CSE 331 Spring 2018 Midterm

Name ____

There are 8 questions worth a total of 93 points. **Please budget your time so that you get as many points as possible.** We have done our best to make a test that folks can complete in 50 minutes, but everyone works at a different pace, and that is just fine!

The exam is closed book, closed electronics, closed classmates, open mind. Many of the questions have short answers, even if the prompt is a little long. Don't worry!

For all questions involving proofs, assertions, invariants, etc., please assume that all integer quantities are unbounded (e.g., overflow cannot happen) and that ***integer division and square root (sqrt) are truncating as in Java***, i.e., 5/3 evaluates to 1 and sqrt(17) evaluates to 4.

If you do not remember the syntax of some command or the format of a command's output, make the best attempt you can. We will not be grading syntactic details.

Relax and have fun! We're all here to learn.

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

1.	__________ / 12	5.	__________ / 12
2.	__________ / 12	6.	__________ / 10
3.	__________ / 10	7.	__________ / 7
4.	__________ / 20	8.	__________ / 10

Remember: For all of the questions involving proofs, assertions, invariants, and so forth, you should assume that all numeric quantities are unbounded integers (i.e., overflow can not happen) and that integer division is truncating division as in Java, i.e., 5/3 => 1.

QUESTION 1: Forward Reasoning (12 points)

Using forward reasoning, write an assertion in each blank space indicating what is known about the program state at that point, given the precondition and the previously executed statements. Your final answers should be simplified. Be as specific as possible, but be sure to retain all relevant information**.**

```
(a) (5 points)

{ x < -1 }

y = x * x;

{ _______________________________________________ }

z = x * y;

{ _______________________________________________ }

w = z < x;

{ _______________________________________________ }
```

[Question 1 continued]

(b) (7 points)

```
{ |x| < 5 }

if (x % 2 == 0) // if x is even...

     { _____________________________________ }

     y = x * x;

     { _____________________________________ }

else

     { _____________________________________ }

     y = x + x;

     { _____________________________________ }


{ ______________________________________________ }
```

QUESTION 2: Backward Reasoning (12 points)

Using backwards reasoning, find the weakest precondition for each sequence of statements and postcondition below. Insert appropriate assertions in each blank line. You should simplify your final answers if possible.

```
(a) (5 points)


{ _____________________________________ }

b = a * a - 5;

{ _____________________________________ }

c = b * 10 - a;

{ c < 0 && b >= 0 }
```

[Question 2 continued]

```
(b) (7 points)

{ _______________________________________________ }

if (x < 0)

     { _____________________________________ }

     x = 2 * y;

     { _____________________________________ }

else

     { _____________________________________ }

     y = x / 2;

     { _____________________________________ }


{ x + y = 10 }
```

QUESTION 3: Loop Invariants and Proofs (10 points)

In this question, we want to verify that the difference between any two elements of an array is less than the result returned by the `range` method below. Fill in invariants to complete the proof.

```
public static int range(int[] a) {

      { ____________________________________________________________
}

      int min = a[0];  int max = a[0];  int i = 1;



      { ____________________________________________________________
}

      while(i < a.length) {

            { ______________________________________________________
      }

            if(a[i] < min) min = a[i];

            { ______________________________________________________
      }

            if(a[i] > max) max = a[i];

            { ______________________________________________________
      }

            i++;

            { ______________________________________________________
      }
      }



      { ____________________________________________________________
}
```

```
    // invariant immediately above should imply
    // { forall p, q. a[p] - a[q] <= max - min }

    return max - min;
}
```

(If you need more space for an invariant, you can use the final blank page of the exam.)

QUESTION 4: Specification and Design (20 points)

(a) `Interval` objects are immutable: (circle one) True False

(b) Provide an example call to the `Interval` constructor that produces an empty interval (i.e., an `Interval` value `v` such that `v.contains(x)` always returns `false`):

____

(c) Give a suitable Representation Invariant (RI) for `Interval` (hint: it may be very simple):

(d) The RI for `Interval` needs to be checked in every method: (circle one) True False

(e) Give a suitable Abstraction Function (AF) for `Interval`:

[Question 4 continued]

(f) Complete the JavaDoc comments below to provide the most suitable specification for `union`. Leave any unneeded parts blank. There may be multiple ways to get full points.

```
/** Return union this with another interval.
  *
  * @param
  *
  * @requires
  *
  * @modifies
  *
  * @effects
  *
  * @throws
  *
  * @returns
  *
  */
public Interval union(Interval other) { /* (see earlier code) */ }
```

(g) Complete the JavaDoc comments below to provide the most suitable specification for `clamp`. Leave any unneeded parts blank. There may be multiple ways to get full points.

```
/** Return integer in this interval closest to x.
  *
  * @param
  *
  * @requires
  *
  * @modifies
  *
  * @effects
  *
  * @throws
  *
  * @returns
  *
  */
public int clamp(int x) { /* (see earlier code) */ }
```

[Question 4 continued]

(h) Complete the JavaDoc comments and the implementation below to provide a `size` method for `Interval`. Leave any unneeded parts blank. Your answer should respect the RI and AF. Hint: it may be helpful to consider the `contains` method.

```
/** Return number of integers in this interval.
  *
  * @param
  *
  * @requires
  *
  * @modifies
  *
  * @effects
  *
  * @throws
  *
  * @returns
  *
  */
public int size() {











}
```

QUESTION 5: Testing (12 points)

For each part below, describe **two** separate, distinct “black box” tests for the `Interval` method in question. For each test give the input values and expected result(s). You do not need to write JUnit tests or other Java code. Reminder: there is a `contains()` observer method defined for this class that might be useful and you are also encouraged to use the `size()` method you defined earlier.

(a) Tests for the `union` method:

(b) Tests for the `clamp` method:

(c) Tests for the `size` method:

QUESTION 6: Equals and Hashcode (10 points)

(a) Implement `equals` for `Interval`. Two intervals should be considered equal if they contain the same set of integers.

(b) Implement `hashcode` for `Interval`. To receive full points, your implementation should be of high quality (i.e., avoid unnecessarily having unequal objects hash to the same value).

(c) What property would a "perfect hashcode" for Interval guarantee?

____

Is it possible to implement such a perfect hashcode method? (circle one) Yes No

QUESTION 7: Equals Equivalence Relation (7 points)

Classes overriding `equals` must implement an *equivalence relation*:
`a.equals(a) == true` (*reflexive*);
`a.equals(b) == b.equals(a)` (*symmetric*); and
`a.equals(b) && b.equals(c) == true` implies `a.equals(c) == true` (*transitive*).

Put a **check** next to the valid overriding implementations of `equals` for `ConstantInt` below.

```
public class ConstantInt {
  private int val;
  public ConstantInt(int v) {
    this.val = v;
  } }
```

	VALID?
`public boolean equals(Object x) {` `  return false; }`	
`public boolean equals(Object x) {` `  return true; }`	
`public boolean equals(ConstantInt x) {` `  return this == x; }`	
`public boolean equals(ConstantInt x) {` `  return this.val == x.val; }`	
`public boolean equals(Object x) {` `  return this.val.equals(x); }`	

`public boolean equals(Object x) {`
`  if(!(x instanceof ConstantInt))`
`    return false;`
`  ConstantInt ci = (ConstantInt)x;`
`  return this.val == ci.val; }`	

`public boolean equals(Object x) {`
`  if(this == x)`
`    return true;`
`  if(!(x instanceof ConstantInt))`
`    return false;`
`  ConstantInt ci = (ConstantInt)x;`
`  return this.val == ci.val; }`	

QUESTION 8: Comparing Specifications (10 points)

Here are four possible specifications for a method that checks whether one integer is a multiple of another.

(S1)
```
@param n
@param f
@returns true if there exists g such that n = f * g
```

(S2)
```
@param n
@param f
@requires f ≠ 0
@returns true if there exists g such that n = f * g, otherwise false
```

(S3)
```
@param n
@param f
@requires f > 0
@returns true if there exists g such that n = f * g, otherwise false
```

(S4)
```
@param n
@param f
@returns true if there exists g such that n = f * g and g > 0,
         otherwise false
```

(a) Circle the specifications as strong as S1: S1 S2 S3 S4

(b) Circle the specifications as strong as S2: S1 S2 S3 S4

(c) Circle the specifications as strong as S3: S1 S2 S3 S4

(d) Circle the specifications as strong as S4: S1 S2 S3 S4

(e) Is it possible for a single method to satisfy both S1 and S4? (circle one) YES NO

(f) Is it possible for a single method to satisfy both S2 and S3? (circle one) YES NO

[Additional space for answers if needed. Please indicate clearly which questions you are answering here, and also be sure to indicate on the original page that the rest of the answer can be found here.]