SWE619–Fall'21: Final Exam

Your Name: Dana Jamous

12/13/2021

Instructions

1. This is an open-book exam. This means that you can access course materials in the book/lecture notes/videos.
2. It is a violation of the honor code to communicate with any other person (except the instructor or TA) about this exam.
3. It is a violation of the honor code to discuss or share the contents of this exam in any way with any student who is currently registered for this course but who has not yet completed this exam.
4. You must type all solutions. You can use plain text format or markdown. If you use something else such as Word or LaTeX, you need to export to PDF and submit the PDF. **Do Not** submit any code (.java) file. if you need to change the code, put the modified code directly in your submission.
5. You need to submit on Blackboard by the deadline. If, for any reason, you have a problem submitting to BB, submit your final on Piazza in a private post. Your post should also explain your problem.

Section	Points	Score
Question 1	20	
Question 2	20	
Question 3	20	
Question 4	20	
Question 5	20	
Question 6	0	
Question 7	0	
Total	100	

1 Question 1

Consider Queue.java.

1. For `enQueue`, write (i) a partial contract and (ii) a total contract. For each part, if you need to change the code for the contract, do so and explain what you did

Answer:

(I) Partial Contract

//Requires: The enQueue calls the add method in Queue interface and based on javaDoc the add method do not have any preconditions

//Effect: add a new element to the to the tail of the queue

(i) Total Contract: No Requires clause (in total contract behavior is defined for all type-correct inputs)

//Effect: add a new element to the Queue

2. Write the rep invs for this class. Explain what they are.

Answer:

(1) rep-inv1 = elements != Null: because all class methods satisfy this invariant

-Constructor: initialize the queue to be not null

-enQueue: when we enter "elements" is not null and after we finish it will only adds an new element

-deQueue: it can make the queue empty but never null after removing all elements

3. Write a reasonable `toString()` implementation. Explain what you did

Answer:

The toString() provided by Object is not informative, it only provides the type name of the object and its hash code. Therefore, we override the toString() method to provide something informative /representative about our Queue as bellow:

```
public String toString ( ) {
     1. if (elements.size( ) == 0) return "Queue: { }";
     2. String string = "Queue: {" + elements.get(0);
     3. for (int i = 1; i < elements.size( ); i++) {
        4.string = string + " , " + elements.get(i); }
     5.return string + " }";
            }
```

Explain each line of code:

- line 1: if the queue is empty return the string"Queue:{ }"
- line 2: get the first element in the queue and concatenate with "{" (start building the string)
- line 3: for loop to iterate queue elements and build a string that have queue elements in representative way
- line 4: on each iteration concatenate the current element with the previous string and ","
- line 5: return the final build string concatenated with "}" at the end

The above code returns the queue in representative way, the outputs of tostring():

- if the Queue is empty toString() output will be => Queue: {}
- if the Queue have the elements "a","b","c" toString output will be => Queue: {"a","b","c"}

1 Question 1

4. Consider a new method, `deQueueAll ()`, which does exactly what the name suggests. Write a reasonable contract for this method and then implement it. Be sure to follow Bloch's advice with respect to generics. Explain what you did

Answer:

Contract

//Requires: elements is not empty (elements.size !=0) and element != null

//Effect: if elements is not empty remove all elements from the queue and return the removed elements else throw `IllegalStateException`

Implementation

Since we can't use for loop and remove the elements one by one because an `ConcurrentModificationException` will be thrown if we try to remove elements and modify the size inside a loop So the only way to deQueueAll is going to be calling **RemoveAll**

```
public List<E> deQueueAll () {
   1.if (size == 0) throw new IllegalStateException("Queue.deQueue");
   2.List<E> result = new ArrayList<E>;
   3.result.addAll(elements);
   4.elements.removeAll(elements);
   5.return result;
}
```

Explain each line of code:

1. If the list is empty throw `IllegalStateException("Queue.deQueue"`
2. Create new local list to use it to save elements data before removing
3. Add elements data to result so we can return it
4. Remove all data form elements
5. Return the removed elements as a generic list

5. Rewrite the `deQueue()` method for an immutable version of this class. Explain what you did

Answer:

```
public Queue<E> deQueue (){
 if (size == 0) throw new IllegalStateException("Queue.deQueue");
 List<E> elementsCopy = new ArrayList<E>();
 elementsCopy.addAll(elements);
 elementsCopy.remove(0);
 return new Queue<E> (elementsCopy); }
```

Changing the deQueue() method for an immutable means that we should not modify /change the state of the variable elements, so what we did was:

1. we create a new list to make the changes (removing the first element) on that newly created list instead of changing elements
2. we copied elements data into elementsCopy using addAll method
3. We made the change(remove) on elementsCopy instead of elements
4. We returned newly created instance of queue

1 Question 1

6. Write a reasonable implementation of `clone()`. Explain what you did.

Answer

I created a new Queue to deep copy the original queue to it

```
public Queue<E> clone() {
Queue<E> result;
try {
result = (Queue<E>) super.clone();
result.elements = new ArrayList<String>(elements); // deep copy
return result;
} catch (CloneNotSupportedException e) {
e.printStackTrace();

return null;}
```

2 Question 2

Consider Bloch's final version of his Chooser example, namely GenericChooser.java.

1. What would be good rep invariants for this class? Explain each.

   Answer:

   ```
   choiceArray != null
   ```

2. Supply suitable contracts for the constructor and the `choose()` method and recode if necessary. The contracts should be consistent with your answer to the previous question. Explain exactly what you are doing and why.

   Answer:

   ```
   public Chooser(Collection choices) {
     choiceArray = choices.toArray();
   }
   ```

3. Argue that the `choose()` method, as documented and possibly updated in your previous answers, is correct. You don't have to be especially formal, but you do have to ask (and answer) the right questions.

3 Question 3

Consider StackInClass.java. Note of the `push()` method is a variation on Bloch's code.

1. What is wrong with `toString()`? Fix it.

   Answer:

   toString() prints the entire contents of elements, instead of just the elements of the stack. This will print junk data

   fix elements.length =size

```
    @Override public String toString() {
    String result = "size = " + size;
   result += "; elements = [";
    for (int i = size - 1; i >= 0 ; i--) {
       if (i > 0)
        result = result + elements[i] + ", ";
              else
           result = result + elements[i] }
    return result + "]";
}
```

2. As written, `pushAll()` requires documentation that violates encapsulation. Explain why and then write a contract for `pushAll()`.

   Answer:

   Contract

   //Effects: Add all the elements of Object[] to the stack, modifies elements

3. Rewrite the `pop()` method for an immutable version of the `Stack` class. Keep the same instance variables. Rewrite what you did.

   Answer:

   Instead of modifying elements directly we create a new CopyElements and we copy the values of the original elements to it, and then we remove the element from the copy instead of the original

```
public Object pop () {
    if (size == 0) throw new IllegalStateException("Stack.pop");
    Object result = elements[--size];
    Object[] elementsCopy elementsCopy.addAll(elements);
    elementsCopy [size] = null;
    return result;
```

4. Implementing the `equals()` method for this class is a messy exercise, but would be much easier if the array was replaced by a list. Explain why. Note: You are not required to provide a implementation inyour answer, but if you find it helpful to do so, that's fine.

   Answer:

   Because using lists instead of array will eliminate the need for many of the instance variables. Because Arrays are covariant and reified. As a consequence, arrays provide runtime type safety but not compile time type and by And Lists play well with generics

   So Object[] elements is replaced with ArrayList<Object> elements. This will facilitates the removal of member "size" since ArrayList dynamically scale. Rep-invariants related to the member size can be removed to simplify the rep-invariant.

4 Question 4

Consider the program below (y is the input).

```
1 {y ≥ 1} // precondition
2
3  x := 0;
4 while(x < y)
5    x += 2;
6
7 {x ≥ y} // post condition
```

1. Informally argue that this program satisfies the given specification (pre/post conditions).

   Answer:

   The program satisfies both the pre and post conditions, to make sure we should assume P holds, if S successfully executes, then Q holds then we know that it satisfy the pre and post conditions

   so we have preconditions y >=1 and x is assignment to 0, x will keep increment it by 2 while its less than y so that means that x is always going to be either equal to y or larger, lets assume the bellow:

   (x =0 , y=7) P holds because 7 >=1 then we execute the assignment statement x=0 and we pass entering the while condition because 0<7 then we keep increment x by 2 until it is no longer smaller than y in this case until x= 8 y=7, then we satisfy the postcondition x>=y (8>7)

   (x =0 , y=8) P holds because 8 >=1 then we execute the assignment statement x=0 and we pass entering the while condition because 0<8 then we keep increment x by 2 until it is no longer smaller than y in this case until x= 8 y=8, then we satisfy the postcondition x>=y (8==8)

2. Give 3 loop invariants for the while loop in this program. For each loop invariant, informally argue why it is a loop invariant.

   Answer:

   Loop-inv1 (y>=1) because we never change the value of y

   Loop-inv2 (x>=0) because x value increases since it incremented by 2 in every iterate so it will always be larger or equal than 0

   Loop-inv3 (True) it only requires to hit the location

3. *Sufficiently strong loop invariants:* Use a sufficiently strong loop invariant to formally prove that the program is correct with respect to given specification. This loop invariant can be one of those you computed in the previous question or something new.

   - Note: show all works for this step (e.g., obtain weakest preconditions, verification condition, and analyze the verification condition).
   - Recall that if the loop invariant is strong enough, then you will be able to do the proof. In contrast, if it is not strong enough, then you cannot do the proof.

   Answer:

   WP(while [I] b do S, {Q}) = I && (I && b => WP(S,I) && (I && !b) => Q)
   WP(while [I] b do S, {Q}) =

   - I :
   - I & b => wp(S,I)
   - I & !b => Q

4 Question 4

Using Loop-inv2 (x>=0)

1. X>=0
2. (x>=0) & (x<y) => wp(x+=2, {x>=0})
   y>=0 => x+2 >=0
   y>=0 => x >=2
   True
3. (x>=0) & !(x<y) => x>=y
   x>=0 & x>=y => x>=y
   y<=0 => x>=y
   True

Creating and checking verification condition vc

p=> WP(while [y>=1] x<y do x+=2, {x>=y})=
p=> x+2>=0
p=> true

4. *Insufficiently strong loop invariants:* Use another loop invariant (could be one of those you computed previously) and show that you cannot use it to prove the program.
   - Note: show all work as the previous question.

Answer:

Using loop inv (y>=1)
WP(while [y>=1] x<y do x+=2, {x>=y})=

1. y>=1
2. (y>=1) &(x<y) => wp(x+=2,{y>0})
   x<1 => y>0
   false
3. (y>=1) & !(x<y) => x>=y
   y>=1 & x>=y => x>=y
   x>=1 => x>=y
   false

so WP of while loop is false thus the WP of the entire program is false
WP(x+=2;{false})

The Vc is
p=> false
y>=1 => false
false

5 Question 5

Note: you can reuse your answers/examples in previous questions to help you answer the following questions.

1. What does it mean that a program (or a method) is *correct*? Give (i) an example showing a program (or method) is correct, an (ii) an example showing a program (or method) is incorrect.

   Answer

   A correct program based on Hoare triple logic: assume P holds, if S successfully executes, then Q holds.
   So of this Hoare triple is valid then ⇒ S(the program) is correct with respect to {P} {Q}
   Example of correct program

   1) P(precondition) = {0 < x < y & y ! = 0 }
      S(program) = z:= x/y
      Q(Postconditions)= {z<1}

   2) P(precondition) = {True}
      S(program) = x:=5
      Q(Postconditions)= x=5

   Example of Incorrect program

   1) P(precondition) = {x<y}
      S(program) = z:= x/y
      Q(Postconditions)= {z<1}

   2) P(precondition) = {True}
      S(program) = x:=5
      Q(Postconditions)= x=6

2. Explain the difference between rep invariants, loop invariants, and contract/specifications (i.e., pre/post conds). Use concrete examples to demonstrate the difference.

   Answer:

   contract/specifications: consistent of preconditions (requires) which are the things that have to be true in order for a client to invoke it. And post conditions. which are the things that will be true after the invocation has completed successfully.

   Examples

```
//Requires/preconditions: Queue != null
//Effects/Postconditions: Remove E from Queue
 public Queue<E> deQueue () {
 if (size == 0) throw new IllegalStateException("Queue.deQueue");
 List<E> elementsCopy = new ArrayList<E>();
 elementsCopy.addAll(elements);
 elementsCopy.remove(0);
 return new Queue<E> (elementsCopy)  }
```

   loop invariants: Loop invariant: captures the meaning of the loop (it is manually provided by the programmer) and it is a property that holds when the loop entered is preserved after the loop body is executed.
   Example:

```
void foo(N){
     int i =0 ;
     //loop inv must hold here at this location (i< = N)
       while (i< N){
            i++;
     //loop inv also holds here (i< = N)
}
```

   rep invariants A statement of a property that all legitimate objects satisfy and it ensures the invariant is true for an object whenever it is being used outside of its class
   For example, for IntSet, we might state the following rep invariant:

```
// The rep invariant is:
// c.els ≠ null &&
// for all integers i. c.els[i] is an Integer &&
// for all integers i , j. (0 <= i < j < c.els.size ⇒
//    c.els[i].intValue ≠ c.els[j].intValue )
```

5 Question 5

3. What are the benefits of using JUnit *Theories* comparing to standard JUnit tests. Use examples to demonstrate your understanding.

Answer:

1) Junit Theories give us the ability to use assume
2) Junit Theories give us the ability dataPoints

Assume : help to help us to make sure that the preconditions are met
Example to write test to ensure symmetry property in equals

```
@Theory public void testEquals(Object a, Object b)

{ assumeTrue (a!= null & b!=null) ;
  assertTrue(a.equals(b) == b.equals(s));
}
```

dataPoints: give us the ability to generate all possible combinations of a given inputs

```
@DataPoints
public static int[] dataPoints = { return new int[] {2,5,6}; };
```

4. Explain the differences between proving and testing. In addition, if you *cannot* prove (e.g., using Hoare logic), then what does that mean about the program (e.g., is it wrong)?

Answer:

Testing is a dynamic analysis which is process of running a program on a set of test cases and comparing the actual results with expected results.

* Strength: Fast, does not need to analyze complex code
* Weakness : could miss corner cases

Proving is a static analysis which analysis source code(static representation) of program without running it.

* Strength: We do not run it on any inputs and therefore attempts to reason about the program on all possible inputs
* Weakness :
  -Slow, infeasible, analyze the program source code
  -For certain domains or applications, failure is not an option

if we cannot prove that Hoare triple is valid then ⇒ S is not correct with respect to {P} {Q}

5. Explain the Liskov Substitution Principle (LSP). Use a concrete example to demonstrate LSP. Note: use a different example than the one given in Liskov.

Answer:

Liskov substitution principle ⇒ help us know when we should subtype property and how to know we subtyped it correctly;
1) If B is a subtype of A, then B can always be substituted for A (whenever you see A we should always be able to put B)
2) An overriding(in B) method must have a stronger (or equal to) specification (pre + post condition) than the original method of A

Example lets say we have the below 2 methods A and B and it satisfy Liskov's principle

```
A_ (int x ){
 //requires x as a positive integer
 //effects: return an intger
 }

B extends A (int x){
//requires x to be an integer
//effects: returns a positive int }
```

so based on Liskov substitution principle a method in B should have stronger specification means that Weaker preconditions and stronger (more precise) postconditions

6 Question 6

This question helps me determine the grade for group functioning. It does not affect the grade of this final.

Who are your group members?

Vu Doan, Qingyang Dai, Dana Jamous, Huan Le

1. For each group member, rate their participation in the group on the following scale:
   (a) Completely absent
   (b) Occasionally attended, but didn't contribute reliably
   (c) Regular participant; contributed reliably

   All team members were Regular participant; contributed reliably

   Huan (C) Regular participant; contributed reliably
   Qingyang (C) Regular participant; contributed reliably
   Vu (C) Regular participant; contributed reliably

7 Question 7

There is no right or wrong answer for the below questions, but they can help me improve the class. I might present your text verbatim (but anonymously) to next year's students when they are considering taking the course (e.g., in the first week of class) and also add your advice to the project description pages.

1. What were your favorite **and** least aspects of this class? Favorite topics?
2. Favorite things the professor did or didn't do? The professor was very good I enjoy this class.
3. What would you change for next time? nothing