Com S 227
Fall 2020
Topics, review problems, and process for Exam 2
Thursday, October 29, 8:15 pm

General information

The format of the exam is similar to Exam 1. It will again be conducted on Canvas using the lockdown browser, and proctored via Zoom. Do the following immediately if you haven't already:

1. *Check Zoom:* Make sure you have the Zoom app on your phone. Join a test meeting using your phone at https://zoom.us/test and make sure the microphone and camera are working. Make sure you know how to "raise your hand". You'll need to keep the microphone un-muted during the exam, but you can turn down/off your speaker. (We'll use the chat window to get your attention if we have to make an announcement.)

2. *Check Location:* Figure out where you're going to take the exam and how you will prop up your phone so that you are visible. You will need to be in a private room, at a desk or table that is clear except for your phone and computer and one piece of scratch paper. It needs to be a quiet place, because you'll have your microphone on during the exam. Make sure your phone is set up to use the local wifi.

3. *Check Canvas:* Try the "practice exam" on Canvas (under "Quizzes) to make sure you can successfully use the Respondus lockdown browser.

However, based on feedback from Exam 1, we are making the following changes:

- Exam time limit will be 75 minutes.
- Check-in and check-out process will be more streamlined:
  - Once everyone is in the zoom room, we will ask you all to simultaneously provide a panorama of your room and desktop[1] - after that, we'll give out the exam password.
  - You'll show your ID at the *end* of the exam, before leaving the zoom room. **Important:** *If you leave the meeting without showing your ID, we will not grade your exam.*

If you cannot run the lockdown browser or if you cannot participate in the zoom meeting with your smartphone, please contact your instructor asap to make an alternative arrangement.

[1] If you uncomfortable about other people seeing your room, please send a private chat to the TA in your zoom meeting, and we can anonymously move you to a breakout room for that portion of the check-in.

Summary of Exam topics

The exam covers everything done in lectures *roughly* through Monday, October 26 and in labs through lab 9. The exam will concentrate on topics covered since the first exam. Most problems will require loops, arrays, lists, or recursion. However, since the subject matter is cumulative, *knowledge of everything from the first exam is assumed*. If you had any trouble with Exam 1 you may also want to refer to the review sheet for that exam. The list below is an overview of the main topics. You will not be tested on JUnit or the Eclipse debugger. **The exam does not cover sorting. We expect that about 15% of the exam will be on recursion.**

- Loops
- Arrays
- Two-dimensional arrays
- **`ArrayList`**s
- Array, list and string algorithms – counting, searching, inserting, deleting, etc.
- Wrapper classes
- Reading and writing text files
- Using **`Scanner`** to parse text
- Recursion

You do not have to memorize methods from the Java API. You should know **`System.out.print`** and **`System.out.println`** . You should know how to use **`String`**, **`Scanner`**, **`Math`**, **`Random`**, **`File`**, and **`ArrayList<E>`**. You should know how to read and write text files and how to read input from **`System.in`**. For specific methods from these classes that are likely to be needed, we'll provide you with the minimal one-sentence descriptions from the API, as seen at the end of this review sheet.

How to prepare

The most important thing you can do to prepare for an exam like this is to *practice solving problems and writing code*. You should write your solutions first in an ordinary text editor (since that is the format of the exam). Then check your work by copying what you've written into Eclipse and testing it. We encourage you to post and discuss your sample solutions in Piazza. **Please remember that we will NOT grade you on formatting or indentation, so don't waste too much time trying to format your code.**

You will need to write quickly and accurately, and to recognize correctly written code without the help of Eclipse. You will *not* be expected to write comments or documentation or define symbolic constants for numbers. (However, including brief comments can sometimes help us interpret what you were trying to do, in case you have errors.)

More practice

Remember that there are many, many more problems to practice on in the book. Another entertaining way to practice writing Java code is the interactive site http://codingbat.com/java. There are lots of problems involving arrays and strings and recursion. (However, note that codingbat.com has no problems involving ArrayList or Scanner.)

Some practice problems

You are encouraged to post your sample solutions on Piazza for discussion.

1) Some loop and array examples. Write a static method for each that has all needed information as parameters. (You don't have to create an entire class.)

a) Given an array of doubles, return the average.

b) Given a sentence, find and return the longest word.

c) Given a string, return the string with all spaces and non-alphabetic characters replaced by the character '#', e.g. "`Hello, world!`" becomes "`Hello##world#`" (You can use the static method `Character.isAlphabetic(char c)` to determine whether a given character is alphabetic.)

d) Interest is added to the balance of a savings account each month. Write a method that, given an annual interest rate and an initial balance, determines how many months it takes for the balance to double.

e) Given an `ArrayList` of `Integer`s, determine whether they are in increasing order.

f) Given a string, return the index of the first vowel (or -1 if there are none).

g) Given a string, determine whether any letter appears two or more times.

h) Given an array of ints, reverse its contents (the method must *modify* the given array and returns void).

i) Given an array of integers, determine whether the array is a permutation of the numbers 0 through n – 1, where n is the length of the array. (A permutation means that each number appears exactly once.)

j) Given a number n, print out a reverse diagonal line of n stars:

```
    *
   *
  *
 *
*
```

k) Given a 2D array of doubles, return a 1D array whose *i*th entry is the average of the *i*th column.

l) Given a 2D array of ints, find the column with the maximum sum.

m) Given positive integers `w` and `h` and an `int[]` array `arr` of length `w * h`, return a 2d array with `h` rows and `w` columns that contains the numbers in `arr`, listed left-to-right and top-to-bottom.

n) Given an integer n, return the smallest prime number that is larger than n. (A number is *prime* if it is greater than 1 and has no divisors other than 1 and itself.)

o) Given an array of positive integers, "collapse" the array to remove duplicates, and fill in the unused cells at the end with zeros. For example, given the array [5, 4, 5, 6, 4, 2], after this method executes the array should be [5, 4, 6, 2, 0, 0]. The method *modifies* the given array, and returns `void`.

p) Given an array of positive integers, return a *new* array containing the same numbers, in the same order, but without duplicates. For example, given the array [5, 4, 5, 6, 4, 2], the method returns [5, 4, 6, 2].

q) Given an instance of Random, generate a list of numbers between 0 and 99, inclusive, stopping when the same number has appeared more than once. The method returns a list of all the generated numbers. (The ArrayList `contains()` method might be useful.)

r) Given a string, return a new string with the words in the opposite order. (E.g. given "He's dead, Jim", return "Jim dead, He's".)

s) Given an array of ints, swap the first half with the second half. The method *modifies* the given array and returns void. If the length is odd, the middle element is not moved. For example, if called on the array [10, 20, 30, 40, 50, 60, 70], after the method executes the array would be [50, 60, 70, 40,10, 20, 30].

2) Write a program that will remove all the //-style comments from a Java file. Your program should prompt the user to enter the name of the input file. The output file should have the same name as the input file but should end with the extension ".out" instead of ".java". The output file should be the same as the input file except that all //-style comments are removed. (You can assume that the sequence "//" does not occur inside any String literals within the program.)

3) Trace the execution of the call **`enigma(12,0)`** and show all output that is produced.

```
public static void enigma (int x, int y) {
  while (x > 0){
    if (x % 2 == 0){
      y = y + 1;
    }
    else {
      x = x + 2;
    }
    x = x - y;
    System.out.println(x + ", " + y);
  }
}
```

4) Write a static method **`getPassword`** that will read a user's password from System.in. The user has to enter the password twice. The method should iterate the following steps *as many times as necessary* until the user successfully enters two values that match:

1. prompt the user and read the password
2. prompt the user and read the password again
3. check that the second entry matches the first

(The method has no parameters and should return the entered password as a String.)

5) Suppose that a file contains lines with a name and phone number having the format

name, xxx-xxx-xxxx

Also suppose you have a class called **`Contact`** with the constructor and methods below:

```
public Contact(String givenName, String givenPhoneNumber)
public String getName()
public String getPhoneNumber() // returns phone number as String
```

Create a class **`ContactDirectory`** suitable for storing a list of **`Contact`**s. The **`ContactDirectory`** should have the methods:

```
// adds the given contact to the directory
void addContact(Contact c)

// add all contacts from a file of the above form
void addFromFile(String filename) throws FileNotFoundException

// returns phone number for name, or null if name is not in the list
String lookUp(String name)
```

6. a) Given the method **mystery** below, determine the output printed by the call **mystery(10)**. (It might be helpful to sketch the call stack as you go.)

```
public static void mystery(int x)
{
  if (x == 1)
  {
    System.out.println("pooh");
  }
  else if (x % 2 == 0)
  {
    System.out.println(x);     // (**)
    mystery(x / 2);            // (*)
  }
  else
  {
    mystery(x - 1);
  }
}
```

b) Suppose we have a method **mystery2** that is the same as mystery except that the lines labeled (*) and (**) are switched. Trace the call **mystery2(10).**

c) What happens when you call **mystery(-1)** ? Explain.

7. a) A child named Beatrice is jumping along on a floor consisting of rectangular tiles. She can jump one tile, two tiles, or three tiles at a time. Write a recursive method to determine the number of different ways she can cross n tiles.

b) The streets of Manhattan are laid out in a rectangular grid. You need to walk to a destination that is **r** blocks to the south and **c** blocks to the east of your current location (where **r** and c are both non-negative). Assume that you never walk west or north. Write a recursive method that determines how many different routes can you take to your destination. (For example: whenever **r** is zero or **c** is zero, there is only one possible route. If both are nonzero, you can start out going one block east, or going one block south.)

c) Write a method that, given a directory (as a **File** object), returns a list of names of the files beneath it whose names end with ".java" (within it, within its subdirectories, and so on). It might help to define a recursive helper method of the form

```
private void findJavaFiles(File file, ArrayList<String> results)
```

Note that the class **java.io.File** includes the following methods:

String getName() – returns the name of this **File**
boolean isDirectory() – returns true if this **File** represents a directory
File[] listFiles() – returns an array of all items (files and directories) in this **File**; returns null if this **File** is not a directory

Excerpts from the Java API documentation

Excerpt from documentation for java.lang.String

`char`	`charAt(int index)` Returns the char at the specified index.
`boolean`	`equals(String anotherString)` Returns true if this string is the same as the given string.
`int`	`indexOf(char ch)` Returns the index within this string of the first occurrence of the given character, or -1 if the character does not occur in this string.
`int`	`indexOf(String str)` Returns the index within this string of the first occurrence of the given substring, or -1 if the given substring does not occur in this string.
`int`	`lastIndexOf(char ch)` Returns the index within this string of the last occurrence of the given character, or -1 if the character does not occur in this string.
`int`	`length()` Returns the length of this string
`String`	`substring(int beginIndex)` Returns a substring of this string starting at beginIndex
`String`	`substring(int beginIndex, int endIndex)` Returns a substring consisting of characters from beginIndex through endIndex - 1
`String[]`	`split(String regex)` Splits this string using the given string expression as the delimiter.
`String`	`trim()` Returns a copy of this string with leading and trailing whitespace removed

Excerpt from documentation for java.util.Random

`int`	`nextInt(int n)` Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

java.util.Random constructors

`Random()` Creates a new random number generator.

Excerpt from documentation for java.lang.Integer

`int`	`parseInt(String s)` Parses the string argument as an integer.

Excerpt from documentation for java.util.ArrayList<E>

`boolean`	`add(E element)` Appends the specified element to the end of this list.
`boolean`	`add(int index, E element)` Appends the specified element at the specified position in this list.
`void`	`clear()` Removes all of the elements from this list.
`boolean`	`contains(E element)` Returns true if this list contains the specified element.
`E`	`get(int index)` Returns the element at the specified position in this list.
`E`	`remove(int index)` Removes the element at the specified position in this list (elements to right shift down)

`boolean`	`remove(Object obj)` Removes the first occurrence of the specified object, returning false if it does not occur
`E`	`set(int index, E element)` Replaces the element at the given position in this list with the specified element.
`int`	`size()` Returns the number of elements in this list.

java.io.File constructor

`File(String filename)` Constructs a new File instance.

Excerpt from documentation for java.util.Scanner

`void`	`close()` Closes this scanner and with associated input stream, if any.
`boolean`	`hasNext()` Returns true if this scanner has another token in its input.
`boolean`	`hasNextDouble()` Returns true if the next token in this scanner's input can be interpreted as a double value
`boolean`	`hasNextInt()` Returns true if the next token in this scanner's input can be interpreted as an int value.
`boolean`	`hasNextLine()` Returns true if there is another line in the input of this scanner.
`String`	`next()` Finds and returns the next complete token from this scanner.
`double`	`nextDouble()` Scans the next token of the input as a double.
`int`	`nextInt()` Scans the next token of the input as an int.
`String`	`nextLine()` Returns all input up to the next line break and advances to the next line.

java.util.Scanner constructors

`Scanner(File source)` Constructs a new Scanner that provides values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new Scanner that provides values scanned from the specified input stream, such as System.in.
`Scanner(String source)` Constructs a new Scanner that provides values scanned from the specified String.