CIS 110 — Introduction to Computer Programming
Fall 2017 — Final Midterm

Name: _____

Recitation Section# : _____

Pennkey *(e.g., paulmcb)*: _____

DO NOT WRITE YOUR ID# ABOVE, YOU WILL LOSE A POINT

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. You will lose 5 points if your cell phone goes off.**

- 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 2 points per page.

- Turn in all scratch paper to your exam. Do not take any sheets of paper with you.

- If you require extra paper for scratch work, 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]

Front Page		1 pt
Question 1		8 pts
Question 2		6 pts
Question 3		9 pts
Question 4		9 pts
Question 5		10 pts
Question 6		16 pts
Question 7		6 pts
Question 8		5 pts
Question 9		10 pts
Total:		**80 pts**

1.) Starting Slow (8 points)
In the space below, illustrate a mergeSort with the numbers [8, 4, 1, 3, 2, 7, 9, 0] **(3 points)**

Splitting

8413 | 2790
84 | 13 | 27 | 90
8 | 4 | 1 | 3 | 2 | 7 | 9 | 0

Merging:
48 | 13 | 27 | 09
1348 | 0279
01234789

Which type of `List` is faster for each function call on a list variable. You can assume someElement and otherElement are both of type Element (which can be sorted). Assume the list is initially unsorted, and has at least 50 elements. **(5 points)**

Operation	ArrayList<Element>	LinkedList<Element>	They are both the same
list.add(0, someElement)		Faster – in a worst case constant time, while ArrayList is O(n)	
list.remove(0)		Faster –constant time, while ArrayList is O(n)	
list.get(25)	Faster –constant time, while LinkedList is O(n)		
list.add(otherElement)		Very slightly faster, since in a worst case ArrayList is O(n)	This answer was accepted
Collections.sort(list)			They are both O(logn * n), though in Java 8 ArrayList is SLIGHTLY faster.

2) Inheritance (6 points)
Consider the following Color class

```
public class Color {
      public int red, green, blue;  //between 0-255

      public Color (int r, int g, int b) {
            red = r; green = g; blue = b;
      }

      public void brighten() {
            //assume this method works
      }
}
```

We want a class called GrayscaleColor such that it extends color. A Grayscale color with a gray value of x would have RGB values of red = x, blue = x, and green = x. Write the constructor for the class below. You should not need more space than provided. Do NOT do any error checking. You may not add ANY variables. **(2 point)**

```
public class GrayscaleColor extends Color{

      public GrayscaleColor(int x) {

            super(x, x, x);

      }

      public void darken() {
            //assume this method works
      }
}
```

If the following lines of code were written in order, clearly circle the lines of code that would produce SYNTAX errors. **(4 points)** (Answer key has illegals lines red and scratch out)

```
i) Color red = new Color(255, 0, 0);

ii) Color black = new GrayscaleColor(0);

iii) GrayscaleColor white = new GrayscaleColor(255);

iv) GrayscaleColor gray = new Color (127, 127, 127);

v) red.blue = 127;

vi) Color c = (Color) black;

vii) red.darken();

viii) white.blue = 127;

ix) white.brighten();
```

3) Comparing Numbers (9 points)

A bank wants to keep track of both its largest debits AND its largest credits. Therefore, it wants to sort a **List<Account>** by the **absolute value** of the balance in **DESCENDING** order using the **Collections.sort** function. You can assume the class Account has a method **getBalance()**, that returns either a positive or negative number representing the amount of money tied to the credit (positive) or debit(negative).

a) **(1 point)** Given the fill in the blank below, would this be done with a **Comparable<Account>** or **Comparator<Account>** ?

    Comparator<Account>

b) **(4 points)** Fill in the blanks below for the class you write to use in **Collections.sort**

```
public class SortAbsValDescending implements Comparator<Account> {

    public int compare (Account a0, Account a1) {

        return Math.abs( a1.getBalance()) -

              Math.abs( a0.getBalance())
    }

}
```

c) **(2 points)** How would you use the above class to sort the List<Account> called accounts?

```
Collections.sort(accounts, new SortAbsValDescending())
```

d) **(2 points)** Above, you answered the question about either Comparable or Comparator (depending on your answer to **a**). For the one you didn't choose, what is the method declaration for the one method need to implement the interface if you also applied it to Account (don't write the method, JUST the method declaration)?

```
public int compareTo(Account a) {
```

4) Birthdays (9 points)
A pop-culture obsessed CIS 110 student wanted to keep track of every celebrity's age. To do this, they wrote a class called **Person** (since celebrities are people too). The class **Person** will keep track of every instance created by adding that instance to a list. This is so when **advanceYear()** is called, everyone ages uniformally. However, the student didn't finish because they were too in shock from learning that James Franco is nearly 40. The student failed to finish the constructor, failed to specify which variables and methods need to be static, and did not yet write **advanceYear().** Answer the questions on the next page about this code. *YOU CANNOT RIP THIS PAGE OFF*.

```java
public class Person {
     private String name;
     private int age;
     private int currentYear = 2017;
    public List<Person> people = new ArrayList<Person>();
     /**
      * Constructor
      */
    public Person(String name, int birthYear) {
          this.name = name;
          age = currentYear - birthYear;

          people.add(this);
    }
     /**
      * Standard toString() method
      */
    public String toString() {
          return name + " : " + age;
    }
     /**
      * Returns the oldest person in the list
      */
    public Person getOldestCelebrity() {
          int max = 0;
          Person out = null;
          for (Person p : people) {
                if (p.age > max) {
                      max = p.age;
                      out = p;
                }
          }
          return out;
    }
     /**
      * Increase the current year by one, and ages every
      * person created by one year.
      */
    public void advanceYear() {
          //TODO: Part D
    }
}
```

a) On the previous page, fill in the blank in the constructor that adds the newly created instance to the list people. **(2 points)**

b) In the space below, list the VARIABLES (not methods) that should be static for the code to work. **(2 points)**

people, currentYear;

c) In the space below, list the METHODS (not variables) that should be static for the code to work. **(2 points)**

getOldestCelebrity(), advanceYear();

d) In the space below, write the function **advanceYear().** This function should age every person in people by one year and increase the current year. **This function body should be no longer than 4 lines of code.** If it is longer than 4 lines, it will NOT be graded, and you will get zero points. **(3 points)**

```
currentYear++;
for(Person p: people) {
    p.age++;
}
```

5) Breaking the Bank (10 points)

The following class below attempts to simulate a bank account. However, there are a number of bugs. On the next page are several test cases. Each time a value or function output is inside of **test()**, fill in the expected output (based on the method description in the header comment) and the ACTUAL output (the result of execution the code). If the test fails, explain the bug fix under fix. If the test does not fail, leave fix blank.

```java
public class BankAccount {
      public double amount; //amount in the bank
      public Account (double startingBalance) {
            amount = startingBalance;
      }

      /**
       * Returns the current balance of the account.
       */
      public double getBalance() {
            return 5; //this is intentionally wrong, see next page
      }

      /**
       * Returns whether or not your account is either at zero or
       * "in the red" (that is, you owe the bank money)
       */
      public boolean isEmpty() {
            return account < 0;
      }
      /**
       * Add money to the account.
       */
      public void deposit(double amount) {
            amount += amount;
      }

      /**
       * Withdraw money. If you try to withdraw too much,
       * do not allow the withdrawal, but subtract $10 from the
       * account as an overdraft penalty
       */
      public void withdraw(int cash) {
            if (cash < amount) {
                  amount -= 10;
            }
            amount -= cash;
      }

      /**
       * Adds 10% interest to the account.
       */
      public void addInterest() {
            amount = amount * 0.1;
      }
}
```

Test Case	Expected	Actual	Fix
`Account a = new Account (50.0);` `test(a.getBalance());`	50.0	5	In getBalance(), change "return 5" to return balance.
`Account a = new Account(0.0);` `test(a.isEmpty());`	true	false	Return account <= 0

`Account a = new Account(10.0);`
`a.deposit(20.0);`
`test(a.amount);`
	30.0	10.0	Should be: this.amount += amount;

`Account a = new Account(15.0);`
`a.withdraw(10.0);`
`test(a.amount);`
	5.0	-5.0	The if statement should be if (cash > amount). Amount -= cash needs to be inside of an else case

`Account a = new Account(11.0);`
`a.withdraw(15.0);`
`test(a.amount);`
	1.0	-4.0	The if statement should be if (cash > amount) Amount -= cash needs to be inside of an else case

`Account a = new Account(100.0);`
`a.addInterest();`
`test(a.amount);`
	110.0	10.0	Change it to: amount += amount * 0.1;

(2 points per row)

6) Tracing A Schedule (16 points)

A new room reservation system for the engineering buildings is being designed. A Reservation class is being tested within the ReservationSystemDemo class. Using the main function in ReservationSystemDemo (next page), fill out the table at the bottom of the next page with the values at each Point (denoted by comments).

```java
public class Reservation {
    private int roomNum;
    private int numPeople;
    private String reserver;
    private int startTime;
    private int endTime;

    public Reservation() {
        this.roomNum = 100;
        this.numPeople = 100;
    }

    public Reservation(int roomNum, int startTime, int endTime) {
        this.roomNum = roomNum;
        this.numPeople = 30;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    public Reservation(int roomNum, int numPeople, String reserver) {
        this.roomNum = roomNum;
        this.numPeople = numPeople;
        this.reserver = reserver;
        this.startTime = 12;
        this.endTime = 12;
    }

    public void addTime(int hours) {
        endTime += hours;
    }

    public boolean changeReserver(String newReserver) {
        if (newReserver != null) {
            this.reserver = newReserver;
            return true;
        }
        return false;
    }

    public String getReserver() {
        return this.reserver;
    }

    public int getEndTime() {
        return this.endTime;
    }
}
```

```java
public class ReservationSystemDemo {

    public static void main(String[] arr) {
        Reservation a = new Reservation();
        Reservation b = new Reservation(309, 12, 14);
        Reservation c = new Reservation(100, 200, "Jane");

        a.addTime(2);
        // Point A

        b.changeReserver(a.getReserver());
        a = c;
        a.addTime(2);
        // Point B

        if (c.getEndTime() == 12) {
            c = new Reservation();
        } else if (b.getEndTime() == c.getEndTime()) {
            b.addTime(-1);
        }
        // Point C

        a = new Reservation(202, 50, "Will")
        b = a;
        c = b;
        b = new Reservation();
        a.addTime(2);

        //Point D
    }
}
```

Point	a.numPeople	a.reserver	b.endTime	c.endTime
Point A	100	null	14	12
Point B	200	"Jane"	14	14
Point C	200	"Jane"	13	14
Point D	50	"Will"	0	14

(1 point per box)

7) Bit Strings (6 points)

Below is a partially complete function to adds two BitStrings of 8 bits each. For example, the BitString representing 5 plus the BitString representing 7 should result in the bitString representing 12.

However, in a misguided attempt to hold the attention of a younger audience, Will randomly named all variables after social networks. While we deal with explaining to Will that he's old now, and he needs to accept that, you fill in the blanks below.

You are not allowed to change any variable names, or add any code. You can only fill in the blanks.

```java
public static String addBitStrings(String twitter,
                                   String facebook) {

    //set initial values

    String instagram = ""; //output String



    int snapchat = 0; //carry bit



    for(int pinterest = 7; pinterest >= 0; pinterest--) {
        //character from twitter
        int imgur = twitter.charAt(pinterest) – '0';

        //character from facebook
        int reddit = facebook.charAt(pinterest) - '0';



        //find the sum bit and add it to the string



        instagram = ((imgur + reddit + snapchat) % 2) +
                    instagram;

    //determine the new carry bit



        snapchat = (imgur + reddit + snapchat) / 2;
    }

    return instagram;
}
```

8) InLinkedListCeption (5 points)

A professor is keeping track of his students grades on a Recursion homework by using a
LinkedList represented with the following Node class.

```
public class Node {
    public Student s;
    public Node next;
}
```

This uses the following Student class:

```
public class Student {
    public String name; //Student's name
    public int score; //score on the recursion Homework
}
```

The professor wants to know which student did worst on the homework. So he has tasked
you with writing a recursive function to return the name of the student with the lowest
score. Luckily, you understand recursion, so this should be easy. Write a function
getLowestScoreStudent(Node node) that returns the Student object of the student
with the lowest score in either **node** or in any **Node** AFTER it. You can assume no two students
have the same score on the homework.

If the first student in the list is called "**head**", then **getLowestScoreStudent(head)**
should return the Student with the lowest score in the class.

**If you write an iterative solution, instead of a recursive solution, you will get _ZERO_ points
on this question. Don't even think about using the words for, while, etc.**

```
public Student getLowestScoreStudent(Node node) {

 if (node.next == null) {
     return node.s;
 } else {
     Student currentLowest = getLowestScoreStudent(node.next);
     if (node.s.score < currentLowest.score) {
         return node.s;
     } else {
         return currentLowest;
     }
 }
```

9) I'll take Merge on Rye, Hold the Sort (10 points)
In this question, you will write a function that combine two SORTED `List<Integer>` and output a single SORTED list. You **cannot** use Collections.sort(), or any sorting algorithm. No recursion, and no nested loops. You can assume the two lists are already sorted. You can also assume that if there are duplicates, their order doesn't matter. You **do not** have to error check if the lists are sorted OR null, but it is possible either or both lists are empty. This function **must not** add or remove elements to the lists a or b, or change those two lists in any way.

```java
public List<Integer> merge(List<Integer> a, List<Integer> b) {

    int aIndex = 0;
    int bIndex = 0;

    List<Integer> output = new ArrayList<Integer>();

    while (aIndex < a.size() && bIndex < b.size()) {
        if a.get(aIndex) < b.get(bIndex) {
            output.add(a.get(aIndex));
            aIndex++;
        } else {
            Output.add(b.get(bIndex));
            bIndex++;
        }
    }

    while (aIndex < a.size()) {
        output.add(a.get(aIndex));
        aIndex++;
    }

    while (bIndex < b.size()) {
        output.add(b.get(bIndex));
        bIndex++;
    }

    return output;
}
```