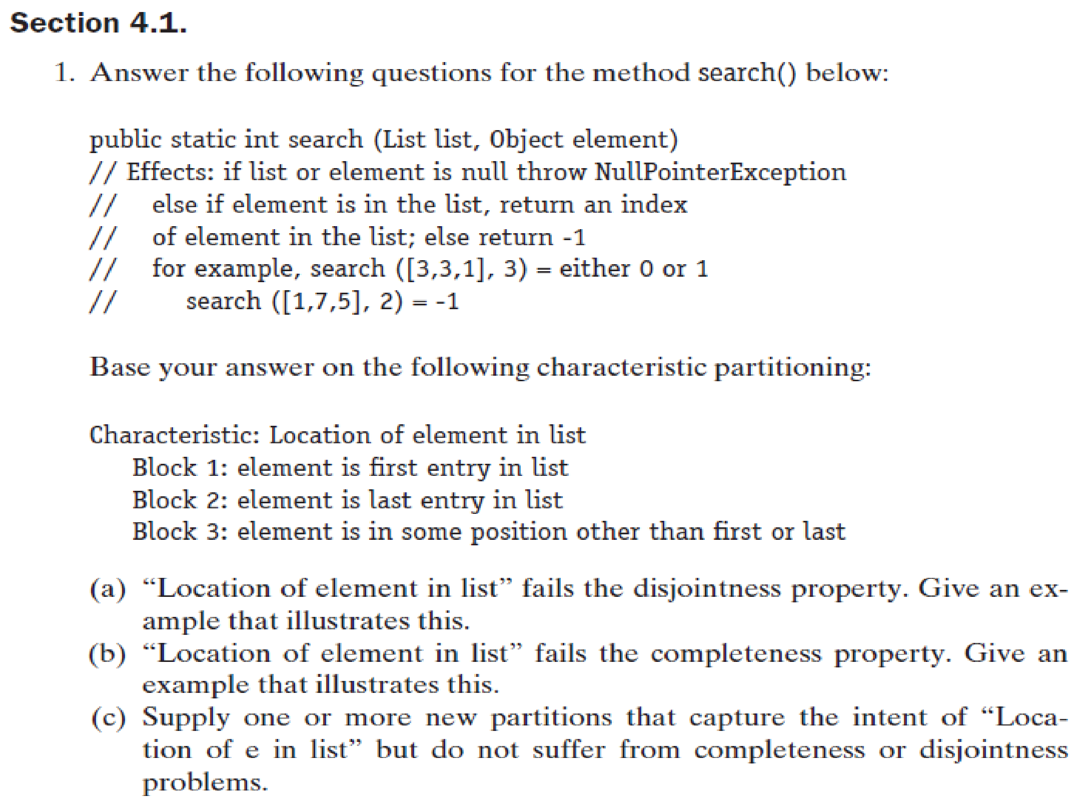
Name: Natnael Kebede

ID: 1001149004



**Section 4.1.**

1. Answer the following questions for the method search() below:

```
public static int search (List list, Object element)
// Effects: if list or element is null throw NullPointerException
//    else if element is in the list, return an index
//    of element in the list; else return -1
//    for example, search ([3,3,1], 3) = either 0 or 1
//        search ([1,7,5], 2) = -1
```

Base your answer on the following characteristic partitioning:

```
Characteristic: Location of element in list
    Block 1: element is first entry in list
    Block 2: element is last entry in list
    Block 3: element is in some position other than first or last
```

(a) “Location of element in list” fails the disjointness property. Give an example that illustrates this.

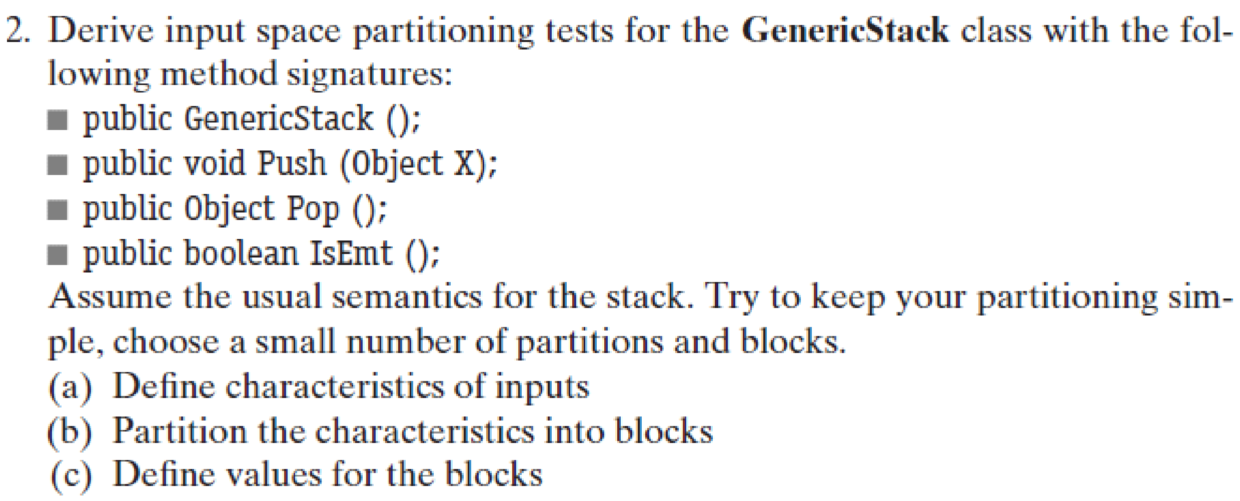
(b) “Location of element in list” fails the completeness property. Give an example that illustrates this.

(c) Supply one or more new partitions that capture the intent of “Location of e in list” but do not suffer from completeness or disjointness problems.

a) If we consider the input ([1, 3, 3], 3) then the location of the element ‘3’ in the list fails the disjointness property. The third member of the list or ‘3’ is in the last entry of the list and belongs to block 2. However, ‘3’ is also the second member of the list (it is between the first and the last members of the list) and belongs to block 3. This shows that the domain element 3 is in two different blocks and hence fails the disjointness property.

b) If we consider the input ([], 3) then, location of element in the list fails the completeness property. None of the blocks are complete to identify the element ‘3’ as a domain in the list.

C) If we consider another partition to handle null list cases when the element doesn’t exist in the corresponding list, it can result in block 4 and satisfy both the completeness and disjointness property of partition.



2. Derive input space partitioning tests for the **GenericStack** class with the following method signatures:
   - public GenericStack ();
   - public void Push (Object X);
   - public Object Pop ();
   - public boolean IsEmt ();

   Assume the usual semantics for the stack. Try to keep your partitioning simple, choose a small number of partitions and blocks.
   (a) Define characteristics of inputs
   (b) Partition the characteristics into blocks
   (c) Define values for the blocks

a) Characteristics of inputs

**public GenericStack ()**

No characters because this is a constructor with no parameters.

**public void Push (Object X)**

Position of the object on the top of stack

**public Object Pop ()**

Remove the object x from the top of the stack and update the stack pointer

**public boolean IsEmt ()**

Existence of an object in the stack

b) Partitioning the Characteristics into blocks

**public GenericStack ()**

There is no input space partition since this is a constructor with no parameters.

**public void Push (Object X)**

Block 1 = all possible input objects to put on the top of the stack
Block 2 = null object pushed on top of stack

**public Object Pop ()**

Block 1 = object x in on the top of the stack and will be popped.
Block 2 = invalid, object x was never pushed on to the top of stack.
Block 3 = no more object on the top of stack to pop
Block 4 = unable to pop object that's not on top of the stack.

**public boolean IsEmt ()**

Block 1 = True, the stack doesn't have any pushed objects in it.

Block 2 = False, the stack still has the pushed object in it.

c) Values for the blocks

**Note: each test case starts by pushing an element on top of an empty stack. When pushed, that element appears on top of the stack.**

**public GenericStack ()**

Since this is a constructor class with no input partitions, there are no blocks.

**public void Push (Object X)**

Block 1, x = 2 // 2 is pushed on top of the stack

Block 2, x = [] //null input, print an error message

**public Object Pop ()**

Push (3) // pushes 3 on top of the stack
Push (4) // pushes 4 on top of the stack
Push (5) // pushes 5 on top of the stack

Block 1, pop () // pops the first object on the stack or x = 3
Block 2, pop () // pop 11, prints the object 11 was never pushed on to the stack
Block 4, pop () // unable to pop object 4 since it is not on the top

Block 1, pop () // pops the first object on the stack or x = 4
Block 1, pop () // pops the first object on the stack or x = 5
Block 3, pop () // no more object on top of the stack to pop and throw exception.

**public boolean IsEmt ()**

Push (3) // pushes 3 on top of the stack

Block2:  False, meaning the stack still has objects that were pushed.
Pop () // 3 is popped from the top of the stack.
Block1: True, the stack is now empty. There are no more objects.