SWE619–Fall'21: Final Exam

Your Name:

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

   Sol: Partial contract: has a precondition/require that `e` is a valid element (non-null), postcondition/-effect is straight forward, e.g., add `e` to top of stack. **Total contract:** no precondition/requirement, effects/postcondition: straightforward but also raise exception when `e` is invalid. Would need to change the code to check that `e` is non-null and raise exception if it is.

   Things to check: make sure the contract is written in general that every stack implementation would have. More specifically, if it talks about this specific implementation, it is not a good contract. Can take of some points

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

   Sol: things like elements cannot be null, $0 \leq$ size $<$ elements.length(), for i $<$ size, elements[i] != null.

   Things to check: off-by-1 error, e.g., $\leq x$ instead of $< x$, is probably OK here, the main idea acounts.

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

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

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

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

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.
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.
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.
2. As written, `pushAll()` requires documentation that violates encapsulation. Explain why and then write a contract for `pushAll()`.
3. Rewrite the `pop()` method for an immutable version of the `Stack` class. Keep the same instance variables. Rewrite what you did.
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 in your answer, but if you find it helpful to do so, that's fine.

4 Question 4

Consider the program below (`y` is the input).

```
{y ≥ 1} // precondition

x := 0;
while(x < y)
  x += 2;

{x ≥ y} // post condition
```

1. Informally argue that this program satisfies the given specification (pre/post conditions).
2. Give 3 loop invariants for the while loop in this program. For each loop invariant, informally argue why it is a loop invariant.
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.
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.

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.
2. Explain the difference between rep invariants, loop invariants, and contract/specifications (i.e., pre/post conds). Use concrete examples to demonstrate the difference.
3. What are the benefits of using JUnit *Theories* comparing to standard JUnit tests. Use examples to demonstrate your understanding.
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)?
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.

6 Question 6

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

1. Who are your group members?
2. 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

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?
3. What would you change for next time?