SWE619–Fall'21: Final Exam
Your Name: Ariunsaikhan Munkhbat
12/13/2021

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

Partial contract:
//requires: e!=null

```
public void enQueue (E e) {
        elements.add(e);
        size++;
}
```

Total contract:
//Effects: if e==null throw NPE or ISE
else: add e and increase size.

```
public void enQueue (E e) {
        if (e==null)throw ISE;
        elements.add(e);
        size++;
}
```

2. Write the rep invs for this class. Explain what they are
- elements !=null
- elements is not empty
- size>= 0
- size is not negative

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

```
public String toString() {
        String result = "size = " + size;
        result += "; elements = [";
        for (int i = 0; i < elements.length; i++) {
            if (i < size)
                result = result + elements[i] + ", ";
            else
                result = result + elements[i];
        }
        return result + "]";
   }
```

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

// Effects: if size ==0, throw ISE
else: remove all elements from queue

```
public E deQueueAll () {
     if (size == 0) throw new IllegalStateException();
     E result = deQueue(elements)
     }
```

5. Rewrite the deQueue() method for an immutable version of this class. Explain what you did
We can make a copy of elements and the remove the element instead of directly removing it. Immutable class also should not be extendable so we can make class final.

```
public E deQueue () {
     if (size == 0) throw new IllegalStateException("Queue.deQueue");
     //instead of
     // E result = elements.get(0); we can make a copy of the element
     E result = duplicate()
     //elements.remove(0);
     size--;
     return result;
   }
```

6. Write a reasonable implementation of clone(). Explain what you did.
The code might not have the exact syntax but the idea is that public clone method will call create queue result using the superclass clone method and make a copy of the elements and put it into the result.

```
public Queue clone() {
        try {
                Queue result = (E) super.clone();
                result.elements = elements.clone();
                return result;
        } catch (CloneNotSupportedException e) {
                throw new AssertionError();
        }
}
```

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.
   - choiceList != null && size(choices) > 0 its greater than) because in my post condition I throw an exception when its empty.

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.

//Precondition is none
//Post condition:
// -If of choices is null, throw IAE
// - if choices is emoty, throw Exception
// - !choices.contains(null), throw exception

```
public GenericChooser (Collection<T> choices) {
        if (choices == isEmpty()){throw exception;}
        if(choices == null){ throw IAE;}
      choiceList = new ArrayList<>(choices);
   }
```

// Requires: None
// Effects: returns random choice in List<T> choiceList

```
   public T choose() {
      Random rnd = ThreadLocalRandom.current();
      return choiceList.get(rnd.nextInt(choiceList.size()));
   }
```

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.

Choose method is documented correctly because there is nothing that is required in the contract about the precondition. Also, we want to have no precondition so that the contract is stronger. Since I said that choices cannot be empty and throws and exception, we can leave the choose() method as it is. Post condition for choose() is that we simply returns a random choice from the choiceList. I would argue this is correct since it is a simple getter method so it doesn't change the rep invariants and it also satisfies the contract.

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.

The problem is that toString() method will print out everything in the Array because it is using elements.length. Instead, it should be based on the size. That way, we don't print out everything and size will only have/print what we want. Not null or something we don't need.

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

The problem with pushAll() contract is that it doesn't require all elements to be not null.
Fix: Throw NPE for null element in the beginning so when we push all elements, we do not want to find out there is null element later.

// Contract: If any element == null, throw NPE
Else: add everything to stack.

```
public void pushAll (Object[] collection) {
        for (Object obj: collection) {
                push(obj); }
}
```

If we want the code as is then we need precondition. But we can change the code to add the null element check so we have no precondition since no precondition is a stronger contract.

```
public void pushAll (Object[] collection) {
        for (Object obj: collection) {
                if obj == null throw NPE;}

        for (Object obj: collection) {
                push(obj); }
    }
```

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

First of all, Pop needs to have a contract
//Effects: If this (stack) is already empty, then throw ISE
else: remove the top of the stack and return it

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

In order to make Pop() immutable, we need to either make the class final or private and result final.

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.

> It would be easier if Array was replaced by list because we can not reuse the equals method for array. Array is checked up to the length of an array but we want to check up to the array size. If we use list, we can reuse equals() method and no need to worry about size and possible null element. Since we no need to override equals(), we don't need to override hashcode() as well. We can just use Bloch's recipe for computing hashcode.

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).

    This program takes y as the input where y is greater than or equal to 1.
    I can argue that post condition is satisfied since we assume x>=y
    When we enter the while loop x is 0 so
        0<y ➔ x = x+2 => x = 2
        loop exits on the next iteration since x=2. Therefore, x is greater than y when y=1.

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

    X<=y
    x>=0
    Y>=1
    True

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.

➔ Answer to questions 3 and 4

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

wp (while [y>=1] x<y do x =x+2, {x>=y})

1. I
   = (y>=1)

2. I && b => wp(S, I)
   = (y>=1 && x<y) => wp(x=x+2, {y>=1})
   = (x<y) => wp(x=x+2, {y>=1})
   =(x<y)=> (x+2)>=1
3. (I&b) => Q
   =((y>=1) & !(x<y)) => x>=y
   =(y>=1 & x>=y) => x>=y
   =(x>=y) => (x>=y)
   =True

VC (p=> wp(…))
P=> wp(S, Q)
y>=1 => 1<=y

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.

We can say that program is correct when it is correct according to the contract. For example, if we look at pop() method from the previous exercise, it satisfies the contract and keeps the rep-invariants.

Correct:

```
//Effects: If this (stack) is already empty, then throw ISE
                else: remove the top of the stack and return it
public Object pop () {
     if (size == 0) throw new IllegalStateException("Stack.pop");
     Object final result = elements[--size];
     // elements[size] = null;
     return result;
   }
```

Incorrect:

```
//Effects: If this (stack) is already empty, then throw ISE
                else: remove the top of the stack and return it
public Object pop () {
     Object final result = elements[--size];
     // elements[size] = null;
     return result;
   }
```

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

Rep-invariants: representation invariant that holds the property as the program executes.

Rep-inv != null or rep-inv >= 0

Loop invariants: property that holds when the loop entered and is preserved after the loop body is executed.

Contracts specifications: pre and post condition of a program. Customer or client need to establish the Precondition and Post condition is the implementation/service. Contract can be partial and total. Contract that has Precondition is partial contract and No Precondition is total contract.

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

The benefit of using JUnit Theories is that it can be parameterized. It takes parameters (precondition) to limit values appropriately and checks the post condition values in the form of assertion after performing a specific action. The contract model for JUnit testing

calls it Assume, Act and Assert as described in the lecture slides. On the other hand, Junit tests can have setup (prefix) and teardown (postfix) to accurately invoke the proper classes to assume that the certain conditions are met.

```
@DataPoints
  public static String[] animals = {"ant", "bat", "cat"};
```

this will check each combination of animals list which is 9 combination instead of checking them one by one.

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)?

Testing is called dynamic analysis and Proving or verification is called static analysis. Basically, the difference between the two is that proving doesn't require you to run the code. You can statically analyze the source code and prove that the program does what it supposed to. But in testing, you actually run the code with some inputs and make sure the program compiles and is okay in runtime. You also don't need to care too much of what the code does as long as it runs so it is generally faster.

In Hoare Logic, we want to assume that P holds, if S successfully execute, then Q holds. If we cannot prove using this logic then it doesn't mean the whole program is wrong. I would argue that it cannot be proved by using hoare logic but not necessarily 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.

Substitution principle defines that an object of superclass should be replaced with the object of its sub classes without breaking the application. For example, if we have an overridden method of subclass, it need to accept same input params as the superclass. Same thing on the return types: return value of subclass needs to comply with return value of the superclass.

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?
    Jacob Lin
    Jesse Gonzalez

2. For each group member, rate their participation in the group on the following scale:
    (c) Regular participant; contributed reliably

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?
    I was glad to work within a group because most of the times my group members were able to explain or help with homework. My recommendation to new students is that, if you don't have Computer science background, this class is too difficult for you.

2. Favorite things the professor did or didn't do? 3. What would you change for next time?
    I had hard time following the lectures. I think it would be easier if we had power point slides to follow along. It is easy to lose track of what the question is or the topic.