Spring 2021 Exam 1 Answer Key

True/False

1. `int x = 4.0;` causes an error. (TRUE)
2. `double x = '4';` causes an error. (FALSE)
3. The following code compiles: (FALSE)

```
int i = 0;
if (3 < 21) {
    i++;
} else {
    i = i - 2;
} else if (10 > 3) {
    i = 1 / 9;
}
```

4. The following code compiles: (FALSE)

```
int i = 0;
if (3 < 21) {
    i++;
} else (10 < 3) {
    i = i - 2;
}
if (10 > 3) {
    i = 1 / 9;
}
```

5. The following code would compile if sequence has type `String` but NOT if sequence has type `String[]` (TRUE)

```
for (int i = sequence.length(); i >= 0; i--) {
    System.out.println(sequence.charAt(i));
}
```

6. Running the following code causes a white rectangle to be visibly drawn over a black background. (FALSE)

```
PennDraw.setPenColor(PennDraw.WHITE);
PennDraw.filledRectangle(0.5, 0.5, 0.1, 0.3);
PennDraw.clear(PennDraw.BLACK);
```

7. A loop control variable must be updated with every iteration of a while loop. (FALSE)
8. `"c" + "d"` evaluates to `"cd"`. (TRUE)
9. For a given array, `arr`, `arr[arr.length]` evaluates to the last element in the array. (FALSE)
10. If a test case for a unit of code passes, that means that there are definitely no bugs in that unit of code. (FALSE)

Short Fill in the Blank

1. What is the return type of this function `returnsBoolean`? ____ (`double`)

```
public static double returnsBoolean(int character) {
    int x = 3 - character;
    int y = 4;
    int str = x + y;
    return str;
}
```

2. The ____ (*base case*) in a recursive function, unlike the recursive case, does not make any more recursive calls and instead allows Java to start clearing activation frames off of the call stack.
3. Look at the following code snippet:

```
char b = ____;
if ('q' < b) {
    if (b > 's') {
        System.out.print("R");
    } else {
        System.out.print("C");
    }
    if ('s' > b) {
        System.out.print("U");
    } else if ('s' == b) {
        System.out.print("T");
    }
} else if (b > 'q') {
    System.out.print("Z");
}
```

    1. What value of `b` would cause the program to print `"RT"`? If there is none, type none. ___ (none)
    2. What value of `b` would cause the program to print `"CU"`? If there is none, type none. ___ (`'r'`)
    3. What value of `b` would cause the program to print `"CT"`? If there is none, type none. ___ (`'s'`)
    4. What value of `b` would cause the program to print `"Z"`? If there is none, type none. ___ (none)

> We accepted `r` and `s` in addition to `'r'` and `'s'`.

Long Fill in the Blank

SEAS just decided that they want to fix their servers but broke everything even worse by wiping the entire mainframe! They've asked you to help them write a function to reset the servers before all their data is lost!

This function is supposed to take a single command line argument of an integer, add 7 and divide it by 2. Then this function will determine if the calculated number is a multiple of 3. If it is, it will print "Success". Otherwise, it will print "Try again". Fill in the blanks so that this program compiles and runs as specified.

Question:

```
public static void main(String[] args) {

    // Convert first command line argument to an int
    int x = ____1____; //#1

    // Add 7 and divide by 2
    int y = ____2____; //#2

    // Figure out what message to print
    ___3__ message; //#3
    if (y __4___ 3 == _5__) { //#4, #5
        message = "Success";
    } ___6__ { //#6
        message = "Try again";
    }

    System.out.println(message);
}
```

Answer:

```
1 = Integer.parseInt(args[0])
2 = (x + 7) / 2
3 = String
4 = %
5 = 0
6 = else
```

Niven Numbers

A niven number is a number that is divisible by the sum of its digits. Write a function that recursively returns the minimum niven number given an integer array, or returns `Integer.MAX_VALUE`.

You are given a function, `sumDigits`, which takes an integer input `n` and returns the sum of the digits in `n`. For instance, `sumDigits(12)` = `3`, `sumDigits(35)` = `8`, `sumDigits(2)` = `2`.

Now, please write the following two functions:

`public static boolean isNiven(int n)` which, using the above `sumDigits` function, returns a `boolean` if a number is a Niven number: `isNiven(7)` = `true`, `isNiven(35)` = `false` [35 is not divisible by 8], `isNiven(36)` = `true` [36 is divisible by 9]

`public static int minNivenNum(int[] arr, int index)` which, using recursion, returns the smallest Niven number, or the value `Integer.MAX_VALUE` if the array contains no niven numbers. For instance, `minNivenNum({15, 7, 10}, 0)` = `7`, `minNivenNum({35, 36, 51}, 0)` = `36`.

isNiven

```
 public static boolean isNiven(int n) {
        return n % sumDigits(n) == 0;
}
```

minNivenNum

```
public static int minNivenNum(int[] arr, int i) {
    if (i == arr.length - 1){
        if (isNiven(arr[i])) {
            return arr[i];
        }
        return Integer.MAX_VALUE;
    }
    if (isNiven(arr[i])) {
        return Math.min(arr[i], minNivenNum(arr,i + 1));
    }
    return Math.min(Integer.MAX_VALUE, minNivenNum(arr,i + 1));
}
```

> We accepted slightly longer implementations of `isNiven`, but it needed to use `sumDigits` to receive full credit. `minNivenNum` could be implemented in a few different ways, this is just one solution.

Buggy Code

Oh no! The University's Covid Test Registration System got a virus and is no longer sending a message to students if they don't get their Covid tests!!!! Help Penn protect us by fixing the bugs in the below function (and by socially distancing, wearing masks, staying inside whenever possible)!

The below function is meant to take in a Boolean array where each index is true if person i has taken their Covid test. It returns a string containing all of the people who haven't taken their test (they will all get a strongly worded email by Amy Gutmann very soon). If there are more than 5 people who haven't taken their Covid test, we want to print to the console saying `"AHHHHHHH"`.

There are seven lines containing bugs in the following function; however, a line may include more than one bug. For each bug, identify the line number, provide a very brief explanation of the bug(s), and rewrite the line to fix the bug.

Please include all errors in the following format, with each error on its own line: Line Number / Description of error / Correct code (Example: Line 42 / Doesn't parse string `x` for integer / `Integer.parseInt(x)`)

Question:

```
public static String covidTests(boolean people) {
    int count = 0;
    String nonTesters;
    int len = people.length();
    for (int i = 0, i < len, i++) {
        if (people[i]) {
            count++;
            nonTesters = nonTesters + i + ", ";
        }
    }
    if (count > 5) {
        System.out.println(AHHHHHHHHHH);
    }
    return count;
}
```

Answer

> There were actually 9 things we considered valid bugs in this code. You still needed to find just 7 of them for full credit. Here is the corrected solution with the bugs commented on their corresponding lines.

```
public static String covidTests(boolean[] people) { //BUG on []
    int count = 0;
    String nonTesters = ""; //BUG, need to initialize instead of leaving
null
    int len = people.length; //BUG on .length, BUG on missing ;
    for (int i = 0; i < len; i++) { // BUG on , ; loop
        if (!people[i]) { //BUG on !
            count++;
            nonTesters = nonTesters + i + ", ";
        }
    }
    if (count > 5) {
        System.out.println("AHHHHHHH"); // BUG missing quotes, BUG on
wrong number of Hs
     }
    return nonTesters; //BUG wrong return value
}
```

Dino Run

Dinosaur Game is a small browser game built into the Google Chrome web browser. In the game, you play as a tiny T-Rex running along a desert landscape, bravely jumping over perilous obstacles. The game has an avid

following among our TAs, and so they'd like to start making their own custom versions. You've been tasked with using your programming skills to help them build the courses that the dinosaur has to run along.

To build Dinosaur Game, we'll implement the courses as arrays of doubles. Each position in the array indicates the height of an obstacle inside that position on the course. For example, we might have the following example of an array: `double[] course = {0, 0, 0, 0, 0, 0, 4.0, 0, 0, 0, 0, 0, 0, 0, 2.6, 2.8, 2.4, 0, 0, 0};`

If we were to visualize this course, we'd see mostly flat ground with two major obstacles. We define an obstacle as a consecutive sequence of cells with non-zero heights. The first obstacle starts and ends at `course[6]`. The next obstacle is wider, ranging from `course[14]` to `course[16]`.

pullSliceFromCourse

To generate a never-ending track for the dinosaur to run on, we'll implement a function `pullSliceFromCourse` that returns a copy of a sequence (called a slice) from an array of doubles `course` starting at index `start` with a length of `length`. Keep in mind, a slice can be as small as one item. `pullSliceFromCourse` should wrap around the input array course if necessary. Wrapping around means that when `start + length >= course.length`, then we start pulling indices from the beginning of the array after we hit the end of it. For example, if `double[] course = {0, 0.3, 0.4, 0, 1.2}`:

- `pullSliceFromCourse(0, 1, course)` should return the array `{0}`
- `pullSliceFromCourse(2, 3, course)` should return the array `{0.4, 0, 1.2}`
- `pullSliceFromCourse(2, 5, course)` should return the array `{0.4, 0, 1.2, 0, 0.3}` (this is a wraparound case, and the sequence of elements we pull is `{course[2], course[3], course[4], course[0], course[1]}`)
- `pullSliceFromCourse(0, course.length, course)` should return an array identical to course. You can assume that `length > 0`, `0 <= start < course.length`, and `course.length > 0`.

SOLUTION:

```
public static double[] pullSliceFromCourse(int start, int length, double[]
course) {
    double[] slice = new double[length];
    for (int i = start; i < start + length; i++) {
        slice[i - start] = course[i % course.length];
    }
    return slice;
}
```

> Other correct solutions could implement the mapping differently by letting the loop control variable range from `0` -> `length - 1`.

checkValidSlice

We also need some way to check to make sure that a particular slice input is valid. If any obstacle is too long or too tall, then the dinosaur will never be able to jump over them! We say that an obstacle is too long if it takes up more than three consecutive indices. For example:

- if `double[] slice = {0, 0.3, 0.4, 0, 1.2, 0, 2.0, 2.0, 2.0}`, `checkValidSlice(slice)` returns `true`. There are three obstacles, and they take up only two, one, and three consecutive positions in the array.
- if `double[] slice = {0, 0.3, 0.4, 0.9, 1.2, 0, 2.0, 2.0, 2.0}`, `checkValidSlice(slice)` returns `false`. There are only two obstacles this time, but the first takes up four consecutive positions in the array.

We say that a slice contains an obstacle that's too tall if any of the heights in slice are greater than 4.0. For example:

- if `double[] slice = {0.4, 0, 1.2}`, `checkValidSlice(slice)` returns `true`.
- if `double[] slice = {4.0, 4.1, 0, 1.0, 1.1}`, `checkValidSlice(slice)` returns `false` because `slice[1]` is `4.1`.

You can assume that `slice.length > 0`.

SOLUTION:

```
public boolean checkValidSlice(double[] slice) {
    int currentLength = 0;
    boolean inObstacle = false;
    for (int i = 0; i < slice.length; i++) {
        if (slice[i] > 4.0) {
            return false;
        }
        if (slice[i] > 0) {
            if (!inObstacle) {
                inObstacle = true;
            }
            currentLength++;
            if (currentLength > 3) {
                return false;
            }
        } else if (inObstacle) {
            inObstacle = false;
            currentLength = 0;
        }
    }
    return true;
}
```

> This solution is far from the most elegant, but it helps to illustrate the constituent steps of the algorithm: check each position, rule out too-tall obstacles, start counting the length of obstacles when encountered, and rule out too-long obstacles.

generateNextPortion

Now that you've written the previous two functions, you can complete the following function `generateNextPortion(int length, double[] course)` to generate the next randomized sequence of the course for the T-Rex to run across. Specifically, `generateNextPortion` should call

`pullSliceFromCourse` to choose a slice of course of length `length` starting at a random starting index chosen uniformly between `0` and `course.length - 1`, inclusive. The slice should be checked for its validity using `checkValidSlice`. If the slice is a valid one, return it. Otherwise, if the slice is not valid, print `"default"` and return an array of length `length` containing just `0`s. Make sure to use the functions you already wrote to do this: you will lose points if you do not.

SOLUTION

```
public static double[] generateNextPortion(int length, double[] course) {
    int start = (int) (Math.random() * course.length);
    double[] candidate = pullSliceFromCourse(start, length, course);
    if (checkValidSlice(candidate)) {
        return candidate;
    } else {
        System.out.println("default");
        return new double[length];
    }
}
```