Prelim Question

What I said:

(b) (2 points) You find a pre-historic computer science textbook, which claims that "the best known search methods have O(n^2) asymptotic complexity." Is this claim still correct today, technically? Why or why not?

What I meant to say:

… sort methods …

The neas:

"Not true. Merge-sort has O(n log n) complexity."

"Not correct. Binary search is O(log n)."

"No. Can bring it down to O(n log n) by sorting and doing binary search."

The yeas:

"Yes. The best known search method now is binary search O(n log n) but that requires data to be sorted … need to use quicksort which takes O(n^2)"

"Yes. The best known search method today is quicksearch, and it is O(n^2)"

"Correct b/c even if we have one that is n log(n) that is also O(n^2)…"

Big-O

- We say XXX = O(YYY) to mean "XXX are *no worse* than YYY"
- Not "XXX is *exactly as bad* as YYY" -- although there is a notation for this concept: XXX = θ(YYY)
- So *technically*:
  - $n = O(n^2)$
  - $n \log n = O(n^2)$
  - $1 = O(n^2)$
  - $\log n \log (\log n \log (n \log n)) = O(n^2)$
- Even if we wouldn't really say it that way in normal conversation
- A better notation: $XXX \in O(YYY)$

All you ever wanted to know about…

How to Cheat Honestly, Dictionaries, Threads, Synchronization, Events & Event-driven programming, Graphical Interfaces, Secure Programming, Capture The Flag, and my quirky programming style…

How to cheat honestly on a prelim
or
Fatherly advice

(b) (2 points) You find a pre-historic computer science textbook, which claims that "the best known search methods have O(n^2) asymptotic complexity." Is this claim still correct today, technically? Why or why not?

(c) (8 points) What is the *tightest* big-O asymptotic complexity of…

…

class E extends C …

(e) objC = objE; **Student says:** "Illegal: C not a subclass of E"

…

class Eleve extends Student …

Eleve a = …;

Student b = a;

…

Just in case it is not clear yet…

(iii) printTree(root, new PriorityQueue())

Student says: R I G H T O N

…

(b) (11 points) Write a method *printDescending* taking only a TreeCell as an argument, and that prints the data stored in the tree rooted at this cell in descending order of priority. For the tree above, it would print R I G H T O N. …

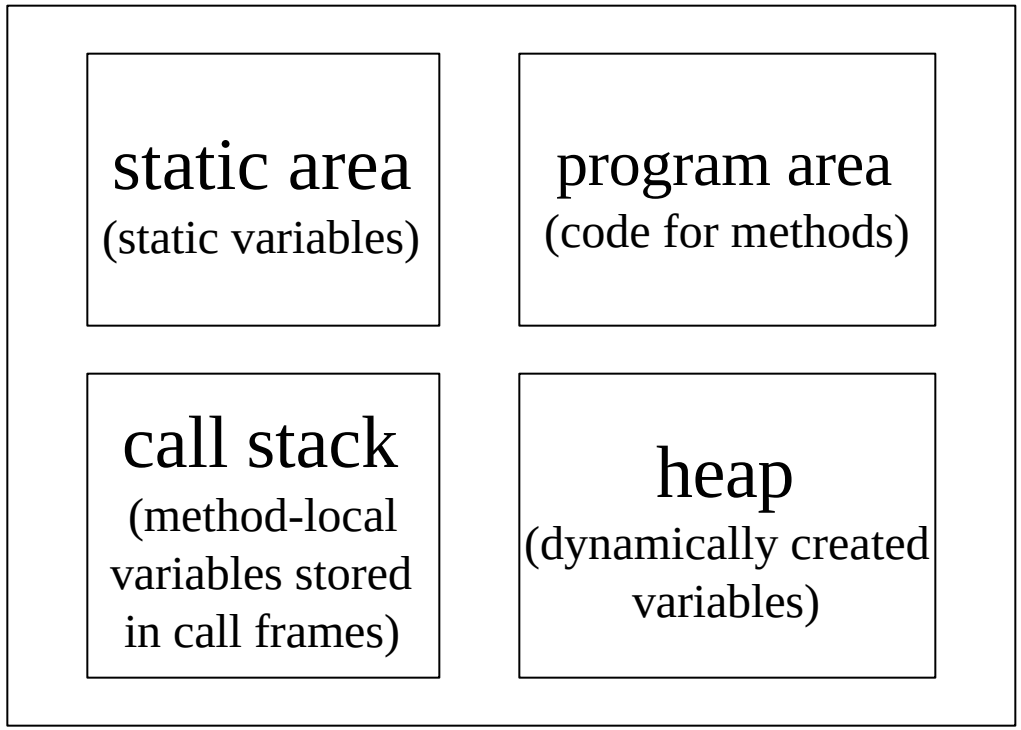
Dictionaries (aka Maps)

- Used a lot in final project
- Refer to “Hashing” lecture notes, last few slides

```
HashMap eyecolor = new java.util.HashMap();
 eyecolor.put(“kevin”, “blue”)
 eyecolor.put(“ben”, “brown”);
 eyecolor.put(new Integer(8), new ChessGame(…));
 …
 String c = (String)eyecolor.get(“meghan”);
 …
```

Threads

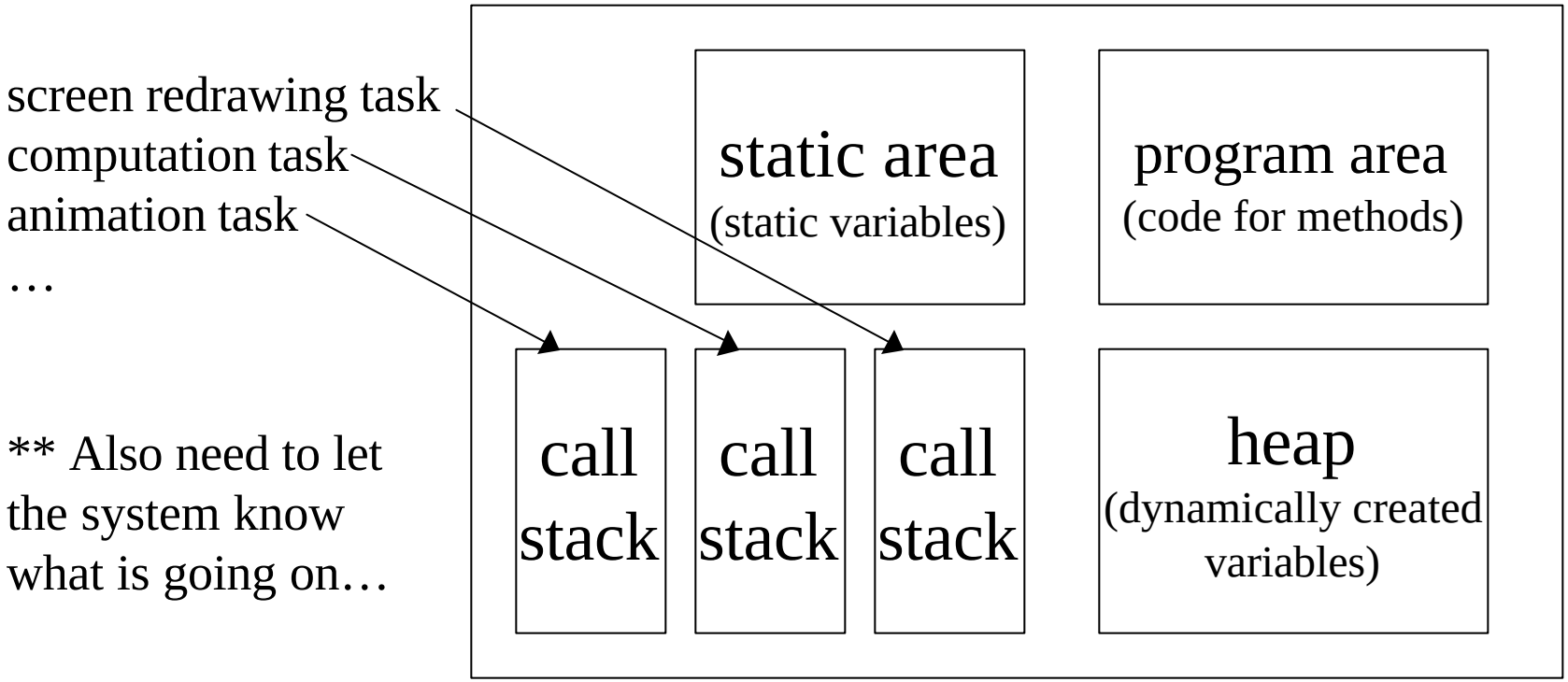
- Used a little in the final project
- We said each running program has four areas



- Program can only be doing one thing at a time… why?

Threads

- Sometimes want multiple things happening at the same time: animation + computation + screen redrawing + handling user input + downloading off network + …
- Solution: need multiple call stacks!



Threads in Java

- Thread = A call stack + some other data internal to the system
  - Thread conceptually is a “task”
  - Can create new ones, start them, stop them, etc.

Thread task = new Thread(…) // creates new call stack

task.start(); // tells the new stack it can start now

System.out.println(“I launched a new task!”);

…

- How to specify what the task should do?
  - Extend Thread and implement your stuff there
  - or Pass a “function object” to Thread constructor

Multi-Tasking / Multi-Threading

- A program with many threads can now *multi-task*!
- General rule: If there are Threads A, B, C, D, …, the system picks which is/are "active" at any one time
  - Do a little of task A
  - Now a little of task C
  - Now a little more of task A
  - Now a little of task B
  - Now a little more of task C
  - …
- *Generally* you don't get to specify what order! (maybe just some priorities…)
  - And a task can get de-activated at any time whatsoever (more or less)

Threaded Execution

```
// thread # 1
print(“a”);
print(“b”);
print(“c”);
print(“d”);
print(“e”);
```

```
// thread # 2
print(“1”);
print(“2”);
print(“3”);
print(“4”);
print(“5”);
```

- Possible outcomes:
  - abcde12345
  - 12345abcde
  - ab1cd2345e
  - 123a45bcde
  - ….

Thread Synchronization

- Problem: Interleaving of threads can cause errors
- Concurrent Access
  - Multiple threads accessing/modifying same data
  - A big problem for linked lists, trees, etc.
  - A problem for arithmetic
    - x = x + 1 is not necessarily “thread safe”
- Atomic actions
  - Several steps might be an “atomic unit” – we don’t want to break an atomic unit!
  - E.g. printing several lines of text (don’t want other text inserted!)
  - Performing a bunch of arithmetic (don’t want variables changing unexpectedly in the middle!)
  - Updating several things together

    x = x + y

    y = x + y

Java's answer

- *synchronized* keyword used on an object

```
public void myMethod(int y) {
        y = 3;
        synchronized (someObject) {
                x = x + y;
        }
        y = 5;
}
```

- *synchronized* keyword used with a method

```
public synchronized void myMethod(String s) { … }
```

same as

```
public void myMethod(String s) {  synchronized (this) { … } }
```

Synchronized keyword

- Basic Rule:
  - Only one thread (at most) can be synchronized on a particular object at any given time
  - Threads take turns entering the synchronized blocks or synchronized methods

```
// thread # 1
print("a");
synchronized (objA) {
    print("b");
    print("c");
    print("d");
}
print("e");
```

```
// thread # 2
print("1");
synchronized (objA) {
    print("2");
    print("3");
    print("4");
}
print("5");
```

Possible outcomes:

a1bcde2345

abcd12345e

1234abcde5

Threads don’t solve evertying

- Each thread has a task, and executes the task from beginning to end. A sort of linear execution + loops (+ recursion)
- How to handle external inputs?
  - User input: Clicks, drags, key presses, etc.
  - Network/disk: data arrives, needs to be processed
  - etc.
- Handling with with a loop:

```
while (true) {
    for (int i = 0;  i < nbuttons; i++) if (button[i] was selected) …
    for (int i = 0;  i < nkeys; i++) if (key[i] was pressed) …
    for (int i = 0;  i < nmenus; i++) if (menu[i] was chosen) …
}
```

Events & Event-driven Programming

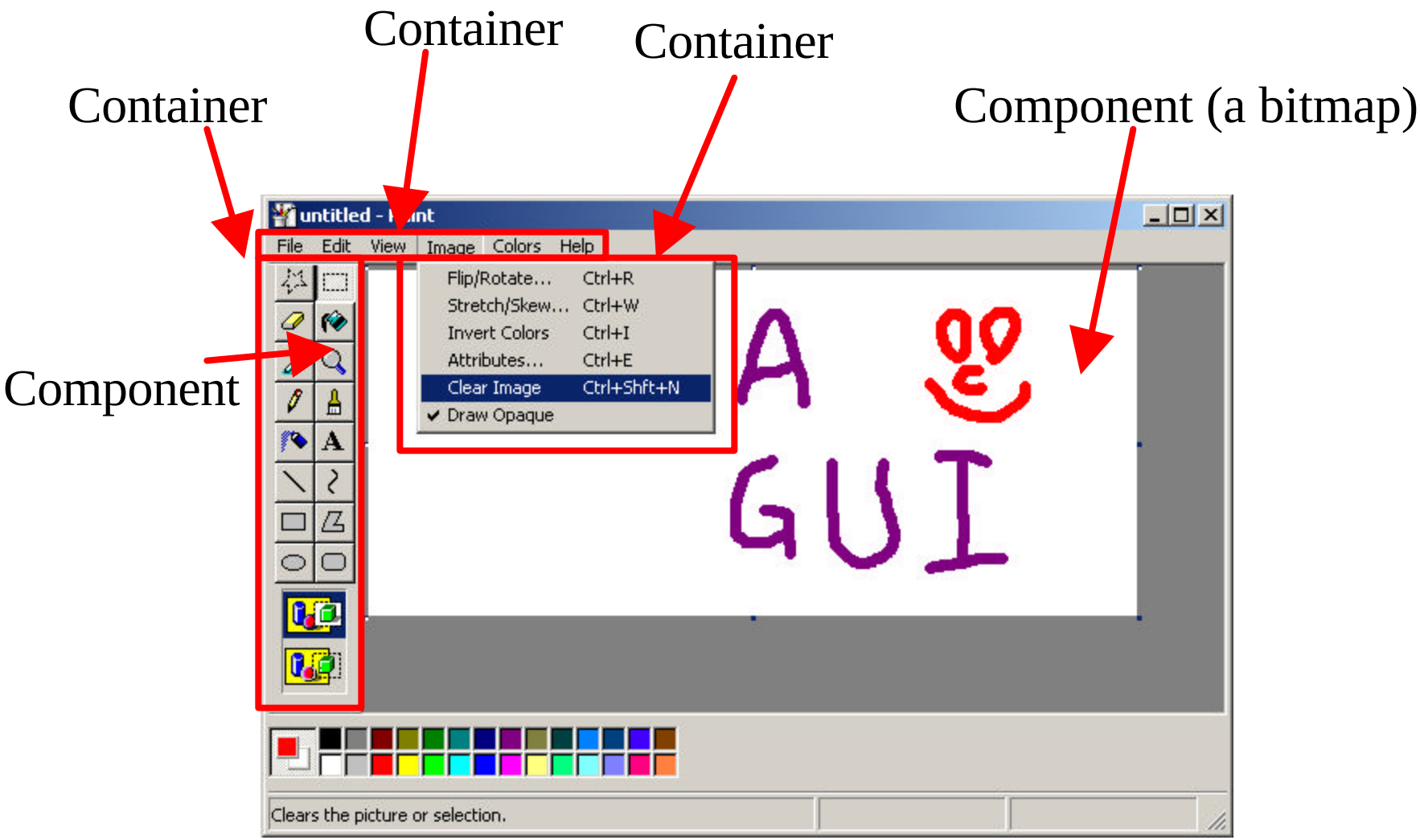
- What we want:
  - specify for each button, what to do if it gets clicked
  - specify for each key, what to do if it gets pressed
  - specify for each menu, what to do if it gets selected
  - etc.
- How?
  - System maintains a *queue* of *events*
  - Whenever *X* happens to *B*, system creates <*B*, *X*> pair (the "event") and puts it on the event queue
  - System "Worker" threads take each event <*B*, *X*> off queue and execute *B.handleEvent(X)*
  - Programmer just writes the *handleEvent* method
    - or specifies it somehow

Java's Graphical User Interface Toolkits

- Java has **two separate toolkits** for GUIs
- java.awt.* ("abstract window toolkit")
  - Mostly written in native code
  - Specific to the operating system
    - AWT implementations available for Mac, Windows, etc.
  - Called *heavyweight*, because it interacts directly with OS
- javax.swing.* ("swing")
  - Mostly written in Java
    - So fully portable to any operating system
  - Called *lightweight*, because it is independent of the OS
  - Built on top of AWT
  - Duplicates much of the functionality of AWT, but nicer

GUI Classes

- *Components* – what you see on the screen (JButton)
- *Containers* – a type of *Component* that contains other components (JPanel may contain JButtons)
- *Layout Managers* – control placement of components in containers (vertical layout, grid, etc.)
- *Events* – are generated by sources (ActionEvent is generated by a JButton component)
- *Listeners* – objects that listen for events to happen (myQuitListener handles ActionEvent from JButton)
- *Helpers* – misc. (Color, Graphics, Font, …)



Layout within each container handled by a Layout Manager object

Programming a GUI

- Step 0: Write all the classes/methods you need to do the "work" of the program and GUI
  - Code to draw pictures, play games, process data, etc.
  - Code that will be run in response to different events
- Step 1: Configure the GUI
  - Create some components
  - Create some containers (which are components too)
  - Pick layout managers for containers
  - Put components in the containers
  - Put everything in a "top level" container (e.g. JFrame)
- Step 2: Set up listeners
  - Create some listener objects to process events
  - Connect listeners to event sources

Step 1 : GUI Statics

- See Weiss Appendix
- See Swing tutorials
- See Capture the Flag code

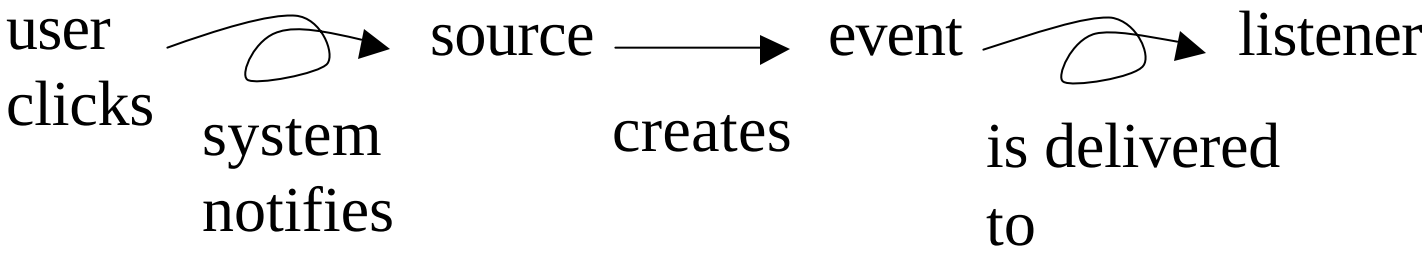
Step 2: GUI Dynamics

- Some common Events:

EventObject	java.util
AWTEvent	java.awt
ActionEvent	java.awt.event
ComponentEvent	java.awt.event
InputEvent	java.awt.event
MouseEvent	java.awt.event
KeyEvent	java.awt.event

- And their reasons and sources:
  - click a JButton → ActionEvent
  - change text in JTextComponent → TextEvent
  - select a JMenuItem → ActionEvent
  - if X generates event E, so can subclasses of X

Step 2: GUI Dynamics



- Diferent kinds of listeners:
  - ActionListener, TextListener, KeyPressListener, etc.
- Any object can be a listener
  - Just needs to implement interface ActionListener or …
- Which object should listen?
  - Create inner class on the fly
  - Make the top-level container be the listener for everything
  - Make the “game” object be the listener for everything
  - mixture of strategies

Learning GUIs

- Look at lots of examples
- Look through the API
  - javax.swing.* (most useful)
  - java.awt.event.* (used extensively, even with swing)
  - java.awt.* (underrlays everything)
- There are classes to do just about everything