CS 106A, Lecture 15
Events and Memory
The River of Java

- Memory
- Events
- Animation
- Graphics

You are here

HW4: Breakout
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
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.
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
}
There are many different types of mouse events.

- Each takes the form:

  ```java
  public void eventName(MouseEvent event) { ... }
  ```

<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>
MouseEvent Objects

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>e.getX()</code></td>
<td>the x-coordinate of mouse cursor in the window</td>
</tr>
<tr>
<td><code>e.getY()</code></td>
<td>the y-coordinate of mouse cursor in the window</td>
</tr>
</tbody>
</table>
Example: 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);
}
• **Instance variable**: A variable that lives outside of any method.
  – The *scope* of an instance variable is throughout an entire file (class).
  – Useful for data that must persist throughout the program, or that cannot be stored as local variables or parameters (event handlers).

  – *It is bad style to overuse instance variables*
We used instance variables and events to make Whack-A-Mole!
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 this?
public void mouseClicked(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    GObject mole = getElementAt(mouseX, mouseY);

    remove(mole);
    score++;
    scoreLabel.setText("Score: " + score);
}
public void mouseClicked(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    GObject mole = getElementAt(mouseX, mouseY);
    remove(mole);
    score++;
    scoreLabel.setText("Score: " + score);
}

Problem: **mole** may be null if the user doesn’t click on a mole! Removing null will crash 😞
public void mouseClicked(MouseEvent event) {
    double mouseX = event.getX();
    double mouseY = event.getY();
    GObject mole = getElementAt(mouseX, mouseY);

    if (mole != null) {
        remove(mole);
        score++;
        scoreLabel.setText("Score: " + score);
    }
}
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
A Boolean Aside

There is one helpful property of how Java evaluates boolean expressions, called short-circuit evaluation.

When evaluating boolean expressions, Java only evaluates as much of the expression as it needs to in order to evaluate it to be true or false.
String str = readLine("? ");

if (str.length() > 0 && str.charAt(0) == 'A') {
    ...
}

String str =.readLine("? "); // what about ""!

if (str.length() > 0 && str.charAt(0) == 'A') {
    ...
}

String str = readLine("? "); // what about ""!

if (str.length() > 0 && str.charAt(0) == 'A') {
    ...
}

A Boolean Aside

```java
String str = readLine("? "); // what about ""!
if (str.length() > 0 && str.charAt(0) == 'A') {
    ...
}
```

&& : true if both parts are true

Java only executes the second part if the first part is **true**!
This means it never crashes.
A Boolean Aside

GObject obj = getElementAt(x, y);

if (obj == null || obj.getX() == 0) {
    ...
}
A Boolean Aside

GObject obj = getElementAt(x, y);  // what about null!

if (obj == null || obj.getX() == 0) {
    ...
}

A Boolean Aside

GObject obj = getElementAt(x, y); // what about null!

if (obj == null || obj.getX() == 0) {
    ...
}
GObject obj = getElementAt(x, y); // what about null!

if (obj == null || obj.getX() == 0) {
    ...
}

|| : true if either side is true

Java only executes second part if the first part is false!
This means it never crashes.
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
A Variable love story

By Chris Piech
A Variable story

origin

Nick Troccoli

By Chris Piech
Chapter 1: Birth
Once upon a time...
...a variable x was born!

```c
int x;
```
...a variable x was born!

```c
int x;
```
x was a primitive variable...

```java
int x;
```

Aww...!

It’s so cuuuute!
...and its parents loved it very much.

```
int x;
```

We should give it... value 27!
...and its parents loved it very much.

\[ x = 27; \]

We should give it... value 27!
A few years later, the parents decided to have another variable.
...and a variable rect was born!

GRect rect;
rect was an object variable...

GRect rect;

Who’s a cute GRect???

It’s so square!
...and its parents loved it very much.

GRect rect;

We should make it... a big, strong GRect!
...and its parents loved it very much.

GRect rect = new GRect(0, 0, 50, 50);

We should make it... a big, strong GRect!
...but rect’s box was not big enough for an object!

GRect rect = new GRect(0, 0, 50, 50);

That box isn’t big enough to store everything about a GRect!
...so they stored the information in a bigger box somewhere else.

```java
GRect rect = new GRect(0, 0, 50, 50);
```

<table>
<thead>
<tr>
<th>x = 0, y = 0</th>
<th>width = 50</th>
<th>height = 50</th>
</tr>
</thead>
</table>

See location 5
Chapter 2: Friends
...x makes a new friend!

```
int y = 27;
```
x makes a new friend!

int y = 27;

Hi! I’m y.
...x makes a new friend!

```c
#include <stdio.h>

int main() {
    int y = 27;  // We have the same value!
    return 0;
}
```
...x makes a new friend!

```c
int y = 27;
```

*blush*
They can use `==` to compare values.

```java
if (x == y) {
    // true!
}
```

27 27
They can use `==` to compare values.

```java
if (x == y) { // true!
...
}
```
See “A Variable Love Story” for more...
rect also makes a new friend!

GRect r = new GRect(0, 0, 50, 50);
Rect also makes a new friend!

GRect r = new GRect(0, 0, 50, 50);
GRect r = new GRect(0, 0, 50, 50);

It looks like we have the same coordinates, size, everything!
Rect also makes a new friend!

GRect r = new GRect(0, 0, 50, 50);
But when they use `==` to compare values...
...something goes wrong.

```java
if (rect == r) { // false!
...
}
```

See location 5

See location 24
...something goes wrong.

```java
if (rect == r) { // false!
...
}
```

...but... but I thought we had so much in common!
...something goes wrong.

```java
if (rect == r) { // false!
...
}
```

See location 5

See location 24
...something goes wrong.

```java
if (rect == r) { // false!
    ...
}
```

You see, `==` compares what is in each variable's box.

- `rect` (See location 5)
- `r` (See location 24)
...something goes wrong.

```java
if (rect == r) { // false!
...
}
```

Primitives store their *actual value* in their box.

- `rect` is a reference variable and its *box* contains the actual value of the object it references.
- `r` is a primitive variable and its *box* contains the actual value of the primitive it holds.

See location 5

See location 24
...something goes wrong.

```cpp
if (rect == r) { // false!
    ...
}
```

But objects store the location where all their information lives.
...something goes wrong.

```java
if (rect == r) { // false!
...
}
```

This means `==` on objects compares their locations, which is not what we want here!
Chapter 3: Twins
One day, they wanted a twin.

Wow, what an awesome int! Let’s make another one.
One day, they wanted a twin.

Wow, what an awesome int! Let’s make another one.

```c
int x2 = x;
```
...so x2 was born!

Wow, what an awesome int! Let’s make another one.

```c
int x2 = x;
```
...so x2 was born!

But let’s increment this one by 2.

```c
int x2 = x;
x2 += 2;
```
int x2 = x;
x2 += 2;

...so x2 was born!

Cool, I’m 29 now!
...so x2 was born!

int x2 = x;
x2 += 2;

And I’m still 27!
Then, they wanted a twin for rect, too.

Wow, what an awesome GRect! Let’s make another one.
...so rect2 was born!

Wow, what an awesome GRect! Let’s make another one.

GRect rect2 = rect;
...so rect2 was born!

Wow, what an awesome GRect! Let’s make another one.

GRect rect2 = rect;
GRect rect2 = rect;
rect2.setColor(Color.BLUE);

...so rect2 was born!

But let’s make this one BLUE.
...so they went to location 5 and changed the color to blue.

x = 0, y = 0
width = 50
height = 50
Color = BLUE
...
GRect rect2 = rect;
rect2.setColor(Color.BLUE);
GRect rect; rect2 = rect;
rect2.setColor(Color.BLUE);

But something went wrong...

Hey, why am I blue TOO?
But something went wrong…

You see, when you set one variable equal to another, the value in its box is copied over.

GRect rect2 = rect;

rect2.setColor(Color.BLUE);
But something went wrong…

For primitive variables, this just means we copy their value.

GRect rect2 = rect;
rect2.setColor(Color.BLUE);
But something went wrong...

But for objects, we copy its location instead.

```java
GRect rect2 = rect;
rect2.setColor(Color.BLUE);
```
But something went wrong...

Therefore, both rect and rect2 think their information is at location 5.

```
GRect rect2 = rect;
rect2.setColor(Color.BLUE);
```

![Diagram showing rect and rect2 each at location 5](image)
But something went wrong…

When you change information for an object, you go to its location first, and then update the information.

GRect rect2 = rect;
rect2.setColor(Color.BLUE);
Since `rect` and `rect2` reference the same location, when you change information for one of them, it changes for both!

```java
GRect rect2 = rect;
rect2.setColor(Color.BLUE);
```
Chapter 4: Leaving the Nest
X grew up, and went to college.

```plaintext
college(x);
```

Bye everyone! I’m going to college.
X grew up, and went to college.

college(x);

We’ll miss you, honey! Don’t forget to call! Don’t stay up too late! Watch out for rogue bicyclists!

Bye everyone! I’m going to college.
X grew up, and went to college.

```java
private void college(int num) {
    ...
}
```

```
x 27
num 27
```
x had an amazing college experience...

```java
private void college(int num) {
    num = 45;
}
```
...but ultimately returned home the same.

```java
private void college(int num) {
    num = 45;
}
```

<table>
<thead>
<tr>
<th>run</th>
<th>college</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>num</td>
</tr>
<tr>
<td>x</td>
<td>45</td>
</tr>
</tbody>
</table>
...but ultimately returned home the same.

college(x);
println(x);

I’m baaaaack! And I’m still 27.
...but ultimately returned home the same.

college(x);
println(x);

Ohhhh honey! You’ll always be our little 27.

I’m baaaaack! And I’m still 27.
rect grew up too, and also went to college.

Bye everyone! I’m going to college.
rect grew up too, and also went to college.

Not you too, rect! We are empty nesters now...

Bye everyone! I’m going to college.
rect grew up too, and also went to college.

private void college2(GRect box) {
    ...
}

run

rect

See location 5

box

See location 5
private void college2(GRect box) {
    box.setColor(Color.PINK);
}

rect also had an amazing college experience...

run

college2

rect

box

See location 5

See location 5
rect also had an amazing college experience...
private void college2(GRect box) {
    box.setColor(Color.PINK);
}

...but it returned home different.

I'm pink!
...but it returned home different.

private void college2(GRect box) {
    box.setColor(Color.PINK);
}

Hey, what gives? I’m pink now too!

I’m pink!
...but it returned home different.

college(rect);
add(rect);

I’m baaaaack! But now I’m pink...
...but it returned home different.

college(rect);
add(rect);

Oh my gosh...what happened to you??!

I’m baaaaack! But now I’m pink...
college(rect);
add(rect);

You see, when a variable is passed as a parameter, the value *inside its box* is copied and given to the method.
...but it returned home different.

college(rect);
add(rect);

For primitives, we make a copy of their *actual value*. Therefore, changes to a parameter do not affect the original.
…but it returned home different.

college(rect);
add(rect);

However, for objects we make a copy of their location. So changes to the parameter do affect the original!
The End
Primitives vs. Objects

- **Primitives** store their *actual value* in their variable box. You can compare values with `==` and `!=`, and the original does not change when passed as a parameter and changed.

- **Objects** store their *location* in their variable box. You can’t compare properties of an object via `==`, and the original *does* change when passed as a parameter and changed.

- **Primitives** are *passed by value*, **Objects** are *passed by reference*
Practice — Chapter 2: Friends
public void run() {
    int x = 5;
    int y = 5;
    println(x == y);
}

Does this print out true or false?
Does this print out true or false?

first.equals(second) would be true
Practice — Chapter 3: Twins
public void run() {
    int x = 5;
    int y = x;
    y += 2;
    println(y);
    println(x);
}

What is printed out?
public void run() {
    GOval oval = new GOval(0, 0, 50, 50);
    oval.setColor(Color.PINK);
    GOval oval2 = oval1;
    oval2.setColor(Color.BLUE);
    add(oval2);
    add(oval);
}

What colors are oval and oval2?
Practice – Chapter 4: Leaving the Nest
public void run() {
    int x = 2;
    addFive(x);
    println(x);
}

private void addFive(int num) {
    num += 5;
}

What is printed out?
public void run() {
    GRect rect = new GRect(0, 0, 50, 50);
    rect.setColor(Color.PINK);
    makeBlue(rect);
    add(rect);
}

private void makeBlue(GRect box) {
    box.setColor(Color.BLUE);
}

What color is rect?
Primitives vs. Objects

Primitives store their actual values. Objects store their locations.

This affects how your code behaves!
**getElementAt** returns a *reference* to an object.

```java
GRect rect = new GRect(0, 0, 50, 50);
add(rect);
...
GObject obj = getElementAt(25, 25);
println(obj == rect);
```

This prints **true**!

Heap

1024

x = 0
y = 0
width = 50
height = 50

Stack

rect
See location 1024

obj
See location 1024
int x = 27;  // In Java, new means create something in heap memory!
int y = x;
GRect rect = new GRect(0, 0, 50, 50);
GRect box = rect;

See location 1024

Strings are weird! Their variables store addresses to a special section of the heap. Detailed explanation on Piazza :)
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
Let’s revisit our Whack-A-Mole program to add some new functionality using what we just learned.
Plan for Today

• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips
Midterm

• Study strategies, in order of exam relevance
  – Sleep well for at least two days beforehand!
  – Do the practice exams under timed conditions
  – (Re-)do section problems from all handouts
  – Code up programs we wrote in lecture
  – Review lecture slides
  – Do CodeStepByStep practice problems
  – Read the textbook
• Strategies for taking the exam
  – Read over all the problems first
    • Seriously, I always have students who run out of time because they got stuck on one problem
  – If you don’t know exactly how to do something, write something approximate
    • If your syntax isn’t too far off, we’ll allow it
    • More importantly, it gets you unstuck so you can keep writing correct parts of the solution
• Methods that *might* be helpful
  - `Integer.parseInt`: converts String to int
    • Example: `int x = Integer.parseInt("27");`
  - `useDelimiter`: tells a Scanner to split on a specified String rather than on whitespace
    • Example: `String line = "a,b,c";`  
      • `Scanner tokens = new Scanner(line);`
      • `tokens.useDelimiter("","");`
      • `println(tokens.next());` // prints "a"
• Review: events and instance variables
• A Boolean Aside
• Memory
• Revisiting Whack-A-Mole
• Midterm tips

Good luck on the midterm!