A7: remember, "Don't look at any other student/group's code, in any form; don't show any other student/group your code".
(The similarity software turned up a few problems on A6, which we are about to start the Academic Integrity violation process for. Note that the checker essentially performs variable-name substitutions, etc., so syntactic modification of the same original program is generally flagged.)

No labs this week, no TA office hours Wed-Fri, see consultant calendar for the updated schedule.

There are "labs" next week, but they will serve as office hours plus an optional exercise on exceptions (covered on final).

Final: Friday Dec 10th, 9-11:30am, Statler Auditorium. 
Register conflicts (same time, or 3 finals in 24 hours) on CMS assignment "final exam conflicts" by Tuesday November 30th.

Please check that your grades on CMS match what you think they are. [For lab-grade issues, contact your lab TA, not the instructors.]
Today’s (and next week's lab’s) topic: when things go wrong (in Java)

Q1: What happens when an error causes the system to abort?
   (NullPointerException, ArrayIndexOutOfBoundsException, …)

   *Understanding this helps you debug.*

Q2: Can we make use of the "problem-signaling mechanism" to handle unusual situations in a more appropriate way?

   *Understanding this helps you write more flexible code.*

   Important example: a “regular person” enters malformed input.

   It is sometimes better to warn and re-prompt the user than to have the program crash (even if the user didn’t follow your exquisitely clear directions or preconditions).
/** Exception example */

public class Ex {

  public static void first() {
    second();
  }

  public static void second() {
    third();
  }

  public static void third() {
    int x = 5 / 0;
  }

}

Call: Ex.first();

System prints the call-stack trace:

ArithmeticException: / by zero
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)

Same structure as our demo:
StockQuoteGUI's actionPerformed calls StockQuote's getQuote, which calls In's constructor and readAll methods.
errors (little e) cause Java to throw a Throwable object as a “distress signal”

Exceptions are signals that intervention may still be possible; they can be “handled”.

Errors are signals that things are beyond help.
/** Exception example */
public class Ex {
  public static void first() {
    second();
  }

  public static void second() {
    third();
  }

  public static void third() {
    int x = 5 / 0; // ArithmeticException: / by zero
  }
}

Call: Ex.first();

Throwable object --- request for help --- is thrown to successive “callers” until caught by a method that declares that it can provide help. (This is a form of communication between methods.)

In this example, the Java system catches it because nothing else does, it just prints the call-stack trace and aborts.

ArithmeticException: / by zero
  at Ex.third(Ex.java:13)
  at Ex.second(Ex.java:9)
  at Ex.first(Ex.java:5)
How can we catch/handle Throwables? With Try/catch blocks.

/** = reciprocal of x. Throws an ArithmeticException if x is 0. 
  (Assume this is third-party code that you can't change.) */
public static double reciprocal(int x) {
    ...;
}

/** = reciprocal(x), or -1 if x is 0. 
  Assume you can't change this spec. */
public static double ourReciprocal(int x) {
    try {
        return reciprocal(x);
    } catch (ArithmeticException ae) {
        return -1;
    }
}

Execute the try-block. If it finishes without throwing anything, fine.
If it throws an ArithmeticException object, catch it (execute the catch block); else throw it out further.
Try-statements vs. if-then checking

/** = reciprocal(x), or -1 if x is 0*/
public static double ourReciprocal2(int x) {
    if (x != 0) {
        return reciprocal(x);
    } else {
        return -1;
    }
}

The previous slide was just to show try/catch syntax. Use your judgment:
• For (a small number of) simple tests and “normal” situations, if-thens are usually better. For more “abnormal” situations, try-catches are better.
[In this case, given the specification, if/then is maybe slightly better; anyone reading the code would expect to see a check for 0.]
• There are some canonical try/catch idioms, such as processing malformed input.
How can we create our own signals?
• We can create new Throwable objects, via new-statements.
• We can write our own Exception subclasses (see demo)

```java
Ex.initArray(-1);
```

java.lang.IllegalArgumentException: initArray: bad value for n, namely -1
at Ex.initArray(Ex.java:20)

```java
/** Illustrate exception handling*/
public class Ex {

    /** = array of n -1’s.
    Throws an
    IllegalArgumentException if n <=0*/
    private static int[] initArray(int n) {
        if (n <= 0) {
            throw new IllegalArgumentException
                (“initArray: bad value for n, namely ” + n);
        }
        …
    }
```
/** Class to illustrate exception handling */
public class Ex2 {
    public static void first() throws OurException {
        second();
    }
    public static void second() throws OurException {
        third();
    }
    public static void third() throws OurException {
        throw new OurException("intentional error at third");
    }
}

A technical point: we may need a “throws” clause to compile

tell the system that an OurException might get thrown

Don’t worry about whether to put a throws-clause in or not. Just put it in when it is needed in order for the program to compile. [runtime exceptions don't require a throws-clause; other kinds do]