CS106A Review Session

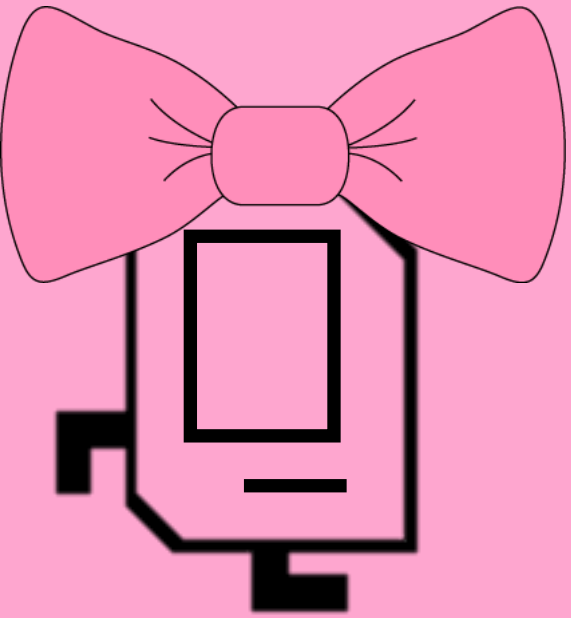
Saturday Feb. 11, 2017

Maria Yang

Topic List

- Karel
- Java constructs
- Graphics + Animation
- Memory (Pass-by-reference vs. pass by value)
- Event-driven programming
- Characters and Strings

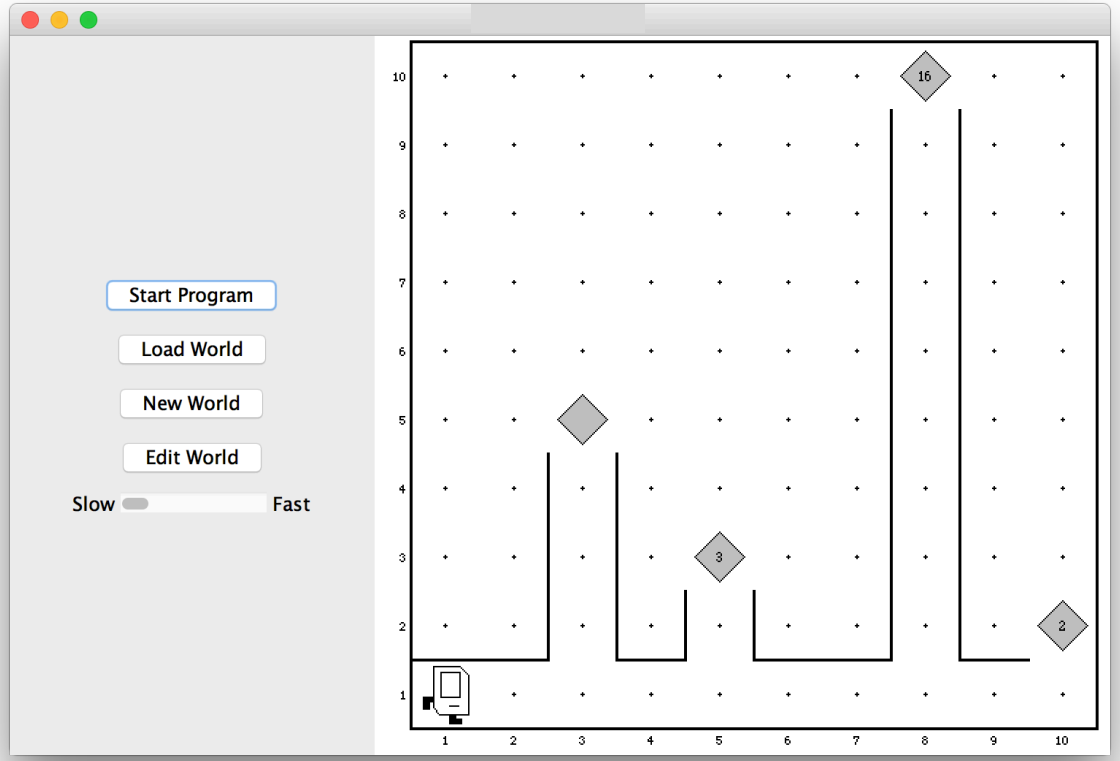
Karel



Karel

- Tips:
  - Pseudocode first
  - Decompose the problem
  - Might be limitations on constructs
    - E.g. no Java features (variables, break, etc.)

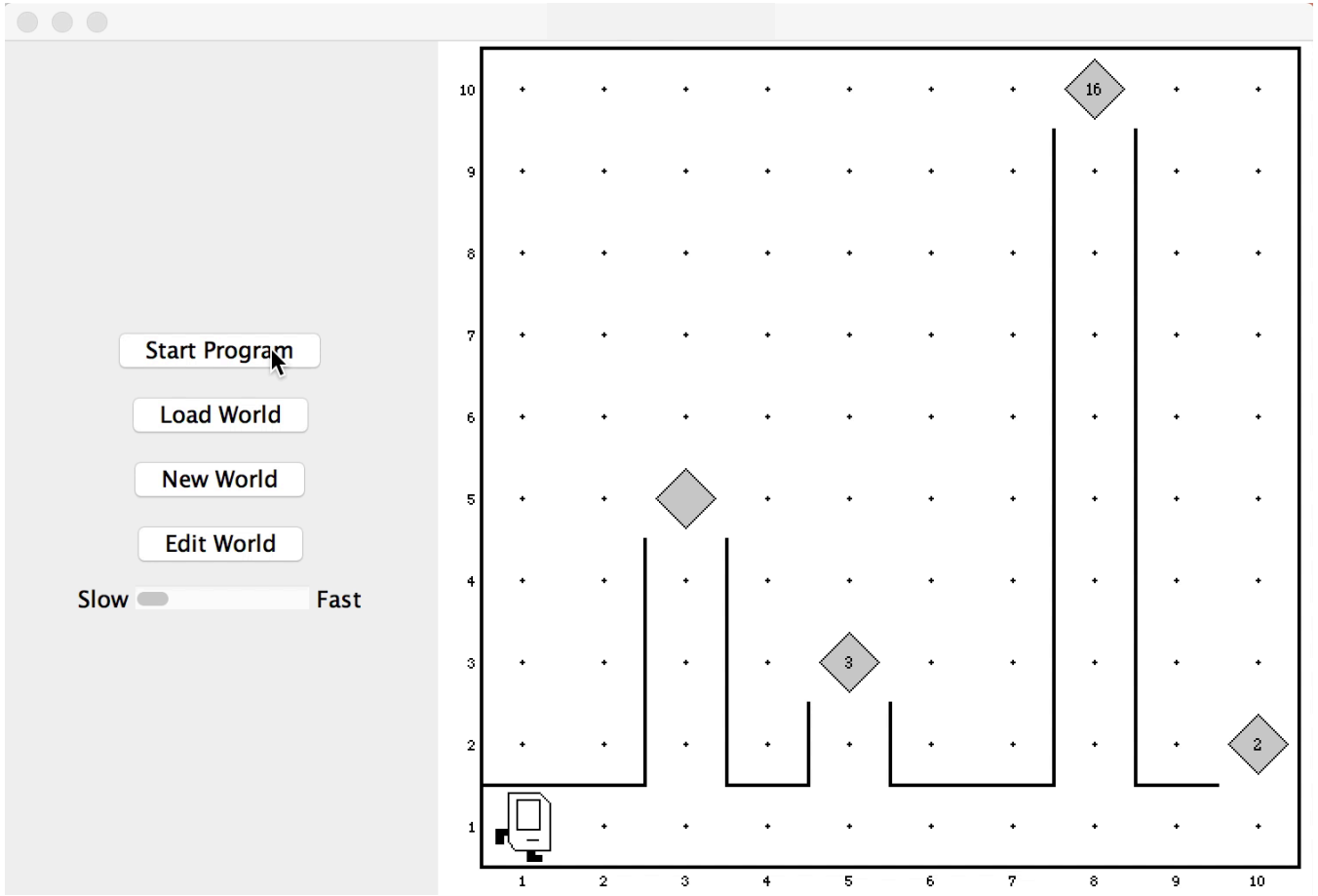
Karel Needs Love



Karel Needs Love

Karel is in a world with walkways to houses that have valentines. Karel should go to every house in order, go up the walkway and take all the valentines (beepers). House walkways can be any distance apart, and have guide walls on the left and right up to the valentine.

Demo: Karel Needs Love



Karel Needs Love

Loop:

- Go to next house
- Get Valentines

Go to next house:

- move along left wall

Get Valentines:

- traverse walkway
- take valentine
- traverse walkway

Karel Needs Love

```
public void run() {
    while (frontIsClear()) {
        goToNextHouse();
        getValentines();
        if (frontIsClear()) {
            move();
        }
    }
}
```

Karel Needs Love

```
private void goToNextHouse() {
     while (leftIsBlocked()) {
          move();
     }
}
```

Karel Needs Love

```
private void getValentines() {
    turnLeft();
    traverseWalkway();
    takeValentine();
    turnAround();
    traverseWalkway();
    turnLeft();
}
```

Karel Needs Love

```
private void traverseWalkway() {
     move();
     while (leftIsBlocked() && rightIsBlocked()) {
          move();
     }
}
```

Karel Needs Love

```
private void takeValentine() {
     while (beepersPresent()) {
          pickBeeper();
     }
}
```

Java Constructs

Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**

Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**

For or While?

WHILE • Read in user input until you hit the SENTINEL

FOR • Iterate through a string

WHILE • Move Karel to a wall

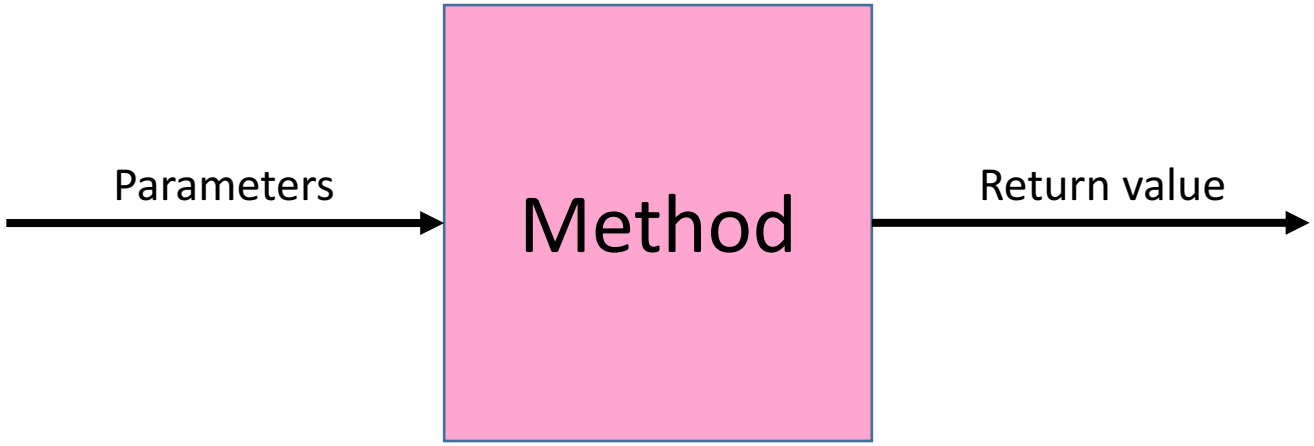
WHILE • Read in a file line-by-line (note: files are **not** on the exam)

Java Constructs

- **Variable types**: primitives (int, double,…) + objects (GRect, Goval,…)
- **Control statements**: if, while, for, switch
  - What is each useful for?
- **Methods**

Java Constructs - Methods

- A method is a routine of instructions that may take some input and give back some output.



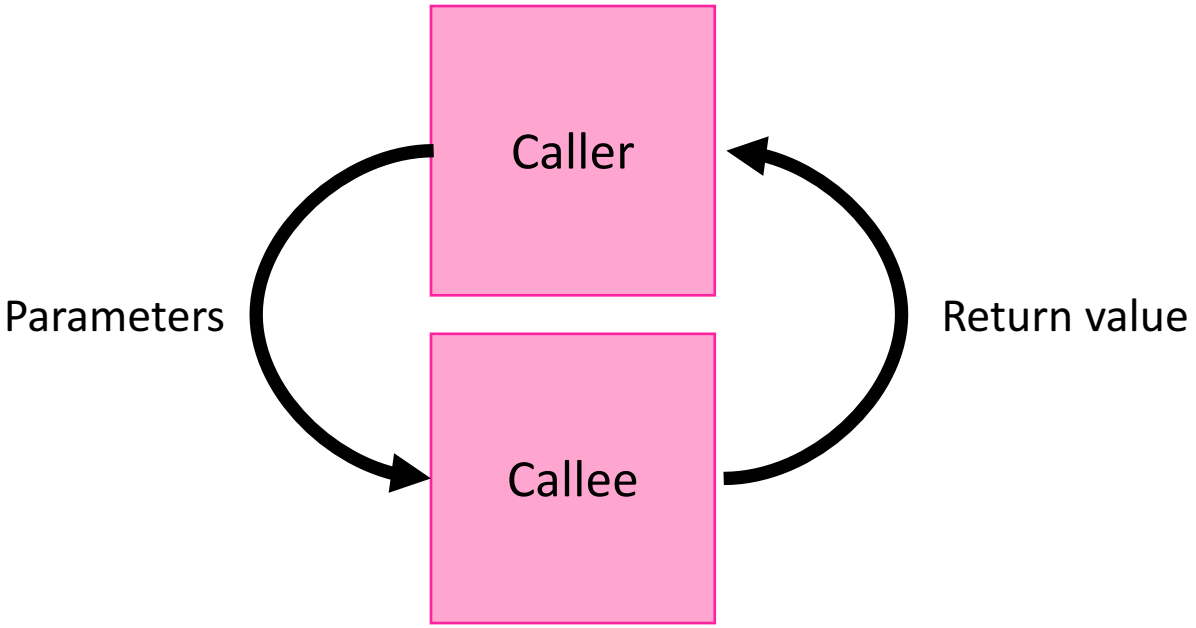
Java Constructs - Methods

```
public void run() {
    println("Hypotenuse of 3 and 4 is: ");
    println(hypotenuse(3.0, 4.0));
}

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

Java Constructs - Methods

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



Java Constructs - 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);
}
```

Java Constructs - 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're passed
  - Write values above variable names as you go through the program (or draw stack card boxes)

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    int candy = 5;
    int love = 6;
    candy = howMuchCandy(candy, love);
    println("I got " + candy + " candy(ies)");
}

private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() -1);
}

private int howMuchCandy(int candy, int love) {
    int num3 = love + candy / 2;
    return num3 % 3;
}
```

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}


private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

run

str Yay!! It is Valentine's Day.

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}


private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

run

str Yay!! It is Valentine's Day.

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}


private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

run

str Yay!! It is Valentine's Day.

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}
```

```
private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

run

str	Yay!! It is Valentine's Day.

getValentine

str	Yay!! It is Valentine's Day.
num1	6

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}

private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

run

str	Yay!! It is Valentine's Day.

getValentine

str	Yay!! It is Valentine's Day.
num1	12

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}

private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

27

run

str: Yay!! It is Valentine's Day.

getValentine

str: Yay!! It is Valentine's Day.

num1: 12

Program Trace

```
public void run() {
    String str = "Yay!! It is Valentine's Day.";
    println(getValentine(str, 6));
    ...
}

private String getValentine(String str, int num1) {
    num1 *= 2;
    return str.substring(num1, str.length() – 1);
}
```

"Valentine's Day"

run

str	Yay!! It is Valentine's Day.

getValentine

str	Yay!! It is Valentine's Day.
num1	12

Program Trace

```
public void run() {
     String str = "Yay!! It is Valentine's Day.";
     println(getValentine(str, 6));
     ...
}
```

run

str Yay!! It is Valentine's Day.

Valentine's Day

(Console)

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love= 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy 5

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy	5
love	6

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy	5
love	6

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy	5
love	6

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy	5
love	6

howMuchCandy

candy	5
love	6

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3;
}
```

run

candy	5
love	6

howMuchCandy

candy	5
love	6
num3	8

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}

private int howMuchCandy(int candy, int love) {
        int num3 = love + candy / 2;
        return num3 % 3; // 8 % 3 = 2
}
```

run

candy	5
love	6

howMuchCandy

candy	5
love	6
num3	8

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}
```

run

candy	8
love	6

Program Trace

```
public void run() {
        ...
        int candy = 5;
        int love = 6;
        candy = howMuchCandy(candy, love);
        println("I got " + candy + " candy(ies)");
}
```

run

candy 8

love 6

Valentine's Day
I got 2 candy(ies)

(Console)

Program Trace

- **Tricky spots:** precedence, parameter/variable names…
- Draw pictures! / Label variable values

Graphics + Animation

Graphics

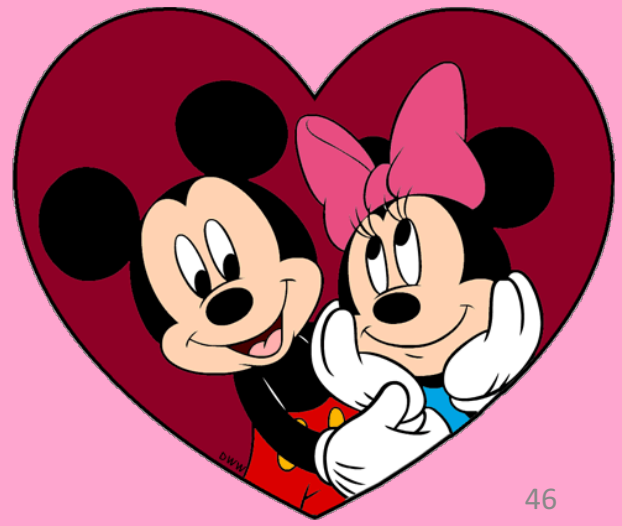
- Look at lecture slides for lists of different GObject types and their methods
- Remember: the x and y of GRect, GOval, etc. is their **upper left corner**, but the x and y of GLabel is its **leftmost baseline coordinate**.
- Remember: a label's height is gotten from **getAscent()**.

Animation

Standard format for animation code:

```
while (CONDITION) {
      updateGraphics();
      performChecks();
      pause(PAUSE_TIME);
}
```

Memory



Memory

- **Stack and heap**
  - **Stack** is where local variables live
  - **Heap** is where objects live
- When you make an object, the local variable (what you named it) is a box that stores an **address on the heap** where the object actually lives.
- When you make a primitive, the local variable is a box that stores the **actual value**.

Memory

- **==** is dangerous because it compares what's in the **variable boxes!**
  - For primitives, ok
  - For objects, compares their addresses! So only true if they're the exact same object living in the exact same place.

Memory

- **Parameters:** when you pass a parameter, Java passes a copy of whatever is in the variable's box.
  - For primitives – a copy of their **value**
  - For objects – a copy of their **address!** So there's still only 1 object version

Memory

```
public void run() {
      GRect rect = new Grect(0,0,50,50);
      fillBlue(rect);
      add(rect);      // rect is blue!
}

private void fillBlue(GRect rect) {
      rect.setFilled(true);
      rect.setColor(Color.BLUE);
}
```

Memory

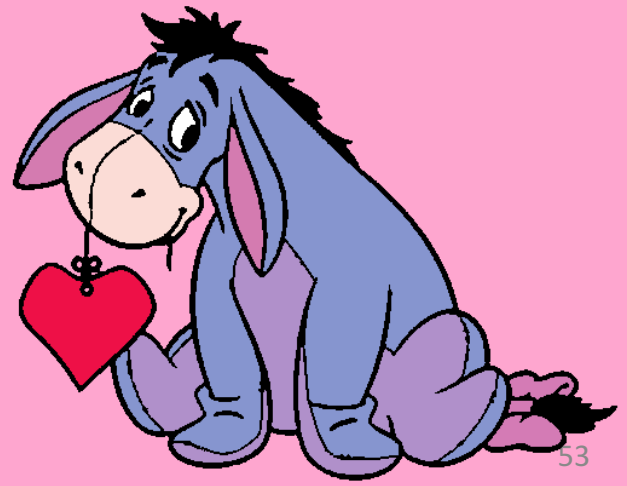
```
public void run() {
      int x = 2;
      addTwo(x);
      println(x);  // x is still 2!
}

private void addTwo(int x) {
      x += 2;          // this modifies addTwo's COPY!
}
```

Memory - Getting Changes to Copies

```
public void run() {
      int x = 2;
      x = addTwo(x);
      println(x);  // x is still 2!
}
private int addTwo(int x) {
      x += 2;          // this modifies addTwo's COPY!
      return x;
}
```

Event-driven Programming



Event-Driven Programming

- **Mouse Events!**
- **Two** ways for Java to execute your code: from run() and from event handler (mouseClicked, mouseMoved, etc.).
- These programs are **asynchronous** – code is not run in order any more, since you don't know when the user will interact with your program!

Event-Driven Programming

1. Sign up for notifications for key or mouse events
2. Implement the method corresponding to what event you care about (e.g. **mousePressed**, **mouseMoved**).
3. Java will call that method whenever the corresponding event occurs.

Event-Driven Programming

Let's write a program like "mystery square", but when you click the shape changes between a square and a circle.

The shapes should always be the same random color, even if you click in between colors changing.

Refresher: Mystery Square

```
public class ColorChangingSquare extends GraphicsProgram {

   /* Size of the square in pixels */
   private static final int SQUARE_SIZE = 100;

   /* Pause time in milliseconds */
   private static final int PAUSE_TIME = 1000;


   public void run() {
      GRect square = new GRect(SQUARE_SIZE, SQUARE_SIZE);
      square.setFilled(true);
      add(square, (getWidth() - SQUARE_SIZE) / 2,
                  (getHeight() - SQUARE_SIZE) / 2);

      /* Note: we meant to have this infinite loop */
      while (true) {
         square.setColor(rgen.nextColor());
         pause(PAUSE_TIME);
      }
   }

/* Private instance variables */
   private RandomGenerator rgen = RandomGenerator.getInstance();
}
```

Color-Changing Shape

Modifications:

- Have a **square and circle** at the same time
- Change the colors of **both** every PAUSE_TIME
- On click, add one and remove the other

Constants + Instance Variables

```
private static final int SQUARE_SIZE = 100;
private static final int PAUSE_TIME = 100;
private GRect square;
private GOval circle;
private boolean isShowingSquare = true;

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

Color-Changing Shape

```
public void run() {
    setup();
    while (true) {
        Color c = rgen.nextColor();
        square.setColor(c);
        circle.setColor(c);
        pause(PAUSE_TIME);
    }
}
```

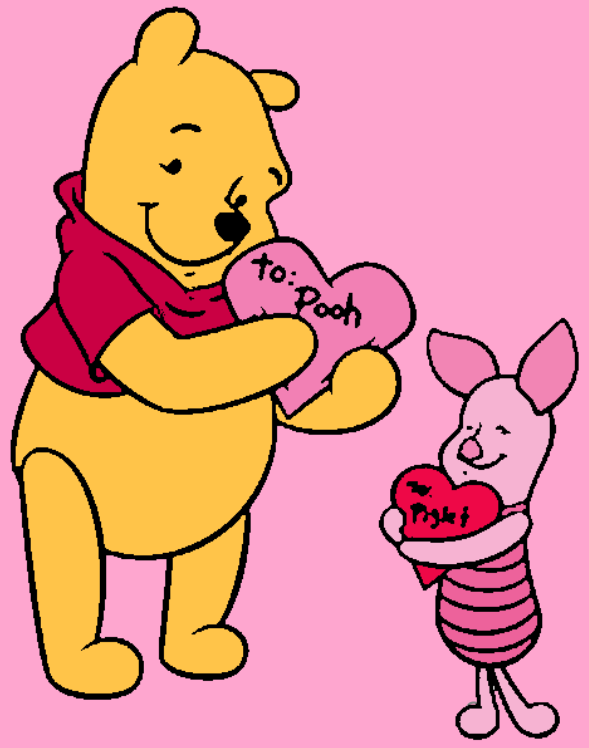
Color-Changing Shape

```
private void setup() {
    double x = (getWidth() – SQUARE_SIZE) / 2.0;
    double y = (getHeight() – SQUARE_SIZE) / 2.0;
    square = new GRect(x, y, SQUARE_SIZE, SQUARE_SIZE);
    circle = new GOval(x, y, SQUARE_SIZE, SQUARE_SIZE);
    square.setFilled(true);
    circle.setFilled(true);
    add(square);   // only add square!
    addMouseListeners();
}
```

Color-Changing Shape

```
public void mouseClicked(MouseEvent e) {
     if (isShowingSquare) {
          remove(square);
          add(circle);
     } else {
          remove(circle);
          add(square);
     }
     isShowingSquare = !isShowingSquare;
}
```

Characters and Strings



Characters and Strings

- A **char** is a primitive type that represents a single letter, digit, or symbol. Uses single quotes (‘’).
- Computers represent **chars** as numbers under the hood (ASCII encoding scheme).
- A string is an immutable object that represents a sequence of characters. Uses double quotes (“”).

Characters

```
char uppercaseA = 'A';
char uppercaseB = (char)(uppercaseA + 1);
int lettersInAlphabet = 'Z' – 'A' + 1;
// equivalent: 'z' – 'a' + 1
// A to Z and a to z are sequential numbers.
```

Characters

Useful Methods in the `Character` Class

`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

Characters

- **Note:** chars are primitives. This means we can't call methods on them!
- Instead we use the **Character** class and call methods on it. We pass in the character of interest as a parameter.
- These methods *do not change the char*! They return a modified char.

Characters

```
char ch = 'a';
Character.toUpperCase(ch);     // does nothing!
ch.toUpperCase();              // won't compile!
ch = Character.toUpperCase(ch);    // ✔

if (Character.isUpperCase(ch)) {
    println(ch + " is upper case!");
}
```

Strings

- **Note:** strings are (immutable) objects. This means we can call methods on them!
- We *cannot change a string after creating it*.
- Strings can be combined with ints, doubles, chars, etc.

Strings

Useful Methods in the `String` Class

`int length()` Returns the length of the string
`char charAt(int index)` Returns the character at the specified index. Note: Strings indexed starting at 0.
`String substring(int p1, int p2)` Returns the substring beginning at **`p1`** and extending up to but not including **`p2`**
`String substring(int p1)` Returns substring beginning at **`p1`** and extending through end of string.
`boolean equals(String s2)` Returns true if string **`s2`** is equal to the receiver string. This is case sensitive.
`int compareTo(String s2)` Returns integer whose sign indicates how strings compare in lexicographic order
`int indexOf(char ch)` *or* **`int indexOf(String s)`** Returns index of first occurrence of the character or the string, or -1 if not found
`String toLowerCase()` *or* **`String toUpperCase()`** Returns a lowercase or uppercase version of the receiver string

Using portions of slides by Eric Roberts

Strings

```
String str = "Hello world!";      // no "new" needed
str.toUpperCase();                // does nothing!
str = str.toUpperCase();          // ✔

for (int i = 0; i < str.length(); i++) {
    println(str.charAt(i));
}
// prints each char on its own line
```

Putting it ALL together

```
String str = "'ello mate!";
str = str.substring(1);
str = 'H' + str;          // str = "Hello mate!"
String newStr = "";
for (int i = 0; i < str.length(); i++) {
    newStr = str.charAt(i) + newStr;
}
// newStr = "!etam olleH"
```

Type Conversion Mayhem

```
println("B" + 8 + 4);
// prints "B84"
println("B" + (8 + 4));
// prints "B12"
println('A' + 5 + "ella");
// prints "70ella (note: 'A' corresponds to 65)"
println((char)('A' + 5) + "ella");
// prints "Fella"
```

Type Conversion Mayhem

- This seems nonsensical - but it's not! (kind of)
- **Just use precedence rules** and keep track of the type along the way.

```
println('A' + 5 + "ella");
// 'A' + 5 is int (70), int + "ella" is string
println((char)('A' + 5) + "ella");
// 'A' + 5 is char ('F'), char + "ella" is string
```

Strings Practice

Lets write a method that, given a string, removes all **strings within asterisks** and returns the result.

```
String str = "int s = 2; * This is 2 *";
println(removeComments(str)); // "int s = 2; "
str = "Hi * Hello * Hello";
println(removeComments(str)); // "Hi  Hello"
str = "No comments!";
println(removeComments(str)); // "No comments!"
```

Strings Practice

- Super helpful Strings pattern: given a string, iterate through and build up **a new string**. (Since strings are immutable!)

```
String oldStr = ...
String newStr = "";
for (int i = 0; i < oldStr.length(); i++) {
    // build up newStr
}
```

Strings Practice

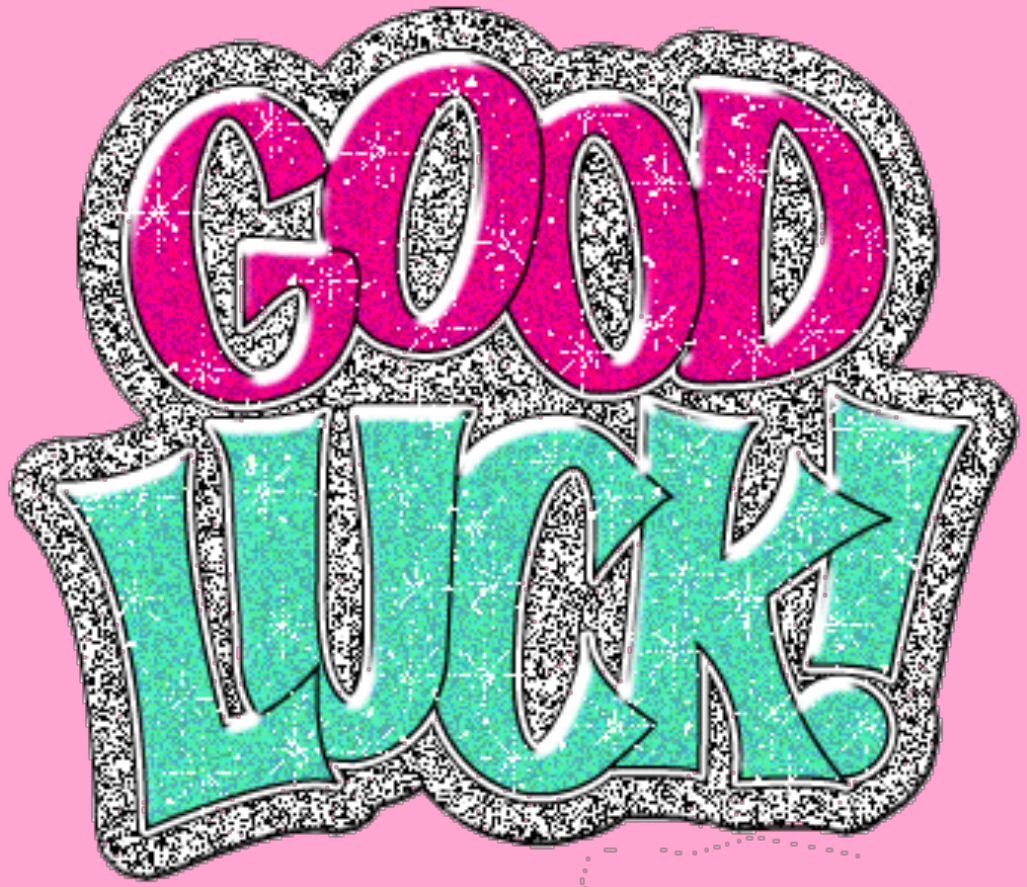
```
private String removeComments(String str) {
        String newStr = "";
        boolean inComment = false;
        for (int i = 0; i < str.length(); i++) {
                if (str.charAt(i) == '*') inComment = !inComment;

                // Note: if at end of comment, inComment already false!
                if (!inComment && str.charAt(i) != '*') {
                        newStr += str.charAt(i);
                }
        }
        return newStr;     // DON'T FORGET!!
}
```

Parting Words

Parting Words

- Try to get to every problem
- Don’t rush to coding too quickly
- Pseudocode!
- Look over the practice midterm


GOOD
LUCK!