CIS 110 — Introduction to Computer Programming
Fall 2019 — Exam 2

Name: _____

Recitation #: _____

Pennkey (e.g. eeaton): _____

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:

- Do not open this exam until told by the proctor.
- You will have exactly 110 minutes to take this exam.
- Make sure your phone is turned OFF (not on vibrate!) before the exam starts.
- Food, gum, and drink are strictly forbidden.
- 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, with one exception**: you <u>must</u> tear off scratch paper and the appendix page. You must turn in both your exam and the separated pages.** Do not take any sheets of paper with you.
- 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. You may use the back of pages for additional scratch work.
- 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 will 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.

1.) **The Easy One** (2 pts)

Write your PennKey (NOT YOUR NAME) on every exam page. This is in case the pages get separated during grading.

2.) **Short and Sweet** (18 points total, 2 pts each)

Determine whether each of the following statements is true or false. You must write either TRUE or FALSE. Writing T, F, t, or f will receive no credit.

It is not possible for a class to implement more than one interface	FALSE
100101 in binary has a decimal value of 37	TRUE
`Math.PI` is a `final` variable	TRUE
The time complexity of `ArrayList.get()` is linear and can be represented as `O(n),` where `n` is the number of elements in the array	FALSE
The first element inserted into a queue with `enqueue()` is the first element to be retrieved by `dequeue()`	TRUE
An object declared as "`Candy c = new TwixBar()`" is able to access all public methods in both the `Candy` and the `TwixBar` classes, including public methods not in the `Candy` interface. Assume that `TwixBar` implements `Candy`.	FALSE
References to objects are stored on the heap.	FALSE
When a class implements an interface, it must provide values for all member variables declared in that interface.	FALSE
If a field in a `private inner` class is `public`, it can be accessed in the outer class	TRUE

3.) **Oh, Sorting!** (12 points total)

```
public static void sort1(int arr[]) {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                arr[j] = key;
                j = j - 1;
            }
            arr[j + 1] = key;
       }
}

public static void sort2(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n-1; i++) {
            int min_idx = i;
            for (int j = i+1; j < n; j++) {
                if (arr[j] < arr[min_idx]) {
                    min_idx = j;
                }
            }
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }

 }
```

What is the name of the algorithm that is implemented in `sort1`? What is its big-Oh runtime? (4 pts)

INSERTION SORT. O(n^2)

What is the name of the algorithm that is implemented in `sort2`? What is its big-Oh runtime? (4 pts)

SELECTION SORT. O(n^2)

If `arr` is nearly sorted (such as {1, 2, 3, 5, 4}), which will be faster: `sort1`, `sort2`, or will they take equal time? (4 pts)
sort 1

4.) **Bugging Out!** (16 points total)

This code is riddled with bugs and unless you can fix it, no one will be able to make new friends at Penn! For each bug that you find, write the line number on the left and write the corrected line (exactly as it should be written in code) in the box on the right. You may or may not need to use all of the spaces provided.
`firstFriends` represents the first friends you make at Penn - you can only make 10! When you make a friend, add them to your array.

```
1 public class PennStudent {
2    String name;
3    PennStudent[] firstFriends;
4
5    //Constructor to make a new PennStudent
6    public PennStudent(String name) {
7       name = this.name;
8       firstFriends = new PennStudent[10];
9    }
10
11    public void makeFriend(PennStudent name) {
12        firstFriends[0] = this.name;
13        for(int i = 0; i <= firstFriends.length; i++) {
14            if(firstFriends[i] = null) {
15                firstFriends[i] = name;
16                break;
17            }
18        }
19        return firstFriends;
20    }
21
22    public static void main(String [] args) {
23        PennStudent michael = PennStudent("Michael");
24        PennStudent sierra = PennStudent("Sierra");
25        PennStudent rachel = PennStudent("Rachel");
26        PennStudent max = PennStudent("Max");
27        michael.makeFriend(max);
28        michael.makeFriend(sierra);
29        michael.makeFriend(rachel);
30    }
31 }
```

List of Valid Bugs

- Line 7. this.name = name
- Line 11, should return type PennStudent[]
- Line 12, delete line.
- Line 13, should be <, not <=
- Line 14, should be ==, not =
- Line 19, delete line.
- Lines 23-26, should be new PennStudent("...");

5.) You can count on it! (22 points)

Implement the function `hasSubsetSum()` that takes as input an array of numbers, determines whether or not a particular target value can be achieved by summing any combination of numbers in the array. Numbers in the array can be used at most once in the sum. The table below contains example interactions.

Arguments		**Return Value (and justification)**
`values`	**`target`**	
`{3, 2, -9, 4, 1}`	`8`	`true (3 + 4 + 1 = 8)`
`{3, 2, -9, 4, 1}`	`2`	`true (2 = 2)`
`{3, 2, -9, 4, 1}`	`-4`	`true (3 + 2 + -9 = -4)`
`{3, 2, -9, 4, 1}`	`18`	`false`
`{3, 2, -9, 4, 1}`	`11`	`false`

Your function must call a recursive helper function (which you must also implement) to receive any credit; the function header and implementation is up to you. If you're provided with a `null` array, throw an appropriate exception.

```
public static boolean hasSubsetSum(int[] values, int target) {
       if (values == null) {
            throw new IllegalArgumentException("Cannot provide null values”);
       }
       return hasSubsetSumRecOpt1(values, target, 0, 0);
   }
```

```
// answer key provides two of the most common ways to implement this

   private static boolean hasSubsetSumRecOpt1(int[] values, int target, int
index, int currSum) {
       if (index >= values.length) {
            return false;
       }
       return target == currSum ||
               hasSubsetSumRecOpt1(values, target, index + 1, currSum +
values[index]) ||
               hasSubsetSumRecOpt1(values, target, index + 1, currSum);
   }

   private static boolean hasSubsetSumRecOpt2(int[] values, int target, int
index, int currSum) {
       if (index >= values.length) {
            return false;
       }
       if (target == currSum) {
            return true;
       }
       boolean include = hasSubsetSumRecOpt2(values, target, index + 1,
currSum + values[index]);
       boolean dontInclude = hasSubsetSumRecOpt2(values, target, index + 1,
currSum);

       return include || dontInclude;
   }
```

What is the big-Oh runtime of `hasSubsetSum()`?

O(2^n)

Now, write 3 JUnit tests for `hasSubsetSum()`. You should have one test for a `true` case, one for a `false` case, and one where the input array is `null`. Assume `hasSubsetSum()` is in a class called `Exam`. Do not use any of the examples from the table above. Assume all relevant import statements are already imported - do not rewrite them.

```
public class ExamTest {

@Test
public void testHasSubsetSumTrue() {
      int[] values = {1, 2, 3, 4};
      int target = 8;
      assertTrue(Exam.hasSubsetSum(values, target));
}

@Test
public void testHasSubsetSumFalse() {
      int[] values = {1, 2, 3, 4};
      int target = 11;
      assertFalse(Exam.hasSubsetSum(values, target));
}

@Test(expected=IllegalArgumentException.class)
public void testHasSubsetSumNullArray() {
      int[] values = null;
      int target = 11;
      Exam.hasSubsetSum(values, target);
}









}
```

6.) Pi(a)zza Par-tay! (18 points)

Professors Eaton and Mally are throwing a pizza party to reward the students who consistently attended recitations. Students who attended recitation at least seven times are invited. The names of invited students are stored in a linked list; however, some students who attended fewer than seven recitations were accidentally added to the list.

To help the professors fix the list of invitees, write a class `LinkedList`. Your class should have whatever private instance variables and constructors you typically have in a linked list. You also must write a public method called `uninvite` which removes from the list students with fewer than seven recitation attendances -- the method header and implementation (recursive or iterative) are up to you. There should be no other public methods (e.g., do not implement `add()`).

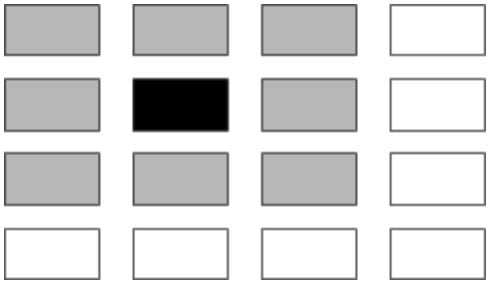
We have provided `Student` and `Node` classes on page 16 that you must use in your implementation.

```
public class LinkedList {

   private Node head;
   public LinkedList(Node head) {
       this.head = head;
   }

   // iterative solution
   public void uninviteIter() {
       // if empty, do nothing
       if (this.head == null) {
            return;
       }

       // edge case where head is issue
       while (this.head != null && this.head.student.recitationVisits < 7) {
           this.head = this.head.next;
       }

       Node curr = this.head;
       while (curr.next != null) {
           if (curr.next.student.recitationVisits < 7) {
               curr.next = curr.next.next;
           } else {
               curr = curr.next;
           }
```

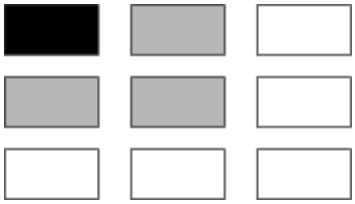
```
        }
    }

    // recursive solution
    public void uninviteRec() {
        while (this.head != null && this.head.student.recitationVisits < 7) {
            this.head = this.head.next;
        }
        uninviteRecHelper(this.head);
    }

    private void uninviteRecHelper(Node curr) {
        if (curr == null || curr.next == null) {
            return;
        }
        if (curr.next.val.recitationVisits < 7) {
            curr.next = curr.next.next;
            uninviteRecHelper(curr);
        }
        uninviteRecHelper(curr.next);
    }


}
```

7.) Seeing Double (22 points total)

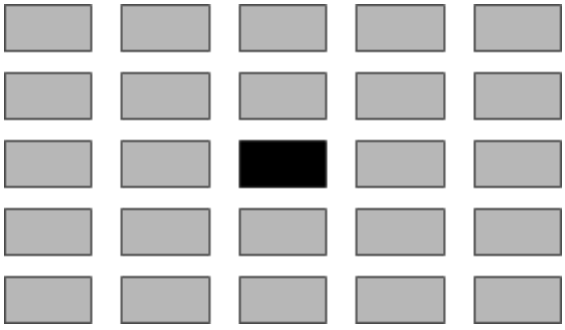
In this problem, you will write a program that outputs a new image based on the input image, performing a box blur on the original image. A box blur entails determining the color for a given pixel by averaging its color with the colors of its neighbors. The colors are represented as 32-bit integers, just like in Steganography. For example, in the diagram below, the black pixel would be replaced by the average of itself and all the grey pixels.



Your implementation must account correctly for the pixels on the edge of the image, such as the case shown below. Incorporate neighboring pixels within the bounds of the image, but be sure not to hit index-out-of-bounds errors.



The box blur can also include a distance that describes how far away neighbors can be to be considered part of the box blur of a pixel. For example, if the distance is 2, the value of a pixel would be the average of all of its direct neighbors as well as all of **their** neighbors.



To implement this program, we will start by first implementing a helper function `blurPixel()` that returns the blurred value for a particular pixel given its location and a search distance. Put your implementation on the next page.

```
public static int[][] blurImage(int[][] img, int radius) {
    int[][] blurredImage = new int[img.length][img[0].length];
    for (int i = 0; i < img.length; i++) {
        for (int j = 0; j < img[i].length; j++) {
            blurredImage[i][j] = blurPixel(img, i, j, radius);
        }
    }
    return blurredImage;
}

private static int blurPixel(int[][] img, int row, int col, int radius) {
    int sum = 0;
    int pixels = 0;
    for (int x = row - radius; x <= row + radius; x++) {
        for (int y = col - radius; y <= col + radius; y++) {
            if (x >= 0 && x < img.length && y >= 0 && y < img[x].length) {
                pixels ++;
                sum += img[x][y];
            }
        }
    }
    return sum / pixels;
}
```

That's it! Relax and check your answers. **Do not discuss the contents of this exam** (especially on Piazza), as there are students who need to take makeup exams.

Be sure that you separated the last three pages from the exam. Put the scrap paper in one hand, and the exam in the other along with your photo ID, and quietly turn in your exam.

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.

Scrap Paper

(This page is intentionally blank)

Any work on this page cannot be graded.

You **must detach this page from the exam**, and you **must turn it in**.

Scrap Paper

(This page is intentionally blank)

Any work on this page cannot be graded.

You **must detach this page from the exam**, and you **must turn it in**.

Appendix

You **must detach this page from the exam**, and you **must turn it in**.

Classes for Question 6:

```
public static class Student {

        public int recitationVisits;
        private String name;
        private int age;


        public Student(int rec, String name, int age) {
            this.recitationVisits = rec;
            this.name = name;
            this.age = age;
        }

        public int getRecitationVisits() {
            return this.recitationVisits;
        }


        public String getName(){
            return this.name;
        }

        public int getAge() {
            return this.age;
        }
 }

public static class Node {
        public Student val;
        public Node next;


        public Node(Student val) {
            this.val = val;
            this.next = null;
        }
}
```