Name: Vaibhav Oza

GNumber: G01337138

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:

Partial contract:

Precondition: Queue e should exist.

Postcondition: Size of e should increase by 1 after adding element in e

Total contract:

Precondition: Queue e should exist.

Queue e should not be NULL.

Postcondition: Size of e should increase by 1 after adding element in e.

Queue e should be in the same sequence provided by the user

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

Answer:

size>=0 : Queue is NOT NULL

According to me this should be the only rep invariant as if size of Queue is NULL then exception will be thrown and code won't execute.

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

Answer:

```
for(Object e : q) {
e = Convert.toString(e);
}
```

Each element in Queue e will be converted to String.

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:

```
public E2 deQueueAll (int size, Queue <String> q,  List<E> elements）  {
for (Object e :q){
      q.dequeue(e);
   size--;
}
   return this.result;
  }
```

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

Answer:

```
public E deQueue (int size, Queue <String> q,  List<E> elements）  {
   if (this.size == 0) throw new IllegalStateException("Queue.deQueue");
   this.elements = elements;
   E result = elements.get(0);
   elements.remove(0);
   size--;
   return this.result;
  }
```

Addition of this helps retrieve data from immutable class and update instances of data of List only inside dequeue, without affecting any actual data in List.

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

Answer:

```
Queue <string> q2 = q.getClass().newInstance();
   for(Object e : q) {
      q2.enQueue(e);
   }
```

Above code will clone each element from Queue q to Queue q2

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:

choiceList.size()>0 : this will represent that there is at least one element in the list and that the list is NOT NULL.

rnd< choiceList.size() : this will represent that the range of random numbers that can be generated will not exceed size of list.

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:

Preconditions:

//List must not be NULL.

Postcondition:

//rnd is integer

//0 <= rnd < choiceList.size()

List being NULL would mean the program will throw an IllegalArgument exception when return statement is executed. Hence, precondition that list must not be NULL is required.

If rnd generated is greater than size of list then IndexOutOfBounds Exception will be thrown. Hence, values of rnd must always be less than size of list.

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.

Answer:

According to me choose() is correct without any changes necessary, although a question may arise of what happens when List is NULL. Hence, NULL list should be handled before entering choose() method.

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:

```
String result = "size = " + size;
      result += "; elements = [";
```

is incorrect
Correct implementation
String result = Convert.ToString(size);

result += elements;

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

Answer:

Each object in collection is pushed and can be changed inside pushall execution hence it violates encapsulation.

Precondition: Data should be available as a clone or a new instance.

Postcondition: Data should be stored in an array. Data should not be changed. Size of new array/List created should be same as size of collection.

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

Answer:

```
public Object pop (int size, Object[] elements) {
this.size = size;
    if (size == 0) throw new IllegalStateException("Stack.pop");
this.elements = elements;
    Object result = elements[--size];
    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 in your answer, but if you find it helpful to do so, that's fine.

Answer:

Array will store different types of data types and if we want to compare specific value, we will have to define it according to the data type of the same value inside array. This may result in breaking of equals method due to difference in datatype. On the other hand List has single data type which makes equals to return correct value as there will only be one datatype that needs to be considered.

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:

Precondition is satisfied as the value of y must be “y>=1” and ii not changes throughout the execution of code.

Consider value of y as 2. So the while loop will execute once as x is 0 and it is less than 2. Hence, value of x will be updated to 2 and loop will end. Thus, satisfying the post condition i.e. x>=2.

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 Invariants:
y>0 is an invariant as if y is 0 loop won’t execute and it also satisfies pre-condition.
x>=0 is an invariant as x is 0 and the value can be equal to y if y is an even
x<=y+1 is an invariant as x can be equal to y+1 if y is odd.

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 analyse 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:

x>0; // weakest precondition

y>1; // weakest precondition

x<=y+1 //strong loop invariants

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:

x>=0 // insufficiently strong loop invariants.

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 satisfies some contract or specifications i.e., it satisfies some set of pre and post conditions for valid and/or required output.

(i). Consider the below code two divide two numbers:

```
double x = 5;
double y= 2;          //preconditions: x >= 0.00, y > 0.00 i.e. are float/double
double z= x/y;        //postcondition : z >= 0.00 i.e. z is a float/double
```

Here, inputs x and y satisfy preconditions and value of z will be '2.5' which satisfies postcondition.

(ii). Consider the below code two divide two numbers:

```
double x = 5;
double y= 2;          //preconditions: x >= 0.00, y > 0.00 i.e. are float/double
int z= x/y;           //postcondition : z >= 0.00 i.e. z is a float/double
```

Here, inputs x and y satisfy preconditions and value of z will be '2' instead of the correct value which is '2.5' which does not satisfy postcondition and is incorrect in terms of calculation.

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:

Rep invariant : It defines the legal or correct definition of a specific variable/condition .

Ex: sorted list of alphabets in ascending order. Here, rem invariant would be *str[0]<str[]1<….str[length_of_list)-1].*

Loop invariant : It is some predicate that stays true for every iteration of the loop.

Ex: "(i>0 &&i<10)" remains true for every iteration of loop

```
int i;
for(i=0; i<10; i++)
{
System.out.print(i);
}
```

Output:

0123456789

Contract/Specifications: They are conditions that should be specific before and after the execution of a specific program.
Ex: Consider the below code two divide two numbers:

```
double x = 5;
double y= 2;                    //preconditions: x >= 0.00, y > 0.00 i.e. are
float/double
double z= x/y;                  //postcondition : z >= 0.00 i.e. z is a float/double
```

Here, inputs x and y satisfy preconditions and value of z will be '2.5' which satisfies postcondition.

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

Answer:

Junit Theories test several data points in each scenario whereas Junit test just test one specific data at a time. This difference helps in experimental testing of a specific data in multiple number for scenarios without specifically mentioning each scenario that may/may not be missed if written individually as Junit tests. For a Theory to be successful all the scenarios must pass i.e. it improves the quality of the code dramatically as multiple scenarios are tested in a single theory test.

Consider a simple mathematical scenario

x,y > 0 then verify if 'x + y > x ' & 'x + y > y'

Example for JUnitTheories:

```
@RunWith(Theories.class)
public class test_xy_JUTheories {
  @DataPoints
  public static int[] xy_IntegerSet() {
```

```
    return new int[]{
                1, 10, 1234567};
 }
 @Theory
 public void x_y_CompareSelf(Integer x, Integer y) {
   assertTrue(x + y > x);
   assertTrue(x + y > y);
 }
}
```

Here all integers defined in datapoints can be compared w.r.t the given scenarios that needs to be tested and it will only return as true(passed) if all integers satisfy conditions in assertTrue.

Example for standard JUnitTests:

```
@Test
 public void x_y_CompareSelf(){
  int x = 2;
  int y = 3;
  assertTrue(x + y > x);
  assertTrue(x + y > y);
 }
```

Here, we can only compare specific values of x and y at a time unlike JUnitTheories.
JUnitTheories increase the range of testing scenarios and reduces the number of tests in terms of lines of code at the same time.

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:

Proving a functionality/condition/sequence gives a deterministic conclusion that the software is correct/incorrect. That said, Proving is a way to determine that a specific functionality/condition/sequence is valid(whether correct or incorrect as required by the software specifications). Testing on the other end is to determine whether it works correctly based on the specifications

mentioned and helps to detect memory leaks or bugs i.e. to verify correctness of flow of execution.

If a program can't be proven (ex: using Hoare Logic) does not necessarily mean that the program is incorrect, it may mean it has partial correctness i.e. it satisfies some pre/post condition and does not necessarily provide an incorrect output.

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:

Statement: "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."

Objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.

If we consider a square, we can say that it is a rectangle as it inherits all its properties.

But if we implement this in code, then a square can be used anywhere instead of the rectangle which may not satisfy all the conditions required which can only be satisfied with a rectangle. Hence, square fails the LSP as Rectangle may require Length and width but square just requires length of one side.

Another example would be a Class bird() exists. There are two other classes that extend bird() namely duck() and ostrich(). Now, bird() has method fly() which is inherited by both duck() and ostrich(), but this method can only be valid for duck() as it is a bird which can use fly() whereas ostrich() can't use fly() and thus, does not satisfy LSP.

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?

Following are my group members including me:

Ramachandra Rao Seethiraju
Mathias F Wiesbauer
Mikaela Goldrich
Vaibhav Vijay Oza (me)

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

Ramachandra Rao Seethiraju: (c)
Mathias F Wiesbauer: (c)
Mikaela Goldrich: (c)
Vaibhav Vijay Oza (me): (c)

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 add your advice to the project description pages.

1. What were your favourite and least aspects of this class? favourite topics?

Favourite aspect was group study. I learned a lot from experienced teammates. Invariants is my favourite topic.

2. favourite things the professor did or didn't do?

Final exam was very length. Professor taught great but the class being online was limiting in terms of learning and interactive experience. Also, professor was not lenient, when I had a midterm right after a lecture and I asked to give quiz later. I understand he was obligated by rules but that put me in

disadvantage for another class. (Note: It is my first semester, and I took 12 Credits, so I had 2 classes every Monday simultaneously).

3. What would you change for next time?

This class should not be online. Interaction in a f2f environment would be much better. Exam should be doable in 3 hours.