Name: Prabhath Surya Palem
Gid: G01341856

Question-1:

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

Ans:

(i) Partial Contract: A partial contract has the requires statement that is the precondition.
Pre condition: (e!=0)element e not equal to zero.
Post condition: add elements into list

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

(ii) Total contract: A total contract has not requires statement that has no pre condition.
Pre condition: No pre conditions.
Modifies: increases the size by 1.
Post condition: if(element == null)→ throws IllegalArgumentException.

```
public void enQueue (E e) {
    if(e == null)
    {
     Throw new IllegalArgumentException("null element");
    }
    elements.add(e);
         size++;
}
```

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

Ans:

I'm considering these rep invariants for Queue class,

a) e! = 0 (elements not null)
b) 0<=size<=e.length (size not equal to 0 & less than length of elements.)

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

Ans:

toString():

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

In the toString() method, prints the size of queue and the elements stored in the wueue.

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

Ans: PreCondition: size greater than 0.

PostCondition: size should be 0.

```
Public void deQueueAll()

{

   while(size>0){

         deQueue();

   }

}
```

Consider trhe post and pre conditions as size reater than 0 as pre and do the dequeue() function and in the end should be 0.

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

Ans:

```
Public List<E>deQueue(List<E>elements){

If (this.size.== 0){

  Throw new IllegalArgumentException ("Empty Queue");

}

e.res= this. elements.remove(0);

this.size --;

return res;

}
```

Is size is 0 then it throws the illegal argument exceptiuon.then reduce the elements and decrease the size.

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

Ans:

```
 Public List<E>clone(List<E>elements)

 {

    List<E>tempqueue = new ArrayList<>();

   tempqueue = (ArrayList) elements.clone();

   Return tempqueue;

 }
```

Consider the clone methos and and using the temporary variable and return the temp as a result.

Question-2:

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

Ans:

Let’s consider Bloch’s GenericChooser.java example, So the good rep invariants for this class are: choiceList! =null &&size(choices)>0 where the selected choiceList must not be empty and also size should be greater than zero.
So, this condition should be satisfied from the GenericChooser example. Therefore, this can be considered as good Rep Invariants.

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.

Ans:

Let us consider some suitable contracts for the constructor where,
The Preconditions would be null and
The Post conditions or the effects can be:

- If(choices==null || choices==0) →throw an IllegalArgumentException()
- If(size==0)→throws an IllegalArgumentException ().

Now Let us consider the total contract for choose() method implementation where,
Similarly, the Preconditions would be null and
The Postconditions: Choose() method returns random choices or elements from the choiceList.
Modified code to satisfy the Pre & Post conditions:

```
public class GenericChooser<T>
{
 private final List<T> choiceList;
 public GenericChooser (Collection<T> choices)
  {
   if(choices. Size () == null){
       throw new IllegalArgumentException();
   }
   If(choices.contains(null)){
       throw new IllegalArgumentException();
  }
   choiceList = new ArrayList<>(choices);
  }
 Public T choose() {
     Random rd = ThreadLocalRandom.current();
     return choiceList.get(rd.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.

Ans:

I’ve taken the example genericchooser.java example as requested and made some changes to the previous code so that there wouldn’t be any problems for the choose() method and I’ve added some additional conditions such as choiceList cannot be empty. The Choose() method throws some random choices from the list. So these conditions are to be updated in constructor there is not point in updating the choose() method and it will return some random elements without any issues.

Question 3:

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

   Ans: As with the toString() method, We need to get the elements till certain size but problem arises as all the elements in the array are getting printed. For the above StackInClass.java the push method pushes the elements into the stack and pop takes out the element out of the stack but on a whole these operations were able to update the size of array but were not removing the elements but for toString() we need to print the elements that are contained in the stack.

   Code after modifications:

```
@Override public String toString() {
      String result = "size = " + size;
      result += "; elements = [";
      for (int i = 0; i < size; i++) {
         if (i < size -1)
            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().

   Ans:
   pushAll() requires the documentation that violates encapsulation , as encapsulation refers to the process of wrapping up of data ie., variables & code into a single unit. And here in the StackInClass.java the encapsulation technique enables to access the data to be accessed only by set of some elementss. Here pushAll() method is declared as public so it might get affected through the process of inheritance as a result it will get affected such as other some other classes can also access the method So on a whole as a result the pushAll() method violates the encapsulation.
   Contract(pushAll()):
   Preconditions: if(element== null)→throw NullPointerException.
   Postcondition: If any element null raise exception otherwise add everything to the stack.

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

Ans:

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

Ans:
As given implementing the equals() method for this class is a messy exercise but it would be more easy if the array was replaced by a list this is because the list on a whole is dynamic which makes it flexible for operations and on a whole using the equals() method is not much required. As all the operations can be performed dynamically through the list the equals() methods can be omitted.

Question 4:

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

Ans.
Yes the given program satisfies the given specification so let us at first consider the requires condition which stands y>=1. So at the point when we consider y as 1 and x as 0 the loop iterates successfully and increments so as a whole the post condition is satisfied. Therefore these specifications are satisfied completely.

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

Ans.

The three loop invariants in here are : (i)x<=y , (ii)x>0, (iii)y>=1

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.

Ans.WP(x:=0; while[a<=b]a<b do a+=10, {a>=b})→

a) x<=y && True && True
we get x<=y

b) WP(x:=0;
c) {x<=y})=
By substituting x we get
y>=1=> 1<=y

d) Compute the verification condition
vc (P => wp(..)) P =>
WP(x := 0; while[x<=y] x<y do x+=2,
{x>=y}) =
P => 1<=y =
y>=1 => 1 <= y (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.

Ans.

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

1.y>=1

2.(y>=1 & x<y) → WP(x+=2,{y>=1})

False

3.y>=1 & !(x<y) -> x>y

y>=1 & x>=y → x>=y →False

(i)So the WP of the while loop is False, and therefore the WP of the entire program is also False as shown below:

WP(x:=0; WP{False}) = WP{False}

(ii)Compute the verification condition:
vc (P => wp(..))
P => False
y>=1 => False
False

(iii) Thus, using this loop invariant we cannot determine or prove the validity of Hoare Tripple. So this insufficiently strong.

Question 5:

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.

Ans. Lets consider the snippet code i:=0;

While x<numb {x==numb;}

Now from the above program the loop invariants can be , x<=numb, x>=0, numb>=0 &true.

As per hoare logic,→ WP(while[I]B do S, {Q}) equals

1. I
2. (I & B)→ WP(S,I)
3. (I&!B)→Q

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

Ans. Rep invariant: Representative invariant gains the set of directives that are required for the algorithms. Representation invariants can be considered as the conditions concerning the state of an object. The condition can be considered true for a given object such that all the further operations performed doesn't violate the condition.

Loop invariant: The set of rules that must be true immediately after running the loop.

Contract/specifications: They are the set of pre and post conditions of algorithm. These conditions are very much required for the correctness fo the program.

Example: Let us consider the above given example chooser from

```
import java.util.*;
import java.util.concurrent.*;

// Bloch's final version
public class GenericChooser<T> {
   private final List<T> choiceList;

   public GenericChooser (Collection<T> choices) {
      choiceList = new ArrayList<>(choices);
   }

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

For the above program ,

Let us consider some suitable contracts for the constructor where,
The Preconditions would be null and
The Post conditions or the effects can be:

- If(choices==null || choices==0) →throw an IllegalArgumentException()
- If(size==0)→throws an IllegalArgumentException ().

Now Let us consider the total contract for choose() method implementation where,
Similarly, the Preconditions would be null and
The Postconditions: Choose() method returns random choices or elements from the choiceList.

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

Ans.

The joint is required to run a series of tests on the developed program and then considers various sets of outputs from those tests.But in junit theories it tests the certain functionality against infinite set of datapoints.

Example:

```
@Theory
public void a+b>a&b(Integer a, Integer b) {
  Assume.assumeTrue(a >0 && b > 0 );
  assertTrue(a + b > a);
  assertTrue(a + b > b);
}
```

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)?

Ans.

When a developer creates a program and inorder for him to check his correctness of the algorithm he need to perform some series of tests which comes under testing process. Whereas proving involves in considering the pre and post conditions and gaining the outcome or the output in such a way that it satisfies all the conditions can be considered as proving.

The syntax of hoare logic: {P}S{Q} p,q=predicates.

A program must prove its correctness through hoare logic so that it indicates it is well written and tested.

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.

Ans.

Liskov's substitution principle states that the objects of the subclasses gets replaced by the objects of the superclass without affecting the running process or application

```
public class BankApp {

  private WithdrawAcc withdraw;



  public BankApp(WithdrawableAccount withdrawableAccount) {

    this.withdraw = withdraw;

  }
```

```
    public void withdraw (BigDecimal amount) {

        WithdrawableAcc.withdraw(amount);

    }

}
```

In the above example the super class is replacing the subclass using the liskov's principle.

Question 6:

1. Who are your group members?
   Jagdish Ramidi
   Suraj Varma
   Hussain Rohawala
   Prabhath Surya Palme

2. For each group member, rate their participation in the group on the following scale:

   (c) Regular participant; contributed reliably: jagdish, suraj, Hussain & Me everyone contributed their part well in the work.

Question 7:

1. What were your favourite and least aspects of this class? Favourite topics?
   Great working loved the course, invariants and reasoning.

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

   He's so supportive

3. What would you change for next time?

   Nothing.