United International University
Department of Computer Science and Engineering

CSE 4495: Software Testing and Quality Assurance
Final Examination : Summer 2022
Total Marks: 40 Time: 2 hours

Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules.

Answer all the questions. Numbers to the right of the questions denote their marks.

1 Unit Testing and Build Scripts(18 marks)

1. (a) Consider the following Java method which takes a student's class performance, mid-term examination and final examination marks as parameter. The method then calculates and returns the student's cumulative letter grade.

String evaluateGrade(double ct, double mid, double final) returns totalGrade;

Here are some requirements regarding how this method should operate- [3 × 4 = 12]

- Marks of each section (ct, mid, final) must be non negative. If any input parameter is less than zero the method throws *InvalidSectionScore* exception.
- The sum of all three sections cannot be greater than 90, in which cases the method throws *ScoreOverflow* exception
- Some test cases are listed on the following table -

Case No.	Input	Expected Output
1	(10, 16, 20)	**D**
2	(20, 30, 40)	**A**
3	(-30, 26, 35)	Exception[InvalidSectionScore]
4	(20, 36, 39)	Exception[ScoreOverflow]

Now devise executable test cases for all of the above specifications for this method in the *JUnit* notation.

(b) Derive the output when the following test suite is executed. [4]

```
int count = 0;
@BeoreEach
public void init(){
    Sytem.out.println("Initializing");
}
@BeoreAll
public void setup(){
    System.Out.println("Setting Up");
}
@AfterEach
public void setup(){
    count++
    System.Out.println("Done with test #" + count);
}
@Test
public void testMethod_normal() {
// Setup
    Student s = new Student();
    s.setName("MrSQA");
    s.setID("1100927001290");
    s.setGPA("3.55")
// Test Steps
    try{
        assertThat(s.getName(), both(containsString("Mr")).and(containsString("SQA"));
        System.out.Println("Name Test Successful");
         assertAll("student",
            () -> assertEquals("1100927001290", s.getID()),
            () -> assertEquals("3.75", s.getGPA()));
```

```
            System.out.Println("Id and cgpa test successful");
        }catch(Exception e){
            fail("failed after the previous step");
        }
    }
```

(c) What are stubs in test scaffolding? In which cases a tester might use a stub? What is a popular Java based library for creating stub or mock objects. [2]

2 Model Based Testing (12 marks)

2. (a) Dracula wants to keep his valuables in a safe that's hard to find. So, to reveal the lock to the safe, Dracula must remove a strategic candle from its holder. This will reveal the lock only if the door is closed.Once Dracula can see the lock, he can insert his key to open the safe. For extra safety, the safe can only be opened if he replaces the candle first. If someone attempts to open the safe without replacing the candle, a monster is unleashed. You must design a state machine for the controller of a secret panel in Dracula's castle. [3]

(b) Consider the following classes in a simple video streaming service named *TenFlix*. After signing up a user can purchase a package of their choice and enjoy the shows. All the methods of the class *TenflixUser* are described.

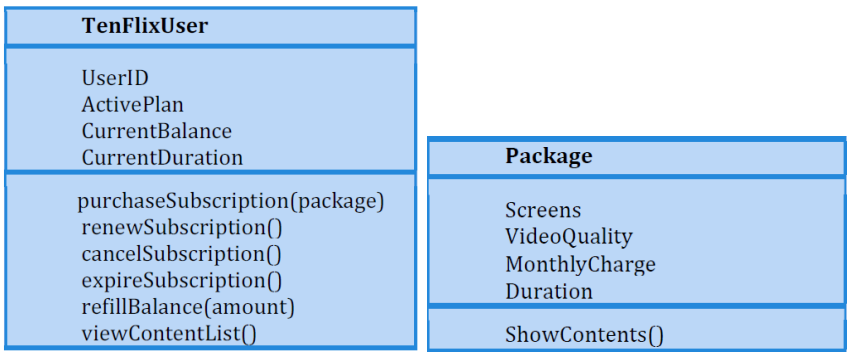

Figure 1: Class diagrams for Ques-3(b)

- ***purchaseSubscription(package)*** – This method can be called only when there are no active plans. Invoking this deducts the package's **MonthlyCharge** from user's **CurrentBalance** and sets the user's current **ActivePlan** to the corresponding package. Also adds the package's **Duration** to the User's **currentDuration**.
- ***renewSubscription()*** – Can be invoked only when the user has an **ActivePlan**. It can be called as many times as needed each time deducing the package's **MonthlyCharge** from user's **CurrentBalance** and adding the package's **Duration** to the User's **currentDuration**.
- ***cancelSubscription()*** – Resets the existing **ActivePlan** to null and **currentDuration** to zero. No refunds are given if user cancels with **currentDuration** remaining.
- ***refillBalance(amount)*** – Can be invoked from any state. Adds the corresponding amount to User's **CurrentBalance**.
- ***expireSubscription()*** – Called when the packages' duration expires ,i.e. user's **currentDuration** is equal to zero.
- ***viewContentList()*** – Displays the list of content that is accessible from this package. Can be invoked if the user has an **ActivePlan**.

Now, identify the states and design a Finite State Model for the *TenflixUser* class with all the relevant labeled transitions. [You don't need to consider viewContentList() as an event]. [5]

(c) Briefly describe what are **Transition coverage** and **Single-transition path coverage metrics**, also describe how these metrics are calculated for a test suite. [2]

(d) Write a test suite (with two or more test cases) that achieves transition coverage for the finite state model you have created in Q-2(b). [2]

3 Finite State Verification (10 marks)

3. (a) Consider a simple microwave controller modeled as a finite state machine using the following state variables:

- Door: Open, Closed – sensor input indicating state of the door
- Button: None, Start, Stop – button press (assumes at most one at a time)
- Timer: 0...999 – (remaining) seconds to cook
- Cooking: Boolean – state of the heating element

Formulate the following informal requirements in **CTL**: [1.5+1.5+1.5 = 4.5]

i. The microwave shall never cook when the door is open.

ii. The microwave shall cook only as long as there is some remaining cook time.

iii. If the stop button is pressed when the microwave is not cooking, the remaining cook time shall be cleared(*Timer* set to zero).

(b) Consider a finite state model of a traffic-light controller, with a pedestrian crossing light and a button to request right-of-way to cross the road. State variables:

- traffic-light: RED, YELLOW, GREEN
- pedestrian-light: WAIT, WALK, FLASH
- button: RESET, SET

Here are some informal requirements for this model :

- The pedestrian light cannot indicate **WALK** when the traffic light is not **RED** or **YELLOW**.
- If the light is **GREEN**, and the button is **SET**, then eventually, the light will turn **RED**.

Based on this model answer the following questions: [2+2+1.5 = 5.5]

i. What is a safety property? From the aforementioned requirements which one is a safety property and why? Express this requirement using **LTL** expression.

ii. What is a liveness property? From the aforementioned requirements which one is a liveness property and why? Also translate this requirement into **LTL**.

iii. *'A trap property is when you write a normal property that is expected to hold, then you negate it (saying that the property will NOT be true).The verification framework will then produce a counter-example indicating that the property actually can be met.'*- now following this definition generate a trap property using the the liveness property you created above .