CIS 110 — Introduction to Computer Programming Summer 2018 — 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

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.

Date

Scores: [For instructor use only]

Question 0	1 pt
Question 1	11 pts
Question 2	8 pts
Question 3	16 pts
Question 4	10 pts
Question 5	11 pts
Question 6	23 pts
Total:	80 pts

- **0) (1 point)** The Easy One:
 - Check that your exam has all 15 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 (11 pts total)

1.1) Exponents

(4 points) Fill in the blank below for a recursive function that finds the value of x / y using integer division (i.e., dropping the remainder/decimal). You can assume x and y are POSTIVE (non-zero, non-negative) integers.

```
public static int division(int x, int y) {
    if (x < y) {
        return 0;
    }
    return division(x-y, y) + 1;
}</pre>
```

1.2) Counting an Array

Consider the function <code>countArray(int[] array</code>, int value, int start) below that finds the number of times value appears in the array. This is done by finding, for each index start of the array, the number of times value appears between start and the end of the array.

Thus, countArray([4,2,1,2,2,0], 2, 0) would return 3 (There are 3 twos in the array starting from zero) while countArray([4,2,1,2,2,0], 2, 3) would be 2 (Since we only starting counting at index 3, so the first two is not counted).

This function is recursive in nature.

a) (2 points) Give an example base case input to the function (i.e., the values of array, value, and start) AND what it would return.

countArray([4,2,1,2,2,0], 2, 6) – returns 0

This is the base case, instead of **countArray([4,2,1,2,2,3], 2, 5)** – **returns 3** since the base case above would work for an array of size 0, the base case below would not work for an array of size 0. And it is a valid question to ask "how many 3's are in this array of size 0", the answer is always 0.

```
b) (5 points) Using the same recursive function above.
public static int countArray(int[] array, int count, int start){
    if (start < 0 || start > array.length) { //Error
        throw new RuntimeException("ERROR");
    if (start == array.length) {
        return 0;
    } else {
        if (array[start] == count) {
            return 1 + countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        } else {
            return countArray(array, count; start + 1);
        }
        return countArray(array, count; start + 1);
        }
        return countArray(array, count; start + 1);
        return countArray(array, count; start + 1);
    }
    }
    }
    }
}
```

}

}

}

2) RECURSION TRACING (8 points total)

Below is a mystery recursion function. Do not try to work out WHAT it's doing, as the functionality is completely made up. It does nothing useful.

```
public static int mystery(int a, int b)
{
    if (a == 0 || b > 12) {
        System.out.println(a + "," + b);
       return 0;
    } else if (b % 2 == 0) {
        System.out.println(a + "," + b);
        return mystery(a - 1, b + 1) + 3;
    } else if (a % 2 == 0) {
        System.out.println(a + "," + b);
        return mystery(a - 1, b + 2) + 2;
    } else {
        System.out.println(a + "," + b);
        return mystery(a - 1, 2 * a) - 3;
    }
}
```

In the boxes below, write whatever prints when the function is called with the given arguments in order. At the bottom of each box, say what the function call ultimate returns.



3) USING OBJECTS (16 points total)

In this problem, you will be using the card class below. In Klondike Solitaire, as well as Freecell Solitaire, a player can move around the traditional French playing cards into columns. The rule is that each card can be placed under a card exactly **ONE RANK** higher, and of the opposite color (the colors are red and black). I.e., a **red 5** can be placed on a **black 6**, but it cannot be placed on a **red 6**, or any card of any other rank. This class is here to model this. Do not worry about specific suits (hearts, clubs, diamonds, spades), just worry about red/black.

```
public class Card {
    private char rank; //'A', or '2', or '3', etc.
    private boolean isRed; //true if card is red, false if it's black
    public final char[] ALL RANKS =
             {'A','2','3','4','5','6','7','8','9','T','J','Q','K'};
    public Card(char rank, boolean isRed) {
        this.rank = rank;
        this.isRed = isRed;
    }
    public char getRank() {
        return rank;
    }
    public boolean isRed() {
        return isRed;
    }
    public int getRankIndex(char ch) {
        for (int i = 0; i < ALL_RANKS.length; i++) {</pre>
            if (ALL RANKS[i] == ch) {
                return i;
            }
        }
        throw new RuntimeException("ERROR: Invalid rank");
    }
    /**
     * This method returns true if child can be placed on parent in
     * Klondike or Freecell Solitaire (i.e., true if parent is black 6 and
     * child is red 5
     */
    public boolean isValidParent(Card parent, Card child) {
        //TODO: SEE NEXT PAGE
    }
}
```

```
Write one (1) line of code to produce a red '5': (2 points)
```

Card c = new Card(5, true);

```
SCORE_____
```

To Complete the Method isValidParent, reorder all the lines of code below. Use **all** of the lines of code below **exactly once**. Do not use any line of code more than once, and **do not write any code not included below**. (You must rewrite the lines in their entiriety). **(8 points)**

```
}
}
}
}
}
}
} else {
childRank = i;
for (int i = 0; i < ALL RANKS.length; i++) {</pre>
if (ALL RANKS[i] == child.rank) {
if (ALL RANKS[i] == parent.rank) {
if (childRank + 1 == parentRank) {
if (parent.isRed() == child.isRed()) {
int childRank = -1;
int parentRank = -1;
parentRank = i;
public boolean isValidParent(Card parent, Card child) {
return false;
return false;
return true;
```

```
public boolean isValidParent(Card parent, Card child) {
    if (parent.isRed() == child.isRed()) {
       return false;
    int childRank = -1;
    int parentRank = -1;
    for (int i = 0; i < ALL RANKS.length; i++) {
       if (ALL RANKS[i] == child.rank) {
         childRank = i;
       if (ALL RANKS[i] == parent.rank) {
         parentRank = i;
       }
    }
    if (childRank + 1 == parentRank) {
       return true;
    } else {
       return false;
    ł
  }
```

In the space below, list the VARIABLE(S) (not methods) in Card that should be static. (2 points)

ALL_RANKS

In the space below, list the METHOD(S) (not variables) in Card that should be static. (2 points)

getRankIndex(); isValidParent();

Which attributes of a Card can you change **AFTER** calling the constructor from **outside** of the file? (*It's possible you can change all of them, none of them, or only certain ones.*) (2 point)

None

4) Object Theory (10 points)

1. How do you get a NullPointerException (you can either describe it or show a code example)? (3 points)

Using a dot operator on a null object.

Object o = null; System.out.println(o.toString());

2. What function do you implement in order to make an object be printed as something other than "ClassName@Address" (write the entire method declaration, including visibility and return type) (2 points)

public String toString()

3. Give a reason you would want a private method in a class, or say why there no reason to ever make a method private. (**3 points**)

A helper method, that is a method that exists inside of the class to help implement things the class needs, should not be visible outside the class.

4. If a class is called Exam and you have exactly two instances (no more) of the class, midterm and finalExam, and there have a **static** String variable called semester, how should you change semester to "Summer 18" for both instances? **(2 points)**

Exam.semester = "Summer 18"

CIS 110 – Final – Summer 2018	SCORE	Page 8
-------------------------------	-------	--------

5) SORTING (11 points)

Sort each array in **ascending** (smallest to largest) order using the specified technique. Show the state of the array after each iteration through the sorting loop, or after each merge.

a) Insertion sort – {8, 3, 4, 1, 5, 2, 5} (4 points)

 $\begin{array}{c} 3 & 8 & 4 & 1 & 5 & 2 & 5 \\ 3 & 4 & 8 & 1 & 5 & 2 & 5 \\ 1 & 3 & 4 & 8 & 5 & 2 & 5 \\ 1 & 3 & 4 & 5 & 8 & 2 & 5 \\ 1 & 2 & 3 & 4 & 5 & 8 & 5 \\ 1 & 2 & 3 & 4 & 5 & 5 & 8 \end{array}$

b) Merge sort- {8, 3, 4, 1, 5, 2, 5} (4 points)

```
(note that you can split either 3 | 4 or 4 | 3 and not lose points

8 3 4 1 5 2 5

8 3 4 1 5 2 5

3 4 1 5 2 5

8 34 15 2 5

8 34 15 25

348 1255

1234558
```

c) Which one of these sorts is recursive in nature? (don't say why, just name it) (1 point)

Mergesort

d) Which of the below is code for a **selection** sort? (circle the one that is a selection sort). Both pieces of code sort an int[] variable called **array**. Both use the swap function as covered in class (that swaps the location of two values in an array). Note that method declarations have been removed, and variable names obfuscated. (1 point)

```
for (int i = 1; i < array.lenth; i++) {
   for (int j = i; j > 0; j--) {
        if (array[j-1] > (array[j])) {
            swap(array, j - 1, j);
        }
   }
}
```

```
for(int i=0; i<array.length-1; i++){
    int a = array[i];
    int b = i;
    for(int j=i+1; j<array.length; j++){
        if(array[j] < a) {
            a = array[j];
            b = j;
        }
    }
    swap(array, i, b);
}</pre>
```

Below are two examples of a sorting algorithm on the same numbers. Which is a selection sort? (1 point)

8, 2, 6, 3, 6, 1 2, 6, 3, 6, 1, 8 2, 3, 6, 1, 6, 8 2, 3, 1, 6, 6, 8 2, 1, 3, 6, 6, 8 1, 2, 3, 6, 6, 8	8, 2, 6, 3, 6, 1 1, 2, 6, 3, 6, 8 1, 2, 6, 3, 6, 8 1, 2, 3, 6, 6, 8 1, 2, 3, 6, 6, 8 1, 2, 3, 6, 6, 8
---	--

6) LINKED DATA STRUCTURES (23 points)

Below is the class LinkedList which is a SINGLY Linked List. Fill in all the blanks below for the constructor and the methods *isEmpty* and *add*.

```
public class Node {
   public int value;
   public Node next;
   //Constructor that creates a node with the input value
   public Node(int value) {
        this.value = value; //1 point
    }
}
public class LinkedList {
   public Node head; //first element of the list
    /**
    * returns true if the list is empty
     */
   public boolean isEmpty() {
        return head == null;//1 point
    }
    /**
     * Adds the integer x as a new node to the END of the List
     */
   public void add(int x) {
        Node newNode = new Node(x);//1 point
        if (isEmpty()) {
            head = newNode;//1 point
        } else {
            Node t = head; //1 point
             while (t.next != null) {//1 point
                t = t.next;//1 point
            }
            t.next = newNode;
        }
   }
}
```

Write a function addRecursive(int x) that implements the same behavior as the existing add function, however does so recursively. You may need to write a helper function to do this.

Use comments to clearly identify your base case or base cases. There will be no partial credit for writing an iterative solution. If you use *either* for or while even once, you will get no credit. **(6 points)**

```
public void addRecursive(int x) {
    if (isEmpty()) {
        head = new Node(x);
    }
    addHelper(head, x);
}
public void addHelper(Node n, int x) {
    if (n.next == null) {
        n.next = new Node(x);
    } else {
            addHelper(n.next, x);
    }
```

}

Using this implementation of LinkedList, write, on the next page, the function:

```
public boolean isSubList(LinkedList a, LinkedList b) {
    //this method should have been static, technically
```

This function returns true if **a** is a *sublist* of **b**. That is, if all of the elements a can be found inside of b and in the same order. For Example, if:

 $a = \underline{5 \to 3 \to 1}$ $b = 7 \to 5 \to 3 \to 1 \to 2$

Then a is a sublist of b (that is, your function should return true). However, if

 $a = 5 \rightarrow 3 \rightarrow 1$ $b = 7 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 1$

A is <u>NOT</u> a sublist of b. This is because while every element of a is in B, they are not continuous. They are interrupted by the 2. Another example is:

$$a = 5 \rightarrow 3 \rightarrow 1$$
$$b = 7 \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 2$$

This is another case where a is **NOT** a sublist of b, because the elements are not in the same order. Another case where a is **NOT** a sublist of b is when:

$$a = 7 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 2$$
$$b = 5 \rightarrow 3 \rightarrow 1$$

This is a case where a is **NOT** a sublist of b, but \mathbf{b} is a sublist of A. Finally, if a and b are the *same* list, such as:

$$a = 5 \rightarrow 3 \rightarrow 1$$
$$b = 5 \rightarrow 3 \rightarrow 1$$

Then a **IS** a sublist of b (that is, return true). Be careful, however. For example, below, a **IS** a sublist of B. If you look, you can see why this case might be tricky.

$$a = \underline{5 \to 3 \to 1}$$

$$b = 5 \to 3 \to 5 \to \underline{5 \to 3 \to 1} \to 5$$

```
As always, there are a number of ways to do this: here's one
public static boolean isSubList(LinkedList a, LinkedList b) {
        if (a.isEmpty()) {
            return true;
        } else if (b.isEmpty()) {
            return false;
        }
        Node travA = a.head;
        Node travB = b.head;
        while (travB != null) {
            if (travA.value == travB.value) {
                 Node tempA = travA;
                 Node tempB = travB;
                 while (tempB != null) {
                     if (tempA == null) {
                         return true;
                     }
                     if ( tempA.value != tempB.value) {
                         break;
                     }
                     tempA = tempA.next;
                     tempB = tempB.next;
                 }
             }
            travB = travB.next;
        }
        return false;
    }
```

10 points