CIS 110 — Introduction To Computer Programming

November 21st, 2011 — Exam 2

Answer key for review problems

CIS 110 Exam 2 Instructions

- You have 50 minutes to finish this exam. Time will begin when called by a proctor and end precisely 50 minutes after that time. If you continue writing after the time is called, you will receive a zero for the exam.
- This exam is *closed-book, closed-notes, and closed-computational devices* except for a one page sheet (8.5" by 11") of double-sided notes.
- When writing code, the only abbreviations you may use are as follows:

$$\begin{aligned} \text{System.out.println} &\longrightarrow \text{S.O.PLN} \\ \text{System.out.print} &\longrightarrow \text{S.O.P} \\ \text{System.out.printf} &\longrightarrow \text{S.O.PF} \end{aligned}$$

  Otherwise all code must be written out as normal.

- Please do not separate the pages of the exam. If a page becomes loose, please make sure to write your name on it so that we don't lose it, and use the provided staplers to reattach the sheet when you turn in your exam.
- If you require extra paper, please use the backs of the exam pages or the extra sheet paper provided at the end of the exam. Clearly indicate on the question page where the graders can find the remainder of your work (e.g., "back of page" or "on extra sheet").
- If you have any questions, please raise your hand and an exam proctor will come to answer them.
- When you turn in your exam, you will be required to show ID. If you forgot to bring your ID, please talk to an exam proctor immediately.

Good luck, have fun!

CIS 110 Exam 2 Cheat Sheet

```
/** 1. For syntax, look at the format of the code presented in the questions
 *     (unless otherwise stated, it is all syntactically correct code). */

/** 2. Useful methods for String objects */
charAt(index)           // Returns the character at the given (zero-based) index.
endsWith(text)          // Returns true if the string ends with the given text.
indexOf(character)      // Returns the (zero-based) index of the given character.
length()                // Returns the length of the string.
startsWith(text)        // Returns true if the string starts with the given text.
substring(start, stop)  // Returns the characters from the start index to just
                        //   before the stop index
toLowerCase()           // Returns a new string with all lower case characters.
toUpperCase()           // Returns a new string with all upper case characters.

/** 3. Useful methods for Scanner objects */
new Scanner(src)        // Makes a new Scanner from the given source.
next()                  // Returns the next token from the Scanner.
hasNext()               // Returns true if there is a token to read.
nextLine()              // Returns the next line from the Scanner.
hasNextLine()           // Returns true if there is a line left to read.
nextX()                 // Returns the next token as a X, e.g., Int, Double.
hasNextX()              // Returns true if there is a token left and it's an X.

/** 4. Useful methods for Random objects */
new Random()            // Creates a new random object.
nextInt()               // Returns a random int.
nextInt(max)            // Returns a random int in the range 0 to max-1.
nextDouble()            // Returns a random double in the range 0.0 to 1.0.
nextBoolean()           // Returns a random boolean.

/** 5. Useful methods from the Arrays class */
Arrays.toString(arr)        // Returns a formatted String representing arr,
                            //   e.g., [1, 2, 3].
Arrays.equals(arr1, arr2)   // Returns true iff arr1 is pairwise equal to arr2.
Arrays.copyOf(arr, len)     // Returns a copy of arr up to the specified length.
                            //   truncating or padding the new array to meet it.
Arrays.fill(arr, val)       // Replaces the elements of arr with val
Arrays.deepToString(arr)        // Variants that work with
Arrays.deepEquals(arr1, arr2)   //   multidimensional arrays.
```

Notes about this exam:

- Exam #2 covers all the new content introduced since the first exam (chapters 4–7). The exam is necessarily cumulative — we can't NOT have a for loop on the exam, for example — but the questions will focus on the new content.
- The purpose of this exam is to *test your algorithmic thinking skills* rather than have you regurgitate facts about computer programming. To prepare for this exam, you should practice those skills by reviewing homeworks and doing practice problems discussed in section or found in the text.
- The exam is *closed book*, *closed notes*, and *closed electronic devices*. You may bring a single 8.5×11" sheet of double-sided notes to help jog your memory.
- In addition to your note sheet, we will include a sheet of commonly-used APIs so that you do not have to write them down.
- This practice exam introduces you to the types of problems that will be on the real exam. That way you can spend more time solving problems instead of trying to understand what the problem asks of you. That being said, while my goal is not to stray far from this format, the format of the actual exam may change slightly if necessary.
- I strongly recommend that you attempt this practice exam without looking at the answers. Afterwards, check your work, and then use the usual channels (e.g., Piazza, your TA, or myself) to resolve any lingering questions that you may have.

Boolean busting machines

1. (5 points) Evalute these Java expressions to their final values.

```
public static int mystery(int n, int k) {
    int r = 0;
    if (n > k) {
        r = n + k;
    } else if (n < k) {
        r = n * k;
    } else {
        r = n - k;
    }
    return r;
}
```

(a) `13 > 4 || true == false` $\longrightarrow$

`true`

(b) `((3 < 1) || !(0 == 1)) && (14 < 2)` $\longrightarrow$

`false`

(c) `mystery(3, -5)` $\longrightarrow$

`-2`

(d) `mystery(10, 2)` $\longrightarrow$

`12`

(e) `mystery(1, 1)` $\longrightarrow$

`0`

It's a mystery to everyone

2. (15 points) Given the following methods, evaluate the following Java expressions to their final values.

(a)
```
public static int mystery1(int n, int k) {
    while (2 * n < k) {
        n += n * 2;
        k += 5;
        System.out.println(">> " + n + ", " + k);
    }
    return n + k;
}

public static boolean mystery2(int m) {
    int val = mystery1(m, m);
    return val == 2 * m;
}
```

i. `mystery1(5, 10)` $\longrightarrow$

`15`

ii. `mystery1(2, 50)` $\longrightarrow$

`119`

iii. `mystery2(100)` $\longrightarrow$

`true`

(b)

```
public static int[] mystery3(int k) {
    int[] arr = new int[k];
    for (int i = 0; i < k; i++) {
        arr[i] = i+1;
    }
    mystery4(arr);
    return arr;
}

public static void mystery4(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        arr[i] = arr[i] * arr[i+1];
    }
}
```

i. `Arrays.toString(mystery3(2))` $\longrightarrow$

`[2, 2]`

ii. `Arrays.toString(mystery3(5))` $\longrightarrow$

`[2, 6, 12, 20, 5]`

Be assertive

3. (15 points) For each of the labeled points in the code fragment below, identify each of the assertions in the table as being *always* true, *never* true, or *sometimes* true or sometimes false.

```
public static int mystery(Console in, int d) {
    boolean b = true;
    int x = 0;
    // POINT A
    while (b) {
        // POINT B
        x = in.nextInt();
        if (x < 0) {
            b = false;
            // POINT C
        } else {
            d += x;
            // POINT D
        }
    }
    // POINT E
    return d;
}
```

	`b == true`	`x > 0`	`d > 0`
A	Always	Never	Sometimes
B	Always	Sometimes	Sometimes
C	Never	Never	Sometimes
D	Always	Sometimes	Sometimes
E	Never	Never	Sometimes

Nom nom nom

4. (20 points) Write a method `processFile` that takes a `Scanner` that is reading from a text file of a particular format and returns the average age of all students found in the file as an integer. The file contains a series of student/faculty records in the university, one record per line in the following format:

```
<last name> <first name> student <age> <gpa>
<last name> <first name> faculty <department> <age>
```

where `student` and `faculty` are actual tokens in the file to distinguish between a student record and a faculty record. For example, the following is a possible text file in the correct format:

```
Dunham Olivia student 19 3.7
Bishop Peter student 18 3.3
Bishop Walter faculty physics 60
Broyles Phillip student 20 3.4
Farnsworth Astrid faculty physics 35
```

If a `Scanner` were passed to `processFile` pointing to this file, then the method would return $(19 + 18 + 20)/3 = 19$.

```
public static int processFile(Scanner file) {
    int sum = 0;
    int count = 0;
    while (file.hasNextLine()) {
        Scanner line = new Scanner(file.nextLine());
        line.next();    // last name
        line.next();    // first name
        if (line.next().equals("student")) {
            sum += line.nextInt();
            count++;
        }
    }
    return sum / count;
}
```

Array whisperer

5. (20 points) Write a method `duplicate` that takes an integer array `arr` and an integer `n` as arguments and returns a new integer array that contains n copies of `arr` laid out end-to-end. Assume that you are passed a non-`null` array and `n` is positive. Here are some example invocations of `duplicate` and their results:

Array	n	Return value
`[1] [2] [3]`	3	`[1] [2] [3] [1] [2] [3] [1] [2] [3]`
`[0]`	5	`[0] [0] [0] [0] [0]`
`[1] [2] [3] [4] [5]`	1	`[1] [2] [3] [4] [5]`

```
public static int[] duplicate(int[] arr, int n) {
    int[] ret = new int[arr.length * n];
    for (int i = 0; i < ret.length; ) {
        for (int j = 0; j < arr.length; j++) {
            ret[i++] = arr[j];
        }
    }
    return ret;
}
```

Gonna (try to, again) fly now

6. (25 points)

(a) Write a method `rotate` that takes an integer array `arr` and a boolean `b` as arguments and changes `arr` so that its elements are rotated one position to the right if `b` is true and one position to the left if `b` is false. The element that is rotated off the end of the array is copied to the vacant position at the other end of the wrap. Note that `rotate` changes its argument array and does not return a result. Here are some example invocations of `rotate` and their results:

Array	b	Return value
`[1] [2] [3]`	true	`[3] [1] [2]`
`[1] [2] [3]`	false	`[2] [3] [1]`
`[0]`	true	`[0]`

Hint: `rotate` does two distinct operations based on `b`, either rotate left or rotate right. Tackle these two operations independently.

```
public static void rotate(int[] arr, boolean b) {
    if (b) {
        int temp = arr[arr.length-1];
        for (int i = arr.length-1; i > 0; i--) {
            arr[i] = arr[i-1];
        }
        arr[0] = temp;
    } else {
        int temp = arr[0];
        for (int i = 0; i < arr.length-1; i++) {
            arr[i] = arr[i+1];
        }
        arr[arr.length-1] = temp;
    }
}
```

(b) Using the `rotate` method from the previous section, write a method `randomRotateWalk` that takes an integer array and a `Random` object and repeatedly rotates its elements one position to the left or one position to the right until the array returns to its original configuration. `randomRotateWalk` should randomly rotate the array once to begin the process. `randomRotateWalk` should print out the initial array as it is passed to the and the resulting array after each random rotation. Here is the sample output of calling `randomRotateWalk` with the array `[0][1][2][3][4][5]`.

```
[0, 1, 2, 3, 4, 5]
[5, 0, 1, 2, 3, 4]
[4, 5, 0, 1, 2, 3]
[3, 4, 5, 0, 1, 2]
[2, 3, 4, 5, 0, 1]
[3, 4, 5, 0, 1, 2]
[2, 3, 4, 5, 0, 1]
[1, 2, 3, 4, 5, 0]
[0, 1, 2, 3, 4, 5]
```

Hint: Keep in mind that `rotate` mutates the original array. You will need some way of remembering the original array so you know when to stop rotating. The `Arrays.equals` and `Arrays.toString` methods will be useful here to compare arrays for equality and petty print the array in the format above.

```
public static void randomRotateWalk(int[] arr, Random rand) {
    int[] orig = Arrays.copyOf(arr, arr.length);
    System.out.println(Arrays.toString(arr));
    rotate(arr, rand.nextBoolean());
    while (!Arrays.equals(arr, orig)) {
        rotate(arr, rand.nextBoolean());
        System.out.println(Arrays.toString(arr));
    }
}
```