Com S 227
Fall 2017
Topics and review problems for Exam 2
Thursday, November 9, 6:45 pm

Exam locations by last name (same as exam 1):

A-L Hoover 2055
M-Z Troxel 1001

***** You must bring your ID to the exam *****

General information

This will be a 70-minute, timed, pencil-and-paper written exam. No books, no notes, no electronic devices, no headphones, no collaboration. The problems will primarily involve writing Java code or reading and interpreting Java code.

Summary of Exam topics

The exam covers everything done in lectures *roughly* through Wednesday, November 2 and in labs through lab 9. The exam will concentrate on topics covered since the first exam. Most problems will require loops, arrays, or lists. However, since the subject matter is cumulative, *knowledge of everything from the first exam is assumed*. So 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. Numbers in parentheses refer to the **current** textbook. If you have an older edition of the book, see http://www.cs.iastate.edu/~cs227/section_number_mapping.pdf .

You will not be tested on JUnit or the Eclipse debugger. The exam does not cover interfaces.

- Loops (Sections 6.1 – 6.3 and 6.7 – 6.8)
- Arrays (7.1, 7.3)
- Two-dimensional arrays (7.6)
- **`ArrayList`**s (7.7, 7.3)
- Array, list and string algorithms – counting, searching, inserting, deleting, etc. (6.7, 7.3)
- Wrapper classes (7.7)
- Reading and writing text files (11.1)
- Using **`Scanner`** to parse text (11.2)
- Recursion (13.1, 13.2)
- Sorting
  - selection sort algorithm (14.1)
  - merge sort (14.4)

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>`**, but for specific methods from these classes that might be needed, we'll provide you with the minimal one-sentence descriptions from the API. You should know how to read and write text files and how to read input from **`System.in`**.

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 on paper (since that is the format of the exam). Then check your work by typing up what you've written in Eclipse. We encourage you to post and discuss your sample solutions in Piazza.

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 textbook. Each chapter has a number of "self-check" questions at the ends of the sections and a section of "Review exercises" at the end of each chapter.

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. (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) Given the array: `int[] test = {6, 7, 4, 3, 5, 2, 7, 9, 8};` Trace the execution of the call `whatever(test)` and show contents of the array after the method returns.

```
public static void whatever (int[] arr)
{
  int i = 0;
  for (int count = 0; count < arr.length; count += 1)
  {
    if (arr[i] % 2 != 0)
    {
      for (int j = i; j < arr.length - 1; j += 1)
      {
        arr[j] = arr[j + 1];
      }
      arr[arr.length - 1] = 0;
    }
    else
    {
      i += 1;
    }
  }
}
```

5) 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.

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

name, xxx-xxx-xxxx

a) Create a class **`Contact`** suitable for storing a name and phone number (the phone number may be stored as a String). It should include an appropriate constructor and the methods:

```
String getName()
String getPhoneNumber()       // returns phone number as String
int[] getPhoneNumberArray()   // returns phone number as an array of 10 ints
```

b) Create a class **`ContactDirectory`** suitable for storing a list of **`Contact`**s. The **`ContactDirectory`** should include 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 void if name is not in the list
String lookUp(String name)
```

7. 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.

8. a) Rewrite the method **`mystery`** from the previous problem so that it produces the same results as the given version for all positive numbers, but does not use recursion.
b) Do the same for **`mystery2`**.

9. One way to efficiently calculate integer powers is by repeated squaring:

$$x^p = \begin{cases} 1, & \textit{if p is zero} \\ \left(x^{\frac{p}{2}}\right)^2, & \textit{if p is even} \\ x\left(x^{\frac{p-1}{2}}\right)^2, & \textit{if p is odd} \end{cases}$$

(For example, to calculate 2^{10}, first calculate 2^5 and then square the result.) Write a recursive function **`pow(x, p)`** that uses this strategy. (Assume that p >= 0.)

10. 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.)

11. 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

12. There are many variations of the mergesort algorithm having different strategies to reduce memory usage and array copying. Suppose that you are *given* a **`merge`** method with the following declaration:

```
/**
 * Merges two sorted subarrays of a given array, storing the result back in
 * the given array. That is, when the method is called,
 *
 *      arr[start] through arr[mid] is already sorted, and
 *      arr[mid + 1] through arr[end] is already sorted
 *
 * When the method returns, arr[start] through arr[end] is sorted.
 */
private static void merge(int[] arr, int start, int end, int mid)
```

Suppose that you are also given the public method:

```
public static void mergeSort(int[] arr)
{
  mergeSortRec(arr, 0, arr.length - 1);
}
```

Write the following recursive helper method that will sort a given subarray using the merge sort algorithm:

```
/**
 * Performs a recursive merge sort of the subarray consisting of
 * arr[start] through arr[end].
 */
private static void mergeSortRec(int[] arr, int start, int end)
```

Note: This problem is NOT asking you to rewrite the merge() method! You can find a sample solution for the problem above, along with two other variations of mergesort, in the November 1 entry here (see MergeSortAlt2.java): http://web.cs.iastate.edu/~smkautz/cs227f17/topics.html

13. Suppose you also have a method that performs a selection sort on a specified subarray:

```
/**
 * Performs a selection sort on the elements
 * arr[start] through arr[end] without modifying the others
 */
selectionSortRange(int[] arr, int start, int end)
```

Modify the base case for the merge sort algorithm so that for any subarray of length 5 or less, it uses a selection sort instead of making a recursive call.

14. Write the method `selectionSortRange` of the problem above.