CSE 331 Midterm 26su *** SOLUTIONS ***

Friday, July 24, 2026 10:50am-11:50am

Name: _____ UW email: _____ @uw.edu

This exam contains 6 pages and 4 problems.

Instructions:

* Closed book, closed notes
* Place all electronics (phones, smart watches, smart glasses) out of sight and out of reach
* You have 60 minutes to complete the problems.
* Answer all problems in the provided space.
* Raise your hand to request scratch paper. Scratch paper will *not* be graded.
  * Please ask for help if you run out of space for your answer.

This page intentionally left blank.

Problem 1

a. What is the right precondition for a user interface to assume? (Answer in 10 words or less.)

no precondition at all, also known as "true"

b. If a method is called when its precondition is false, which of the following might happen?
(Select all that apply.)
[X] Throw an exception.
[X] Exit the program.
[X] Print an error message.
[X] Return normally.

Suppose some client C calls some method implementation I, and the client and implementor have agreed on specification S1. Now suppose they decide to upgrade to a new specification S2. Answer the following questions.

c. Suppose S2 is stronger than S1. Which of the following are true?
(Choose 1 from each group of 3.)
(X) Client C is guaranteed to work with S2.
( ) Client C may or may not work with S2.
( ) Client C is guaranteed *not* to work with S2.

( ) Implementation I is guaranteed to satisfy S2.
(X) Implementation I may or may not satisfy S2.
( ) Implementation I is guaranteed *not* to satisfy S2.

d. Now suppose S2 is weaker than S1. Which of the following are true?
(Choose 1 from each group of 3.)
( ) Client C is guaranteed to work with S2.
(X) Client C may or may not work with S2.
( ) Client C is guaranteed *not* to work with S2.

(X) Implementation I is guaranteed to satisfy S2.
( ) Implementation I may or may not satisfy S2.
( ) Implementation I is guaranteed *not* to satisfy S2.

e. Now suppose S2 is incomparable with S1. Which of the following are true?
(Choose 1 from each group of 3.)
( ) Client C is guaranteed to work with S2.
(X) Client C may or may not work with S2.
( ) Client C is guaranteed *not* to work with S2.

( ) Implementation I is guaranteed to satisfy S2.
(X) Implementation I may or may not satisfy S2.
( ) Implementation I is guaranteed *not* to satisfy S2.

f. When a test fails, which of the following are possible? (Select all that apply.)
[X] There is a bug in the code.
[X] There is a bug in the test.
[X] There is a bug in the spec.
[X] The code satisfies the spec. (a failing test need not mean buggy code)

g. Code that passes all tests in a test suite with 100% branch coverage is guaranteed to be correct.
( ) True
(X) False

h. A method that throws an exception must have a specification that mentions that exception.
( ) True
(X) False

i. We discussed many error signaling techniques (return a boolean, throw an exception, etc.). Which technique can a method implementor use so that the caller's code will not compile if they forget to check for the error? (Answer in 10 words or less.)

a checked exception

j. The Java type checker is precise, meaning all type errors reported at compile time would actually happen if the program were allowed to run.
( ) True
(X) False

Problem 2

Read the code for the following method. Then fill in the assertions as explained below.

Recall that a[x..y] means "the elements of a at indices x, x+1, ..., y-1" (inclusive x, exclusive y)

We use "sum(a)" to mean "the sum of the elements of a". And similarly "sum(a[0..i])" to mean
"the sum of the elements of a at indices 0, 1, ..., i-1".

```
/** @requires a != null
  * @returns sum(a) > 0 */
public static boolean hasPositiveSum(int[] a) {
    int s = 0;



    {{ P1: a != null and s = 0                                                    }}
    int i = 0;



    {{ P2: a != null and s = 0 and i = 0                                          }}

    // Inv: s = sum(a[0..i])
    while (i != a.length) {



        {{ P3: s = sum(a[0..i]) and i != a.length                                 }}
        s = s + a[i];



        {{ P4: s - a[i] = sum(a[0..i]) and i != a.length                          }}
        i = i + 1;



        {{ P5: s - a[i-1] = sum(a[0..i-1]) and i-1 != a.length                    }}



        {{ Q: s = sum(a[0..i])                                                    }}
    }



    {{ P6: s = sum(a[0..i]) and i = a.length                                      }}
    return s > 0;
}
```

a. Fill in P1 and P2 using forward reasoning.
b. Explain in 10 words or less why P2 implies Inv.

    i = 0, so sum(a[0..0]) is the empty sum = 0 = s

c. Fill in P3 based on the Floyd logic loop rule. (What do you know at the top of the loop body?)
d. Fill in Q based on the Floyd logic loop rule. (What do you need at the bottom of the loop body?)
e. Fill in P4 and P5 using forward reasoning.
f. Explain in 10 words or less why P5 implies Q.

    rearranging P5 we get s = sum(a[0..i-1]) + a[i-1] = sum(a[0..i])

g. Fill in P6 based on the Floyd logic loop rule. (What do you know when the loop exits?)
h. Explain in 10 words or less why P6 implies the return value (s > 0) satisfies the postcondition.

    s = sum(a), so returning s>0 returns sum(a)>0

Problem 3

Write tests for hasPositiveSum() according to the 331 testing conventions.

For each of your tests, explain what cases it covers.

We have given you one example test.

```
@Test
public void testHasPositiveSum() {

    assertTrue(hasPositiveSum(new int[]{1, -2, 3}));  // justification: 2+ iterations

    assertFalse(hasPositiveSum(new int[]{-1}));       // 1 iteration, and a case that returns false

    assertFalse(hasPositiveSum(new int[]{}));         // 0 iterations
```

}

Problem 4

```
/** A mutable set of integers that is guaranteed to have a positive (> 0) sum.
 * For example, {1, -2, 3} has a sum of 1 - 2 + 3 = 2, so it is a valid PositiveSumSet. */
class PositiveSumSet {
    private Set<Integer> set;
    private int sum;

    // RI: this.sum = sum(this.set) and sum > 0

    // AF: AF(this) = this.set
    /**
     * Constructs a PositiveSumSet with the given elements.
     * @requires the sum of initial elements to be strictly greater than zero
     * @modifies this
     * @effects this = initialElements
     */
    public PositiveSumSet(Set<Integer> initialElements) {
        this.set = new HashSet<>(initialElements);   // copy in (avoid rep exposure)
        this.sum = 0;
        for (int x : this.set) { this.sum += x; }



    }
    /** @return the sum of this */
    public int getSum() {
        return this.sum;
    }
    /** @return this */
    public Set<Integer> getSet() {
        return new HashSet<>(this.set);   // copy out (avoid rep exposure)


    }  // write your spec and impl for add() below (see part e.)
    /**
     * Adds x to this set.  @param x the value to add
     * @modifies this
     * @effects adds x to this; no change if x is already present
     * @throws IllegalArgumentException if x is new and would make the sum <= 0
     */
    public void add(int x) {
        if (this.set.contains(x)) return;    // duplicate: set and sum unchanged
        if (this.sum + x <= 0)
            throw new IllegalArgumentException("adding x would make sum <= 0");
        this.set.add(x);
        this.sum += x;
    }
}
```

a. Fill in the RI and AF.
b. Write code to implement the constructor according to its specification.
c. Prove getSum() correct as follows:
    i. What concrete precondition can getSum() assume when it runs? (Even though it has no abstract precondition.)

        the RI

    ii. The abstract postcondition is that the method returns "the sum of this". Explain why the code's actual return value satisfies this postcondition. You will need to refer to the concrete precondition and explain the connection between the abstract and concrete "worlds".

        AF(this) = this.set, so "sum of this" = sum(this.set), and by the RI, this is equal to this.sum.

d. Write code to implement getSet() according to its specification.
e. Specify and implement a method to add() which adds an element to the set. Write your spec and code for add() in the blank space provided above.