CIS 110 — Introduction To Computer Programming

December 19th, 2011 — Final

Answer key

CIS 110 final instructions

- You have 120 minutes to finish this exam. Time will begin when called by a proctor and end precisely 120 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:

  System.out.println $\longrightarrow$ S.O.PLN
  System.out.print $\longrightarrow$ S.O.P
  System.out.printf $\longrightarrow$ S.O.PF

  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 of 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 final 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.
```

The critter class

```
// The Critter class is the base class for all critters in the simulation.
public abstract class Critter {
    // Defines the types of food a critter can eat: grass or meat.
    public static enum FoodType { GRASS, MEAT }

    // Defines the possible directions a critter can move: the four cardinal
    // directions along with "no move" (CENTER).
    public static enum Direction { NORTH, EAST, SOUTH, WEST, CENTER }

    // The speed at which a critter moves: fast, medium, or slow.
    public static enum Speed { FAST, MEDIUM, SLOW }

    // Returns the direction this critter should move for the next step of the
    // simulation.
    public abstract Direction getMove();

    // Returns the type of food this critter eats.
    public abstract FoodType getFoodType();

    // Returns the color of this critter.
    public abstract Color getColor();

    // Returns the speed of this critter.
    public abstract Speed getSpeed();

    // Returns the String representation of this critter.  This should be a
    // a single character which will be rendered onto the simulation board.
    public String toString() {
        throw new UnsupportedOperationException(
                "Implementors must provide their own toString() methods!");
    }
}
```

Express yourself

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

   (a) `20 / (5 * 3) - 10 / 5 + 2` $\longrightarrow$

   $\boxed{\texttt{1}}$

   (b) `83 % 2 + 100 % 2 - 101 % 2` $\longrightarrow$

   $\boxed{\texttt{0}}$

   (c) `(12 >= 5 && 6 < 4) || (19 > 3 && !true)` $\longrightarrow$

   $\boxed{\texttt{false}}$

   (d) `!(!(12 < 3) || !(9 > 5))` $\longrightarrow$

   $\boxed{\texttt{false}}$

   (e)
   ```
    String s = "Last test!";
   s.substring(5, 9) + s.charAt(9) + s.substring(0, 4)
   ```
   $\longrightarrow$

   $\boxed{\texttt{"test!Last"}}$

Method mysteries: the next generation

2. (10 points) Given methods `mystery1` and `mystery2`, evaluate the following Java expressions to their final values. Note that `mystery2` is found on the next page.

```
public static int mystery1(int x, boolean b) {
    if (b) {
        if (x > 0) {
            x = x - 1;
        } else {
            x *= -2;
        }
    } else {
        if (x <= 0) {
            x++;
        }
        if (x == 0) {
            x += 100;
        }
    }
    return x;
}
```

(a) `mystery1(-5, false)` ⟶

$\boxed{\texttt{-4}}$

(b) `mystery1(3, true)` ⟶

$\boxed{\texttt{2}}$

(c) `mystery1(-1, false)` ⟶

$\boxed{\texttt{100}}$

(d) `mystery1(-2, true)` ⟶

$\boxed{\texttt{4}}$

(e) `mystery1(10, false)` ⟶

$\boxed{\texttt{10}}$

```
public static int[] mystery2(int[] arr) {
    if (arr.length <= 1) {
        return null;
    }
    int[] ret = new int[arr.length/2];
    for (int i = 0; i < ret.length; i++) {
        if (arr[i * 2] > arr[i * 2 + 1]) {
            ret[i] = arr[i * 2];
        } else {
            ret[i] = arr[i * 2 + 1];
        }
    }
    return ret;
}
```

(a) `mystery2(new int[]{3})` $\longrightarrow$

$\boxed{\texttt{null}}$

(b) `mystery2(new int[]{3, 5})` $\longrightarrow$

$\boxed{\texttt{[5]}}$

(c) `mystery2(new int[]{1, 3, 9, 8, 5, 3})` $\longrightarrow$

$\boxed{\texttt{[3, 9, 5]}}$

(d) `mystery2(new int[]{4, 2, 0, 0, 1, 5})` $\longrightarrow$

$\boxed{\texttt{[4, 0, 5]}}$

(e) `mystery2(new int[]{2, 10, 8, 5, 4})` $\longrightarrow$

$\boxed{\texttt{[10, 8]}}$

Be assertive

3. (15 points) For 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. You may abbreviate always, never, and sometimes as A, N, and S respectively.

```
public static int mystery(int x, int y) {
    if (y < 0 || y > 2 || x <= 0) {
        return 0;
    }
    Random r = new Random();
    int z = x * x;
    // POINT A
    while (z >= x) {
        // POINT B
        if (y == 0) {
            z += 1;
            y = r.nextInt(5) + 1;
            // POINT C
        } else {
            x += y;
            z -= 2;
            y = r.nextInt(2);
            // POINT D
        }
    }
    // POINT E
    return z;
}
```

	`z >= x`	`y >= 0 && y <= 2`	`z < 0`
A	Always	Always	Never
B	Always	Sometimes	Never
C	Always	Sometimes	Never
D	Sometimes	Always	Sometimes
E	Never	Always	Sometimes

Polymorph

4. (15 points) For the class hierarchy below, fill out the table with the results of method calls to `speak()` on objects in that class hierarchy. For example, the top-left corner of the table is the result of the call:

```
Animal a = new Goat();
a.speak();
```

Several of the combinations of variables and assignments below cause compiler errors. For those that cause compiler errors, write "Error" in place of the output.

```
class Animal {
    public String speak() { return "Noise!"; }
}
class Goat extends Animal {
    public String speak() { return "Bleat!"; }
}
class MountainGoat extends Goat {
    public String speak() { return super.speak() + " going up!"; }
}
class Cat extends Animal {
    public String speak() { return "Meow"; }
}
class Dog extends Animal {
    public String speak() { return "Woof"; }
}
```

	`= new Goat()`	`= new Cat()`	`= new MountainGoat()`
`Animal a`	`"Bleat!"`	`"Meow"`	`"Bleat! going up!"`
`Goat g`	`"Bleat!"`	Error	`"Bleat! going up!"`
`MountainGoat m`	Error	Error	`"Bleat! going up!"`
`Cat c`	Error	`"Meow"`	Error
`Dog d`	Error	Error	Error

Flying one more time

5. (20 points) Write a static method `printTriangle` that takes an integer `n` and prints a triangle of height `n` in a particular format. The rows of the triangle are made up of periods (.). However, certain rows are made up of different characters as described below. We consider the first row of the grid to have row number 1.

(a) If the row number is a multiple of 3, then the row is made up of greater-than signs (>).

(b) If the row number is a multiple of 5, then the row is made up of less-than signs (<).

(c) If the row number is both a multiple of 3 and 5, then the row is made up of exclamation marks (!).

If `n < 1`, then `printTriangle` does nothing. Here are some examples of invocations of `printTriangle`.

printTriangle(5)

```
.
..
>>>
....
<<<<<
```

printTriangle(1)

```
.
```

printTriangle(15)

```
.
..
>>>
....
<<<<<
>>>>>>
.......
........
>>>>>>>>>
<<<<<<<<<<
...........
>>>>>>>>>>>>
.............
..............
!!!!!!!!!!!!!!!
```

```java
public static void printTriangle(int n) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            if (i % 3 == 0 && i % 5 == 0) {
                System.out.print("!");
            } else if (i % 3 == 0) {
                System.out.print(">");
            } else if (i % 5 == 0) {
                System.out.print("<");
            } else {
                System.out.print(".");
            }
        }
        System.out.println();
    }
}
```

Nom nom nom

6. (20 points) Write a static method `getWins` that takes a `Scanner` that reads from a text file in a particular format and a String representing a player's name and returns the number of wins for that player found in the file. The file contains a series of game entries where each record represents the outcome of a single game, one record per line, in the following format:

```
<player1name> <player2name> <winner>
```

The winner field can either be 1 or 2 corresponding to the first player winning or the second player winning, respectively. Say that our scanner points to the following file, then here are some example invocations of `getWins` and the values it returns. If a person is not found in the file, then `getWins` returns zero. You may also assume that the input file is well-formed.

```
john james 1
john jane 2
john julia 1
james jane 2
james julia 2
jane julia 1
```

`getWins(file, "john")`	2
`getWins(file, "james")`	0
`getWins(file, "julia")`	1
`getWins(file, "jane")`	3

```
public static int getWins(Scanner file, String name) {
    int numWins = 0;
    while (file.hasNextLine()) {
        Scanner line = new Scanner(file.nextLine());
        String p1 = line.next();
        String p2 = line.next();
        String winner = "";
        if (line.nextInt() == 1) {
            winner = p1;
        } else {
            winner = p2;
        }
        if (winner.equals(name)) { numWins++; }
    }
    return numWins;
}
```

I've got 99 problems and an array ain't one

7. (20 points) Write a method `intersperse` that takes a String `s` and an array of Strings `arr` and returns a new String array that "intersperses" `s` among the elements of `arr`. By intersperse, we mean that the new array is a copy of `arr` except between each element of `arr` we insert a copy of `s`. That is, the returned array is laid out as follows:

```
[arr[0], s, arr[1], s, ..., s, arr[arr.length-1]]
```

In the case of a null array, `intersperse` returns `null`. In the case of an array of length zero, `intersperse` returns a new array of length zero. Note that an array of length zero is initialized in the usual way except that you pass zero for its length.

Here are some example calls of `intersperse` and their results.

`String s`	`String[] arr`	Result
`"pie"`	`["hello", "world", "wow!"]`	`["hello", "pie", "world", "pie", "wow!"]`
`"sigh"`	`["huh?"]`	`["huh?"]`
`"hrm"`	`[ ]`	`[ ]`

```
public static String[] intersperse(String s, String[] arr) {
    if (arr == null) { return null; }
    if (arr.length == 0) { return new String[0]; }
    String[] ret = new String[arr.length * 2 - 1];
    for (int i = 0; i < arr.length - 1; i++) {
        ret[i * 2] = arr[i];
        ret[i * 2 + 1] = s;
    }
    ret[ret.length - 1] = arr[arr.length - 1];
    return ret;
}
```

Didn't we just do this?

8. (20 points) Write a subclass of `Critter` called `Rhino` that meets the following specification:

Constructor	`Rhino(int numAdditionalSteps)`: `numAdditionalSteps` is the number of additional steps a Rhino takes in each direction.
Food type	Grass
Movement	Chooses a random direction (from north, south, east, or west) and moves in that direction for `numAdditionalSteps + 3` steps. After this, it chooses a new random direction and repeats the process.
Color	Normally gray. However, when the Rhino is on its last 3 steps of its current direction, it becomes red.
Speed	Slow
toString	Normally `"R"`. However, when the Rhino is on its last 3 steps of its current direction, it becomes `"!"`.

For reference, the essential pieces of the critter class are reproduced as part of your cheat sheet. (Solution on the next page)

```
public class Rhino extends Critter {
    private int counter;
    private int numAdditionalSteps;
    private Direction direction;
    private Direction chooseRandomDirection() {
        Random rand = new Random();
        int val = rand.nextInt(4);
        if (val == 0) { return Direction.NORTH; }
        else if (val == 1) { return Direction.SOUTH; }
        else if (val == 2) { return Direction.EAST; }
        else { return Direction.WEST; }
    }
    public Rhino(int numAdditionalSteps) {
        this.numAdditionalSteps = numAdditionalSteps;
    }
    public Direction getMove() {
        if (counter == numAdditionalSteps + 3) {
            direction = chooseRandomDirection();
            counter = 0;
        }
        counter++;
        return direction;
    }
    public FoodType getFoodType() { return FoodType.GRASS; }
    public Color getColor() {
        if (counter >= numAdditionalSteps) {
            return Color.RED;
        } else {
            return Color.GRAY;
        }
    }
    public Speed getSpeed() { return Speed.SLOW; }
    public String toString() {
        if (counter >= numAdditionalSteps) {
            return "!";
        } else {
            return "R";
        }
    }
}
```

That's a wrap!

9. (25 points)

   (a) Write a method `swapRandom` that takes a `Random` object, an integer array `arr`, an integer index `k`, and an integer length `len` and swaps the element at index `k` with the element found at a random index `i` where `k <= i < k + len`. Note that `swapRandom` does not return anything nor does it output anything to the console.

   If `k` is an invalid index, `len` is negative, or `k + len > arr.length` then `swapRandom` does nothing. Here are some example invocations of `swapRandom` over the example array `arr`

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

   and the resulting changes to `arr`. The example calls to `swapRandom` here are independent and do not affect the results of the other example calls. Remember that `swapRandom` behaves randomly, so these examples represent one such possible result for each method invocation.

`k`	`len`	Changed `arr`
0	10	[**7**, 1, 2, 3, 4, 5, 6, **0**, 8, 9]
3	2	[0, 1, 2, **4**, **3**, 5, 6, 7, 8, 9]
6	5	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
public static void swapRandom(Random rand, int[] arr, int k, int len) {
    if (k < 0 || k >= arr.length ||
        len < 0 || k + len > arr.length) { return; }
    int index = rand.nextInt(len) + k;
    int temp = arr[k];
    arr[k] = arr[index];
    arr[index] = temp;
}
```

(b) Use `swapRandom` to write a method `fisherYatesShuffle` that takes an integer array `arr` and shuffles `arr` according to the Fisher-Yates shuffle algorithm. The Fisher-Yates algorithm shuffles the elements of the input `arr` as follows:

- For each i from 0 to `arr.length - 1`:
  - Choose a random index k in the range $i \leq k < arr.length$.
  - Swap `arr[i]` with `arr[k]`.

For example, say that `arr` is the following array:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Then one possible way that Fisher-Yates may shuffle `arr` is:

```
[5, 8, 1, 2, 7, 6, 4, 9, 0, 3]
```

Note that `fisherYatesShuffle` does not return anything nor does it output anything to the console. Also if a `null` array is passed to `fisherYatesShuffle`, then the method does nothing. You may not use the `Collections.shuffle` method to shuffle the elements of the input array.

```
public static void fisherYatesShuffle(int[] arr) {
    if (arr == null) { return; }
    Random rand = new Random();
    for (int i = 0; i < arr.length; i++) {
        swapRandom(rand, arr, i, arr.length - i);
    }
}
```

Scratch paper