

**MSc in Computer Science**  
**MSc in Computer Science with Data Analytics**  
**MSc in Computer Science with Cyber Security**  
**MSc in Computer Science with Artificial Intelligence**

DEPARTMENT OF COMPUTER SCIENCE

**SOFTWARE ENGINEERING**

**Time restricted exam (SAIFF)**

Time allowed: 120 (one hundred and twenty) minutes plus an additional 30 (thirty) minutes for file uploads.

The exam requires you to upload some of your answers. Hand written answers are acceptable. Alternatively, you may also use a software tool of your choice. Your uploads must be in either a PDF, a JPG or a PNG format.

The exam paper will total up to 100 marks, and will cover the following sections:

- Section A: Object constraint language
- Section B: Use case and class modelling
- Section C: Interaction and state Modelling
- Section D: Software design, testing and management

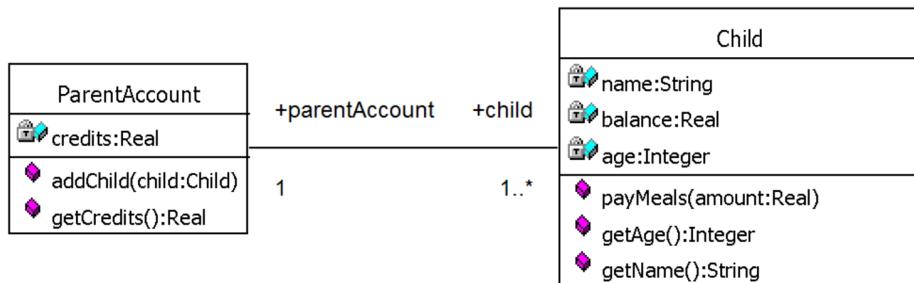
**Answer all the questions**

Please read the following instructions before attempting the exam:

Instructions for the Software Engineering SAIFF exam.pdf 

## Section A Object constraint language [20 marks]

*Parentpay* is a trusted cashless system for school meal payments. Each parent account holds information on the credits available. A number of children can be added to a parent account. The name and age for each child are stored alongside the credit balance for the child account. A child is able to pay for the meals by using the credits from his/her parent account. The following diagram gives the skeleton of a UML class diagram describing the parent account and associated children. In the class diagram, all attributes are *private* and all operations are *public*. Write down the OCL expressions that represent the following requirements:



[Alt text: This is a class diagram with two classes: ParentAccount and Child.

The ParentAccount class has a private attribute: credits:Real, and two public methods: addChild(child:Child), and getCredits():Real.

The Child class has three private attributes: name:String, balance:Real and age:Integer. The Child class also has three public methods: payMeals(amount:Real), getAge():Integer and getName():String.

There is a two-way association line between the ParentAccount class and the Child class with +parentAccount and 1 at the ParentAccount end, and +child and 1..\* at the Child end.]

1. [6 marks] There are three pre-requisites for a parent account: i) its available credits cannot be below zero, ii) there must be at least one child associated with the parent account, and iii) all children associated with the parent account should be no more than 18 years of age.
2. [7 marks] The operation *addChild()* is used to add a child to a parent account. It has three pre-requisites: i) the input object should be of type *Child*, ii) the child name should contain less than 30 characters, and iii) the child is not currently associated with the parent account. The post-requisite is to successfully add the child to the parent account.
3. [7 marks] The operation *payMeals()* is used by a child to pay for the meals by using the parent account credits. It has two pre-requisites that the payment amount should be greater than zero, and the payment amount should be no more than the available credits in the parent account. It has two post-requisites that the amount is added to the balance of the child account, and the amount is reduced from the available credits of the parent account.

## **Section B Use case and class modelling [35 marks]**

You are required to develop a software system for a private Medical Centre at York (MCY) to manage their appointments.

MCY employs many staff members including doctors and administrators. When a new staff joins, an administrator records details of their ID, name and job title. A doctor's speciality and an administrator's skills are also recorded.

A patient is required to register online by themselves prior to making their first appointment. Upon registration, the details of every patient including name, date of birth, personal email and address are recorded.

For registered patients, they can book appointments via phone or in person. One of the administrators will book the appointment for the patient. An appointment ID together with the date and time for the appointment are recorded. A patient may book many appointments. Each appointment is with a doctor, and a doctor can be scheduled for many appointments.

After seeing a patient, the doctor needs to sign off the appointment in the system. As part of the signing off process, a bill is auto-generated with the amount and status of each bill (i.e. outstanding or paid in full) recorded by the system. At the end of each working day, newly generated bills are automatically emailed to the patient via an external email system.

MCY currently accepts cash, credit card and cheque payment through one of the administrators. A patient may make multiple payments on a single bill. When a payment is made, the amount that has been paid, the date/time of the payment, and the method of payment are recorded.

4. [10 marks] Draw a use case diagram that covers all functionalities of the system solely based on the information given above. You do not need to show the use case that enables a user to log onto the system.
5. [25 marks] Draw a conceptual class diagram solely based on the information given above. Your class diagram should show entity classes, attributes, associations and multiplicities. You don't need to show boundary, control classes, operations and data types on your class diagram

## Section C Interaction and state modelling [20 marks]

6. [8 marks] Given the preliminary state transition table for a microwave below, draw a UML state diagram representing states, events and transitions for the microwave, assuming that *Unplugged* is both the initial and the final pseudostate. You also need to show the actions including the entry and internal actions in the diagram.

State	Entry action	Internal action	Event	Action	Next State
Unplugged			switchOn()		Awaiting
Awaiting	beep()	displayTime()	placeItem()		Item placed
			switchOff()		Unplugged
Item placed			setTimer()		Ready to cook
			removeItem()		Awaiting
Ready to cook			start()	startTimer()	Cooking
			cancel()		Item placed
Cooking		updateTimer()	[time expired]		Item placed
			cancel()	stopTimer()	Item placed

7. [12 marks] Read the Java code below, produce a sequence diagram for the *book()* operation in the *MyTravelAgent* class. You are not required to show returning message and branching in the diagram.

```

public class MyTravelAgent {
    Hotel hotels[]; //list of hotels that the agent can access

    public void book(Holiday h) {
        Hotel hotel=search (h);
        hotel.reserve(this);
    }

    //Search for the appropriate hotel
    public Hotel search (Holiday h) {
        int i;
        for (i=0; i<hotels.length; i++) {
            if (hotels[i].match(h)) break;
        }
        return hotels[i];
    }

    public void provideDetails() {};
}

class Hotel{
    String location;
    int unitPrice;

    public boolean match (Holiday h) {
        if(h.getDestination().equals(location) &
           h.getDailyBudget()<=unitPrice)
        return true;
        else return false;
    }

    public void reserve(MyTravelAgent agent) {
        agent.provideDetails();
    }
}

class Holiday{
    private int days;
    private int budget;
    private String destination;

    public String getDestination() {
        return destination;
    }
    public int getDailyBudget() {
        return budget/days;
    }
}

```

## Section D Software design, testing and management [25 marks]

8. [3 marks] with no more than 50 words, explain why the peer-to-peer architecture is more difficult to maintain than the client-server architecture.
9. [2 marks] With no more than 30 words, explain why quality assurance team should be independent from the software development group?
10. [5 marks] Given the decision point at the following code snippet: if ( $A < 100 \ \& \ B > 200$ ), design a set of tests that achieve the Modified Condition/Decision Coverage (MC/DC). Please write the set of tests in the form of

T1 = ( $A=a_1, B=b_1$ ), outcome= $o_1$   
T2 = ( $A=a_2, B=b_2$ ), outcome= $o_2$

...

where  $a$  and  $b$  are your input data and  $o$  is the expected outcome of the decision point.

11. [12 marks] This is a two-part question: Part 1.

Read the Java code below and draw a control flow graph to represent the *printNumbersAfter()* method in the *NumberAfter* class. You are required to use a rectangle to represent sequence of uninterrupted program statements, a diamond to represent a decision, and a circle to represent a junction.

```
1 import java.util.*;
2 public class NumbersAfter {
3     int [] a= {6, 8, 9, 4, 5, 7, 2};
4     /*print the elements including and after the target,
5      * print empty array if the target is not in the array.
6      */
7     public void printNumbersAfter(int target) {
8         ArrayList result=new ArrayList();
9         int i=0;
10        boolean copy=false;
11        while(i<a.length){
12            if (a[i]==target){
13                copy=true;
14            }
15            if (copy) {
16                result.add(a[i]);
17            }
18            i++;
19        }
20        System.out.println(result);
21    } //end of method
22 } //end of class
```

12. [3 marks] Part 2. Write a minimal set of tests that can achieve the full branch coverage for the *printNumbersAfter()* method. Please write the set of tests in the form of

T1 = ( $target=x_1$ ), output= $o_1$   
T2 = ( $target=x_2$ ), output= $o_2$

...

where  $x$  is your input data and  $o$  is the expected terminal output from the search method.

## Model Answers

### Section A Object constraint language [20 marks]

#### Question 1 Solution [6 marks]

context ParentAccount	[1 mark]
inv: credits>=0	[1 mark]
inv: self.child->size()>=1	[2 marks]
inv: self.child->forAll(getAge()<=18)	[2 marks]

#### Question 2 Solution [7 marks]

context ParentAccount::addChild(child:Child)	[1 mark]
pre: child.ocllsTypeOf(Child)	[2 marks]
pre: child.getName().size()<30	[2 marks]
pre: self.child->excludes(child)	[1 mark]
post: self.child->includes(child) or self.child=self.child@pre->including(child)	[1 mark]

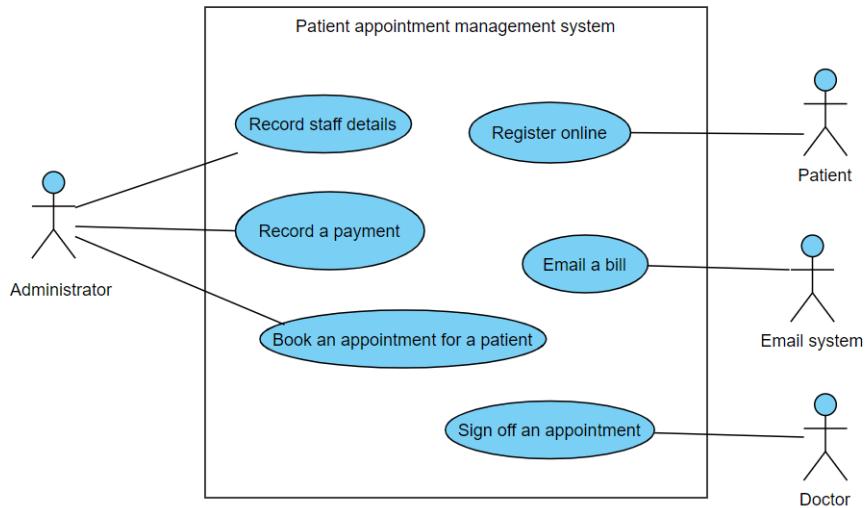
#### Question 3 Solution [7 marks]

context Child::payMeals(amount: Real)	[1 mark]
pre: amount>0	[1 mark]
pre: amount<=parentAccount.getCredits()	[1 mark]
post: balance=balance@pre+amount	[2 marks]
post: parentAccount.getCredits()=parentAccount@pre.getCredits()-amount	[2 marks]

Flexible with different expressions achieving the same effect. Partial marks can be allocated as appropriate.

## Section B Use case and class modelling [35 marks]

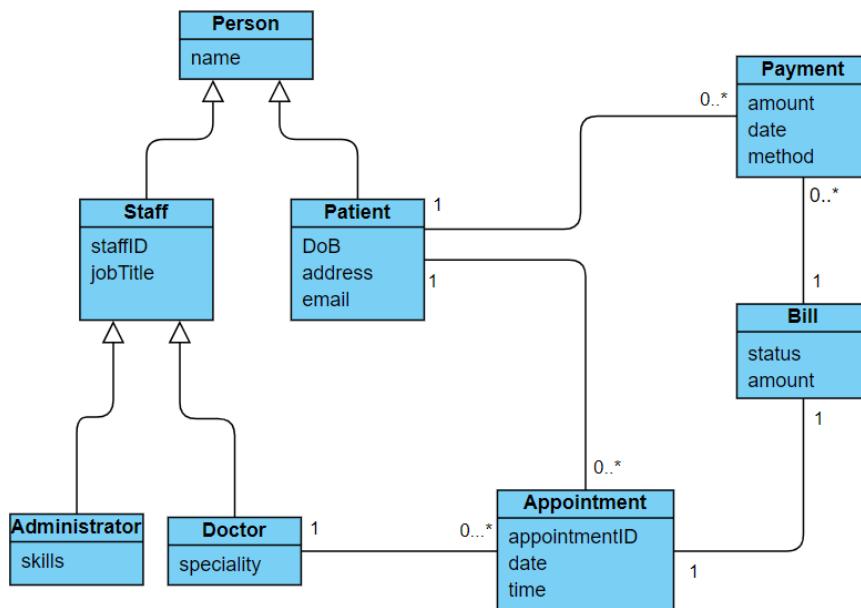
### Question 4 Solution [10 marks]



4 marks for the correct identification of actors, 1 mark each.

6 marks for the correct identification of use cases, 1 mark each. Use case names may differ slightly but the semantics should be the same.

### Question 5 Solution [25 marks]



8 marks for the correct identification of classes, 1 mark each.

8 marks for the correct identification and allocation of the attributes, 1 mark for the attributes of each class.

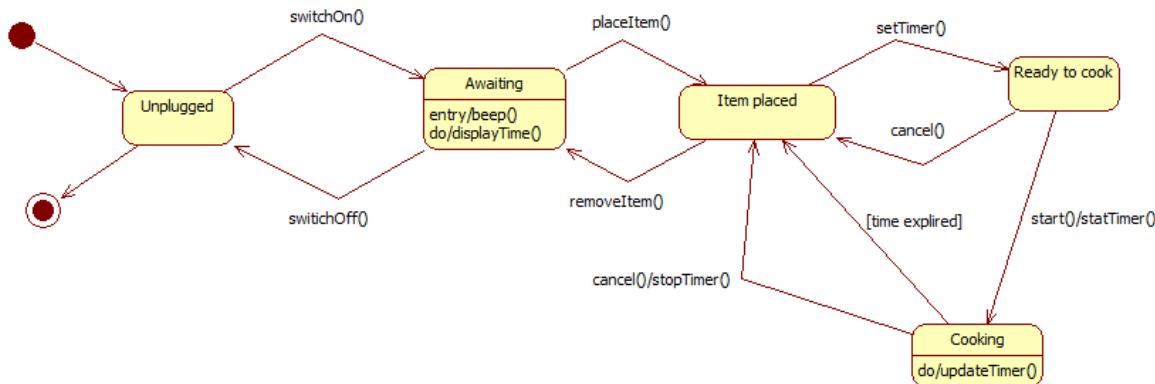
5 marks for the correct use of normal associations with the correct labelling of multiplicities, 1 mark each.

4 marks for the correct use of generalisations: 1 mark each.

Element names might differ slightly as long as the semantics are correct.

## **Section C Interaction and State modelling [20 marks]**

### Question 6 Solution [8 marks]:

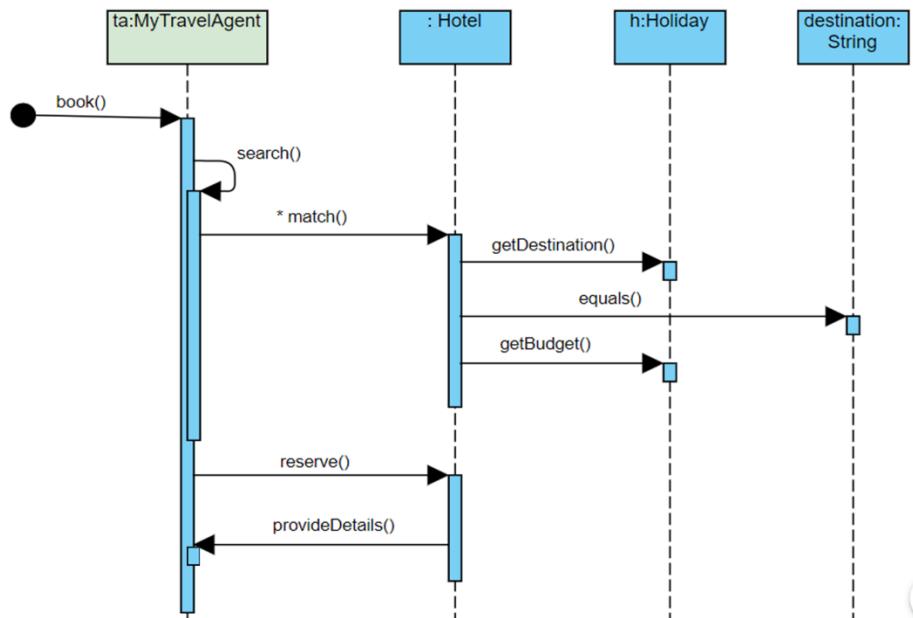


2 marks for the Awaiting and Cooking state with correct entry and internal actions, 1 mark each.

Half a mark for each of the following 12 elements:

- The remaining states: 3
  - Transitions with correct event/action labels: 9

### Question 7 Solution [12 marks]:



Message could optionally show parameters and iteration symbol \*.

4 marks for the correct labelling of participating objects, 1 mark each,

8 marks for the correct labelling of the messages, i.e. correct sender, receiver, name, 1 mark each. Be flexible with the activation boxes.

## **Section D Software design, testing and management [25 marks]**

### Question 8 Solution [3 marks]:

With client-server architecture, the client knows the server, while for the peer to peer architecture, the peers know each other [1 mark]. Peer to peer is more difficult to maintain because of the increased coupling due to extra dependencies [2 marks].

### Question 9 Solution [2 marks]

QA people are independent from the software development group so that they can make objective view of the quality of the software.

### Question 10 Solution [5 marks]

A<100	B>200	Outcome
True	True	True
True	False	False
False	True	False
False	False	False

The table is not part of the answer, it is for the makers' reference only.

Exact three tests (with yellow shade) are needed, for example:

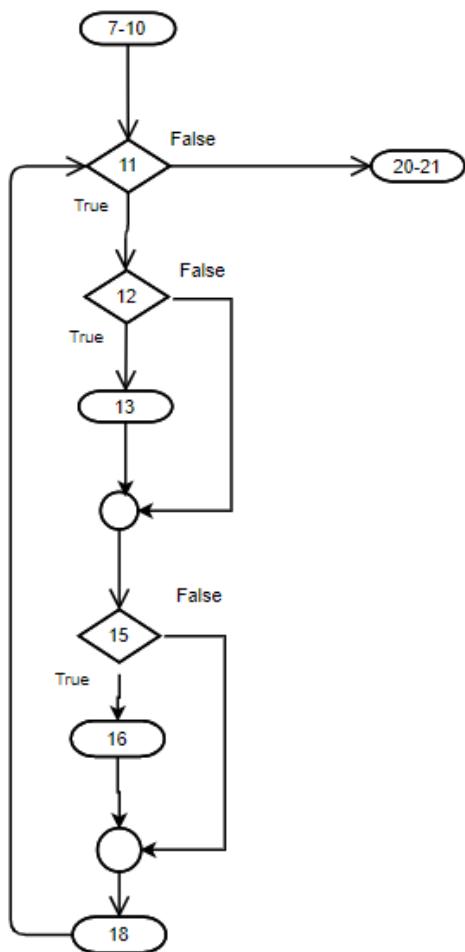
$$T1 = (A=99, B=201), \text{ outcome=True}$$

$$T2 = (A=99, B=199), \text{ outcome=False}$$

$$T3 = (A=101, B=201), \text{ outcome=False}$$

The exact values for A and B are flexible of course as long as the truth value are correct. Partial marks may be awarded for the correct number of tests but with minor errors e.g. missing or wrong outcome.

Question 11 Solution [12 marks]:



Types of arrows don't really matter, the key is the correct control flow structure,

[6 marks] 2 marks for each of the following branches:

- 12 -> joint
- 15 -> joint
- 18->11

[4 marks] 1 mark for each of the following branches:

- 11->12
- 11->20
- 12->13->joint
- 15->16->joint

[2 marks] 2 marks for the presentation of the diagram e.g. the correct use of the different shapes and labelling of truth values etc.

[Question 12 Solution \[3 marks\]:](#)

Need exact 1 test with any number from the list apart from the first number,  
effectively any number from the sub list {8, 9, 4, 5, 7, 2}

e.g. T1=(target=8), output=[8, 9, 4, 5, 7, 2] or  
T1=(target=2), output=[2]

Answers with more than 1 test will receive 0. Partial marks may be allocated for one test but with wrong outputs.