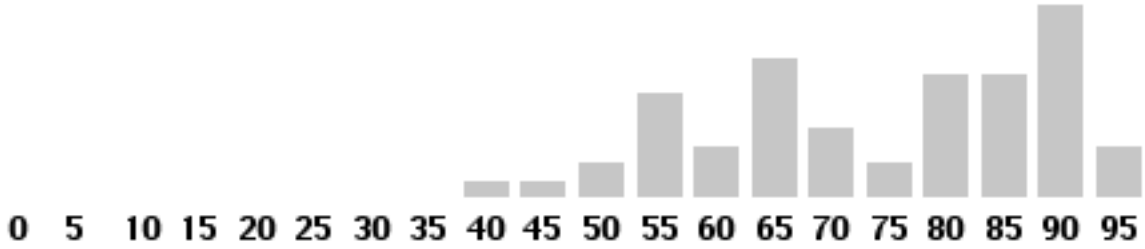
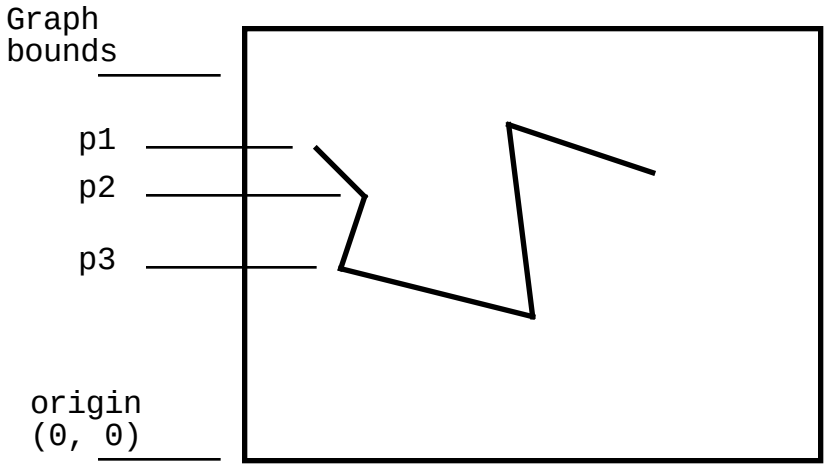
CS193j Solution



I thought the exam was fairly difficult, so I was pleased that most people were able to come up with reasonable looking code for most of the exam. The median score was 80. As you can see from the above histogram, the scores were rather spread out. This document includes the original text of the exam with solution notes added.

1. Swing/GUI (20 points)

For this problem, you will write a Graph class. Graph is a subclass of JComponent. Given a sequence of (x, y) points, p1, p2, ...pn, Graph draws the sequence of lines p1-to-p2, p2-to-p3, and so on. The (x, y) points should be drawn with the origin in the lower-left corner, and ignore the issue of (x, y) points that are larger than the current width or height of the Graph. Just draw all the lines, and the too-large parts of the drawing will be clipped.



To help you get started, the client code to create and set up the Graph is given in the static main() below. The Graph should support...

- Constructor -- takes a pointer to the status JLabel
- add(int x, int y, String text) -- add a point to the Graph. Called by main() to load data into the Graph. Each point may have an associated text string, or it may be null. The text is used for mouse clicks.
- Mouse clicks: if the user clicks within +/-1 pixel of a point and that point has a non-null text, set the "status" JLabel to display the text. The status label is set up for you by the starter code.

```
class Graph extends JComponent {

// provided code: main() creates a JFrame, creates a JLabel,
// creates a Graph, reads (x, y, text) out of a file and uses add()
// to add it to the Graph.
public static void main(String[] args) {
   JFrame frame = new JFrame("Graph");
   JComponent container = (JComponent) frame.getContentPane();
   container.setLayout(new BorderLayout(6,6));

   JLabel status = new JLabel();
   container.add(status, BorderLayou.SOUTH);

   Graph graph = new Graph(status);
   container.add(graph, BorderLayout.CENTER);


   // Assume loop to read file is here, calls add() to add data,
   // text may be null
   int x,y;
   String text;
   ...graph.add(x, y, text)


   frame.pack();
   frame.setVisible(true);
}


// Your code here
// add(), ctors, other methods, ivars, etc.

public void add(int x, int y, String text) {




Solution  notes:

*  Obviously, you need some kind of data structure to hold all the points.
   An ArrayList is a good choice here.
```

* Each point has an x-coordinate, a y-coordinate, and some text (which could be null). It's probably best to make this into a separate class (say TextPoint), so you can just add TextPoint objects into the ArrayList. An alternative is to use Point objects (which just have x and y), and use these as keys to a HashTable where the texts are stored. Either method is easier than using two or three ArrayLists to hold different attributes for each point (although no points were taken off for doing this).

* The constructor needs to initialize the ArrayList (or you'll get a NullPointerException when you try to add()).

* There also needs to be a MouseListener that figures out what to do when the mouse is clicked. In this case, when a click occurs, we go through our ArrayList until we find a point that is close enough to the clicking point, and call JLabel.setText() when a match is found (and the corresponding text is not null). This should be in the constructor as well.

* The line drawing needs to be done in paintComponent(). All this needs to do is to iterate through the points and draw the lines.

* One sublety that many people didn't get right: the origin is at the bottom-left corner, not the upper-left corner (which is what Swing assumes). So you need to do some coordinate-mapping. The easiest way to do this correctly is to map the coordinates within the add() method; then both the drawing and the mouse-clicking will operate under the new coordinate system. If you only mapped coordinates in paintComponent(), then clicking also needs to do the transform. The transform is simply to subract the y value from the component height:

  The correct mapping is (x, y) -> (x, height - y)

2. Inheritance (10 points)

There are two types of grad-student: Sleepy, Grumpy. Grad students are mostly similar, but a little bit different. All grad students have a current happiness factor which is between in 10 and -100 (initially 0). They all respond to the flame() message when they are flamed by e-mail. Here's what they do when flamed:

1. Print "ouch" and decrement happiness by 1, unless it is already at -100. Sleepy grad students are a little different— they go through the above process twice.

2. Then they read their favorite newsgroup to relax (print "reading"). This causes the happiness factor to go up by 4, although it never goes above 10. In addition, the Grumpy grad student posts to the newsgroup after reading (print "posting").

Write the code for Sleepy, Grumpy, and any support classes. You may omit constructors, and don't worry about the public/private/etc. keywords. Just write the code required by the flame() message, and use inheritance to minimize code repetition.

```
Solution notes: the class structure should look something like

class Grad { ... }
class Sleepy extends Grad { ... }
class Grumpy extends Grad { ... }

The Grad superdlass could be abstract or not (the question
did not specify)

There are many ways to implement flame() correctly.  A solution that does
so without repeating code unnecessarily got full credit.

abstract class Grad {
  int happiness;
  public Grad() { happiness = 0; }
  public void flame() {
    ouch();
    reading();
  }
  public void ouch() {
    System.out.println("ouch");
    happiness--;
    if (happiness < -100)
      happiness = -100;
  }
  public void reading() {
    System.out.println("reading");
    happiness += 4;
    if (happiness > 10)
      happiness = 10;
```

```
    /* Note: depending on how you interpreted the problem, you may have
       left happiness unchanged if the original value was 7, 8, or 9. */
  }
}

class Sleepy extends Grad {
  public void ouch() {
    super.ouch();
    super.ouch();
  }
}

class Grumpy extends Grad {
  public void reading() {
    super.reading();
    posting();
  }
  public void posting() {
    System.out.println("posting");
  }
}
```

3. Threading (20 points)

For this problem, you will use threading to speed up a decryption process.

Suppose there is an existing static method `int decrypt(String key)` that attempts a decryption using the given key. It returns 0 if the decryption was not successful, or a non-zero int code on success. The decryption process is very slow.

Write a class SuperCrypt that can use multiple CPUs to do decryption concurrently. As always, you are free to add ivars, methods, inner classes, and so on.

- In its constructor, SuperCrypt takes an array of String keys to try. (code provided)
- The check() method should fork off 4 worker threads. Collectively, the workers should try to decrypt all the keys. The check() method should return 0 if all of the decrypts return 0. If a key decrypts successfully (non-zero), then check() should return its non-zero code. Check() should return immediately when a non-zero code is found.
- When done with one string, each worker should get the next available string from the array.

- When a successful decrypt is found, we could interrupt() the other workers. However, **we will not do this**. It is acceptable to allow the other workers to continue to run.

```
class Crypt {
   // Attempts to decrypt with the given key.
   // Returns 0 on failure, otherwise a non-zero success code
   public static int decrypt(String key) {...}
}


public class SuperCrypt {
   private String keys[];

   public SuperCrypt(String[] keys) {
      this.keys = keys;
   }


   public int check() {



Solution notes:

Minor errors -1 each, -5 max.
* Run() not public
* Instantiates instance of crypt() to use decrypt()
* returns wrong code
* Calls wait/notify() on object without holding lock for it (inside a
synchronized(..) {} block or in a synchronized function)


Major errors
* Traversed keys array in check() instead of in each thread in a synchronized
fashion (-5)
* Does decryption in groups of four (-2)
* Did not traverse through entire array (-7)
* Did not synchronize keys-array traversal properly (-4)
* Got around synchronization by assigning each thread a specific part of the key
space (-4)
* Instanciated keys.length number of threads, or did not reuse the four threads
(-2)
* Check returns after all keys are tried (did not short cut return) -7
* Did code that would have short-cut the return correctly but with minor
mistake -2
* Check could return before all keys are tried (did attempt shortcut)
  but have to wait for unncessary decryptions to finish -4
* Busy waiting -4
* wait() without proper test for condition -4
* Did not synchronize the "done" condition properly (-2)
* Used fix-time sleeping to wait for "done" -7
* Did not wait for proper "done" condition -7
* returns from check prematurely -4 (specifically, it should not return from
check until either all
  threads are done or one of the decrypt is successful.  Just testing that the
```

```
last key has been
  consumed or decrypted is not enough, since that could happen before one of
threads decrypting
  keys prior to the last key is finished)
* Could never return from check -5 to -7 depending on how it happens
* Tried to use interrupt() with join() to short cut the early return however
incorrectly -4
* Started more than four threads at the same time  -2
* Assumes array references are atomic -0
* Calls start() again to rey to restart thread -2
* Each thread ran decrypt on every key -10
* Calls missing function -2



/**
 * SuperCrypt.java
 *
 * Title:                        SuperCrypt
 * Description:         . In its constructor, SuperCrypt takes an array of
String keys to try.
(code provided)
. The check() method should fork off 4 worker threads. Collectively, the
workers should try to decrypt all the keys. The check() method should
return 0 if all of the decrypts return 0. If a key decrypts successfully
(non-zero), then check() should return its non-zero code. Check() should
return immediately when a non-zero code is found.
. When done with one string, each worker should get the next available
string from the array.
. When a successful decrypt is found, we could interrupt() the other
workers. However, we will not do this. It is acceptable to allow the
other workers to continue to run.
 * @author                       yl314
 * @version
 */

class Crypt {
// Attempts to decrypt with the given key.
// Returns 0 on failure, otherwise a non-zero success code
// OF COURSE EVERYTHING HERE IS FAKE!!
    private static java.util.Random r = new java.util.Random();
    public static int decrypt(String key) {
        for (int i=0; i<5; i++) {
            System.out.println(Thread.currentThread().getName() + ": Decrypting
key " + key + ", step " + i);
            try {
                Thread.sleep(r.nextInt(1000));
            } catch (InterruptedException e) {}
        }
        if (key.equals("The Key")) {
            return 1;
        }
        return 0;
    }
}

public class SuperCrypt {
    private String keys[];      // track the keys
```

```
private int keyIdx;         // which key is being tried next
private int code;           // final result
private int threadsDone;    // How many threads/workers has finished

public SuperCrypt(String[] keys) {
    this.keys = keys;
}

public int check() {
    // Reset ivars
    keyIdx = 0;
    code = 0;
    threadsDone = 0;

    // Instanciate and start threads
    for (int i=0; i<4; i++) {
        Thread worker = new Thread() {
            // Anonymous instance of an inner subclass of thread

            // Thread code
            public void run() {
                String key;
                // no need to synchronize reading the ivar code
                while (code==0 && null != (key=getKey()) ) {
                    // We haven't found the key and we still have keys
                    // to try
                    int code = Crypt.decrypt(key);

                    // if code is non-zero, we have to wake up the
                    // main thread
                    if (code != 0) {
                        setCode(code);
                    }
                }

                // Log this worker as being done
                threadDone();
            }
        };

        //Start the thread
        worker.setName("Worker " + i);
        worker.start();
    }

    // Wait for the final result
    return getFinalCode();
}

// Must be synchronized because of wait()
public synchronized int getFinalCode() {
    // Note that we have to check the number of
    // finished threads instead of checking whether
    // the last key has been consumed or finished checking,
    // because the a non-zero result code may still
    // be produced by one of the executing threads!
    // (Note the case when checking the last key starts
    // and/or finishes earlier than some of the other keys.)
```

```
    //
    // We could avoid having a threadsDone ivar if
    // we send each thread the isAlive() message,
    // however it is not as efficient.
    //
    // Also note that the while() loop is technically
    // unnecessary because in this design no call to
    // notifyAll() could be a false positive; however
    // it is still a good practice to leave it in.
    //
    // There is also a debate on whether the
    // InterruptedException generated by wait() should be
    // handled inside the while loop or outside the while
    // loop.  It would depend on the semantic of interruption
    // In general however it is safer for it to be inside
    // the loop
    while (code == 0 && threadsDone != 4) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    return code;
}

// Must be synchronized because of notifyAll()
// We use notifyAll() because the condition being
// notified on would not change; however since in
// this problem only one thread could be possibly waiting
// notify() would be fine.
public synchronized void setCode(int code) {
    this.code = code;
    notifyAll();
}

// Must be synchronized because of increment
// to threadDone and notifyAll()
public synchronized void threadDone() {
    threadsDone ++;
    if (threadsDone == 4) {
        notifyAll();
    }
}

// Must be synchronized to serialize the
// traversal of the keys array
public synchronized String getKey() {
    if (keys == null || keyIdx >= keys.length) {
        // No keys available or left
        return null;
    }
    return keys[keyIdx++];
}

// Main entry point
static public void main(String[] args) {
    String [] keys = {"key1", "key2", "key3", "key4", "key5", "The Key"};
    System.out.println("Final result: " + (new SuperCrypt(keys)).check());
}
```

```
}
```

4. Networking (20 points)

Suppose you have the Binky class that does some simple networking.

```
public class Binky {
   public Binky(String url) { ... }

   public BufferedReader connect() throws BinkyException { ... }
}
```

For the BinkyChecker class below, the readAll() method is given an array of URLs to connect to. Define readAll() so it forks off a worker thread for each URL. Each worker should create a Binky for its URL, make a connection, and read all the lines of text from it. If a worker encounters an error, it should exit silently. When all of the workers are finished, readAll() should return a Collection (an ArrayList for example) of all the lines of text the workers read. The lines of text may be in any order. Use join() to wait for the workers. If interrupted, return the strings gathered so far.

Reminder: BufferedReader responds to readLine():
`String readLine() throws IOException;`

Solution:

```
public class BinkyChecker {

   Collection readAll(String[] urls) {

      // allocate storage
      Thread [] threads = new Thread[urls.length];
      result = new ArrayList();

      // launch all the workers
      for (int i=0; i<threads.length; i++) {
         threads[i] = new Worker(urls[i]);
         threads[i].start();
      }

      // wait for them to finish
      try {
         for(int i=0; i<threads.length; i++) {
            threads[i].join();
         }
      }
      catch(InterruptedException e) { }

      return(result);
   }
```

```
   private ArrayList result;

   // Workers use this to add lines -- synchronizeed
   // Note: must synch on the BinkyChecker, not thw worker.
   private synchronized add(String line) {
      result.add(line);
   }

   class Worker extends Thread {
      private Binky binky;
      private Worker(Binky b) {
         binky = b;
      }

      public void run() {
         try {
            BufferedReader read = binky.connect();
            String line;
            while ((line = read.readLine()) != null) {
               add(line);  // synch add line
            }
            read.close();
         }
         catch (BinkyException e) { }
         catch (IOException e) { }
      }
   }
```

5. Misc, parts a-f (30 points)

a. Suppose you are adding a capability to the XEdit class from HW4. You would like to iterate over the tree, searching for a node of the given name, returning true if found and false otherwise. So for the following XML, searching for node "a" "b" "c" or "foo" will return true, and others will return false.

```
<a>
 <b> <c>blah blah</c> </b>
 <b> <c>yatta yatta</c> <foo>woo hoo</foo> </b>
</a>


Reminder:
int type = node.getNodeType() -- returns TEXT_NODE, ELEMENT_NODE
String s = node.getNodeName() -- returns tag name
NodeList list = node.getChildNodes();
int len = list.getLength() -- list length
Node child = (Node)list.item(i) -- node from list


public boolean search(Node n, String target) {
```

```
   if (n.getNodeName().equals(target)) return(true);  // base case

   if (n.getNodeType() == ELEMENT_NODE) {
      NodeList list = n.getChildNodes();
      int len = list.getLength();
      for (int i=0; i<len; i++) {
         if (search((Node)list.item(i), target)) return(true);
         // note: break out immediately on found
      }
   }
   return(false);
}

Key issues:
-iterate over children
-recur / compute correctly
-on found, return right away

equals vs. == on strings -1
short circuit -1
missing return(true) -1
missing bottom return(false) -0
It turns out text and element nodes both respond to
getChildNodes() and getNodeName(), so you can sloppy
with the node type and it still works.
getChildNodes() on text node -0 -- returns a 0 length collection
getNodeName() on elem node -0 -- returns empty string
```

b. Suppose the classes A and B are defined, and A is the superclass of B. True or false: the following line will compile.

```
   A a = new B();

True -- this is just basic upcasting (2 points)
```

c. Suppose you have a constructor of a Swing component that creates and installs a label and a button (coded below). Add a listener to the button so that when it is clicked, a thread is forked off that waits 10 seconds or until interrupted, and then sets the text of the label to "hello".

Reminder: Thread.sleep(milliseconds)

```
class MyComponent extends JComponent {

   public MyComponent() {

      // Create and install the button and label
      JButton button = new JButton("Hello");
      add(button);
      JLabel label = new JLabel();
      add(label);

      // YOUR CODE HERE
```

```
      // add listener to the button


      final JLabel temp = label;

      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               Thread t = new Thread() {
                  public void run() {
                     try {
                        Thread.sleep(10000);
                     }
                     catch (InterruptedException ignored) {}
                     SwingUtilities.invokeLater(   // Note: this is key (-3)
                        new Runnable() {
                           public void run() {
                              temp.setText("hello");
                           }
                        }
                     );
                  }
               };
               t.start();
            }  // run()
         }     // ActionListener
      );



Key points:
-Create an action listener, override actionPerformed
-Use Thread.sleep to wait
-Use SwingUtilities.invokeLater() w/ runnable, since not on swing thread
```

d. Optimization

Suppose you have a Foo class that contains an int[] array. The **len** ivar marks how full the array is -- the ints in the range 0..len-1 are valid, and ints beyond that range are not valid. The **max** ivar stores the max int value in the array, or -1 if the array is empty. The computeMax() method below is correct -- it will compute the correct value of **max** based on the array. We assume that the array does not change while computeMax is running (i.e. we only support one thread).

```
public class Foo {
   private int ints[];     // an array of ints
   private int len;
   private int max;        // the current, max int value in the ints

   public void computeMax() {
      int i=0;
      max = -1;
      for (i=0; i<len; i++) {
         if (ints[i] > max) max=ints[i];
      }
   }
```

```
}
```

Here is a second copy of the outer shell of computeMax() -- keeping the algorithm the same (look at all the ints, compute their max) re-write the loop so that it is likely to run faster. (Of course, how much faster the new version runs will depend on the specific VM and its optimizer.)

```
Solution: pull ivars into locals

   public void computeMax() {
      int i=0;

      // Solution: don't use ivars, use stack vars
      int tLen = len;
      int tMax = -1;
      int[] tInts = ints;
      for (i =0; i<tLen; i++) {
         if (tInts[i] > tMax) tMax = tInts[i];
      }

      // now store back to ivar
      max = tMax;
   }

Criteria:
5/5: pull two things into locals (must be values used every iteration)
-2: pull one thing into local
-4: some minor optimization
caching ints[i] in a local is not a great optimiztion, since the value
is not used very often
```

e. Suppose we are given a Binky object that supports messages a(), b(), and c(), each of which returns an int. However, a(), b(), and c() can each throw a BinkyException. In the foo() message below, send the a(), b(), and c() messages to the Binky object, and add the ints they return to the "sum" ivar. However, if any of the messages throw an exception, the sum should not be changed -- it should be left with the same value it had before foo() was called.

```
class Client {
   private int sum;

   //
   void foo(Binky x) {
      // YOUR CODE HERE
      int temp = 0;
      try {
         temp += x.a(); // these may throw
         temp += x.b();
         temp += x.c();

         sum += temp;      // now safe to update
      }
```

```
      catch (BinkyException e) {
         // do nothing
      }
   }

criteria:
-ok if do = instead of +=
```

f. Suppose you have a very simple Binky class that just stores a single number that never changes. Define a countBinkys() method that returns the number of Binky objects created so far. So if the program has created3 Binky objects, countBinkys() sent to any of them will return 3. Add your solution into the following structure...

```
public class Binky {
   private int num;



   public Binky(int num) {
      this.num = num;



   }


   public int getNum() {
      return(num);
   }


   // Returns the number of binkys created up until now
   public int countBinkys() {



}


Solution:
-add a "static" int count
-count++ in the ctor
-countBinkys just returns count
```