Cairo University	**Midterm Exam – Third Year**	**Date: 12 April 2022** **Time:1 - 2**
Faculty of Computers and Artificial Intelligent	**Software Testing**	**Instructor: Dr. Manar Elkady**
Software Engineering Program	**Model (1)**	**No of Pages: 5** **Total Mark: 25 Marks – SCALLED to 20**

STUDENT NAME: ________________________________

STUDENT ID :________________________________

Question 1[3 MARKS]

(a) Determine the Priority and severity level for the following case with JUSTIFICATION. there are few actual users who are still using the older IE versions like IE8. The banking application when accessed in older versions of IE, the page is not loaded completely and the form fields are overlapped. This makes the accessibility of the website difficult for IE users using the older versions. **[1.5 MARKS]**

Severity 2, Priority 3 (0.5 marks)
OR: Severity high, priority low

Hence, the severity is high because the whole application is impacted. However, there will be very few actual users who will use IE 8 and other older versions, this makes the priority Low because this fix can wait. (1 marks)

(b) A tester defined three characteristics based on the input parameter *car*: **Where Made**, **Energy Source**, and **Size**. The following partitions for these characteristics have at least two mistakes. Correct them. **[1.5 MARKS]**

	Where Made	
North America	Europe	Asia
	Energy Source	
gas	electric	hybrid
	Size	
2-door	4-door	hatch back

Solution
Pair Disjoint property violation: 2-door, 4-door blocks are not disjoint.
Size overlaps, a hatch-back could be 2-door or 4-door. Either add \2-door + hatch-back," and \4-door + hatch-back," or create two new characteristics:
Side Doors: 2, 4
Hatch-back: yes, no
(0.75 marks)

Competence property violation: other made in countries are missing. Where Made is not complete. Add \other (0.75 marks)

<u>Question 2 [5 MARKS]</u>

```
/* Effect: if x==null throw NullPointerException.
Otherwise, return the index of the last element in the array 'x' that equals integer 'y'.
Return -1 if no such element exists. */
public int findLast (int[] x, int y) {
        for (int i=x.length-1; i>0; i--) {
                if (x[i] == y) { return i; }
        }
        return -1;
}
```

Read this faulty program, which includes a test case that results in failure. Answer the following questions.
a) Identify the fault, and fix the fault.
b) If possible, identify a test case that does not execute the fault.
c) If possible, identify a test case that executes the fault, but does not result in an error state.
d) If possible identify a test case that results in an error, but not a failure. Hint: Don't forget about the program counter.
e) Identify the first error state. Be sure to describe the complete state.

<u>ANSWER</u>
(a) The for-loop should include the 0 index: (1 marks)
for (int i=x.length-1; i >= 0; i--)

(1 marks) (b) The null value for x will result in a NullPointerException before the loop test is evaluated, hence no execution of the fault.
Input: x = null; y = 3
Expected Output: NullPointerException
Actual Output: NullPointerException

(1 marks) (c) For any input where y appears in a position that is not position 0, there is no error. Also, if x is empty, there is no error.
Input: x = [2, 3, 5]; y = 3;
Expected Output: 1
Actual Output: 1

(1 marks) (d) For an input where y is not in x, the missing path (i.e. an incorrect PC on the final loop that is not taken, normally i = 2, 1, 0, but this one has only i = 2, 1,) is an error, but there is no failure.
Input: x = [2, 3, 5]; y = 7;
Expected Output: -1
Actual Output: -1

(1 marks) (e) Note that the key aspect of the error state is that the PC is outside the loop (following the false evaluation of the 0>0 test. In a correct program, the PC should be at the if-test, with index i==0.
Input: x = [2, 3, 5]; y = 2;
Expected Output: 0
Actual Output: -1
First Error State:
– x = [2, 3, 5]
– y = 2;
– i = 0 (or undefined);
– PC = return -1;

Question 3 [6 MARKS]

Derive input space partitioning tests for the **GenericStackclass** 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, choosea small number of partitions and blocks
(a) List ALL inputs variables [1 MARK]
(b)Define characteristics of all input variables [2 MARK]
(c) Partition the characteristics into blocks. [2 MARK]
(d) Define values for the blocks [1 MARK]

Solution

Note that there are 4 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 theObject xparameter inPush().

The constructor has neither inputs nor implicit parameters.

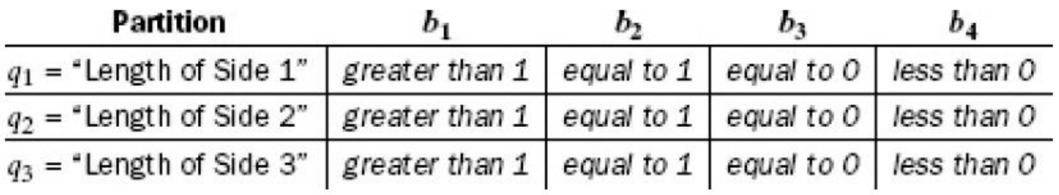
Typical characteristics for the implicit state are
- Whether the stack is empty
  - true (Value stack = [])
  - false (Values stack = [”cat”], [”cat”, ”hat”])
- The size of the stack.
  - 0 (Value stack = [])
  - 1 (Possible values stack = [”cat”], [null])
  - more than 1 (Possible values stack = [”cat”, ”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”])

A typical characteristic for Object x is
- Whether x is null.
  - true (Value x = null)
  - false (Possible values x = ”cat”, ”hat”, ””)

There are also charactistics 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”]))

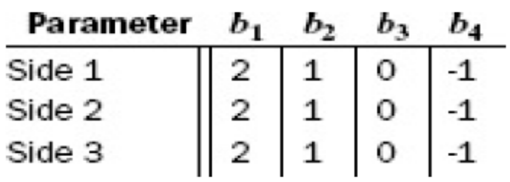
Question 4 [5 MARKS]

Given the triang() function studied in lectures that takes the lengths of the three sides of a triangle as input parameters, the triang()'s inputs (interface-based) are exist in the following table



Partition	b_1	b_2	b_3	b_4
q_1 = "Length of Side 1"	*greater than 1*	*equal to 1*	*equal to 0*	*less than 0*
q_2 = "Length of Side 2"	*greater than 1*	*equal to 1*	*equal to 0*	*less than 0*
q_3 = "Length of Side 3"	*greater than 1*	*equal to 1*	*equal to 0*	*less than 0*

The Possible values for blocks in the above partitioning are listed in the table below.



Parameter	b_1	b_2	b_3	b_4
Side 1	2	1	0	-1
Side 2	2	1	0	-1
Side 3	2	1	0	-1

(a) If you use the Pair-wise Criterion (PWC), how many test needed to satisfy this method.
(b) Write down all the tests that satisfy this Criterion.

Solution:

a) 16 (1 marks)

b) The tests are: (4 marks)

$$
\begin{array}{l}
\{(2,2,2),\\
(2,1,1),\\
(2,0,0),\\
(2,-1,-1),\\
(1,2,1),\\
(1,1,2),\\
(1,0,-1),\\
(1,-1,0),\\
(0,2,0),\\
(0,1,-1),\\
(0,0,2),\\
(0,-1,1),\\
(-1,2,-1),\\
(-1,1,0),\\
(-1,0,1),\\
(-1,-1,2)\}
\end{array}
$$

Question 5 [JUNIT][6 MARKS]

For the below code snippet:

(a) Write a JUnit Test class which tests all the functions in the StringUtils class.

(b) Mention at least **3 different test cases** for each function.

```
public class StringUtils {

    public int getLastIndexOf(String str, char x) {
        for (int i = str.length(); i >= 0; i--) {
            if (str.charAt(i) == x) {
                return i;
            }
        }
        return -1;
    }

   public String toUpperCase(String str) {
        if (!str.isEmpty())
            return str.toUpperCase();
        return null;
   }

   public boolean isPalindrome(String str) {
        StringBuilder plain = new StringBuilder(str);
        StringBuilder reverse = plain.reverse();
        return (reverse.toString()).equals(str);
   }
}
```

(a)

```
public class StringUtilsTest { (0.25 mark)

StringUtils stringUtils; (0.25 mark for declaration)

@Before
public void init() {
   stringUtils = new StringUtils(); (0.25 mark for initialization)
}

@Test
public void testCountLowerCaseLetters() { (0.5 mark for annotation + function header)
   assertEquals(4, stringUtils.countLowerCaseLetters("AaBbCcDd")); (0.5 mark)
}

@Test
public void testToUpperCase() { (0.5 mark for annotation + function header)
   assertTrue("HELLO".equals(stringUtils.toUpperCase("Hello"))); (0.5 mark)
}
```

```
    @Test
    public void testIsPalindrome() { (0.5 mark for annotation + function header)
        assertTrue(stringUtils.isPalindrome("racecar")); (0.5 mark)
    }
}
```

Note: The student may use any type of assert functions such as assertTrue(), assertFalse() or assertEquals() … etc. All of them are correct.

(b) Student should write a total of **9** test cases either in the code of the JUnit functions or mention as text the different **inputs** and **expected outputs** (0.25 mark for each test case = 0.25 * 9 = 2.25 marks).

Note: Test Cases for each function should be **different,** for example toUpperCase() can have these test inputs:

a. All letters are lower case
b. All letters are upper case
c. All letters are a mix between lower case and upper case
d. String is empty
e. String has spaces or special characters … etc.