveInt, 0, 50);
    add(intLabel);
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}

These are not the same variable. These are two variables with the same name. The `primitiveInt` inside of changeInt copied the value of the `primitiveInt` inside of run.
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}

These are not the same variable. These are two variables with the same name. The `primitiveInt` inside of `changeInt` copied the value of the `primitiveInt` inside of `run`. They are stored in different locations in the computer’s memory. One just copied the value of the other variable.
public void run()
{
    int primitiveInt = 0;
    changeInt(primitiveInt);

    primitiveInt += 10;
}

private void changeInt(int primitiveInt)
{
    primitiveInt += 10;
}
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: "+primitiveInt, 0, 50);
    add(intLabel);
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    primitiveInt += 10;
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
}

private void changeInt(int primitiveInt){
    primitiveInt += 10;
}
What Happened: Primitive

```java
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}
```
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}
public void run(){
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);
    ...
}

primitiveInt: 0
public void run()
{
    int primitiveInt = 0;
    changeInt(primitiveInt);
    GLabel intLabel = new GLabel("primitiveInt: " + primitiveInt, 0, 50);
    add(intLabel);

    ...
}

primitiveInt: 0
What Happened: Object

```java
public void run(){
    //...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
```

We’ll hide these for now!
public void run(){
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
public void run() {
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
public void run(){
    ...

    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
public void run() {
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
public void run(){
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

voidExample

primitiveInt: 0

run
changeRect

objectRect
objectRect
What Happened: Object

```java
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.BLUE);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
```
public void run(){
private void changeRect(GRect objectRect){
    objectRect.setFilled(false);
    objectRect.setColor(Color.GREEN);
}
add(objectRect, 100, 100);
changeRect(objectRect);
}
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

Unlike a primitive variable, it wasn’t the value that was copied here.
```java
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
```

Unlike a primitive variable, it wasn’t the value that was copied here.

Our `changeRect objectRect` variable **kept track of the location in memory** where our `run objectRect` was stored.
What Happened: Object

```java
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
```

Basically, whenever we change our `changeRect` `objectRect` variable, we are actually changing our `run` `objectRect` variable.

The way we've programmed this, they are connected together.
What Happened: Object

```java
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}
```
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
public void run(){
    private void changeRect(GRect objectRect){
        objectRect.setFilled(false);
        objectRect.setColor(Color.GREEN);
    }
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
public void run(){
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
public void run(){
    ...
    GRect objectRect = new GRect(100, 100);
    objectRect.setFilled(true);
    objectRect.setColor(Color.BLUE);
    add(objectRect, 100, 100);
    changeRect(objectRect);
}

primitiveInt: 0
Pass by Reference vs Value

What does this mean?

If something is passed by *reference*, it *can* be altered simply by passing it into a method.

If something is passed by *value*, it *cannot* be altered simply by passing it into a method.
Types of Errors
Types of Errors

Syntax Errors:

A programming “typo”. Usually causes a red squiggly line in code.
Types of Errors

Syntax Errors:
A programming “typo”. Usually causes a red squiggly line in code.

Execution Errors:
Something that crashes the program after you run it.
Types of Errors

Syntax Errors:

A programming “typo”. Usually causes a red squiggly line in code.

Execution Errors:

Something that crashes the program after you run it.

Logic Errors:

All of your code runs, but it produces unexpected results.
How Can We Debug Errors?
How Can We Debug Errors?

Using the Eclipse Debugger!
How Can We Debug Errors?

Could we have debugged our previous program using the Eclipse Debugger?
Debugger Commands

**Suspend.** Stops the program immediately as if it had hit a breakpoint.

**Terminate.** Exits from the program entirely.

**Step Into.** Executes one statement of the program and then stops again. If that statement includes a method call, the program will stop at the first line of that method. As noted below, this option is not as useful as **Step Over**.

**Step Over.** Executes one statement of the program at this level and then stops again. Any method calls in the statement are executed through to completion unless they contain explicit breakpoints.

**Step Return.** Continues to execute this method until it returns, after which the debugger stops in the caller (the method that called the current method).

*The debugger allows you to step through the execution of a program, one line at a time.*
I can view what my variables are currently equal to by opening the “Variables” View.

How?
- Click **Window**
- Click **Show View**
- Click **Other**…
- Search for **Variables**
- Click **Variables**
Setting a Breakpoint

To start the debugger, it helps to set a breakpoint. Double click next to the line number where you want to set a breakpoint. This will create a little blue dot, or **breakpoint**.

Your code will stop and start the debugger when it sees the breakpoint.
Let’s Debug Our Program!
Ummm, I have a problem. Can you help?
I wrote my whole program without testing it and now I’m stuck!
Let’s Help Duke!
Plan for Today

- Review: Null, Events, Instance Variables
- Pass by Reference vs. Pass by Value
- Types of Errors
- Eclipse Debugger
- Practice!