CIS 1100 — Spring 2024 — Exam 1

Full Name: ______________________________

PennID (e.g. 80472698): ______________________________

“My signature below certifies that I have complied with the University of Pennsylvania’s Code of Academic Integrity in completing this exam.”

Signature ______________________________

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.
- There is scratch space at the end of the 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: 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 **"compilation error"** 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 = "cis".equals("cs");`	____________________
`______ x = "code".charAt(3) == "d";`	____________________
`______ x = 5 + Integer.parseInt("34");`	____________________
`______ x = new int[13];`	____________________
`______ x = 0 <= Math.random() < 1;`	____________________
`______ x = "caesa" + 'r';`	____________________
`______ x = 4 + (28 / (8 / 2.0));`	____________________

Q2: Values Fill in the Blank

Write the value that gets printed, or write **"runtime error"** if there is an error during the execution of these lines of the program. Each code snippet is independent of the others

Question 2.1

```
System.out.println(3 + Integer.parseInt("2" + "1"));
```

Answer:

Question 2.2

```
int x = 5;
int y = 2;
System.out.println(x / y);
```

Answer:

Question 2.3

```
int[] arr = {2, 4, 6, 8};
arr[arr.length - 1] = arr[arr.length - arr[1]] * arr[2];
System.out.println(arr[3]);
```

Answer:

Question 2.4

```
int a = 12;
String s = "4.0"
System.out.println(a / Double.parseDouble(s));
```

Answer:

Question 2.5

```
// Math.random() evaluates to a random double between 0 and 0.999...
System.out.println((int) Math.random());
```

Answer:

Question 2.6

```
String a = "midterm";
int i = a.length();
System.out.println(a.charAt(i));
```

Answer:

Q3: Tracing

Here's a class that features a few functions.

```
public class TracingExercise {
  public static boolean mysteryOne(String word) {
    boolean check1 = word.charAt(0) == 'A';
    boolean check2 = word.charAt(word.length() - 1) == 'w';
    boolean check3 = word.length() > 6;

    return (check1 || check2) && check3;
  }

  public static int mysteryTwo(String word, int x, int y) {
    boolean check = mysteryOne(word);
    int value = x * y;

    if (!check && value < 15) {
        return mysteryThree(value, word.length());
    } else {
        return 1 + mysteryThree(x + x, y + y);
    }
  }

  public static int mysteryThree(int a, int b) {
    int value1 = a % b;
    int value2 = a / b;
    value2--;

    return value1 + value2;
  }

  public static void main(String[] args) {
    System.out.println(mysteryTwo("Arrows", 2, 4));
    System.out.println(mysteryTwo("Sparrow", 7, 2));
    System.out.println(mysteryTwo("wow", 4, 3));
    System.out.println(mysteryTwo("alive", 9, 6));
  }
}
```

When the program is run as `java TracingExercise`, four lines are printed. Fill in the blanks in the "Printed Line" column (on the next page) to show what values get printed.

Please write your solution to Q3 on this page.

Print Statement	Printed Line
`System.out.println(mysteryTwo("Arrows", 2, 4));`	____________________
`System.out.println(mysteryTwo("Sparrow", 7, 2));`	____________________
`System.out.println(mysteryTwo("wow", 4, 3));`	____________________
`System.out.println(mysteryTwo("alive", 9, 6));`	____________________

Q4: Complete the Program: `Outfit.java`

Bhrajit and Priya can never decide what to wear when they want to go out to events on campus. They decided to put their wardrobes together to widen their selection of shirts. Combine their wardrobes and randomly pick an outfit for both Priya and Bhrajit!

1. This program should first create a combined array to store all of Bhrajit's and Priya's shirts. It should fill up the array first with all of Bhrajit's shirts, then with Priya's.
2. Once the array is full, the program should randomly select any shirt from the array and print out its description. This will be Bhrajit's shirt. Then, repeat the random selection for Priya.
3. The program should not be able to select the same shirt for both Bhrajit and Priya, so if Priya draws the same shirt as Bhrajit, Priya should get the shirt at the next index. If Priya and Bhrajit both draw the last shirt, Priya should get the first shirt in the array.

```java
public class Outfit {
    public static void main(String[] args) {
        // starter arrays with individual clothes
        String[] bhrajitShirts = { "yellow", black, "button down brown", "denim" };
        String[] priyaShirts = { "red", "sparkly mesh", "black tank", "fruits", "purple" };

        // combined array initialization
        String[] combinedWardrobe = new String[bhrajitShirts.length * 2];
        for (int i = 0; i < combinedWardrobe.length(); i++) {
            // fill up array first with Bhrajit's clothes
            if (i < bhrajitShirts.length) {
                combinedWardrobe[i] = bhrajitShirts[i];
                // after Bhrajit's clothes, fill up with Priya's clothes
            } else {
                combinedWardrobe[i] = priyaShirts[i];
            }
        }
        // Select ANY valid index from the combinedWardrobe array twice.
        int bhrajitIndex = (int) (Math.random() * (combinedWardrobe.length + 1));
        int priyaIndex = (int) (Math.random() * (combinedWardrobe.length));
        if (priyaIndex == bhrajitIndex) { // check if both selected the same index
            priyaIndex = (priyaIndex + 1) % 1; // if so, pick the next index
        }
        String bhrajitFit = combinedWardrobe[bhrajitIndex];
        String priyaFit = combinedWardrobe[priyaIndex];

        System.out.println(bhrajitFit);
        System.out.println("priyaFit");
    }
}
```

There are seven bugs in total. For each, write the line number where the bug is found. Then, describe the bug and then write the entire fixed line. Each bug is contained on one line and can be fixed by changing one line. Some bugs may be related to syntax, while others may be related to incorrect implementation of the requirements listed above.

Line Number	Bug Description	Entire Fixed Line

Q5. Coding: `HikePlanning.java`

The following three subquestions can be completed in any order. Although you will use `isExciting` *and* `firstDramaticSegment` *in your implementation for* `likeHike`*, you do not need to implement the former two correctly to earn credit for the latter.*

Harry is picky about the kinds of hikes he likes to go on. A hike can be modeled as an array of doubles (a double[]) representing the elevations that the hiker will encounter every mile. A hike is said to be **exciting** if it reaches an elevation that is at least twice as high as its starting elevation. A hike has a **dramatic segment** if there are any two consecutive elevations that differ by at least `1000` feet. Harry likes to go on hikes that **are exciting and that have at least one dramatic segment as long as that segment is not the very first segment of the hike.** (He doesn't like a hike that has no dramatic segments, and he doesn't like a hike if the first segment is dramatic.)

Question 5.1 (`isExciting`)

`isExciting()` returns `true` if the hike reaches a height that is at least twice as high as its starting elevation.

```
double[] hikeOne = {100, 200, 300, 400, 500, 600, 700, 800, 900, 1000};
System.out.println(isExciting(hikeOne)); // prints true since 1000 is at least twice as high as 100

double[] hikeTwo = {500, 300, 900, 400, 800};
System.out.println(isExciting(hikeTwo)); // prints false since 900 is not at least twice as high as 500
```

```
/*
 * Input: an array of doubles representing the elevation of a hike
 * Output: a boolean indicating whether the hike is exciting
 */
public static boolean isExciting(double[] hike) {




}
```

Question 5.2 (`firstDramaticSegment`)

A dramatic segment refers to two consecutive points of the hike where the absolute value of the difference between the two elevations is greater than `1000`. A segment's index is the index of the first point. Return the index of the first dramatic segment, or -1 if none exists.

```
[4000, 4500, 6000, 5400] --> return 1 since Math.abs(4500 - 6000) = 1500
[5800, 4500, 5400, 3000] --> return 0 since Math.abs(5800 - 4500) = 1300
[4000, 4500, 4600, 5400, 5000] --> return -1 since no dramatic segment
```

```
/*
 * Input: an array of doubles representing the elevation of a hike
 * Output: the index of the first dramatic segment
 */
public static int firstDramaticSegment(double[] hike) {

}
```

Question 5.3 (`likeHike`)

Harry likes when hikes are exciting and when they have a dramatic segment. But Harry does not like a hike that starts with a dramatic segment. (He likes a calm start.) Write a method `likeHike` that returns `true` if the hike is exciting and the hike has a dramatic segment but the dramatic segment is not the first segment of the hike.

Complete this function in **one line of code** to receive full credit. Solutions in more than one line of code will receive less credit. *Answer on the next page*

```
[4000, 4500, 6000, 8000] --> return true; exciting & dramatic segment at index 1
[500, 300, 900, 400, 800] --> return false; not exciting
[3000, 3500, 3800, 6500] --> return true; exciting & dramatic segment at index 2
[100, 1100, 1300, 1400, 1500] --> return false; dramatic segment at index 0
```

```
/*
 * Input: an array of doubles representing the elevation of a hike
 * Output: a boolean indicating whether Harry would like this hike
 * Complete this function in ONE LINE for full credit.
 */
public static boolean likeHike(double[] hike) {




}
```

Question 6: 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 Space

You can use this page for scratch work. Make sure to note on the original problem that you are using this space. Mark clearly which answers correspond to which questions.

Extra Space

You can use this page for scratch work. Make sure to note on the original problem that you are using this space. Mark clearly which answers correspond to which questions.