CSE 331 Midterm Exam Sample Solution 2/20/13

Question 1. (16 points) (assertions) 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 answers if possible.

(a)

{ **x > x + y - 3** } **=> { 0 > y – 3 } => { 3 > y }**

```
z = x + y;
```

{ **x > z - 3** }

```
y = z - 3;
```

```
{ x > y }
```

(b)

{ **a + b + a – b = 42** } **=> { 2*a = 42 } => { a = 21 }**

```
p = a + b;
```

{ **p + a – b = 42** }

```
q = a - b;
```

```
{p + q = 42}
```

Question 2. (12 points) Specification madness. Suppose we are implementing a BankAccount class that has the following heading and instance variable:

```
public class BankAccount {
  int balance;  // current account balance
```

Here are three possible specifications for a method `withdraw(amount)` to withdraw funds from the account. To save space, we have omitted the `@modifies this` clause, which would be part of all of the specifications.

A @effects decreases balance by amount

B @requires amount <= balance and amount >= 0
@effects decreases balance by amount

C @throws InsufficientFundsException if balance < amount
@effects decreases balance by amount

Now, here are four possible implementations of method `withdraw`:

```
I    void withdraw(int amount) {
         balance -= amount;
     }

II   void withdraw(int amount) {
          if (balance >= amount) balance -= amount;
     }

III  void withdraw(int amount) {
         if (amount < 0) throw new IllegalArgumentException();
         balance -= amount;
     }

IV   void withdraw(int amount) throws InsufficientFundsException {
         if (balance < amount) throw new InsufficientFundsException();
         balance -= amount;
     }
```

(continued on next page – you may remove this page for reference.)

Question 2. (cont.) In the following table, put an X in each square where the implementation whose number is given on the left properly implements the specification whose letter is given at the top. If a given implementation does not meet a specification, or if a specification is improperly formed, inconsistent, or otherwise defective, leave the square blank.

	Spec A	Spec B	Spec C
Impl I	X	X	
Impl II		X	
Impl III		X	
Impl IV		X	X

The next several questions concern the following class that represents immutable, integer-valued polynomials, similar to the one sketched in examples in lecture.

The basic class definition and rep are as follows:

```
/**
 * An IntPoly is an immutable, integer-valued polynomial
 * with integer coefficients.  A typical IntPoly value
 * is a_0 + a_1*x + a_2*x^2 + ... + a_n*x_n.  An IntPoly
 * with degree n has coefficent a_n != 0, except that the
 * zero polynomial is represented as a polynomial of
 * degree 0 and a_0 = 0 in that case.
 */
public class IntPoly {
  // rep
  int a[];    // Coefficients

  // AF(this) = a has n+1 entries, and for each entry,
  //            a[i] = coefficient a_i of the polynomial.
```

Two of the methods of the class are the following:

```
  /**
   * Return the coefficients of this IntPoly
   */
  public int[] getCoeffs() {
    return a;
  }

  /**
   * Return a new IntPoly that is the sum of this and other
   */
  public IntPoly add(IntPoly other) {
    // implementation omitted
  }
```

See the questions concerning this class on the following pages. You may remove this page for reference if you wish.

Question 3. (10 points) The above `add` method does not have a careful specification. Below, give a complete, appropriate specification for method `add`. If some part of the specification would be empty or "none", say so explicitly.

```
/**
 * Return a new IntPoly that is the sum of this and other
 *

 * @requires  other != null *

 * @modifies  none

 * @effects  none

 * @return  A new IntPoly that is the sum of this
            and other *

 * @throws  none

 */
public IntPoly add(IntPoly other) {
  ...
}
```

***The specification ought to say something about whether the parameter `other` can be `null` or not. One way to handle it, which makes as much sense as anything, is to require `other!=null`. Another possibility would be to add to the `@return` clause a description of what happens if `other` is `null` – maybe return a copy of `this`. A third possibility would be to specify that the method `@throws` an unchecked exception if `other` is `null` – but not a checked exception since no exception is included in the method heading.**

Question 4. (12 points) (rep exposure) The observer `getCoeffs` method shown above returns to the client an array with the coefficients of this `IntPoly`.

```
/**
 * Return the coefficients of this IntPoly
 */
public int[] getCoeffs() {
  return a;
}
```

(a) One of your colleagues is worried that this creates a potential representation exposure problem. Another colleague says there's no problem since an `IntPoly` is immutable. Is there a problem? Give a brief justification for your answer.

Yes there is a problem. The return value is a reference to the same coefficient array stored in the `IntPoly` and the client code could alter those coefficients.

(b) If there is a representation exposure problem, give a new or repaired implementation of `getCoeffs` that fixes the problem but still returns the coefficients of the `IntPoly` to the client. If it saves time you can give a *precise* description of the changes needed instead of writing the detailed Java code.

Create a new array the same length as `a`, copy the contents of `a` to it, and return the new array.

Question 5. (20 points) Loop development. We would like to add a method to this class that evaluates the `IntPoly` at a particular value x. In other words, given a value x, the method `valueAt(x)` should return $a_0 + a_1x + a_2x^2 + ... + a_nx^n$, where a_0 through a_n are the coefficients of this `IntPoly`.

For this problem, develop an implementation of this method and prove that your implementation is correct. For full credit, you must invent an appropriate loop invariant and show it is established by any initialization code, is maintained as the loop executes, and that after termination of the loop plus any additional code, the proper result (value) is returned to the caller. Your proof does not need to be completely formal, but needs to be sufficiently careful to convince the reader that the code and proof are correct (i.e., you can skip over tedious, obvious logic steps and simplifications – provided they really are obvious).

You may not call any other methods. This method should be self-contained.

Write
Your
Answer
On the
Next page …

(Use the rest of this page for scratch work, but please put your answer **on the next page**.)

Question 5. (cont.) Write your code and correctness proof below.

```
/** Return the value of this IntPoly at point x */
public int valueAt(int x) {

  int val = a[0];

  int xk = 1;

  int k = 0;

  int n = a.length-1;  // degree of this, n >= 0

  { inv: xk = x^k && val = a[0]+a[1]*x+...+a[k]*x^k }

  while (k != n) {

    { inv && k != n }

    xk = xk * x;

    { xk = x^(k+1) & val = a[0]+a[1]*x+...+a[k]*x^k }

    val = val + a[k+1]*xk

    { xk=x^(k+1) & val=a[0]+a[1]*x+...+a[k+1]*x^(k+1) }

    k = k + 1;

    { inv }

  }

  { inv && k=n => val = a[0]+a[1]*x+...+a[n]*x^n }

  return val;

}
```

During the exam we announced that it was ok to assume that `a` had at least one element, i.e., the polynomial degree was >= 0. We also, in a moment of weakness, said that it was ok to use `Math` functions, like `Math.pow`, although the solution is clearer and likely faster without that.

Question 6. (18 points) Suppose we are defining a class to represent items stocked by an online grocery store. Here is the start of the class definition, including the class name and instance variables (rep):

```
public class StockItem {
  String name;          // item name (e.g., "Fancy Feast")
  String size;          // size ("small", "12oz", etc.)
  String description;   // item description (e.g., "yummy food")
  int    quantity;      // number of copies of this item in stock

  /** Construct a new StockItem with the given data */
  public StockItem(String name, String size,
                   String decription, int quantity) { ... }
```

A summer intern was asked to implement an equals function for this class that treats two `StockItem` objects as equal if their `name` and `size` fields match. Here's the result:

```
  /** return true if the name and size fields match */
  public boolean equals(StockItem other) {
    return name.equals(other.name) && size.equals(other.size); }
```

(a) (4 points) This `equals` method seems to work some of the time but not always. Give an example showing a situation where it fails.

```
Object s1 = new StockItem("thing", 1, "stuff", 1);
Object s2 = new StockItem("thing", 1, "stuff", 1);
System.out.println(s1.equals(s2));  // prints "false"
```

Reason (not required in your answer): The equals method in `StockItem` overloads `equals(Object)` rather than overriding it since its parameter type is `StockItem`, not `Object`. When the compiler type-checks `s1.equals(...)` the only available equals method is `equals(Object)` defined in class `Object`, which returns true only if the two variables refer to the same object.

(continued next page)

Question 6 (cont.) (b) (4 points) Show how to fix the `equals` method given above so it works properly and has the intended meaning (e.g., `StockItem`s are equal if their names and sizes are equal)

(you won't need all this space probably….)

```
@Override
public boolean equals(Object o) {
  if (! (o instanceof StockItem))
    return false;
  StockItem other = (StockItem)o;
  return name.equals(other.name) &&
         size.equals(other.size);
}
```

Answers that used `getClass()` instead of `instanceof` were also fine.

(continued next page)

Question 6. (cont.) (8 points) (c) Here are four possible `hashCode` methods for class `StockItem`. For each of these `hashCode` methods, circle *legal* if the method is a correct implementation of `hashCode` for `StockItem`, as specified on the previous pages. Circle *wrong* if it is not a correct implementation.

(i) [✓ legal] wrong

```
public int hashCode() {
  return name.hashCode();
}
```

(ii) [✓ legal] wrong

```
public int hashCode() {
  return name.hashCode()*17+size.hashCode();
}
```

(iii) legal [✓ wrong]

```
public int hashCode() {
  return name.hashCode()*17+quantity;
}
```

(iv) legal [✓ wrong]

```
public int hashCode() {
  return quantity;
}
```

(d) (2 points) Of the four `hashCode` methods above, which one is the best and (briefly) why?

(ii) will likely do the best job since it takes into account both the size and name fields. (i) is also legal but it gives the same `hashCode` for `StockItems` that have different sizes as long as they have the same name, so it doesn't differentiate between different `StockItems` as well as (ii).

Question 7. (6 points) (specifications) Suppose we are specifying a method and we have a choice between either requiring a precondition (e.g., @requires: n > 0) or specifying that the method throws an exception under some circumstances (e.g., @throws IllegalArgumentException if n <= 0).

Assuming that neither version will be significantly more expensive to implement than the other and that we do not expect the precondition to be violated or the exception to be thrown in normal use, is there any reason to prefer one of these to the other, and, if so, which one? Give a brief (couple of sentences) justification along with your answer.

It would be better to specify the exception. That reduces the domain of inputs for which the behavior of the method is unspecified. It also will cause the method to fail fast for incorrect input, which should make the software more robust – or at least less likely to continue execution with erroneous data.

Question 8. (6 points) (specifications) Suppose we are trying to choose between two possible specifications for a method. One of the specifications S is stronger than the other specification W, but both include the behavior needed by clients. In practice, should we always pick the stronger specification S, always pick the weaker one W, or is it possible that either one might be the suitable choice? Give a brief justification of your answer, including a brief list of the main criteria to be used in making the decision.

Neither is necessarily better. What is important is picking a specification that is simple, promotes modularity and reuse, and can be implemented efficiently.

(Many answers focused narrowly on which would be easier to implement. While that is important – we don't want a specification that is impossible to build – it isn't the main thing that determines whether a system design is good or bad.)