Breakout YEAH hours

Brahm Capoor
Road Map

- Lecture Review
- Using the debugger
- Assignment Overview
- Q&A!
Primitive variables

```java
int x = 7;  // declare and initialize a variable
x = 9;     // change the value of x
x = x + 1; // increment (add 1 to) x. A.K.A. x++
x = x + 2; // add 2 to x. A.K.A. x += 2
x /= 2     // divide x by 2, and truncate result

double d = 3.5;

boolean isThisTrue = true;
isThisTrue = !isThisTrue; // flip isThisTrue
```
Graphics

GRect rect = new GRect(50, 50, 200, 200);
rect.setFilled(true);
rect.setColor(Color.BLUE);

GOval oval = new GOval(0, 0, getWidth(), getHeight());
oval.setFilled(false);
oval.setColor(Color.GREEN);

GLabel text = new GLabel("banter", 200, 10);

add(text);
add(rect);
add(oval);

Things to remember

● Coordinates are doubles
● Coordinates are measured from the top left of the screen
● Coordinates of a shape are coordinates of its top left corner
● Coordinates of a label are coordinates of its bottom left corner
● Remember to add objects to the screen!
● Use the online documentation!
● These are class variables!
Variable scope

---

Variables live inside the block in which they’re declared

```java
for (int i = 0; i < 5; i++) {
    int y = i * 4;
}
i = 3;  // Error!
y = 2;   // Error!

... // in some code far, far away
int y = 0;
for (int i = 0; i < 5; i++) {
    y = i * 4;
}
y = 2; // Ayy!
```
Methods & parameters
private returnType methodName(type parameter1, type parameter2,...)

private int returnsInt() {...}
private void drawsRect(int width, int length) {...} //void is no type
public boolean frontIsClear() {...} //look familiar?

Parameters and a return value are both optional!
public void run() {
    println("Choose 2 numbers!");
    int n1 = readInt("Enter n1"); //5
    int n2 = readInt("Enter n2"); //7

    int total = addNumbers(n1, n2);
    println ("The total is " + total);
}

private int addNumbers(int num1, int num2) {
    int sum = num1 + num2;  //12
    return sum;
}

Example: Methods and Parameters

- run() is a public method that runs the program.
- addNumbers is a private method that adds two numbers.

GET n1 AND n2

run() → addNumbers(n1, n2) → total = 12 → PRINT RESULT

num1 = 5, num2 = 7

addNumbers() → sum = 12
private int multipleReturns(int x) {
    if (x == 5) {
        return 0;
    }
    return 1; // this only happens if x != 5
    return 5; // never gets to this line
}

// note: every path through the method ends with a single return statement

// note: a function ends immediately after it returns
Mouse Movement

We’re not specifying what our program should do, we’re specifying how it should react.

The question we’re answering:

“When the mouse does something interesting, how should our program respond”?

When your mouse does that interesting thing, your program pauses.
Mouse Movement

Step 1: Figure out the important mouse events you need to deal with

- mouseMoved
- mouseClicked
- mouseDragged
- mousePressed
- mouseReleased
Anatomy of a Mouse Method

Public so other programs can call it

```java
public void mouseMoved(MouseEvent e) {
}
```
Anatomy of a Mouse Method

```java
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more sick code here
}
```

Doesn’t return anything
Anatomy of a Mouse Method

It must have one of the mouse event names

```java
public void mouseMoved(MouseEvent e) {
}
```
Anatomy of a Mouse Method

A collection of information about the Mouse Event

```java
public void mouseMoved(MouseEvent e) {

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

Anatomy of a Mouse Method

Get information about the event
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more sick code here
}
An annoying nuance

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 sick code here
}
```
An annoying nuance

You don’t call this method, so you can’t specify its parameters.

So how can we give `mouseMoved` access to our other variables?

```java
public void mouseMoved(MouseEvent e) {
    double mouseX = e.getX();
    double mouseY = e.getY();
    // more sick code here
}
```
Instance variables

private int x; // belongs to the instance of the program

public void run() {
    x = 2;
    addTwo();
    println(x); // prints 4
}

private void addTwo() {
    x += 2;
}

Should you use an instance variable?

YES
- You access & change the variable everywhere
- You use it in mouseListener methods
- You have literally no other choice

NO
- It makes information flow more annoying to visualize (parameters are easier)
- Poor style to build up unnecessary instance variables

The opposite of an instance variable is a local variable
Breakout!
Due Wednesday, February 6th
Tips and tricks:

● Is it easier to conclude that a number is prime or that it isn’t?

● What do you need to guarantee that a number is prime?

Prime Checker

(A sandcastle)
Mouse Reporter
(Another sandcastle)

Tips and tricks:
- The starter code stores the label as an instance variable
- `getElementAt` might be useful here!
Breakout
(The actual assignment)

(What we’re making!)
What you’re given: Constants

- Use `getWidth()` and `getHeight()` for dimensions of window, not the ones in the constants!
- You might need to add more instance variables...
What you’re given: **starter code**

```java
public void run() {
    // Set the window's title bar text
    setTitle("CS 106A Breakout");

    // Set the canvas size. Remember to ALWAYS use getWidth()
    // and getHeight() to get the screen dimensions, not constants!
    setCanvasSize(CANVAS_WIDTH, CANVAS_HEIGHT);

    /* You fill this in, along with any subsidiary methods */
}
```
MILESTONE 1: BRICKS

- Similar to pyramid!
- Drawing multiple rows:
  - Figure out how to draw one row first
  - Bricks should be centered horizontally
- Reasonable coloring for any number of rows
MILESTONE 2: PADDLE

- How do you make the mouse control the paddle?
- Chapter 9: GObject Methods
- Chapter 10: Event Driven Programs (responding to mouse events)
- Things to consider:
  - Paddle only needs to move in the x direction
  - Paddle can’t move off the screen
MILESTONE 3: PLAY BALL!

- How do we move the ball?
- How do you choose the direction of the ball?
- What information do we need in the G Oval constructor?
MILESTONE 3: PLAY BALL!

/* Animation: */
while(condition) {
    // update graphics
    obj.move(5, 5);
    pause(DELAY);
}

MILESTONE 3: **PLAY BALL!**

/* Moving the ball: */
double vx;
double vy;
...
while (condition) {
  // update graphics
  ball.move(vx, vy);
  pause(DELAY);
}
MILESTONE 3: **PLAY BALL!**

/* Randomizing the ball’s initial velocity: */
// make a random generator instance variable
private RandomGenerator rgen = RandomGenerator.getInstance();

// give the ball an initial direction
vx = rgen.nextDouble(1.0, 3.0); // choose speed
if(rgen.nextBoolean(0.5)) vx = -vx; // choose left or right

// wait until player clicks the screen
waitForClick();
MILESTONE 4: COLLISIONS

- Handle bouncing off walls first
- Collisions with objects: check if there's anything at each of the 4 corners and return one GObject
- Useful method:

```
GObject getElementAt(double x, double y);
```
MILESTONE 4: COLLISIONS

/* Handling collisions: */
private GObject getCollidingObject() {
    // sick code
    // return a GObject
}
...
GObject coll = getCollidingObject();
// bounce vertically if collider is brick or paddle
// also need to handle collisions with walls--separate logic!
Ending the game

- Remove the ball when it goes off the screen
  - remove(ball);
- Determine wins and losses by the count of bricks
Tragedy strikes: the sticky paddle 😞
Testing the program

- Check if it deals with changed constants
- Win condition / loss condition
- Try mega paddle
- Try sticky paddle
Wrapping up

- Read the spec (seriously, read the whole thing)!
- Comment your code!
- Incorporate IG feedback!
- Asking for help
- Extensions
Questions?