CSE 331 Midterm Exam 2/13/12

Name _____

There are 10 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc.

Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading.

Relax, you are here to learn.

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

Score _____ / 100

1. _____ / 14
2. _____ / 25
3. _____ / 12
4. _____ / 12
5. _____ / 12
6. _____ / 6
7. _____ / 6
8. _____ / 6
9. _____ / 6
10. _____ / 1

Question 1. (14 points) (assertions) Using backwards reasoning, find the weakest precondition for each sequence of statements and postconditions below. Insert appropriate assertions in each blank line. You should simplify your answers if possible.

(a)

```
{ _____ }
a = x + 1;

{ _____ }
b = y - x;
{ a*x + b > 0 }
```

(b)

```
{ _____ }
y = y + 1;

{ _____ }
x = y + 3;
{ x > 0 & y < 0 }
```

Question 2. (25 points) Loop development. The factorial function, as you probably remember, is defined as $n! = 1 * 2 * 3 * \ldots * n$, i.e., the product of the numbers 1 through *n*. For this problem, complete the following **non-recursive** method to compute and return *n*! and prove that it computes and returns the correct answer. The method heading is provided for you as is the declaration of variable `ans` and the `return` statement at the end. You should declare additional variables as you need them. Your answer should supply the method code and include assertions, preconditions, postconditions, and invariants as needed to prove it is correct.

```
// return the value n! = 1 * 2 * ... * n
// pre: n > 0
int fact(int n) {
   int ans;
```

{ `ans` == *n*! (where *n* is the original argument value) }

```
   return ans;
}
```

Question 3. (12 points) (Specifications) Consider the following specifications for a method that has one integer argument:

(a) Returns an integer ≥ the argument
(b) Returns a non-negative integer ≥ the argument
(c) Returns argument – 1
(d) Returns argument^2 (i.e., the square of the argument)
(e) Returns a non-negative number

Consider these implementations, where `arg` is the function argument value:

```
(i)     return arg + 5;
(ii)    return arg * arg;
(iii)   return arg % 10;
(iv)    return Math.abs(arg);
(v)     return Integer.MAX_VALUE;
```

Place a check mark in each box for which the implementation satisfies the specification. If the implementation does not satisfy the specification, leave the box blank.

Implementation	**Specification**				
	(a)	(b)	(c)	(d)	(e)
(i)					
(ii)					
(iii)					
(iv)					
(v)					

Question 4. (12 points) A bit of code reading. Consider the following ADT, which we wish to use to represent a line of people waiting to buy tickets at a movie theater.

```
public class WaitLine {

   // instance variable
   private List<String> people;

   // AF: people represents a list of persons in a line (say
   // at a movie theater).  They are enqueued at the end of
   // the line when they arrive and dequeued in that order.

   // RI: entries in people are not null and, for persons A
   // and B, A appears in people before B if enqueue(A) was
   // called before enqueue(B).

   // constructor
   public WaitLine() {
      people = new LinkedList<String>();
      checkRep();
   }

   // mutators

   public void enqueue(String p) {
      people.add(p);
      checkRep();
   }

   public String dequeue() {
      if (people.size() == 0) return null;
      String p = people.remove(0);
      checkRep();
      return p;
   }

   // observers

   public int size() { return people.size(); }

   public List<String> getEveryone() { return people; }

   // internal method
   private void checkRep() {
      for (String p: people) assert p!= null;
   }

}
```

Answer the questions about this class on the next page. You may remove this page for reference if you wish.

Question 4. (cont.) Answer the following questions about the `WaitLine` class on the previous page. **Be brief:** you shouldn't need more than a short sentence or two for each answer, but you do need to justify your conclusions.

(a) The `dequeue` method returns `null` if the queue is empty. Is this a reasonable way for the method to behave if the queue is empty? Does it create problems or ambiguities for clients using this class?

(b) Are there any potential representation exposure problems with this class? If so, what are they, and how could they be fixed, preferably without major changes to the class?

(c) Is the definition and use of `checkRep` reasonable, given the operations in the class?

(d) Our new summer intern wants to add a `sellTicket` method to this class to sell a ticket to the first person in line and remove them from the queue. Is this a good design decision? Why or why not?

Question 5. (12 points) Testing. In homework 4 one of the questions involved a queue that was implemented using a finite array as a "circular list". New items were added at the end of the array, but after filling the last slot in the array, we wrapped around and stored the next queue element at the beginning – which worked provided that the oldest element(s) had been previously removed to make room.

(a) Describe two good **black box** tests for this queue implementation. The two tests should be from different revealing subdomains – i.e., they should not detect exactly the same set of errors. You do not need to give JUnit code – just describe the tests.

(i)

(ii)

(b) Describe two good **white box** (or glass box) tests for this queue implementation. As with the black box tests, the two tests should be from different revealing subdomains. Again, no JUnit code required.

(i)

(ii)

A few short answer questions…

Question 6. (6 points) Given the representation in an ADT and the representation invariant (RI) that it satisfies, an abstraction function (AF) describes the meaning of this representation as an abstract value. Is the AF also a function in the other direction, i.e., do the abstract value and the AF determine a unique representation? If yes, give a brief justification for your answer; if no, give a counterexample explaining why not.

Question 7. (6 points) You are implementing your graph ADT using appropriate specifications, documentation, JUnit tests, and version control. You find a bug. What strategy should you use to deal with the problem and ensure it is unlikely to occur again? (i.e., what steps in the development process should come next after discovering the bug?)

Question 8. (6 points) Suppose we create an ADT to implement an immutable data type. The ADT contains only creator, observer, and producer methods – no mutators. True or false: no method in the ADT can modify the ADTs instance variables once an instance has been created. Give a brief justification for your answer.

Question 9. (6 points) Java contains both checked and unchecked exceptions. A method that might cause a checked exception either has to catch the exception or include a `throws` clause in its heading to indicate that it might generate that exception. Why aren't all exceptions treated this way? In particular, why are there unchecked exceptions that don't have to be handled or included in the method's heading? After all, they are as much a part of the method's possible behavior as the checked ones.

Question 10. (1 point – all honest answers receive the point) What is the one question (if any) that you really thought would be on this test that we forgot to include??