Name: Hussain Rohawala.
G Number: G01286796.

Question 1:
Consider Queue.java

1.
Total contract:
//Effects: No pre-conditions.
//Modifies: It will increase the Size by 1.
//Requires: Throw a NullPointerException if element is null or add the element to the queue.

//As there is no pre condition we can update the code for the total contract.

```
public Queue() {
try(){
                elements.add(e);
                Size++;
        }
        catch(NullPointerException e){
                System.out.println("Null elements are not allowed inside the queue.");
        }
}
```

Partial contract:
//Requires: Element cannot be null.
//Modifies; Increases the Size by 1.
//Effects: It modifies the queue by adding the new element .

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

  }
```

2. I am considering the below rep-invariants:

a)0<=size<=elements.length()
-I am considering the above rep invariant to avoid the possibility of junk or null elements inside the queue.

b)e!=null
-If the element is null it should not be added in the queue.

```
public E deQueue () {
    if (size == 0) throw new IllegalStateException("Queue.deQueue");
    E result = elements.get(0);
```

```
    elements.remove(0);
    size--;
    return result;
  }
```

3.

```
public String toString() {
                String answer = “”;
                for (int i = 0; i < size; i++) {
                if (i < size-1)
                        answer= answer + elements.get(i) + ", ";
                else
                        Answer = answer + elements.get(i);
                }
                return answer;
        }
```

toString(E e) implementation.

```
 if (Size=0) { return " "; }
StringBuilder sb = new StringBuilder();
Int elementsSize=size;

        for(int i=0;i<elementsSize;i++)
                {
                        E result = elements.get(I);
                        elementsSize--;
                        sb.append(result) ;
                }
return    sb.toString();

}
```

4.

```
//Requires: Size should be greater than 0.
```

Post-condition: size should be equal to 0.

```
public void deQueueAll() {

                while(size>0){
                        deQueue();
                }
  }
```

5.

```
public List<E> deQueue(List<E> elements)
        {
        if(isEmpty())
        {
                throw new IllegalArgumentException("Queue is Empty!!");
        }
        List<E> temporaryElements= new ArrayList<E>();
        temporaryElements.addAll(elements);
         temporaryElements.remove(0);
        return new ArrayList<E>(temporaryElements);
    }
```

The original list named elements is loaded into a new array list named temporaryElements. First element is removed by using and it returns a new updated array list.

6.

```
public List<E> clone(List<E> elements)
        {
                List<E> temporaryQueue = new ArrayList<>();
                temporaryQueue = (ArrayList) elements.clone();
                return temporaryQueue ;

        }
```

I have initialized an empty array list temporaryQueue and cloned all the elements from the original array list elements. Finally, the method returns the temporaryQueue which is the copy of the original array list.

Question 2:
Consider Bloch's final version of his Chooser example, namely GenericChooser.java.

1)Rep-Invariants:

- choiceList.size()>0
  - Size of the choiceList should be greater than zero i.e it cannot be empty.
- choiceList !=null
  - choiceList cannot be null.

2)
For constructor,

```
public GenericChooser (Collection<T> choices) {

            if(choices.size() == 0)

            {

                  throw new IlegalArgumentException();

            }

            if(choices == null)

            {

                  throw new IlegalArgumentException();

            }

            choiceList = new ArrayList<>(choices);

      }
```

There is no precondition for the constructor.

Post conditions are as follows:

a) If choice is empty or null it should give an IllegalArgumentException or NullPointerException.

For choose() method,

There is no precondition for the constructor.

Post conditions are as follows:

a) If choiceList is empty it should give an IllegalArgumentException.

b)ChoiceList cannot contain null elements else it will give an exception in the choices method.

c)If choiceList is null, it should throw NullPointerException.

The above-mentioned post-condition are based on the rep-invariants. To satisfy the above postcondition, the code needs to be modified as follows:

```
public T choose() {

if(choiceList .size() == 0)
            {
                  throw new IlegalArgumentException()
            }
      if(choiceList  == null)
            {
                  throw new NullPointerException();
                  }
  if(choiceList .contains(null))
            {
                  throw new NullPointerException();
                  }
      Random rnd = ThreadLocalRandom.current();
      return choiceList.get(rnd.nextInt(choiceList.size()));
}
}
```

3)

I have updated the choose method by handling all the exceptions occurring in the method. In the constructor the choice cannot be empty or null; it should give an NullPointerException or

IllegalArgumentException. constructor as well as the choose() method will be modified to check the exception to handle the conditions in which the list cannot be empty or null.

Question 3:
Consider StackInClass.java. Note that the push() method is a variation on Bloch's code?

1) In the toString() method, we only need the elements of length size but we are using elements.length in the for loop. While performing operations, the size attribute has been updated so we should consider size as a condition in the for loop and print only the elements that are stored in the stack and not all the elements.

```
        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)

Encapsulation mentions that the data should be accessed only by a required set of elements.Public methods can be accessed from anywhere throughout the project by creating objects of class. Public methods show the internal working of the program. pushAll() method depends on push() method. As the push method can be accessed from anywhere and hence this will result in violation of the encapsulation principle. We can make the push() method as final so that the push method will not be modified or updated by any other class.

Contract for pushAll

Pre condition: If any element is null throw NullPointerException.

Post condition: elements should be added to the stack.

3)

Immutable pop() method will create a new object array which stores all the elements from the existing object array and remove an element from the new object array and return it back. Instead of updating the array we will create a new array every time.

```
public Object[] pop (Object[] elements) {

        if (size == 0) throw new IllegalStateException("Stack.pop");

        Object[] temporaryObject= elements;

        int temporarySize= size-1;

        temporaryObject[temporarySize] = null;

        return temporaryObject;

}
```

4)

Using list would be easy as there is no need of implementing equals() method, we can directly use inbuilt method. Arrays equals() method checks the array length but here we need to check the stack size so it would be difficult to use equals() method.. For Example, if there is a null value inside the array it will increase the size of the array which is not correct implementation for the equals method.

Question 4:

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

A. Yes, it satisfies the given specification because firstly the pre condition is y>=1. Let's consider y as 10 and x=0. So, it satisfies the while loop (0<10) and so the x becomes 2. Finally, which satisfies the post condition which is x>=y. Therefore it satisfies the given specification .

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 loop invariants for the while loop :

a)x>0

X greater than 0 can be taken as loop invariant as it satisfies all the conditions.

b)y>0

y>0 is present as the precondition so it is one of the best choices for loop invariant.

c)x<=y

As it satisfies both the pre and post conditions.

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.

We are taking x<=y as the loop invariant for this condition.

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

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

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

a) x<=y

b) (x<=y & x<y) => WP (x+=2, {x<=y})

By simplifying it further, we get

x<y => x+2<y

x<y => x<y (True)

c) x<=y & !(x<y) => x>=y

We can rewrite this as

x<=y & x>=y => x>=y

x=y => x=y (True)

1) x<=y && True && True

we get x<=y

2) WP (x:=0; {x<=y})=

y>=1=> 1<=y

3) 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.

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

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

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

a) y>0

b) (y>0 & x<y) => WP(x+=2, {y>=1})

(y>0 & x<y) => y>=1 (False)

We Cannot simplify this further.

False

c) y>0 & !(x<y) => x>=y

We can rewrite this as

y>0 & x>=y => x>=y

False

1)WP of the entire program is also False

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

2)Verification condition:

vc (P => wp())

P => False

y>0 => False

False

(iii)Analyzing the vc to determine whether the program is proved or not

Using the above loop invariant we cannot determine or prove the validity of Hoare Tripple. So this is insufficiently strong.

Question 5:

1) If it does not have any compile time exceptions and it satisfies the Hoare Condition proves the correctness of the method

{N >= 0} // precondition

i := 0 ;

while(i < N)

{ i =i+1; }

{i == N} // post condition

For the above code

1. i <= N

2. (i <= N & i < N) => WP(i=i+1, {i<=N})

After simplifying, we get: i < N => i+1 <= N i < N => i < N

True

3. i <= N & !(i<N) => i == N

i<=N && i>=N => i==N

After simplifying we get: i == N => i == N True = i <= N && True && True = i <=N
WP(i := 0; {i<=N}) = By substituting i value we get: 0<=N => N>=0

c) Verification condition vc (P => wp(..)) P => WP(i := 0; while[i<=N] i < N do i=i+1, {i == N}) = P => 0<=N = N>=0 => 0 <= N (True)

d) Analyze the vc to determine whether the program is proved or not

Thus using this loop invariant we can prove the corre ctness of code.

2)

Loop Invariant: Statement that should be true when entering a loop and after every iteration of the loop.

Rep Invariant: Capture all the major specifications and rules about data structure and its correctness.

Contract/specifications: Pre-conditions and postconditions defined for any program can be described as a contract for the program. Pre-condition comes under required conditions and post-conditions comes under the effects of the program.

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

Rep-Invariants example from above for the above given piece of code:

- choiceList.size()>0
  - Size of the choiceList should be greater than zero i.e it cannot be empty.
- choiceList !=null
  - choiceList cannot be null.

b)Loop invariant as well as the contract Example:

Loop invariant taken for the following code is i<=N

{N >= 0} // precondition

i := 0 ;

while(i < N)

{ i =i+1; }

{i == N} // post condition

Loop Invariant : i <= N

3)A Theory is a piece of functionality (a method) that is executed against several data inputs called data points. To make a test method a theory you mark it with @Theory. To create a data point you create a public field in your test class and mark it with @DataPoint.

It's common to execute a series of tests which differ only by input values and expected results.

As an example, if you are testing a method that validates email IDs, you should test it with different email ID formats to check whether the validations are done. Testing each email ID format separately, will result in duplicate or boilerplate code. It is better to abstract email ID test into a single test method and provide it with a list of all input values and expected results. JUnit supports functionality through parameterized tests.

4)

In testing, observable properties are determined. It verifies programs for one execution, and it is the most practical approach.It is helpful to check for different implementations in the same program.In Proofs any program property is determined. It verifies programs for all the executions.ness using pre-conditions and post-conditons are:

{P}{ S} {Q}

P and Q are predicates and S is a program.

If we start in a state where P is true and execute S, then S will terminate in a state where Q is true.

Program proves correctness using the hoare logic

5)

Liskov Substitution Principle:

It is a concept in Object Oriented Programming Language which states that the function that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

Example:

public class Bird{ public void fly(){} } public class Duck extends Bird{}

The duck can fly because it is a bird, but what about this:
public class Ostrich extends Bird{}

Ostrich is a bird, but it can't fly, Ostrich class is a subtype of class Bird, but it shouldn't be able to use the fly method, that means we are breaking the LSP principle.

Correct Implementation:
public class Bird{}
public class FlyingBirds extends Bird
{ public void fly(){} }
public class Duck extends FlyingBirds{}
public class Ostrich extends Bird{}

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?
-Jagdish,Prabhat,Suraj.

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
Jagdish- Regular participant; contributed reliably
Prabhat-Regular participant; contributed reliably
Suraj-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?
In class exercise
Favorite topics?
Loop-invariants.
2. Favorite things the professor did or didn't do?
In class Exercise
3. What would you change for next time?
Online to in-person.