CIS 110 — Introduction to Computer Programming
Summer 2017 — Final

Name: _____

Recitation # (e.g., 201): _____

Pennkey (e.g., paulmcb): _____

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 120 minutes to finish it.
- **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
- Food, gum, and drink are strictly forbidden.
- **You may not use your phone or open your bag for <u>any</u> 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 out in proper java format, including all curly braces and semicolons.
- **<u>Do not separate the pages.</u>** You may tear off the one scratch page at the end of the exam. This scratch paper must be turned in or you lose 3 points.
- Turn in all scratch paper to your exam. Do not take any sheets of paper with you.
- If you require extra paper, please use the backs of the exam pages or the extra pages provided at the end of the exam. **Only answers on the FRONT of pages will be grading. The back is for scratch work only.**
- 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 answer them.
- When you turn in your exam, you may be required to show ID. **If you forgot to bring your ID, talk to an exam proctor <u>immediately</u>.**
- We wish you the best of luck.

Scores: [For instructor use only]

Question 0		1 pt
Question 1		14 pts
Question 2		6 pts
Question 3		11 pts
Question 4		8 pts
Question 5		10 pts
Question 6		6 pts
Question 7		24 pts
Total:		**80 pts**

0) (1 point) The Easy One:

- Check that your exam has all 16 pages (excluding the cover sheet).
- Write your name, recitation number, and PennKey (username) on the front of the exam.
- Sign the certification that you comply with the Penn Academic Integrity Code.

1.) RECURSION (14 pts total)

1.1) Exponents

(2 points) Fill in the blank below for a recursive function that finds the value of x^y where y is a non-negative integer, and x is not zero. It's worth noting that $3^4 = 3 * 3^3$, $3^2 = 3 * 3^1$. Note that any non-zero number to the power of zero is 1.

```
public static int power(int x, int y) {
    if (y == 0) { //exponent is 0
        return 1;
    }
    return x * power(x, y-1);
}
```

1.2) Summing an Array

Consider function `sumArray(int[] array, int start)` below that finds the sum of an array recursively. This is done by finding, for each index `start` of the array, the sum of all elements of the array from start to array.length – 1 by adding start to the sum of all remaining elements.

Thus, sumArray([4,2,1],0) would return 7 (adding every element from zero forward) while sumArray([4,2,1],1) would be 3 (since we would only add the elements 2 and 1 together.

This function is recursive in nature.

a) **(2 points)** Give an example **base case** input (in code) and what the output would be.

sumArray([4,2,1], 3)

output = 0

b) **(4 points)** Using the base case above, fill in the blank to make the function work

```
public static int sumArray(int[] array, int start){
    if (start < 0 || start > array.length) { //Error
        throw new RuntimeException("ERROR");

    if (start == array.length) {  //base case

            return 0; //base case output

    } else {    //recursive step

            return array[start] + summArray(array, start+1);
    }
}
```

.1.3) (6 points) Writing Recursion

Given the following implementation of singly linked list of integers, answer the question on the next page:

```
public class LinkedList{
    public IntNode first;

    public LinkedList(){
        first = null;
    }

    public boolean isEmpty(){
        return (first == null);
    }

    public void insert(int x){
        //Code redacted
        //You can assume insert works
    }
}
----------------------------------------
public class IntNode{
    public int value;
    public IntNode next;

    public IntNode(int value){
        this.value = value;
        this.next = null;
```

```
    }
}
```

Below, write a **recursive** method `count(IntNode node, int x)` that returns the number of times x appears in an instance of `LinkedList` .

Start by identifying the base case, then the recursive step.

If you find you are not able to write a recursive solution, you may write an iterative (looping) solution. However, you will receive an **automatic 2 point deduction** for iterative solutions.

```
public int count(IntNode node, int x) {
      if (node == null) {
            return 0;
      } else {
            if (node.value == x) {
                  return 1 + count(node.next, x);
            } else {
                  return count(node.next, x);
            }
      }
}


//Iterative solution – note this lost points

public int count (IntNode node, int x) {
      int out = 0;
      while (node != null) {
            if (node.value == x) {
                  out++;
            }
      }
      return out;
}
```

2) RECURSION TRACING (6 points total)

I brought in JJ Abrams because I wanted to reboot an old exam question. Like most things he does, he ended up ruining it. I can't even recognize what this code does anymore.

```
public class Nostalgia {
    public static int lensFlare(int emoSpock, int superEight) {
        if (emoSpock < superEight) {
            return emoSpock;
        } else {
            return lensFlare(emoSpock - superEight, superEight);
        }
    }

    public static void main(String[] args) {
        int newHopeRemake = lensFlare(args[0], args[1]);
        System.out.println(newHopeRemake);
    }
}
```

To help me out, tell me what prints as a result of the following calls (use the following blank page for notes).

(1 point each)

a) java Nostalgia 1 2 ___1_____

b) java Nostalgia 2 1 ____0_____

c) java Nostalgia 12 5 ___2_____

d) java Nostalgia 22 6 ___4_____

e) **(2 points)** What built in Java operation does this mimic?

Modulus Function

3) USING OBJECTS (11 points total)

You may not know this, but Professor McBurney is a diehard SportsBall fan (go "Fightin' Fighters"). SportsBall is truly the greatest modern game, fit for the attention spans of today's audiences, where every game between two teams is decided by a coin flip. **That's the entire game.** Imagine we had two teams:

Team 1) "Penn Quakers" – 5 wins, 3 losses.

And

Team 2) "Princeton Tigers" – 3 wins, 4 losses.

If the two teams played, they would have a 50% chance of either team winning.

If Penn wins, the result would be Penn increasing their number of wins from 5 to 6, Princeton increasing their number of losses from 4 to 5. Penn would still have 3 losses, Princeton would still have 3 wins.

If, however, Princeton wins, then Princeton goes from 3 wins to 4 wins, and Penn goes from 3 losses to 4 losses. Penn would still have 5 wins, and Princeton would still have 4 losses.

You are given an incomplete Team.java on the next page. Each Team instance represents a SportsBall team. The play method is used to play a game between the two teams.

This could be tested using the following code:

```
public static void main(String[] args) {
     // create two teams that both have initially
     // zero wins and zero losses
     SportsBallTeam penn = new SportsBallTeam("Penn");
     SportsBallTeam princeton = new SportsBallTeam("Princeton");

     // play a match between them
     penn.play(princeton);

     //now one team should have 1 win, and the other
     //should have one loss
}
```

a) (7 points) Fill in the blanks to finish this class.

```
public class SportsBallTeam {
    public final String name;
    private int wins, losses;
    //constructor
    public SportsBallTeam(String name) {
        this.name = name;
        wins = 0;
        losses = 0;
    }
    //getters
    public int getWins() {

        return wins;
    }

    //getters
    public int getLosses() {

        return losses;
    }
    public void play(SportsBallTeam opponent) {

        if ( Math.random() < 0.5  ) { //50% chance opponent wins

             opponent.wins++;    //increment opponent wins

             losses++; //increment calling instance's losses

        } else {

            opponent.losses++; //increment opponent losses

            wins++;//increment calling instance's wins
        }
    }
}
```

b) (4 points) For both of the lines below, explain why they would result in a compile-time error if called **from another class** (assuming `penn` and `princeton` are existing SportsBallTeam variables):

```
penn.name = "The best team ever";
```

Name is a final String, meaning once it is initially set, it cannot be changed

```
princeton.wins = -10;
```

Wins is a private variable, meaning it cannot be modified externally.

4.) STATIC (8 points total)

Consider the following object.

```
public class RGBColor {
    public final double r, g, b;

    public RGBColor RED = new RGBColor(1.0, 0.0, 0.0);
    public RGBColor GREEN = new RGBColor(0.0, 1.0, 0.0);
    public RGBColor BLUE = new RGBColor(0.0, 0.0, 1.0);
    public RGBColor WHITE = new RGBColor(1.0, 1.0, 1.0);

    public RGBColor(double r, double g, double b) {
        this.r = r;
        this.g = g;
        this.b = b;
    }

   /**
    * Returns the grayscale (0.0 - 1.0) value of this color
    */

    public double toGrayscale() {
        return (r + g + b) / 3.0;
    }

    /**
     * Returns the grayscale (0.0 - 1.0) value of the input color triplet
     */
    public double tripletGrayscale(double r, double g, double b) {
        RGBColor temp = new RGBColor(r, g, b);
        return temp.toGrayscale();
    }

    public String toString() {
        return "Red: " + r + " , Green: " + g + " , Blue: " + b;
    }

    public static void main(String[] args) {
        RGBColor yellow = new RGBColor (1.0, 1.0, 0.0);
        System.out.println(yellow);
    }
}
```

a) **(3 points)** Some of the fields should be static (in fact, if you don't make some of the fields static, this will crash when you create an instance). List the **fields** that should be static: (Do not list any methods, we are only talking about fields right now).

RED, GREEN, BLUE, WHITE must be static

b) (3 points) What prints as a result of running main (assuming you made the proper corrections in a)?

Prints:

Red: 1.0 , Green: 1.0 , Blue: 0.0

c) (1 point) Which method should be static (there is only one)?

tripletGrayscale

d) (1 point) How would you call the method from c **outside of the class**? Keep in mind how static methods are called in other classes. Just give one example of a line of code that calls this method and sets an output to a variable.

```
RGBColor.tripletGrayscale(0.25, 0.5, 0.75);
```

5.) SORTING (10 points)
Sort each array in **ascending** order using the specified technique. Show the state of the array after each iteration through the sorting loop.

a) Insertion sort – {8, 3, 4, 1, 5} **(4 points)**

83415
38415
34815
13485
13458

b) Selection sort– {8, 3, 4, 1, 5} **(4 points)**

83415
13485
13485
13485
13458

Students were expected to write the three intermediate steps (in directions "AFTER EACH ITERATION")

c) **(1 point)** Both of these sorts are far slower than what type of **recursive** sort?

MergeSort

d) **(1 point)** Describe in plain English (not code) what the base case this sorting technique.

MergeSort on an array of size 1 or size 0.

6) INTERFACES (6 points)

Answer the follow questions about the interface below:

```
public interface GeoCoordinates{
  public double getLongitude();
  //get the longitude coordinate

  public double getLatitude();
  //get the latitude coordinate
}
```

(6 points) Write the java code for a class that implements this interface. This class should have a constructor that takes in two numbers, `(double longitude, double latitude)`. DO NOT do **any** error checking on the coordinates. This will waste your time working on the exam and our time grading it. This means you don't have to check of longitude is between -180 and 180, or if latitude is between -90 and 90. Select your own name for the class.

```
public class MyGeoCoordinates implements GeoCoordinates {

     private double longitude, latitude;

     public MyGeoCoordinates(double longitude, double latitude){
          this.longitude = longitude;
          this.latitude = latitude;
     }

     public double getLongitude() {
          return longitude;
     }

     public double getLatitude() {
          return latitude
     }
}
```

7) LINKED DATA STRUCTURES (24 points)

7.1) Based on what we discussed in class, illustrate these data structures (indicating where the "first" node is for both, and the "last" node for queue). There is no fixed way to draw these lists, but order should be clear, as well as where the head node is.

a) **(3 points)** A queue "q" implemented as a LinkedList after the instructions

```
q.queue("cat");
q.dequeue();
q.queue("river");
q.queue("dog");
q.dequeue();
q.queue("bag");
q.queue("cat");
q.dequeue();
```

bag[first] -> cat[last] -> null

b) **(3 points)** A stack "s" implemented as a LinkedList after the instructions

```
s.push("cat");
s.pop();
s.push("river");
s.push("dog");
s.pop();
s.push("bag");
s.push("cat");
s.pop();
```

bag[first] -> river -> null

7.2) In this question, you will write the insert method of a Sorted Doubly Linked List. Each element is added to the list in sorted order. For example, if the list contains:

(first) 5 → 8 → 10 (last)

And you inserted 7, the end result would be a list:

(first) 5 → 7 → 8 → 10 (last)

You CAN have duplicate values. For example, if you inserted 7 again, you would get

(first) 5 → 7 → 7 → 8 → 10 (last)

Notice it doesn't matter if you add the 7 before or after the previous 7.

Using the following code, answer questions on the next page. **YOU MAY NOT RIP THIS PAGE OFF.**:

```
public class SortedDoublyLinkedList{
    public IntNode first;
    public IntNode last;

    public SortedDoublyLinkedList(){
        first = null;
        last = null;
    }

    public boolean isEmpty(){
        return (first == null);
    }

    public int size() {
       //TODO: Return the size of the list;
    }

    public void insert(int x){

        //TODO: Insert into the Linked List in sorted Order
    }
}
-----------------------------------------------------------------
public class IntNode{
    public int value;
    public IntNode next;
    public IntNode prev;

    public IntNode(int x){
        value = x;
        next = null;
        prev = null;
    }
}
```

a) (4 points) Write the `size()` function to return the number of elements in the list. YOU MAY NOT ALTER THE OBJECT IN ANY WAY. You cannot assume a size variable exists. You should not need more space than given. The lower half of this page is for you to use as scratch space for both a) and b)

```
public int size() {
  int size = 0;
  for (IntNode node = first; node != null; node = node.next)

       size++;
  }
  return size;
}
```

b) (14 points) Write the function `insert(int x)` such that the list inserts x in sorted ASCENDING ORDER. This means the **`first`** of the list must be the smallest value, the **`last`** of the list must be the largest value. Your insert function must work for all cases:

- Inserting into an empty list
- Inserting a new largest value into the list
- Inserting a new smallest value into the list
- Inserting a value that goes in the middle of the list

```
public void insert(int x) {
	IntNode newNode = new IntNode(x); //create new Node

	//insert into empty list
	if (isEmpty()) {
		first = newNode;
		last = newNode;
	}

	//insert BEFORE first
	else if (x < first.value) {
		newNode.next = first;
		first.prev = newNode;
		first = newNode;
	}

	//insert AFTER last
	else if (x > last.value) {
		last.next = newNode;
		newNode.prev = last;
		last = newNode;
	}

	else {
		//insert in middle
		IntNode temp;
		for(temp = first; temp.next.value < x; temp = temp.next){
			//empty for loop, gets temp to BEFORE the insertion occurs
		}
		//temp is BEFORE the node being insert
		//temp.next is AFTER the node being inserted
		IntNode after = temp.next;

		//connect the links
		temp.next = newNode;
		newNode.prev = temp;
		after.prev = newNode;
		newNode.next = after;
	}
}
```

EXTRA ANSWER SPACE: You may NOT rip this page off, but work written on the <u>FRONT</u> of this page may be graded.

SCRATCH PAPER: You may rip this page off, but must turn it in at the end of the exam.
If you take this page with you, you lose 3 points.