CIS1100.java — Fall 2024 — Exam 1

Full Name: ____________________________________

PennID (e.g. 12345678): ____________________________

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

________________________ ____________________

Signature Date

Instructions are below. Not complying will lead to a 0% score on the exam.

- Do not open this exam until told by the proctor.
- You will have exactly 60 minutes to take this exam.
- Make sure your phone is turned OFF (not on vibrate!) before the exam starts.
- Food and gum are not permitted—don't be noisy or messy.
- You may not use your phone or open your bag for any reason, including to retrieve or put away pens or pencils, until you have left the exam room.
- This exam is closed-book, closed-notes, and closed computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written in proper Java format, including all curly braces and semicolons.
- Do not separate the exam pages. Do not take any exam pages with you. The entire exam packet must be turned in as is.
- Only answers on the FRONT of pages will be graded. There are two blank pages at the end of the exam if you need extra space for any graded answers.
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to you.
- When you turn in your exam, you may be required to show your PennCard. If you forgot to bring your ID, talk to an exam proctor immediately.
- We wish you the best of luck!

Q1	Q2	Q3	Q4	Q5	Q6 (bonus)

Q1. Types Fill In The Blank

In the column marked **“Type,”** choose the type (e.g. `int`, `char`, etc.) for the variable that would allow the line to compile, or write **"CE"** if there is an error in the expression that makes its type undefined. You do not need to write the value of the expression.

Statement	Type
`_______ x = (5 >= 5) && !false;`	
`_______ x = 5 >= !false;`	
`_______ x = 5 / 2;`	
`_______ x = (5 / 2) + 7.0 * (3 / 2);`	
`_______ x = (char) 72 + "ELLO WORLD";`	
`_______ x = 72 + "ELLO WORLD";`	
`_______ x = Double.parseDouble("3");`	
`_______ x = double[3];`	

Q2. Values Fill In the Blank

Write the value that gets printed, or write **"error"** if there is an error during the execution of these lines of the program.

Question 2.1

```
System.out.println(true && !(false || true));
```

Answer:

Question 2.2

```
System.out.println(Integer.parseInt("three"));
```

Answer:

Question 2.3

```
int y = 4 / 3;
double x = y;
System.out.println(x);
```

Answer:

Question 2.4

```
int[] numbers = {1, 2, 3, 4};
int sum = 0;
for (int i = 0; i < 4; i++) {
    sum += numbers[i];
}
System.out.println(sum);
```

Answer:

Question 2.5

```
int[] numbers = {1, 2, 3, 4};
int sum = 0;
for (int i = 0; i < 0; i++) {
    sum += numbers[i];
}
System.out.println(sum);
```

Answer:

Q3. Debugging

Harry is behind on inputting grades! He decides to (hastily) write a function **homeworkAverage** that, given a double array of homework scores, returns the average homework score, dropping the lowest if and only if ALL homework scores are at least 30%. Otherwise, no score is dropped. Below is a buggy implementation of this function (he was in a hurry!). Read the code below.

Then, in the table that follows, identify the line on which each of the bugs appears, the part of the code that is incorrect, and the code you can replace that part with to make it correct. We filled in an example bug on the first line. Note that there are no further bugs in the function signature — i.e., the input and return types are correct. **There are six other bugs.**

```
 1.  public stochastic double homeworkAverage(double[] scores) {
 2.      // check if all are at least thirty
 3.      String allAtLeastThirty = true;
 4.      for (i = 0; i < scores.length; i++) {
 5.         if (scores[i] < 30) {
 6.            allAtLeastThirty = false;
 7.         }
 8.      }
 9.      // find minimum score
10.      double minScore = scores[0];
11.      for (int i = 0; i < scores.length(); i++) {
12.         if (scores[i] < minScore) {
13.            minScore = scores[i];
14.         }
15.      }
16.      // find sum of scores
17.      double sum = -1;
18.      for (int i = 0; i < scores.length; i++) {
19.         sum += scores[i];
20.      }
21.      // return the average
22.      if (allAtLeastThirty) {
23.         double average = (sum - minScore) / scores.length;
24.         return average;
25.      } else if {
26.         double average = sum / scores.length;
27.         return average;
28.      }
29.  }
```

Line Number	Incorrect Code	Replacement Code
1	`stochastic`	`static`

Q4. Complete the Program: *Calendar.java*

The following program models a person's calendar of monthly recurring events. The program maintains two arrays: **int[] days** and **String[] events**. The arrays always have the same lengths. For each index **i**, we store in **days[i]** the day on which a particular event takes place. The name of that event is **events[i]**. The program should read a day in (1-31) as the first command line argument and then search through the **days** array, printing out all of the names of events that take place on the input day. The program should also print out a warning about events that are scheduled to take place on the following day. Keep in mind that the day following **31** would be **1**! Finally, if the day provided as a command line argument is not a number between 1 and 31 inclusive, print an error message. There are some example executions on this page; the code to complete and the table for writing answers are found on the next page.

Examples:

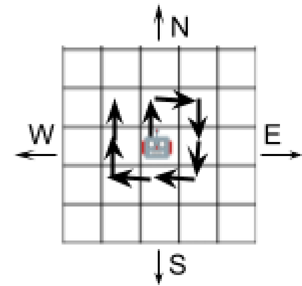
$ java Calendar 15 TODAY: Bowling TODAY: Movie Night TOMORROW: Pay Bills	$ java Calendar 31 TOMORROW: Pay Rent TODAY: Deep Clean	$ java Calendar 0 Invalid day!

```
public class Calendar {
    public static void main(String[] args) {
        int[] days = {1, 5, 15, 15, 5, 16, 31};
        String[] events = {"Pay Rent", "Team Meeting", "Bowling",
                           "Movie Night", "Grocery Shopping",
                           "Pay Bills", "Deep Clean"};

        int inputDay = ___BLANK_0___; // read from command line
        if (____BLANK_1____) { // check if inputDay isn't a valid day
            System.out.println("Invalid day!");
            return; // exit the program since the day isn't valid
        }
        int nextDay = (____BLANK_2____) + 1; // find the next day & wrap around
                                             // if necessary

        for (int i = 0; ____BLANK_3____; i++) {
            if (____BLANK_4_____) { // found an event for today
                System.out.println("TODAY: " + events[i]);
            } else if (days[i] == nextDay) { // found an event for tomorrow
                System.out.println("TOMORROW: " + ____BLANK_5_____);
            }
        }
    }
}
```

Blank #	Code
0	
1	
2	
3	
4	
5	

Q5. Coding: FarmBot

FarmBot is a little robot responsible for inspecting crops in a field. He records an array of moves that he took on his travels, called a **history**.



In a history, a 0 indicates one move north, a 1 indicates one move east, a 2 indicates one move south, and a 3 indicates one move west. For example, the history {0, 1, 2, 2, 3, 3, 0, 0} would indicate the path on the right, assuming that FarmBot starts at the location marked by 🤖.

First, implement a function that converts a history into a **summary**, which is an array that counts the total number of moves north, east, south, and west. The summary for the history above would be {3, 1, 2, 2}. (A summary array always has length four.)

Question 5.1

```
/*
 * Input: a history of FarmBot's moves
 * Output: a summary of the moves, as defined above
 * Examples: historyToSummary({}) => {0, 0, 0, 0}
 *           historyToSummary({0}) => {1, 0, 0, 0}
 *           historyToSummary({3, 3, 3, 3}) => {0, 0, 0, 4}
 *           historyToSummary({0, 1, 2, 2, 3, 3, 0, 0}) => {3, 1, 2, 2}
 */
public static int[] historyToSummary(int[] history) {












}
```

Second, implement a function that determines <u>from a summary</u> whether FarmBot is **burnt out**, which means he moved more than five times in any direction. This is not based on FarmBot's final location but instead on the total number of times he moved in each direction.

Question 5.2

```
/*
 * Input: a summary of FarmBot's moves
 * Output: whether FarmBot is burnt out, as defined above
 * Examples: isBurntOut({1, 2, 3, 4}) => false
 *           isBurntOut({5, 5, 5, 5}) => false
 *           isBurntOut({3, 4, 5, 6}) => true
 *           isBurntOut({6, 0, 0, 6}) => true
 */
public static boolean isBurntOut(int[] summary) {




}
```

Finally, implement a function (on the next page) that determines <u>from a history</u> whether FarmBot **needs to be picked up**. He needs to be picked up if he took more than 10 moves overall or if he is burnt out. Use the previous two functions you wrote to complete this one in **one line only**!

Question 5.3

```java
/*
 * Input: a history of FarmBot's moves
 * Output: whether FarmBot needs to be picked up, as defined above
 * Examples: needsPickup({0, 1, 2, 3, 0, 1, 2, 3}) => false
 *           needsPickup({0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3}) => true
 *           needsPickup({0, 1, 0, 0, 0, 2, 0, 0}) => true
 */
public static boolean needsPickup(int[] history) {




}
```

Q6. (Bonus)

Create a piece of art (e.g. drawing, poem, short program, anything you like)!
If you are unsure of what to make, select one member of the course staff and make art about that person. All exams will get full credit for this question, no pressure to make anything in particular!

Extra Answers Page (This page is intentionally blank)

You may use this page for additional space for answers; keep it attached to this exam. Clearly note on the original question page that your answer is on this extra page, and clearly note on this page what question you are answering.