CSE 131 Introduction to Computer Science **Spring 2016**

Exam II

Given: 21 March 2016 Due: End of Exam Session

This exam is closed-book, closed-notes, no electronic devices allowed. The exception is the "sage page" on which you may have notes to consult during the exam. Answer questions on the pages of the exam. Do not unstaple the pages of this exam nor should you attach any other pages to the exam. You are welcome to use the blank space of the exam for any scratch work.

Your work must be legible. Work that is difficult to read will receive no credit. Be sure code is indented to show its structure. Comments are only needed if you think you need to clarify your work.

You must sign the pledge below for your exam to count. Any cheating will cause the students involved to receive an F for this course. Other action may be taken. If you need to leave the room for any reason prior to turning in your exam, you must leave your exam and any electronic devices with a proctor.

YOU MUST FILL IN YOUR IDENTIFYING INFORMATION CORRECTLY. Failure to do so is grounds for a zero on this exam.

Print clearly the following information:
Name (print clearly):
Student 6-digit ID (print *really* clearly):
Your answers below tell us where to return your graded exam.
What time do you actually attend studio/lab? 11:30am 1pm 2:30pm 4pm
What room? 222 , 218, 216, or 214

Problem Number	Possible Points	Received Points
1	15	
2	17	
3	23	
4	10	
5	25	
6	10	
Total	100	

Pledge: On my honor. I have neither given nor received any unauthorized aid on this exam.

Signed:_____

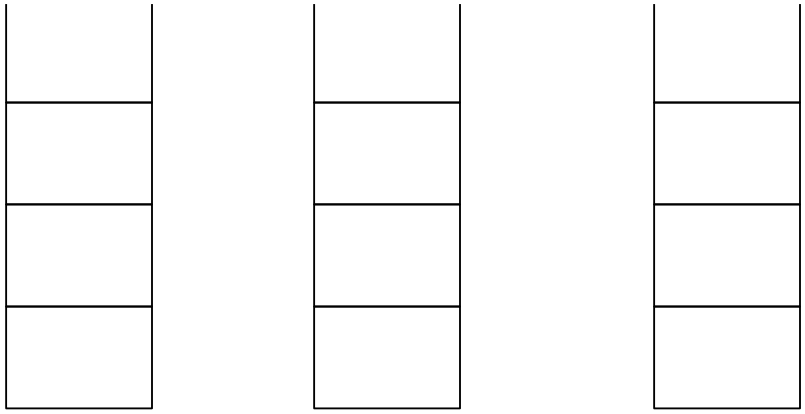
1. (15 points) Consider the following methods:

```
public static int foo(int x, int y, int z ) {
      return  (3*x) + (y*z);
}

public static int bar(int x, int y) {
      return x + y;
}
```

and a call to the methods:

```
int a = foo(bar(2, 3), 4, 3);
```



(a) (b) (c)

a) Show the stack, just prior to **bar** executing
b) Show the stack, just prior to **foo** executing
c) Show the stack, just prior to **foo** returning the answer that will be assigned to **a**

2. (17 points) Consider the following user story:
A **Card** has-a

- integral number value
- suit name (e.g. in a normal deck of cards this might be heart, diamond, spade or club.)
- indication of whether the card has been dealt or is available.

In terms of behavior, the **Card** object should offer the following functionality:

- Initially, Card should be available
- There should be getters but no setters for the Card's value and suit
- There should be a getter and setter for the Card's dealt status

Note that the above requirements do not print out anything. Your methods therefore should not print anything. Instead they should affect the object by changing its instance variable(s) or by returning appropriately typed values to provide the desired functionality.

Below, and on the facing page, write the code for your **Card** class, placing elements of your class between the comments as indicated.

```
public class Card {
// Instance variable(s) below here. For each instance variable
// choose a type that best fits its purpose and usage (4 points)




//Constructor below here (5 points)




//toString() below here (3 points)




//Other methods below here. DO NOT provide equals or hashCode (5 points)
```

```
}
```

3. (23 points) Using the **Card** class from Question 2. Consider the following user story:

A **CardSuit** has-a:

- suit name
- set of 4 **Card**s (numbered 1-4)

In terms of behavior, the **CardSuit** object should offer the following functionality:

- There are getters but no setters for suit name
- It should be possible to retrieve the current number of cards dealt (i.e. those that are not available)
- It should be possible to deal a card. Meaning, the next available **Card** in the set of 4 cards is returned. If no **Card**s are available, return the 4th card. Once a **Card** is dealt, it cannot be dealt again until the suit is reset and the card becomes available again
- It should be possible to reset the suit making all cards available for dealing.

As in Question 2, the above requirements do not print out anything. Your methods therefore should not print anything. Instead they should affect the object by changing its instance variable(s) or by returning appropriately typed values to provide the desired functionality.

Extra Credit: Earn 2 extension points if you can implement the set of Cards with an array. You may answer the question below without an array and later if you have time write your answer with the array on the back page of the exam. If you are *very* confident, feel free to answer below with the array, but it counts as your exam answer.

```
public class CardSuit{
// Instance variable(s) below here. For each instance variable
// choose a type that best fits its purpose and usage (5 points)
```

```
//Constructor below here (8 points)
```

```
//Other methods below here. DO NOT provide equals or hashCode or toString()
//(10 points)
```

}

4. (10 points) Think about how you might use the CardSuit object from Question 3 to create a CardDeck object representing a deck of cards with suits of hearts, diamonds, spades, clubs.

(a) (5 points) What are the has-a's?

_____

_____

_____

(b) (5 points) What input parameters would the constructor have?

---

5. (25 points) In this problem, we are interested in computing the number of digits in the representation of a positive integral number N using recursion. For example, if N is 1234567, N has 7 digits. If N is 1000, the answer is 4 digits.

(a) (5 points) Identify the recursive substructure by circling the next smaller problem for the example number

1 2 3 4 5 6 7

(b) (5 points) Now, generalize the smaller problem in relation to the number N. If N is the original number you want to count the digits of, write the smaller problem in terms of N.

(c) (5 points) Assume you have a recursive method **`numDigits`**`(int x)` that counts the number of digits in **`x`**. Describe the base case?

(d) (10 points) Write the recursive method **`numDigits`** which takes in an **`int x`** and returns the number of digits in **`x`**.

```
public static _______ numDigits(int x)  //(1 point) {
//put the base case below here: (4 points)
```

```
//put the recursive call below here: (5 points)

}
```

6. (10 points) Considering the following recursive formula:

$$f(n) = \begin{cases} f(n-2) + 2 * f(n-1), \text{ if } n \geq 2 \\ n, \text{ otherwise} \end{cases}$$

Below, using substitution, show the evaluation of $f(4)$, making sure you transcribe lines faithfully down the page, so that each line you write is truly equal to $f(4)$ (you may or may not need all the lines provided):

$f(4)$ =

=

=

=

=

=

=

=

=

=