

University of Washington  
CSE 331 Software Design & Implementation  
Spring 2012

**Midterm exam**

**Friday, May 4, 2012**

**Name:** \_\_\_\_\_

**CSE Net ID (username):** \_\_\_\_\_

**UW Net ID (username):** \_\_\_\_\_

\_\_\_\_\_

This exam is closed book, closed notes. You have **50 minutes** to complete it. It contains 16 questions and 12 pages (including this one), totaling 100 points. Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. **Write your initials on the top of ALL pages.**

**Please write neatly;** we cannot give credit for what we cannot read.

Good luck!

Page	Max	Score
2	6	
3	12	
4	12	
5	10	
6	4	
7	10	
8	20	
10	12	
11	14	
Total	100	

## 1 True/False

(2 points each) Circle the correct answer. T is true, F is false.

1. **T / F** If you use proper testing methodology (specification tests and implementation tests, black-box and clear-box tests, achieve coverage, etc.), and your component passes all those tests, then your code is correct.
2. **T / F** Adding more preconditions to a method sometimes strengthens and sometimes weakens the specification.
3. **T / F** An exception indicates an unrecoverable failure within your program.

## 2 Multiple choice

(3 points each) Mark the single best choice, by circling the appropriate letter.

4. The stronger a specification is, the more it is:
  - (a) Easier for the client to use, easier for the implementer to satisfy
  - (b) Harder for the client to use, harder for the implementer to satisfy
  - (c) Easier for the client to use, harder for the implementer to satisfy
  - (d) Harder for the client to use, easier for the implementer to satisfy
5. You can **weaken** a specification by:
  - (a) Weakening the preconditions or strengthening the postconditions
  - (b) Strengthening the preconditions or weakening the postconditions
  - (c) Weakening the preconditions or weakening the postconditions
  - (d) Strengthening the preconditions or strengthening the postconditions
6. Regression testing is the process of
  - (a) Adding new tests when you add a new feature
  - (b) Adding new tests when you find a bug
  - (c) Adding new tests until you find a bug
  - (d) Writing tests starting with larger components and working your way down (“regressing”) to smaller components
  - (e) Running all tests after refactoring part of the program
  - (f) Running all tests at a regular interval
7. The method `get(int index)` in `java.util.List` throws an `IndexOutOfBoundsException` when `index` is less than 0. Suppose we impose a precondition that requires that `index` is non-negative.
  - (a) The new specification is stronger.
  - (b) The new specification is weaker.
  - (c) The new specification is equally strong.
  - (d) The new specification is incomparable to the original one (neither weaker nor stronger).

Mark all of the following that can be true, by circling the appropriate letters.

For questions 8–9, assume the following declarations:

```
Object o;  
Number n;  
Integer i;  
PositiveInteger p; // suppose PositiveInteger and NegativeInteger are subclasses of Integer
```

8. (6 points) Given the following additional declaration, mark all of the expressions that are legal (that type-check).

```
Collection<? extends Integer> cei;
```

- (a) `cei.add(o)`
- (b) `cei.add(n)`
- (c) `cei.add(i)`
- (d) `cei.add(p)`
- (e) `cei.add(null)`
- (f) `o = cei.get(0)`
- (g) `n = cei.get(0)`
- (h) `i = cei.get(0)`
- (i) `p = cei.get(0)`

9. (6 points) Given the following additional declaration, mark all of the expressions that are legal (that type-check).

```
Collection<? super Integer> csi;
```

- (a) `csi.add(o)`
- (b) `csi.add(n)`
- (c) `csi.add(i)`
- (d) `csi.add(p)`
- (e) `csi.add(null)`
- (f) `csi.add("Miami")`
- (g) `o = csi.get(0)`
- (h) `n = csi.get(0)`
- (i) `i = csi.get(0)`
- (j) `p = csi.get(0)`

### 3 Specifications

10. (10 points) Fill in the missing parts of the specification. If a part would be empty or “none”, say so explicitly.

(Hint: Do not worry about the apparent mismatch between uses of `int` and `Integer`. Because of auto-boxing, the code is type-correct.)

```
// A set of integers.
class IntegerSet {

    private List<Integer> set = new LinkedList<Integer>();

    // Inserts the argument in this IntegerSet.
    public void add(int x) { ... }

    /**
     * Return true iff the given value x is in this IntegerSet.
     *
     * @requires _____
     *
     * @modifies _____
     *
     * @effects _____
     *
     * @return _____
     *
     * @throws _____
     */
    public boolean contains(int x) {
        int index = set.indexOf(x);
        if (index != -1) {
            set.remove(index);
            set.add(0, x);
        }
        return index != -1;
    }
}
```

## 4 Reasoning about code

11. (4 points) Write the strongest postcondition for this code. Show your work at each statement by filling in all the blanks. Let  $x$  and  $y$  be of type `double`.

Precondition:  $x < 0$

$x = -2 * x$

\_\_\_\_\_

$y = x - 4$

\_\_\_\_\_

$y = 4 * y$

\_\_\_\_\_

12. (10 points) Write the weakest precondition for this code. Show your work at each statement by filling in all the blanks. Let  $x$ ,  $y$ , and  $z$  be of type `double`.

\_\_\_\_\_

```
y = x;
```

\_\_\_\_\_

```
y = 2 * y;
```

\_\_\_\_\_

```
if (y == 0) {
```

\_\_\_\_\_

```
    z = 4;
```

\_\_\_\_\_

```
} else {
```

\_\_\_\_\_

```
    z = y*2;
```

\_\_\_\_\_

```
}
```

postcondition:  $z \geq 4 \vee z \leq -4$

13. (20 points) Consider the following specification, along with Java code that implements it. You will prove total correctness of this code. You are not permitted to change the code.

```
// Computes x*n.  
// requires: n <= 0  
// returns: x*n  
int mult(int x, int n) {  
    int result = 0;  
    while (n != 0) {  
        result = result - x;  
        n = n + 1;  
    }  
    return result;  
}
```

- (a) Prove partial correctness (correct output if the routine terminates).

- (b) Prove termination.

Questions 14–16 use this code. There is another copy of the code on page 12 that you may tear off.

```
// Digit represents a single decimal digit, from 0 to 9.
//
// @specfield value: The value of the digit (0 through 9 inclusive).
public class Digit {

    private int v;

    private static Digit[] instances = new Digit[10];

    // Constructs a Digit representing the given number, such that digit.value = i.
    // @param i the value of the returned digit
    // @requires 0 <= i <= 9
    public Digit(int i) {
        if (i < 0 || i > 9)
            throw new IllegalArgumentException();
        this.v = i;
    }

    // Returns a Digit representing the given number.
    // @requires 0 <= i <= 9
    // @param i the value of the returned digit
    // @returns a digit such that digit.value = i
    public static Digit factory(int i) {
        if (instances[i] == null) {
            instances[i] = new Digit(i);
        }
        return instances[i];
    }

    public int getValue() {
        return this.v;
    }

    // Counts the number of unique digits in the given number.
    // For example, the number 2012 has three unique digits: 0, 1, and 2.
    // @requires number is not null, and contains no null elements
    // @param number a number, represented as a list of Digits
    // @returns the number of unique digits in the given number;
    // the result is >= 0 and <= 10.
    public static int numDigits(List<Digit> number) {
        Set<Digit> s = new HashSet<Digit>;
        for (Digit d : number) {
            s.add(d);
        }
        return s.size();
    }
}
```

14. (6 points) A client calls the `numDigits()` method (from the code that appears on both pages 9 and 12). The client is surprised when the method incorrectly returns 11. In 1–2 sentences, explain how this failure is possible.

---

---

---

---

---

15. (6 points) Write the smallest JUnit test method that you can, which exposes this problem in the implementation.

16. (14 points) Give two ways the Digit class could be modified to prevent the failure, *without modifying the specification or implementation of the numDigits method*. Give two ways that are as different as possible. Answer in 1–2 sentences each. Be specific. You may use a *small* amount of code (a few lines) to augment your answer if necessary, but you can get full credit without doing so.

(a)

(b)

This is a duplicate of the code that appears on page 9 and is used in questions 14–16. You may tear it off if you find that convenient. You do not need to hand it in.

```
// Digit represents a single decimal digit, from 0 to 9.
//
// @specfield value: The value of the digit (0 through 9 inclusive).
public class Digit {

    private int v;

    private static Digit[] instances = new Digit[10];

    // Constructs a Digit representing the given number, such that digit.value = i.
    // @param i the value of the returned digit
    // @requires 0 <= i <= 9
    public Digit(int i) {
        if (i < 0 || i > 9)
            throw new IllegalArgumentException();
        this.v = i;
    }

    // Returns a Digit representing the given number.
    // @requires 0 <= i <= 9
    // @param i the value of the returned digit
    // @returns a digit such that digit.value = i
    public static Digit factory(int i) {
        if (instances[i] == null) {
            instances[i] = new Digit(i);
        }
        return instances[i];
    }

    public int getValue() {
        return this.v;
    }

    // Counts the number of unique digits in the given number.
    // For example, the number 2012 has three unique digits: 0, 1, and 2.
    // @requires number is not null, and contains no null elements
    // @param number a number, represented as a list of Digits
    // @returns the number of unique digits in the given number;
    // the result is >= 0 and <= 10.
    public static int numDigits(List<Digit> number) {
        Set<Digit> s = new HashSet<Digit>;
        for (Digit d : number) {
            s.add(d);
        }
        return s.size();
    }
}
```