CSE 331 18su Midterm Exam

Name _____

Welcome to the 331 Midterm!

Please wait to turn the page until everybody is told to begin.

Friendly reminders:

1. **Attempt every problem.** You can receive partial credit.
2. **Write clearly!** We can't give you credit for answers that we can't read.
3. **If you make a mistake** and need to correct it, be sure that your final answer is very clearly indicated.
4. **If you run out of room** for any problem, you may continue on the last page. At the original location, be sure to indicate that your answer continues on the back.

The exam has 6 parts with the following point values:
Part 1 (26 points)
Part 2 (14 points)
Part 3 ~~(20 points)~~ 21 points
Part 4 (12 points)
Part 5 (16 points)
Part 6 (4 points)

This is an opportunity to show off what you have learned. Good luck and have fun!

Part 1: Hoare Logic (26 points)

Circle the correct answer in each row. (2 points each)

1.1	P: x > 5 Q: x >= 3	**P is stronger than Q**	Q is stronger than P	P and Q are incomparable
1.2	P: x % 10 = 0 Q: x % 5 = 0	**P is stronger than Q**	Q is stronger than P	P and Q are incomparable
1.3	P: | x - 1 | >= 2 Q: x >= 2	P is stronger than Q	Q is stronger than P	**P and Q are incomparable**
1.4	P: x > x Q: x = x	**P is stronger than Q**	Q is stronger than P	P and Q are incomparable

1.4: The most frequently missed problem on the exam. P is stronger than Q because x > x => x = x. If you are skeptical about whether a false statement could imply anything, look up the truth table for logical implication on any credible internet source.

1.5 (18 points) Let a be an array of floats. Write code to compute the average value in a and use Hoare Logic to prove that your code is correct. Your code must use a non-trivial loop.

One possible solution:

```
public static float average(float a[]) {
    { a != null & a.length > 0 }
    n = a.length;
    sum = 0;
    k = 0;
    { inv: sum = sum(a[0] ... a[k-1]) }
    while ( k != n ) {
        { inv & b: sum = sum(a[0] ... a[k-1]) & k != n}
        sum = sum + a[k];
        { sum = sum(a[0] ... a[k]) & k != n}
        k = k + 1;
        { inv: sum = sum(a[0] ... a[k-1]) }
    }
    { inv & !b: sum = sum(a[0] ... a[k-1])  & k = n }
    { sum = sum(a[0] ... a[n-1]) }
    avg = sum / n;
    { avg = ( sum(a[0], ... a[n-1] ) / n }
    return avg;
}
```

End of Part 1. I believe in you!!

Part 2: Specifications (14 points)

2.1 (4 points) Ada works for a small company, and Chandra is their customer. In January, Ada and Chandra had a meeting to discuss an application that the company would implement for Chandra. After the meeting, Ada wrote down a specificaion for the app. Over the next two months, Ada and teammates worked hard to implement the app. In March, Ada met with Chandra again to demonstrate the new software. Chandra was not pleased with the app. Here is a snippet of dialogue from the meeting:

Chandra: This software is wrong!
Ada: How is that possible? It's implemented exactly according to spec!

Assuming that the software does indeed match the spec, describe at least one idea for what the problem might be. Justify your answer(s).

Answer: The spec that matches the software does not match the customer's needs.

Possible Justification:
- The customer's needs changed over time.
- The spec that Ada wrote down did not match what Chandra said in the meeting.
- What Chandra said in the meeting was not a clear description of their actual needs.

Let **`IntStack`** be a class that represents a stack of inetgers. Consider the following specification for a method of the **`IntStack`** class:

```
/**
 * Look at the Integer that is n places from the top of the stack
 * without removing it from the stack. peek(0) looks at the top
 * Integer in the stack.
 * @param n - number of places from the top of the stack to peek
 * ...
 */
public Integer peek(int n)
```

The "..." in the spec above will be replaced with S1, S2, or S3.

(turn the page)

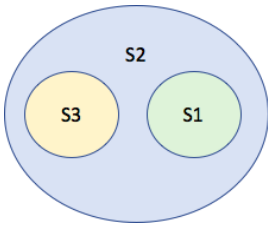
S1
```
@return the Integer n places from the top of the stack,
or null if n >= stack.size()
```

S2
```
@requires n < stack.size()
@return the Integer n places from the top of the stack
```

S3
```
@return the Integer n places from the top of the stack
@throws IllegalArgumentException if n >= stack.size()
```

Circle the correct answer in each row (2 points each):

2.2
S1 stronger than S2 S2 stronger than S1 S1 and S2 incomparable



2.3
S1 stronger than S3 S3 stronger than S1 **S1 and S3 incomparable**

2.4
S2 stronger than S3 **S3 stronger than S2** S2 and S3 incomparable

2.5 (4 points) Here is another potential spec for **`peek`**. Is it valid? If not, explain why.

S4
```
@requires stack.size() >= n-1
@return the Integer n places from the top of the stack
@throws IllegalArgumentException if stack.size < n-1
```

S4 is invalid because the requires tag contradicts the throws tag.

Note to graders: This spec also contains an unintended off-by-one error. We'll also give full credit if they said it's invalid because of the off-by-one error. This does not technically make the spec invalid, but it does make the question misleading. Point out that the real reason is that there's a contradiction in the spec.

Part 3: Abstract Data Types (ADTs) (21 points)

3.1 (2 points) Complete the analogy.

Method Specification is to **Method Implementation** as **Abstract Data Type** is to **Data Structure (partial credit for "class implementation" or "concrete implementation")**

3.2 ~~(6 points)~~ (7 points) Which of the following are parts of an Abstract Data Type? For each thing, write "yes" if the thing is part of an ADT or "no" if it is not part of an ADT.

Note: The nos on this question are unambiguous. The yesses are a bit ambiguous. The answer I had in mind is that public method javadocs are part of the ADT since they're the specification of an operation and not tied to any one implementation. However, it's possible to argue that javadocs are attached to a java class, and the ADT is something more abstract than that. Similar argment for each answer marked yes below. In the end, we made this question out of 7 points, scoring only the nos listed below, and accepting any answer for the others.

Class-level Javadocs	**yes**
Public method Javadocs	**yes**
Private method Javadocs	**no**
Abstraction Function	**no**
Representation Invariant	**no**
Method bodies	**no**
Public method signatures	**yes**
Private method signatures	**no**
Private fields	**no**
Internal comments	**no**
Class declaration	**yes**

3.3 (4 points) What is a representation invariant? Why is a representation invariant needed?

A rep invariant is a set of assertions about the concrete value of a data structure. Rep invariant differentiates between well formed and poorly formed concrete values.

alternate answer: Rep invariant tells whether a concrete value is valid (it represents an abstract value of the ADT) or invalid (does not represent an abstract value of the ADT).

Consider the class **`AxisAlignedRectangle`**:

```
/**
* A quadrilateral with four right angles whose edges are parallel
* to the x and y axes.
*/
public class AxisAlignedRectangle {
  private double a, b, c, d;
  // Abstraction Function:
  // For example, if a=1, b=2, c=3, and d=4, it represents the
  // rectangle with vertices
  //   (1,2), (1,4), (3,2), and (3,4)
  ...
}
```

3.4 (4 points) Describe what is wrong with the abstraction function for AxisAlignedRectangle.

The abstraction function itself is missing, there's only an example.

3.5 (4 points) Below, write a valid abstraction function for AxisAlignedRectangle.

```
// Abstraction Function:
// AF(this) = a rectangle with vertices at
//  (this.a, this.b), (this.a, this.d),
//  (this.c, this.d), and (this.c, this.b)
```

Finished with Part 3. You're more than halfway through!

Part 4: Design (12 points)

4.1 (4 points) Consider the **`AxisAlignedRectangle`** class from Part 3. Identify four bad variable names from that class, and suggest better names for each one.

one possible answer: a, b, c, d should be x1, x2, y1, y2

Consider the following code:

```
1   public String stringify(String a, String b, boolean flag) {
2     StringBuilder ret = new StringBuilder();
3     // strip characters in b from a
4     if (flag) {
5       for (int i = 0; i < a.length(); i++) {
6         if (!b.contains(a.get(i))) {
7           ret.append(a.get(i));
8         }
9       }
10    }
11    // concatenate b to the end of a
12    else {
13      ret.append(a);
14      ret.append(b);
15    }
16    return ret.toString();
17  }
```

4.2 (8 points) Describe at least two design issues from **`stringify`** and write a specific and detailed description of how you would fix them. (You do not need to actually rewrite the code.)

Some possible answers:
- The original method does two things. New code separates into two methods.
- Original method has vague method name. New methods are called `strip` and `concatenate`.
- Original method has vague names `a` and `b`. In `strip`, the parameters are `src` and `charsToStrip`. In `concatenate`, the parameters are `head` and `tail`.
- In the original method, the second part replicates functionality of a library method (`concat`) and a string operator (`+`). Use the library method or string operator instead of re-inventing this functionality. (Also give credit if they say the same thing for strip, though stringify is actually doing something slightly different than existing library methods.)
- The first half of `stringify` uses a for loop. In the new code, use a for-each loop.
- Parameters a and b should be checked to see if they're null.

End of Part 4. Great progress so far!

Part 5: Testing (16 points)

5.1 (4 points) What is wrong with this statement: "All the tests passed, therefore my program is correct!"

Tests cannot show the absence of bugs.
(Small partial credit for saying the tests might be wrong or incomplete.)

Consider the following method:

```
/** Find the maximum value in the array.
 * @param arr - an array of ints
 * @requires arr.length > 0
 * @return the minimum i such that arr[i] >= arr[j] for all j
 *    that are valid indices of the array
 */
public static int indexOfMax(int[] arr) {
  int max = Integer.MIN_VALUE;
  int idxOfMax = -1;
  for (int i = 0; i < arr.length-1; i++) {
    if (arr[i] > max) {
      max = arr[i];
      idxOfMax = i;
    }
  }
  return idxOfMax;
}
```

5.2 (12 points) Write at least three test cases for the **`indexOfMax`** method. In your test cases, you should identify inputs and expected outputs, but do not write actual test code. All of your tests must be for boundary cases. For each test, describe why it is a boundary case.

Grading criteria: 4 points per test case
+ 1 point for identifying concrete inputs and outputs.
+ 1 point for having the inputs and outputs match. (Correct outputs are those that the program would produce if it matches its spec.)
+ 1 point if the case is a boundary case
+ 1 point for a correct explanation of why it is a boundary case
(If student writes >3 test cases, grade the best 3)

Some boundary cases:

boundary: Single element array
input: {0}
output: 0

boundary: Max is first element
input: {1, 0, 0, 0}
output: 0

boundary: Max is last element (catches off-by-one bug)
input: {0, 0, 0, 1}
output: 3

boundary: Max is the greatest negative integer
input: {-1, -2, -2, -2}
output: 0

boundary: Max is the smallest positive integer
input: {-1, 0, 1, 0}
output: 0

boundary: Max is MAX_INT
input: {0, MAX_INT, 0}
output: 1

boundary: Max is MIN_INT (catches MIN_INT bug)
input: {MIN_INT}
output: 0

boundary: Multiple elements equal to max
input: {0, 1, 1, 1}
output: 1

Part 5 is finished. Look how far you've come!

Part 6: Equals and Hashcode (4 points)

Consider the following code:

```
public class FluffyKitten extends Cat implements Fluffy {
  ...
  public boolean equals(FluffyKitten kitten) { ... }
}
```

6.1 (2 points) What is the problem with the signature for the equals method?

This signature overloads Object's equals method. Instead it should override.

6.2 (2 points) Rewrite the signature for the equals method.

```
public boolean equals(Object o) { ... }
```

Congrats! You made it to the end! I hope you enjoyed this exam :)

Room for more answers

Be sure to clearly indicate which question(s) you are continuing here.