Breakout YEAH hours

Brahm Capoor & Jared Wolens

Road Map

— — —

YEAH hour schedule

Deadline: Due Wednesday, February 8th

Lecture Review

Using the debugger

Assignment Overview

Q&A!

YEAH hours this quarter

Assignment	Hours
3: Breakout	Right here, right now!
4: Hangman	Thursday February 9th, 7:30 - 9:30 P.M.
5: Yahtzee	Tuesday February 21st, 7:30 - 9:30 P.M.
6: NameSurfer	Wednesday March 1st, 7:30 - 9:30 P.M.
7: FacePamphlet	Thursday March 9th, 7:30 - 9:30 P.M.

All YEAH sessions are in Bishop Auditorium

Methods and parameters

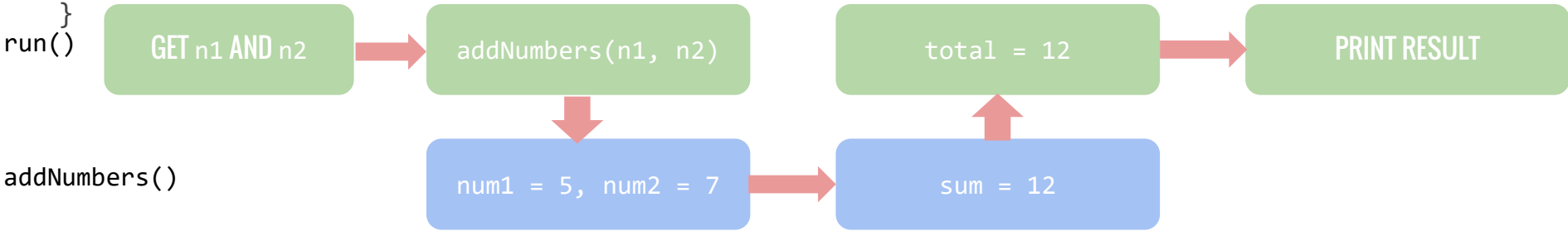

```
private returnType methodName(parameter1, parameter2,...)

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!

An example

```
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);
}
```

```
                                        5          7
private int addNumbers(int num1, int num2) {
        int sum = num1 + num2;//12
        return sum;
}
```



Variable scope

Variables live inside the block in which they're declared

```
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!
```

```
SCOPE A
for(..) {
        SCOPE B
}
SCOPE A

public void run() {
        SCOPE A
}

private void function() {
        SCOPE B
}
```

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
- 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

Multiple returns

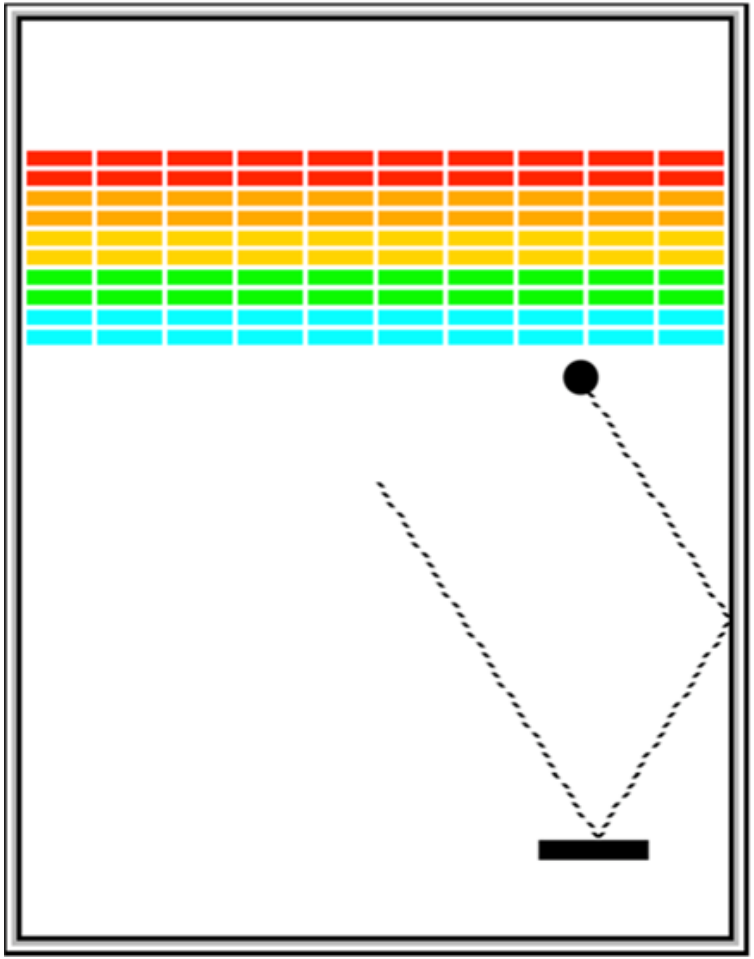
```
private int multipleReturns(int x) {

            if (x == 5) {
                        return 0;
            }

            return 1; // this only happens if x
!= 5
}
```

```
// note: we can only add return whenever
another return does not collide
```

Live demo: Using the debugger

Breakout!

What we're making



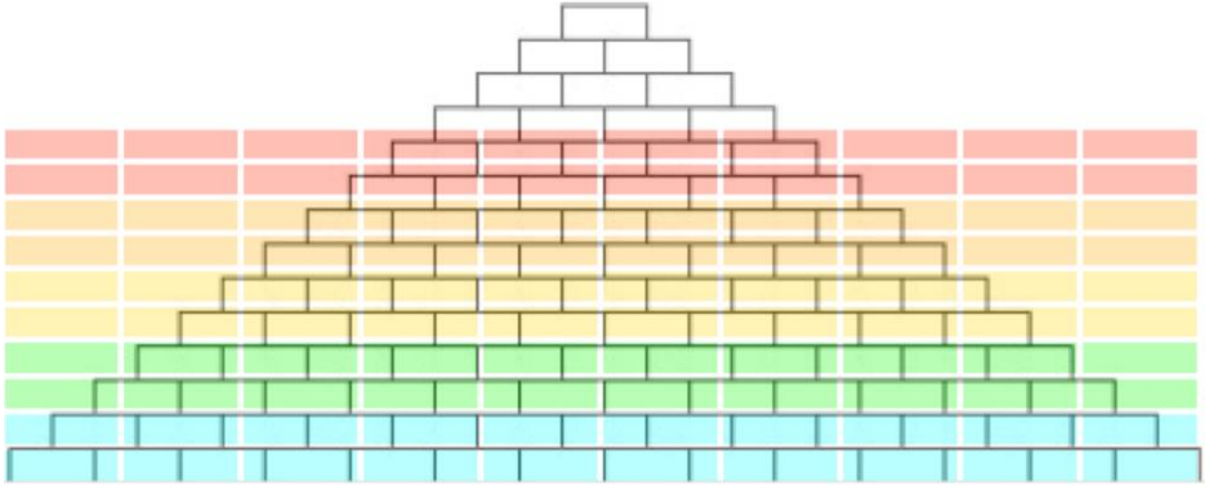
What you're given

- These are constants
- Use `getWidth()` and `getHeight()` for dimensions of window, not the ones in the constants!
- You might need to add more instance variables…

```
/**
 * Width and height of application window, in pixels.
 * These should be used when setting up the initial size of the game,
 * but in later calculations you should use getWidth() and getHeight()
 * rather than these constants for accurate size information.
 */
public static final int APPLICATION_WIDTH = 420;
public static final int APPLICATION_HEIGHT = 600;

/** Dimensions of game board (usually the same), in pixels */
public static final int BOARD_WIDTH = APPLICATION_WIDTH;
public static final int BOARD_HEIGHT = APPLICATION_HEIGHT;

/** Number of bricks in each row */
public static final int NBRICKS_PER_ROW = 10;

/** Number of rows of bricks */
public static final int NBRICK_ROWS = 10;

/** Separation between neighboring bricks, in pixels */
public static final int BRICK_SEP = 4;

/** Width of each brick, in pixels */
public static final double BRICK_WIDTH =
    (BOARD_WIDTH - (NBRICKS_PER_ROW + 1.0) * BRICK_SEP) / NBRICKS_PER_ROW;

/** Height of each brick, in pixels */
public static final int BRICK_HEIGHT = 8;

/** Offset of the top brick row from the top, in pixels */
public static final int BRICK_Y_OFFSET = 70;

/** Dimensions of the paddle */
public static final int PADDLE_WIDTH = 60;
public static final int PADDLE_HEIGHT = 10;

/** Offset of the paddle up from the bottom */
public static final int PADDLE_Y_OFFSET = 30;

/** Radius of the ball in pixels */
public static final int BALL_RADIUS = 10;

/** initial random velocity that you should choose */
public static final double VELOCITY_MIN = 1.0;
public static final double VELOCITY_MAX = 3.0;

/** Animation delay or pause time between ball moves (ms) */
public static final int DELAY = 1000 / 60;

/** Number of turns */
public static final int NTURNS = 3;
```

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

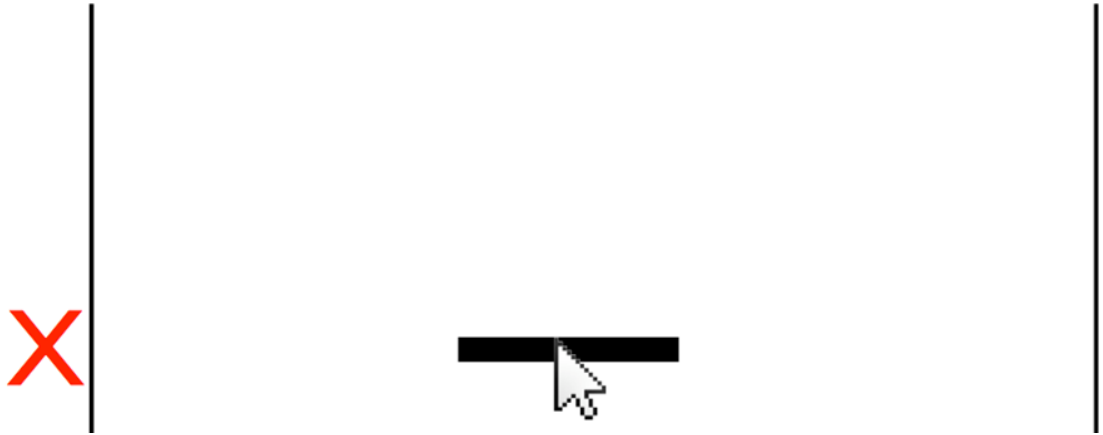
How do you make the mouse control the paddle?

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



A brief aside: Mouse Movement

———

```
addMouseListeners(); // this needs to happen before the program can respond to the mouse!

public void mouseMoved(MouseEvent e) { // remember to make this public!
          double mouseX = e.getX(); // get the x-coordinate of where the mouse moves to
          double mouseY = e.getY(); // get the x-coordinate of where the mouse moves to
          ...
}
```

Things to remember:

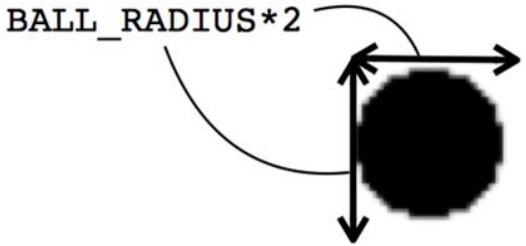
- Other things you can do with the mouse: `mouseClicked(mouseEvent e)`, `mouseDragged(mouseEvent e)`
  - Check the textbook and the online documentation for more!
- `mouseListener`s are called parallel to your code, they happen as soon as you move the mouse
  - as long as you've called `addMouseListeners()` already!

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 GOval constructor?



Animation

```
while(executing condition) {
     // update graphics
     obj.move(dx, dy);
     pause(PAUSE_TIME_MILLISEC);
}
```

Moving the ball

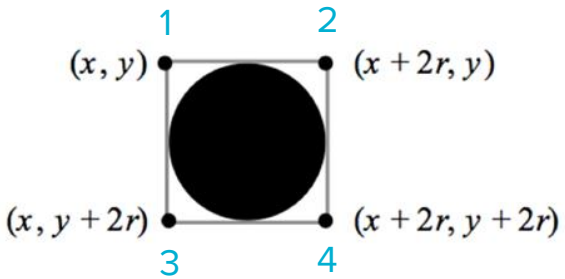
```
double vx;
double vy;
…

while(existing condition) {
     // update graphics
     ball.move(vx, vy);
     pause(PAUSE_TIME_MILLISEC);
}
```

Choosing the direction of the ball

```
//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();
```

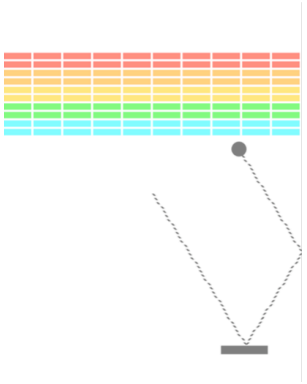
MILESTONE 4: COLLISIONS

Main idea: Check if there's anything at each of the 4 corners and return one GObject



Useful method: `public GObject getElementAt(double x, double y);`

Handling collisions redux



```
private GObject getCollidingObject() {
          // insert impressive code
}
```

…

```
GObject collider = getCollidingObject();
// only need to bounce vertically for collisions with brick, top wall and paddle
// only need to bounce horizontally for collisions with side walls
```

Things to think about: what direction needs to be flipped when?

This is just like the bouncing ball example in lecture!

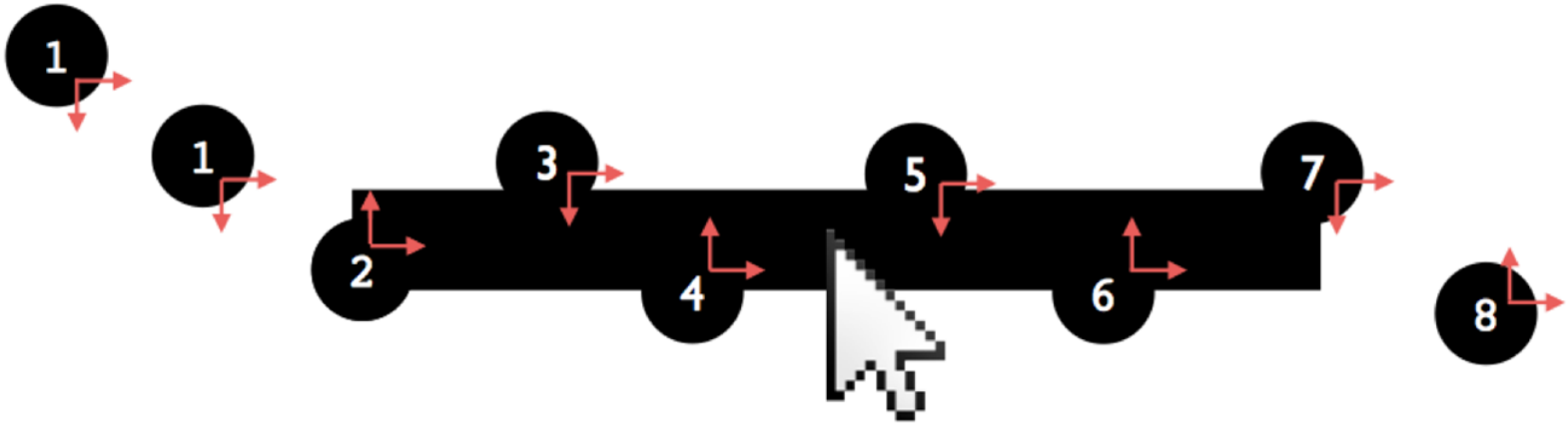
Ending the game

Remove the ball when it goes off the screen

```
remove(obj);
```

Winning and losing

How? Bricks!



The sticky paddle (you saw something similar in lecture)

Testing your program

- Check if it deals with changed constants
- Mega paddle
- Sticky paddle
- Crazy random player

Wrapping up

Read the spec!

Extensions!

Commenting!

Ask for help!

Incorporate IG feedback!