CSE 331 19su Midterm Exam 7/22/19 Sample Solution

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 cannot happen and there are no fractional parts to numbers) and integer division is truncating division as in Java, i.e., 5/3 => 1.

Question 1. (5 points) (Forward reasoning) Starting with the given assertion, insert appropriate assertions in each blank line. You should simplify your final answers if possible.

```
{ x > 3 }
x = x + 1;
{  x > 4   }
y = x * 10;
{  x > 4  && y = x * 10   }
x = x - 2;
{  x > 2  && y = (x+2)  * 10   } =>
            { x > 2 && y = 10*x + 20 } =>
            { x > 2 && y > 40 }
```

The final simplification is useful if we want to know the possible final values for x and y, although it loses some information about the exact relationship between them. In fact, we can also assert y>50 based on what we know about the value of x when y was assigned 10*x.

Question 2. (8 points) (Backward reasoning). Find the weakest precondition for the sequence of statements below to establish the given postcondition. Write appropriate assertions in each line and simplify your final answer if possible.

```
{  (x<y && x!=y-1 && x>-1) || (x>=y && y-1!=x && x>0)  } =>
        { (-1 < x < y-1) || x>=y && x>0) }
if (x < y) {
  {  y != x+1 && x+1 > 0   } => { y != x+1 && x > -1 }
  x = x + 1;
  {  y != x && x > 0   }
} else {
  {  y-1 != x && x > 0   }
  y = y - 1;
  {  y != x && x > 0   }
}
{ x != y && x > 0 }
```

Question 3. (16 points) Loops. The following loop finds the two largest distinct values in an array `a` that has `n` elements. Your job is to prove that it works correctly and establishes the postcondition that is given at the end. According to the precondition, the array length `n` is at least 2, there are no duplicate values in the array. Write a suitable loop invariant and add the necessary assertions to complete the proof. To save writing, use the notation `a[i..j]` to refer to the array elements starting with `a[i]` and ending with `a[j]`. Hint: the Java expression `b?x:y` used below evaluates to `x` if `b` is true and to `y` if `b` is false. (It works correctly – you don't need to prove it.)

```
{ pre: n ≥ 2 && a has no duplicate elements }
// initialize max and max2nd (you can assume this is correct)
int max    = (a[0]<a[1]) ? a[1] : a[0];
int max2nd = (a[0]<a[1]) ? a[0] : a[1];
int k = 2;
{ inv: max is largest in a[0..k-1] and max2nd is 2nd largest in a[0..k-1] }
while(k != n) {
    { inv and k != n }
    if (a[k] > max) {
        { inv and a[k] > max = largest in a[0..k-1] (so max = 2nd largest in a[0..k]
                and a[k] is largest in a[0..k]) }
        max2nd = max;
        { max = max2nd = 2nd largest in a[0..k] and a[k] is largest in a[0..k] }
        max = a[k];
        { max = largest in a[0..k] and max2nd = 2nd largest in a[0..k] }

    } else if (a[k] > max2nd) {
        { inv and max2nd < a[k] < max = largest in a[0..k-1]
              (so a[k] = 2nd largest in a[0..k]) }
        max2nd = a[k];
        { max = largest in a[0..k] and max2nd = 2nd largest in a[0..k] }
    } else {
        // nothing needed
        { inv && a[k] < max2nd } =>
                { max = largest in a[0..k] and max2nd = 2nd largest in a[0..k] }
    } // end if-elseif-else
    { max = largest in a[0..k] and max2nd = 2nd largest in a[0..k] }
    k = k + 1;
    { inv }
} // end loop
{ inv and k=n } =>
{ post: max = largest in a[0..n-1] && max2nd = 2nd largest in a[0..n-1] }
```

Note: the original problem did not leave a blank line for an assertion between the end of the `if` statement and the assignment `k=k+1`. We added that above for completeness, but as long as appropriate assertions were written at the end of each part of the `if`, a solution received full credit.

The next several questions concern the following code, which contains a class that represents a shopping cart and a second class that represents items that can be added to the cart. There may be some logic bugs in the code (to be explored later), but it does compile and execute without any reported errors. There is a second copy of this page at the end of the exam that you can remove for convenience.

```
/** A ShoppingCart holds a list of n Items i1, i2, ..., in */
public class ShoppingCart {
  private List<Item> items;
  public ShoppingCart() { items = new ArrayList<Item>(); }
  public void addItem(Item item) {
    items.add(item);
  }
  public void applyDiscount(double scaleFactor) {
    for(Item item : items) {
      item.setPrice(scaleFactor * item.getPrice());
    }
  }

/** An item in a shopping cart */
public class Item {
  private String name;
  private double price;

  public Item(String name, double price) {
    this.name = name;
    this.price = price;
  }

  public String getName() { return this.name; }
  public double getPrice() { return price; }
  public void setPrice(double price) { this.price = price; }

  /** Items are considered equal if they have the same name.   */
  @Override
  public boolean equals(Object o) {
    if(!(o instanceof Item)) return false;
    Item other = (Item) o;
    return this.name.equals(other.name);
  }

  /** an attempt at a hashCode for an Item */
  @Override
  public int hashCode() {
    return (31 * Double.hashCode(price)) + name.hashCode();
  }
}
```

Do not remove this page from the exam, but feel free to tear off the copy of this page at the end of the exam. Continue with questions about this code on the next page.

Question 4. (10 points) As usual, whoever writes these exams doesn't provide proper specifications for things. Below, fill in a correct CSE 331-style specification for the constructor and the `applyDiscount` method in `ShoppingCart`. For CSE 331-specific custom tags, you can write `@spec.xyz` or just `@xyz` – whichever you prefer.

```
/** Construct a new, empty ShoppingCart
 *
 *
 * @effects make a new empty ShoppingCart
 *
 *
 *
 *
 *
 *
 *
 *
 *
 *
 */
public ShoppingCart() { ... }

/** Apply a discount to the Item prices in this ShoppingCart
 *
 * @param scaleFactor Amount to use to adjust each Itme price.
 *                    Normally would be less than 1.0.
 *
 * @modifies this
 *
 * @effects The price of each item in this is multiplied by
 *          scaleFactor
 *
 *
 *
 *
 *
 */
public void applyDiscount(double scaleFactor) { ... }
```

Note: This method could have been better named `adjustPrice` or something similar since `applyDiscount` seems to imply that the method should only reduce an `Item`'s price. Some solutions had a precondition requiring the `scaleFactor` to be positive or strictly between 0.0 and 1.0, and as long as that was reasonable it received full credit.

Question 5. (12 points, 3 each) Testing. Describe four distinct black-box tests that could be used to verify that the `applyDiscount` method specified in the previous problem works properly. Each test description should describe the test input and expected output. For full credit each test should be different in some significant way from the other tests (think about boundary conditions and subdomains, etc.). You **should not** provide JUnit or other code, just a clear, precise description of each test, and your descriptions should be a few lines each, at most. If you want to describe a specific `Item` as part of a test you can write it with the notation (name, price), i.e., (magicwand, 17.42).

Note: This question should probably have come after problem 7 and should have said to assume that a `getTotal` method was available to compute the total value of a `ShoppingCart`. Because there are no other obvious ways in the starter code to observe changes in a `ShoppingCart`, we gave credit to answers as long as they described a precise test setup and how that would change the internal state of the `ShoppingCart` and/or the `Items` in it The answers below assume that `getTotal` can be used to observe the value of a `ShoppingCart`. Here are several tests:

Input or test setup: **Create an empty `ShoppingCart s`. Call `s.applyDiscount(0.5)`.**

Expected output: **`getTotal()` returns 0.0.**

Input or test setup: **Create an empty `ShoppingCart s`. Call: `s.addItem(new Item("Apple", 5.0); s.applyDiscount(1.0)`.**

Expected output: **`getTotal()` returns 5.0.**

Input or test setup: **Create an empty `ShoppingCart s`. Call: `s.addItem(new Item("Apple", 5.0); s.applyDiscount(0.8)`.**

Expected output: **`getTotal()` returns 4.0.**

Input or test setup: **Create an empty `ShoppingCart s`. Call: `s.addItem(new Item("Apple", 5.0); s.addItem(new Item("Apple", 5.0); s.applyDiscount(0.8)`.**

Expected output: **`getTotal()` returns 8.0.**

Input or test setup: **Create an empty `ShoppingCart s`. Call: `s.addItem(new Item("Apple", 5.0); s.applyDiscount(0.8); s.addItem(new Item("Banana", 3.0`.**

Expected output: **`getTotal()` returns 7.0.**

Question 6. (6 points, 2 each) We would like to add a `getTotal()` method to `ShoppingCart` that returns the total price of the items in the `ShoppingCart`. Here are three possible specifications for this new method.

Spec. A

```
/**
 * @return The total price of all items in the shopping cart.
 * @throws IllegalStateException if the shopping cart is empty
 */
```

Spec. B

```
/**
 * @return The total price of all items in the shopping cart.
 * @requires The shopping cart is not empty.
 */
```

Spec. C

```
/**
 * @return The total price of all items in the shopping cart, or $0 if there are none.
 */
```

Describe the relationship between each pair of specifications by circling the correct answer below:

a) **Spec A is** weaker than [✓ stronger than] incomparable to **Spec B**

b) **Spec A is** weaker than stronger than [✓ incomparable to] **Spec C**

c) **Spec B is** [✓ weaker than] stronger than incomparable to **Spec C**

Question 7. (9 points, 3 each method) Here are three possible implementations of `getTotal()` for `ShoppingCart`. For each implementation, circle *all* of the names of the specifications from the previous page that it satisfies (if any).

```
public double getTotal1() {
    if(items.isEmpty()) {
        throw new IllegalStateException();
    }
    double total = 0.0;
    for(Item item : items) {
        total += item.getPrice();
    }
    return total;
}
```

[✓ Satisfies Spec A]

[✓ Satisfies Spec B]

Satisfies Spec C

```
public double getTotal2() {
    if(items.isEmpty()) {
        return -1.0;
    }
    double total = 0.0;
    for(Item item : items) {
        total += item.getPrice();
    }
    return total;
}
```

Satisfies Spec A

[✓ Satisfies Spec B]

Satisfies Spec C

```
public double getTotal3() {
    double total = 0.0;
    for(Item item : items) {
        total += item.getPrice();
    }
    return total;
}
```

Satisfies Spec A

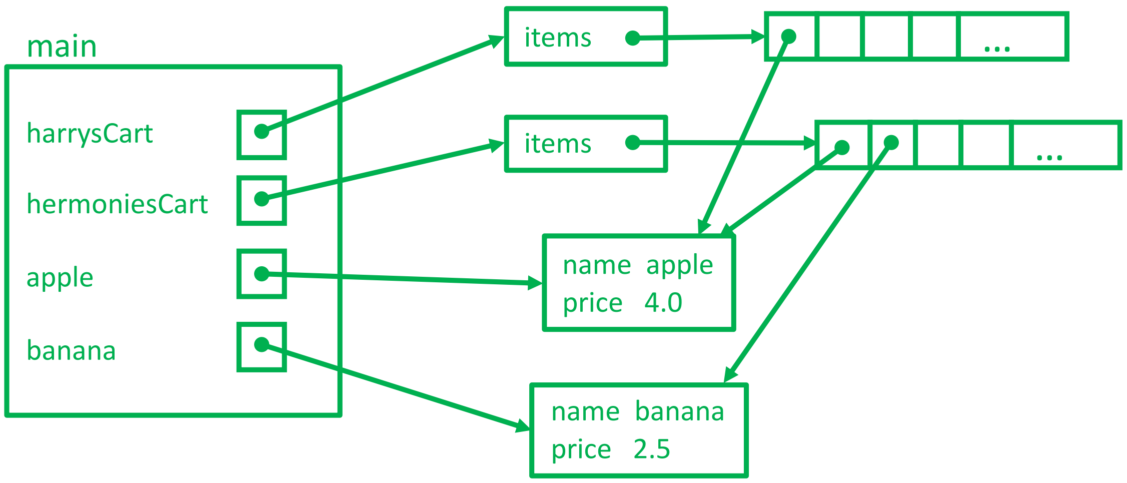
[✓ Satisfies Spec B]

[✓ Satisfies Spec C]

Question 8. (14 points). We've decided to add method `getTotal3` from the previous page to the existing `ShoppingCart` code, and we've written a small program to try it out. One of our lucky customers is given a discount. Here is the code:

```
  public static void main(String[] args) {
    ShoppingCart harrysCart = new ShoppingCart();
    ShoppingCart hermoniesCart = new ShoppingCart();
    Item apple = new Item("apple", 5.0);
    Item banana = new Item("banana", 2.5);
    harrysCart.addItem(apple);
    hermoniesCart.addItem(apple);
    hermoniesCart.addItem(banana);
    harrysCart.applyDiscount(0.8);
    /**** HERE!!! ****/
    System.out.println("Harry: " + harrysCart.getTotal3());
    System.out.println("Hermione: " + hermoniesCart.getTotal3());
}
```

(a) (6 points) Draw a diagram showing the variables and objects as they exist when execution of `main` reaches the first `println` statement (i.e., where the `HERE!!!` comment is, right after `applyDiscount` returns).



Note: Technically the `name` Strings in `Item` object are separate objects, so a truly careful diagram would show that. For this question it's fine to omit that detail.

(continued on next page)

Question 8. (cont) (b) (2 points) There's a bug somewhere. In method `main`, the discount should only apply to `harrysCart` (that's what the client expects), but it appears to have also affected `hermoniesCart` too! What output is printed when the program is executed?

```
Harry: 4.0
Hermione: 6.5
```

(c) (3 points) What is the bug? You should assume that the problem is *not* in the client code – `main` should work as written. Describe what went wrong in a couple of sentences. If this bug has a specific name be sure to include that in your description.

There is a representation exposure bug. If the same `Item` object is added more than once to one or more `ShoppingCarts`, all of the `ShoppingCarts` share a reference to the same object. If any `ShoppingCart` changes the `Item`, the change will affect all of the references to it in the `ShoppingCart`(s).

(d) (3 points) Describe an appropriate way to fix this bug, also briefly.

One simple way is for each `ShoppingCart` to make a local copy of each `Item` that is added so it has its own private copy or copies.

Question 9. (4 points) Class `Item` contains a `hashCode` method, but, as the comment in the code implies, it might not be correct. Does the given method satisfy the specification for `hashCode`? If so, give a brief justification for why it does; if not, describe what's wrong and how to fix it.

It is not correct. `Item.equals` only compares `Item` names when testing for equality but the provided `hashCode` method uses both `name` and `price` to compute the hashcode. The fix is easy: `Item.hashCode` should just return `name.hashCode()` as its result.

Question 10. (14 points, 2 each) Overloading, overriding, and `equals`. We've found the following code, which defines classes for 2-D and 3-D points, but doesn't quite get equality right – notice that the parameter types in the `equals` methods look suspicious. But the code does compile and execute without reporting any errors. Answer questions about this code on the next page. There is a second copy of this page at the end of the exam that you can remove for convenience.

```
/** Point on a 2-D plane with x,y coordinates */
public class Point {
  private int x, y;
  public Point(int x, int y) {
    this.x = x; this.y = y;
  }
  public boolean equals(Point o) {
    if (! (o instanceof Point)) {
      return false;
    }
    Point p = (Point) o;
    return this.x == p.x && this.y == p.y;
  }
}
/** Point on 3-D plane with x,y,z coordinates */
public class Point3d extends Point {
  private int z;
  public Point3d(int x, int y, int z) {
    super(x,y);
    this.z = z;
  }
  public boolean equals(Object o) {
    if (! (o instanceof Point)) {
      return false;
    }
    if (! (o instanceof Point3d)) {
      return super.equals(o);
    }
    Point3d p3 = (Point3d) o;
    return super.equals(p3) && this.z == p3.z;
  }
  public static void main(String[] args) {
    Point pta = new Point(1,2);
    Point ptb = new Point(5,6);
    Point3d p3a = new Point3d(1,2,3);
    Point3d p3b = new Point3d(1,2,4);
    Object o2a = pta;
    Object o3a = p3a;
    Object o3b = p3b;
    ______________________ ;  // insert code from questions here
  }
}
```

Do not remove this page from the exam, but feel free to tear off the copy of this page at the end of the exam. Continue with questions about this code on the next page.

Question 10. (cont.) For each line of code below, indicate what happens if it is inserted by itself at the end of the `main` method on the previous page and executed. For each one, indicate which method is called during execution (`Object.equals`, `Point.equals`, or `Point3d.equals`) and whether the method call returns true or false. Circle the correct answers.

(a) `pta.equals(ptb);`

Class whose `equals` method is executed: `Object` [✓ `Point`] `Point3d`

Result: `true` [✓ `false`]

(b) `pta.equals(p3a);`

Class whose `equals` method is executed: `Object` [✓ `Point`] `Point3d`

Result: [✓ `true`] `false`

(c) `p3a.equals(pta);`

Class whose `equals` method is executed: `Object` [✓ `Point`] `Point3d`

Result: [✓ `true`] `false`

(d) `p3a.equals(p3b);`

Class whose `equals` method is executed: `Object` [✓ `Point`] `Point3d`

Result: [✓ `true`] `false`

(e) `o3a.equals(p3a);`

Class whose `equals` method is executed: `Object` `Point` [✓ `Point3d`]

Result: [✓ `true`] `false`

(f) `o2a.equals(p3a);`

Class whose `equals` method is executed: [✓ `Object`] `Point` `Point3d`

Result: `true` [✓ `false`]

(g) `o3a.equals(o3b);`

Class whose `equals` method is executed: `Object` `Point` [✓ `Point3d`]

Result: `true` [✓ `false`]

Question 11. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

Design and implement a full Java compiler. Be sure to include all necessary libraries and utility programs for it to be useful.

(b) (1 point) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

[✓ $!@$^*% No !!!!!]

No opinion / don't care

None of the above. My answer is ________________________.