3. 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
```

Solutions:

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

Lots of examples can be found. One is: list $= [3, 4, 3]; e = 3$

Another is: list $= [3]; e = 3$

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

The problem is that e may not be in the list: list $= [5, 3]; e = 4$

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

The easiest approach is to separate the characteristics into separate partitions:

- *Whether e is first in list: true, false*
- *Whether e is last in list: true, false*

You might also consider:

- *Whether e is in list: true, false*

But this is not really covered by the original characteristic.

4. Derive input space partitioning test inputs for the **GenericStack** class with the following method signatures:

   - `public GenericStack ();`
   - `public void push (Object X);`
   - `public Object pop ();`
   - `public boolean isEmpty ();`

   Assume the usual semantics for the **GenericStack**. Try to keep your partitioning simple and choose a small number of partitions and blocks.

(a) List all of the input variables, including the state variables.

(b) Define characteristics of the input variables. Make sure you cover all input variables.

(c) Partition the characteristics into blocks.

(d) Define values for each block.

Solutions:

(a and b)

Note that there are four testable units here (the constructor and the three methods), but that there is substantial overlap between the characteristics relevant for each one. For the three methods, the implicit parameter is the state of the **GenericStack**. *The only explicit input is the* `Object x` *parameter in* `Push()`. *The constructor has neither inputs nor implicit parameters.*

Typical characteristics for the implicit state are

- *Whether the stack is empty.*
  - *true (Value stack = [])*
  - *false (Values stack = ["cat"], ["cat", "hat"])*
- *The size of the stack.*
  - *0 (Value stack = [])*
  - *1 (Possible values stack = ["cat"], [null])*
  - *more than 1 (Possible values stack = ["cat", "hat"], ["cat", null], ["cat", "hat", "ox"])*
- *Whether the stack contains null entries*
  - *true (Possible values stack = [null], [null, "cat", null])*
  - *false (Possible values stack = ["cat", "hat"], ["cat", "hat", "ox"])*
- *Whether x is null.*
  - *true (Value x = null)*
  - *false (Possible values x = "cat", "hat", "")*

There are also characteristics that involves the combination of `Object x` *and the stack state. One is:*

- *Does* `Object x` *appear in the stack?*
  - *true (Possible values: (null, [null, "cat", null]), ("cat", ["cat", "hat"]))*
  - *false (Possible values: (null, ["cat"]), ("cat", ["hat", "ox"]))*

(c) See solution to (a and b)

(d) See solution to (a and b)