CS 106A Midterm Review

Rishi Bedi, adapted from slides by Kate Rydberg and Nick Troccoli
Summer 2017

Details

- Only the textbook is allowed
  - *The Art and Science of Java*
  - Karel Course Reader
  - You will be provided a reference sheet
- Unless mentioned in the problem you are graded only on functionality
  - Commenting/decomposing not required, but use them to your advantage
  - Naming variables intelligently will also only help you

Major Topics

- Expressions and Variables
- Java Control Statements
- Karel
- Console Programs
- Methods, parameters, and returns
- Randomness
- Strings and chars
- Scanners and file processing
- Graphics Programs
- Memory

Tips

- Common causes of lost points
  - Not understanding concepts
  - Bugs while using concepts
  - Edge cases
- Two kinds of questions: read and write

Tips

- Reading questions
  - Write out everything clearly
  - Pay attention to details

Tips

- Writing questions
  - Plan your code ahead of time!
  - What kinds of variables/loops will you need?
  - Write out steps in pseudocode
  - Can you decompose it to make it easier? (You are allowed to write as many helper methods as you need!)
  - What edge cases might there be?

Where to go for more practice?

- Practice Exam
- Section Problems
- CodeStepByStep (“Practice” link under each lecture)
- The Book
- Review concepts from the assignments

Expressions and Variables

Variables

```
int

double

boolean

char

String
```

+ Variables

```
int count = 0;

double height = 5.2;

boolean readyForMidterm = true;

char letter = ‘a’;

String str = “I love CS106A”;
```

Expressions

- Evaluate the following expressions:

```
3.0 * (23 % 5) / 2 + 2 * 7 / 3                   = 8.5

13 / 2 / 2.0 + 5 / 2.0 / 2                        = 4.25

6 == 3 * 2 && !(7 < 6) && 1 + 1 != 3             = true

2 + 2 + "[" + 2 + 4 * 2 + "]" + 3 + 5            = “4[28]35”
```

+

Java Control Statements

Java Control Statements

- `if`
  - Doing something **once** if a condition is true
- `while`
  - Doing something while a condition is true
- `for`
  - Doing something a given number of times

+ Java Control Statements

For or while?

- Read in user input until you hit the SENTINEL
  - WHILE
- Iterate through a string
  - FOR
- Move Karel to a wall
  - WHILE
- Read in a file line-by-line
  - WHILE

The “Fencepost” Structure

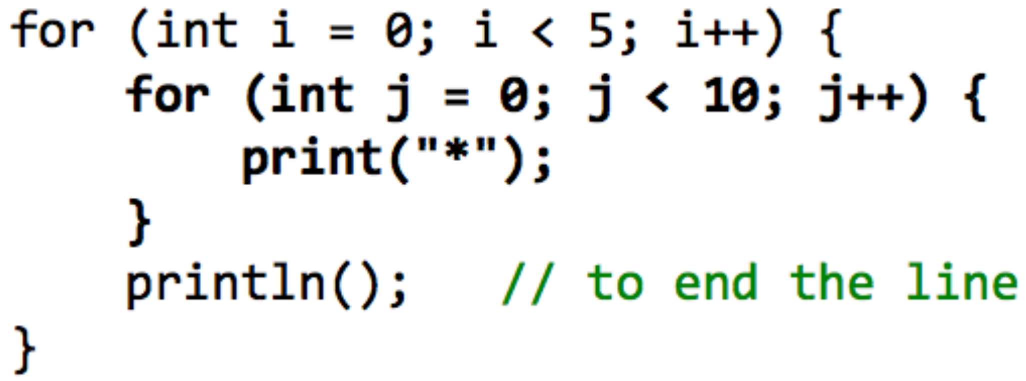
- Loop a set of statements, but do some part of those statements *one additional time*
- Frequently comes up in Karel and user input

```
putBeeper();                // post
while (frontIsClear()) {
    move();                 // fence
    putBeeper();            // post
}
```

```
int sum = 0;
int num = readInt("Enter a number: ");
while (num != -1) {
        sum += num;
        num = readInt("Enter a number: ");
}
println("Sum is " + sum);
```

Nested Loops

- What does this code do?



```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 10; j++) {
        print("*");
    }
    println();   // to end the line
}
```

```
**********
**********
**********
**********
**********
```

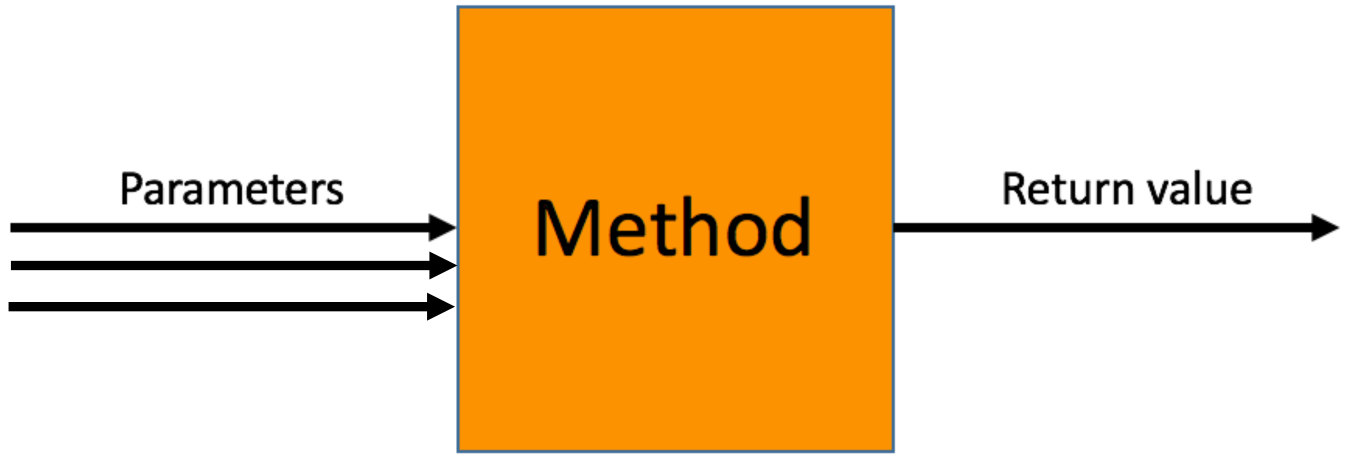
- Inner loop repeats 10 times each time the outer loop repeats

Karel

- Only Karel features!
- Not allowed:
  - Variables (other than “int i” in for loop)
  - parameters/return
  - break

+ Methods, Parameters, and Returns

Methods

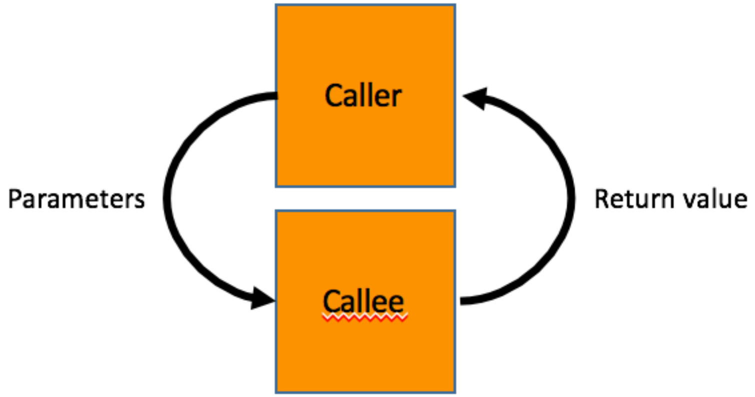


Methods

```
public void run() {
     println("Hypotenuse of 3 and 4 is: ");
     println(hypotenuse(3, 4));
}

private double hypotenuse(double a, double b) {
     return Math.sqrt(a*a + b*b);
}
```

Parameters and Returns

- Parameters are how the caller gives information to the callee
- A return value is how the callee gives information back to the caller



Methods

- Approaching program traces
  - Local variables in the caller are distinct from local variables in the callee
  - Parameters are just assigned names by the order in which they are passed
- Tricky spots
  - Precedence
  - Parameter/variable names
  - What's in scope??
- Draw pictures and label variable values!

Methods : Trace

```
public void run() {
    int a = 1;
    int b = 2;
    int c = 3;
    c = foo(b, a, 5);
    foo(b, c, a);
    println(a + “,” + b + “,” + c);
}

public int foo(int a, int b, int c) {
    b = a + c;
    println(a + “,” + b + “,” + c);
    return a + b * c;
}
```

Randomness

Randomness

- RandomGenerator

```
RandomGenerator rgen = RandomGenerator.getInstance();
```

- Can be used to generate…
  - Integers: `rgen.nextInt(min, max);`
  - Doubles: `rgen.nextDouble(min, max);`
  - Colors: `rgen.nextColor();`
  - Booleans: `rgen.nextBoolean();`

Strings and Chars

Strings (see syntax reference sheet for the rest)

		`String s = "Hello, world!";`

`s.charAt(`**`index`**`)`	Returns the character at the given index	`s.charAt(2);` **`// 'l'`**
		`s.charAt(7);` **`// 'w'`**

`s.substring(`**`start, end`**`)`
`s.substring(`**`start`**`)`
	Returns the part of the string between the given indices	`s.substring(1, 4);` **`// "ell"`**
		`s.substring(7);` **`// "world!"`**

`s.length()`	Returns the length of the string	`s.length();` **`// 13`**

`s1 += s2`
`s1 = s1 + s2`
	Concatenates string s2 to the end of string s1	`s += "!!"` **`// "Hello, world!!!"`**

Strings: Indexing

Substring: remember that that first index is **inclusive** while the second is **exclusive**

```
Hello, world!
0 1 2 3 4 5 6 7 8 9 10 11 12
```

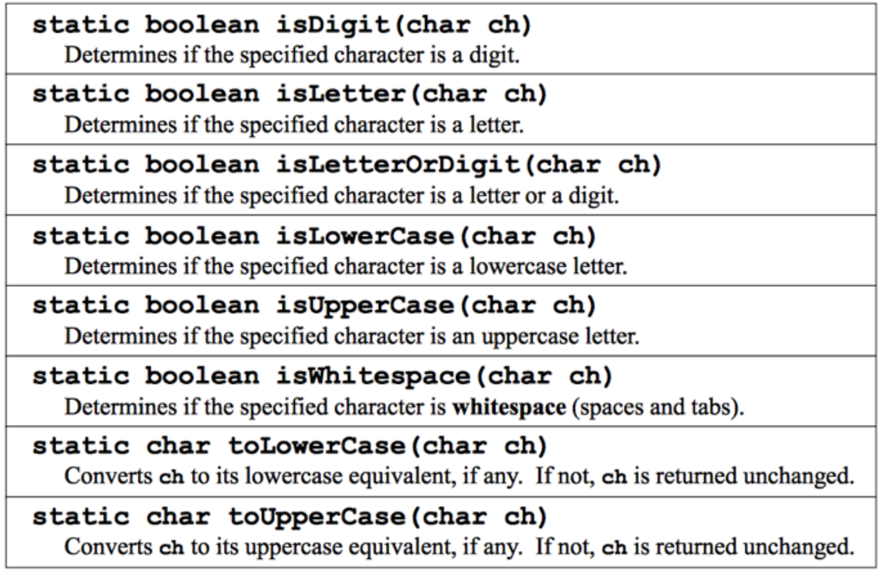
```
s.substring(4, 10) // “o, wor”
```

Strings: Don’t forget that

- We compare strings using `str1.equals(str2)`, NOT `str1 == str2`
- Single quotes are for chars, double quotes are for strings.
- To go from a char to a string, you can concatenate with the empty string: `‘a’ + “” => “a”`
- If a string has N characters, you can index it from 0 to N-1
- Strings are immutable

Characters

Characters



`static boolean isDigit(char ch)`	Determines if the specified character is a digit.
`static boolean isLetter(char ch)`	Determines if the specified character is a letter.
`static boolean isLetterOrDigit(char ch)`	Determines if the specified character is a letter or a digit.
`static boolean isLowerCase(char ch)`	Determines if the specified character is a lowercase letter.
`static boolean isUpperCase(char ch)`	Determines if the specified character is an uppercase letter.
`static boolean isWhitespace(char ch)`	Determines if the specified character is **whitespace** (spaces and tabs).
`static char toLowerCase(char ch)`	Converts **ch** to its lowercase equivalent, if any. If not, **ch** is returned unchanged.
`static char toUpperCase(char ch)`	Converts **ch** to its uppercase equivalent, if any. If not, **ch** is returned unchanged.

Using portions of slides by Eric Roberts

Remember that these methods do not modify the characters that are passed in:

```
char ch = ‘a’;
ch = Character.toUpperCase(ch);
```

Scanners and File Processing

File Reading and Scanners

Use your reference sheet for syntax if unsure!

`sc.next()`	Returns the next token (as separated by a space)
`sc.nextLine()`	Returns the next line

`sc.nextInt()`
`sc.nextDouble()`
	Returns the next int or double

`sc.hasNext()`
`sc.hasNextLine()`
`sc.hasNextInt()`
`sc.hasNextDouble()`
	Returns a **`true`** or **`false`** value indicating whether or not the scanner has any more of the given token lined up

+ Strings and Scanners Practice

- Let's write a method that, given a string, returns its acronym
  - "Throw Back Thursday" -> T.B.T.
  - "All Day I Dream About Soccer" -> A.D.I.D.A.S.
  - "Come Late And Start Sleeping" -> C.L.A.S.S
  - "Come Late And then you Start Sleeping" -> C.L.A.S.S.

- Every capitalized word contributes one letter to the acronym

Strings and Scanners Practice

```
private String acronym(String str) {
        String result = “”;
        Scanner scanner = new Scanner(str);
        while (scanner.hasNext()) {
                String token = scanner.next();
                if (Character.isUpperCase(token.charAt(0))) {
                        result += token.charAt(0) + “.”;
                }
        }
        scanner.close();
        return result;
}
```

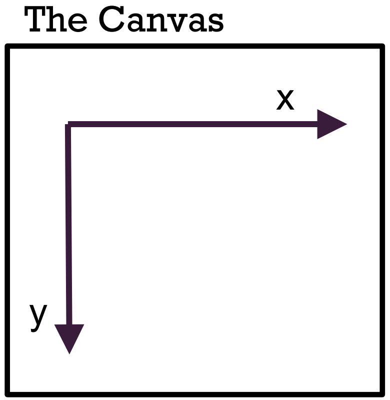
File Reading Practice

```
try {
        Scanner input = new Scanner(new File("data.txt"));
        while (input.hasNextLine()) {
                String line = input.nextLine();
                println(line);
        }
        input.close();
} catch (IOException ex) {
        println("Error reading the file: " + ex);
}
```

+

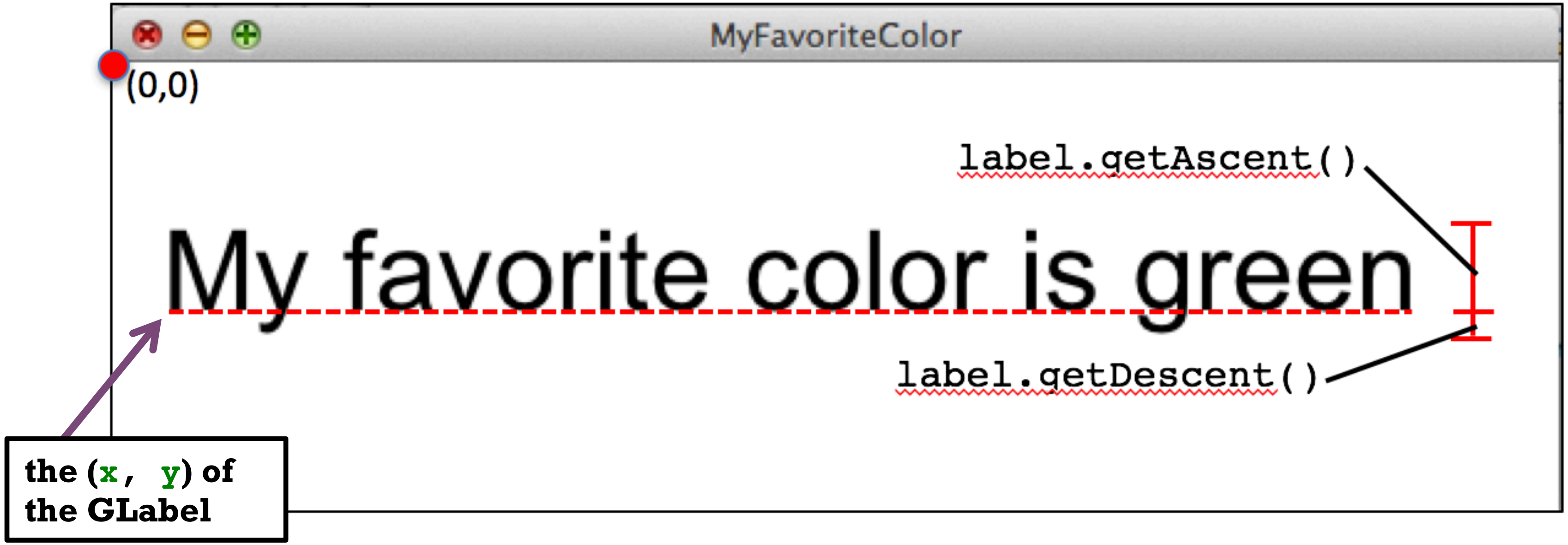
Graphics Programs

Graphics



- Remember to extend GraphicsProgram
- add/remove shapes
- Origin at **top left**! +x to the right, +y down
- GLine/GRect/Goval
- The x, y values of GRect, GOval, etc. is the **upper left corner**, but the x, y of a Glabel is the **leftmost baseline coordinate**
- Label's height gotten from getAscent

GLabels



```
GLabel glabel = new GLabel(str, x, y);
```

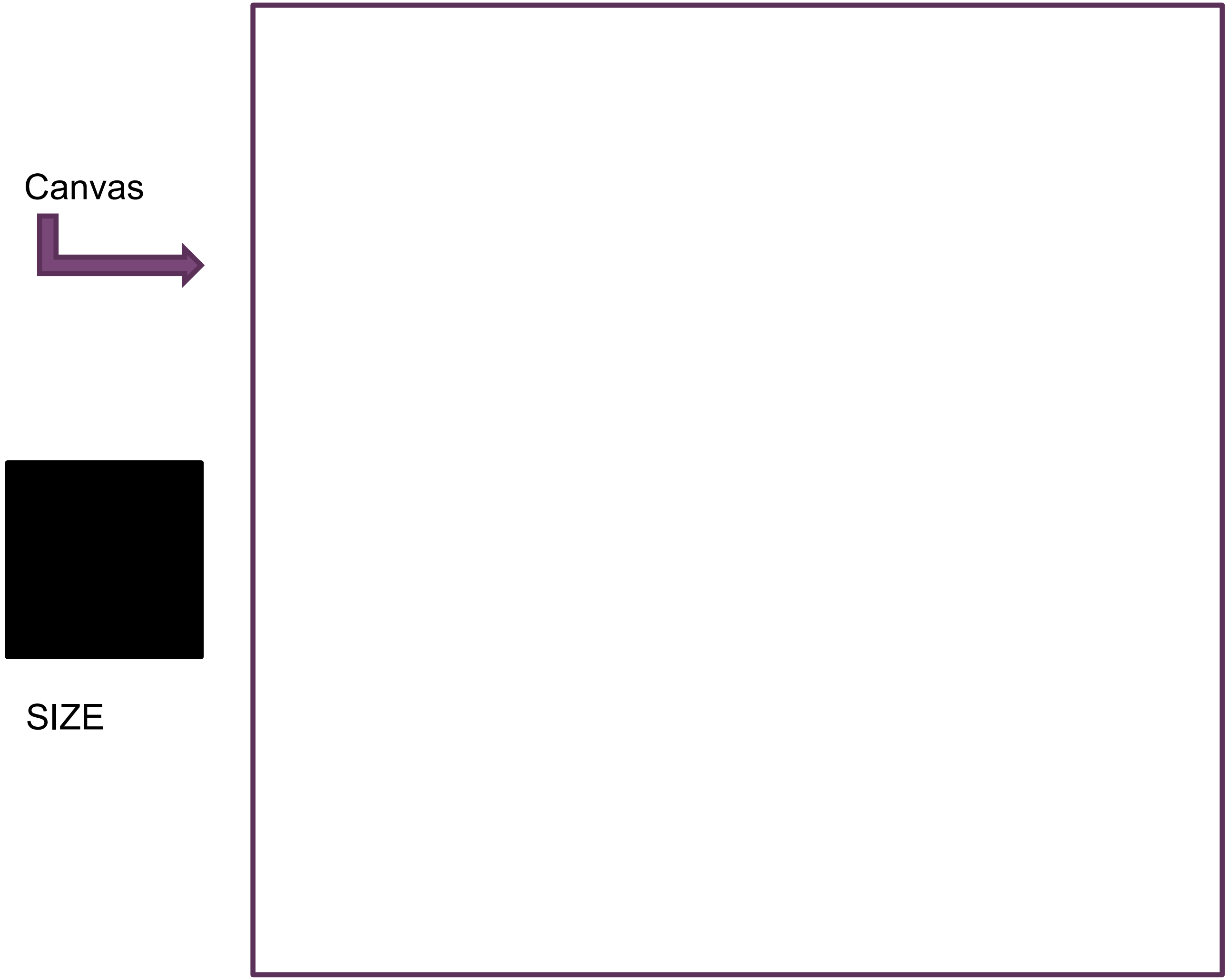
Graphics - Animation

- Standard format for animation code:

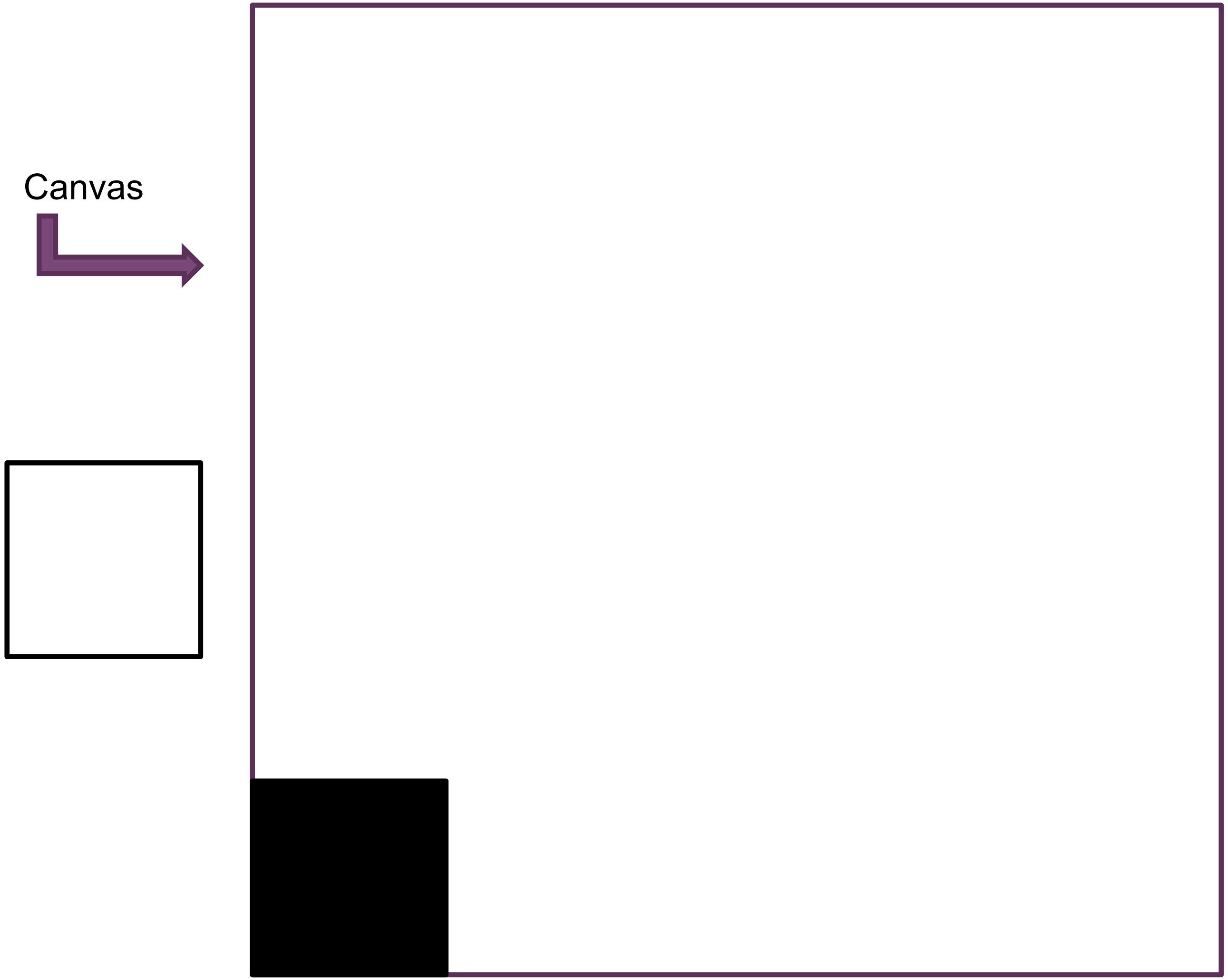
```
public void run() {
    ...
    while (test) {
        update the position of shapes;
        pause(milliseconds);
    }

}
```

Practice Problem: Checkerboard Graphics

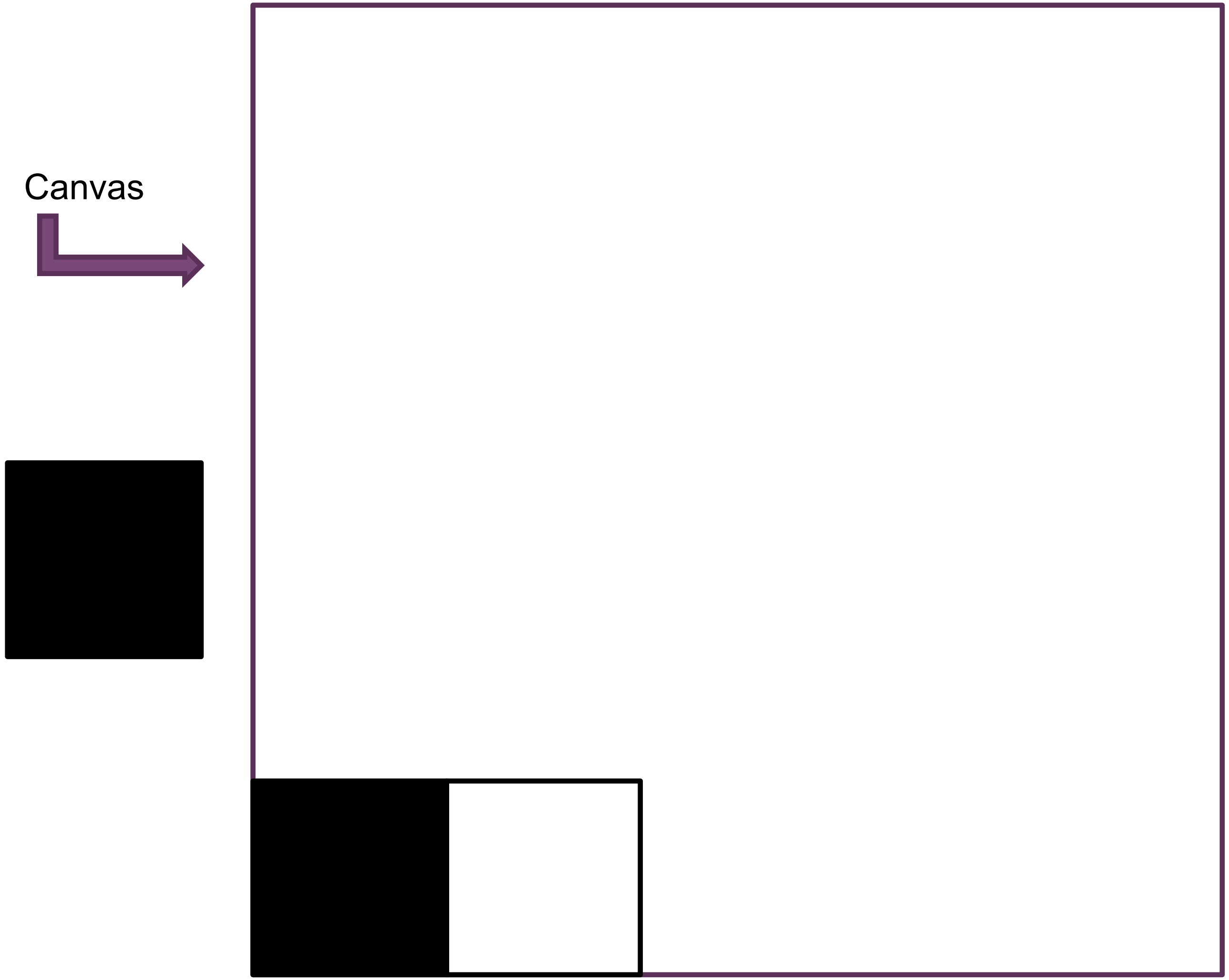
Write the method `drawCheckerboard(width, height)` that draws a checkerboard on the canvas with top left corner at the origin with width # of squares horizontally and height # of squares vertically. Assume there is a class constant SIZE defined for the size of the squares. Alternate black/white with the top left square black.



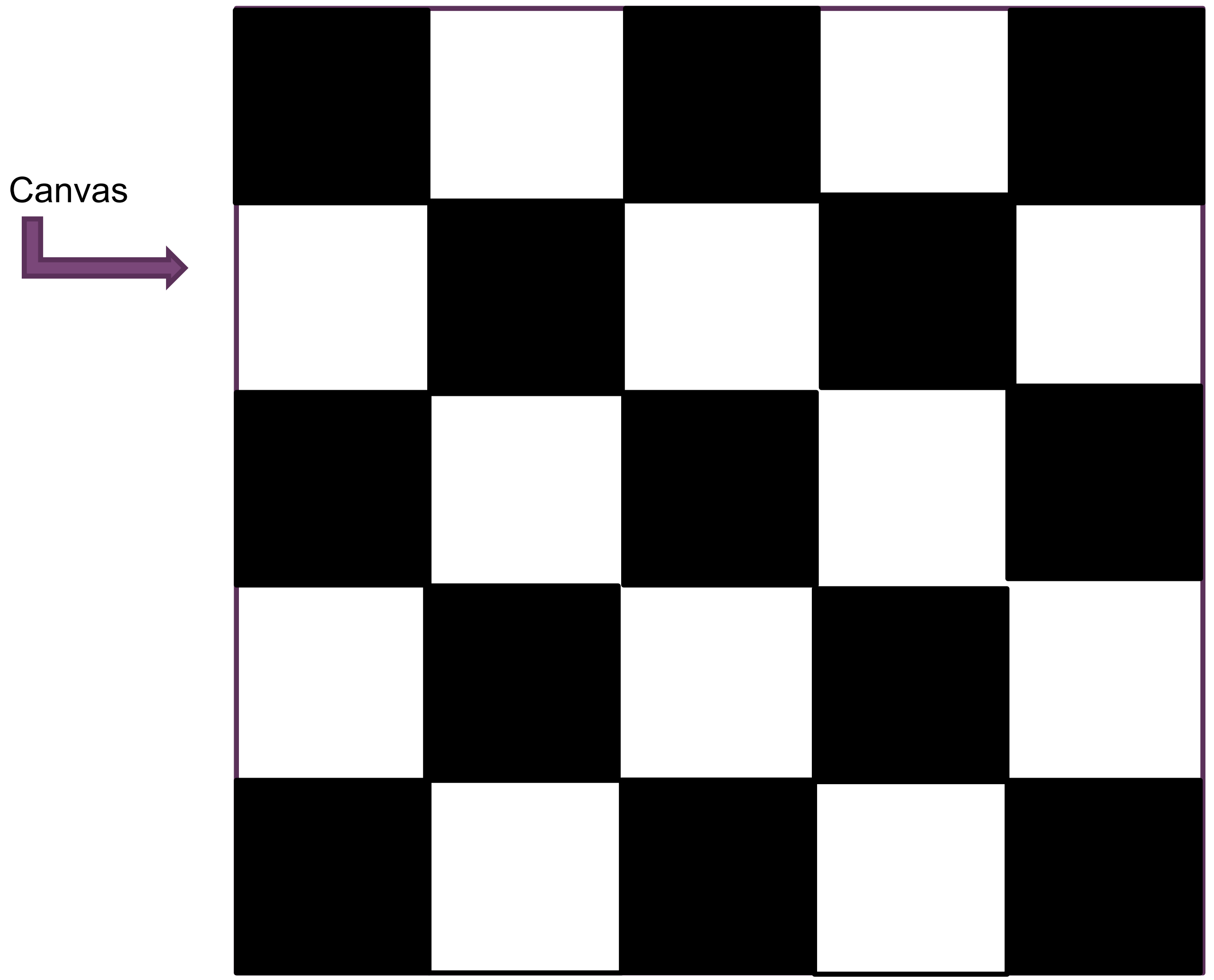
```
height = 5
width = 5
```



```
height = 5
width = 5
```



```
height = 5
width = 5
```



```
height = 5
width = 5
```

Solution: Checkerboard Graphics

```
public void drawCheckerboard(int width, int height) {
    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
            if ((row + col) % 2 == 0) {
                GRect box = new GRect(col * SIZE, row * SIZE, SIZE, SIZE);
                box.setFilled(true);
                box.setFillColor(Color.BLACK);
                add(box);
            }
        }
    }
}
```

Event Handlers

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

public void mouseClicked(MouseEvent e) {
    // Java runs this when mouse is clicked
}

public void mouseMoved(MouseEvent e) {
    // Java runs this when mouse is moved
}
```

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

`public void` ***`eventMethodName`*** `(MouseEvent event) { ...`

Event Handlers

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

```
public void eventMethodName(MouseEvent event) { ...
```

… and contain, at least, the following information:

Method	Description
`e.getX()`	the *x*-coordinate of mouse cursor in the window
`e.getY()`	the *y*-coordinate of mouse cursor in the window

Memory

Instance Variables

```
private type name;  // declared outside of any method
```

- **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*

Primitives vs. Objects

	Primitives	Objects
What do they store in their variable box, directly?	actual value	location of the object
Can you compare using == and !=?	Yes	No
How are they passed as parameters?	A copy of the **value**	The actual **location** ("reference") of original
Does the original change when it's passed as a parameter?	No	Yes

“null”

Can an integer be **null**?

Answer: no, all **primitives** cannot be null.

What about a GOval?

Answer: yes, **object** variables can be set to null.

How do you check if a variable is **null**?

```
if (maybeAnObject == null) { …
```

Why would you do this?

Calling methods on an object that is **null** will crash your program!

```
// 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!
}
```

Questions?

Good luck on the midterm!