


National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Software Design & Analysis	Course Code:	CS3004
	Program:	BS (CS)	Semester:	Spring 2022
	Duration:	Three hours	Total Marks:	50
	Paper Date:	21-Jun-2022	Weight:	
	Section:	(All)	Page(s):	6
	Exam Type:	Final exam		

Student : Name: _____ Roll No. _____ Sec. _____

Write your answers on this question paper; do not attach any additional sheet.

Question 1 (5+5 marks) [CLO 2]

a) Write name of a design principle in front of each of the following statement:

- If a module is going to replace another module then it needs to provide all the features of that module
(Liskov's substitution)
- A module should focus on a single aspect **(Single responsibility)**
- Principle of least privilege **(Interface segregation)**
- If a module can be replaced in future then other modules should not refer it directly **(dependency inversion)**
- A design should be extensible without requiring changes in the existing artifacts **(open closed principle)**

b) Write name of a design pattern in front of each of the following statement:

- A class can have only a single object **(singleton)**
- The child class will create the appropriate object **(factory method)**
- Incorporate a class from a third-party vendor **(adapter)**
- A tree of objects (of different classes) **(composite)**
- Traverse elements of any type of data structure **(iterator)**

Question 2 & 3 (10+10 marks) [CLO 3 & 4]

Consider the following system description:

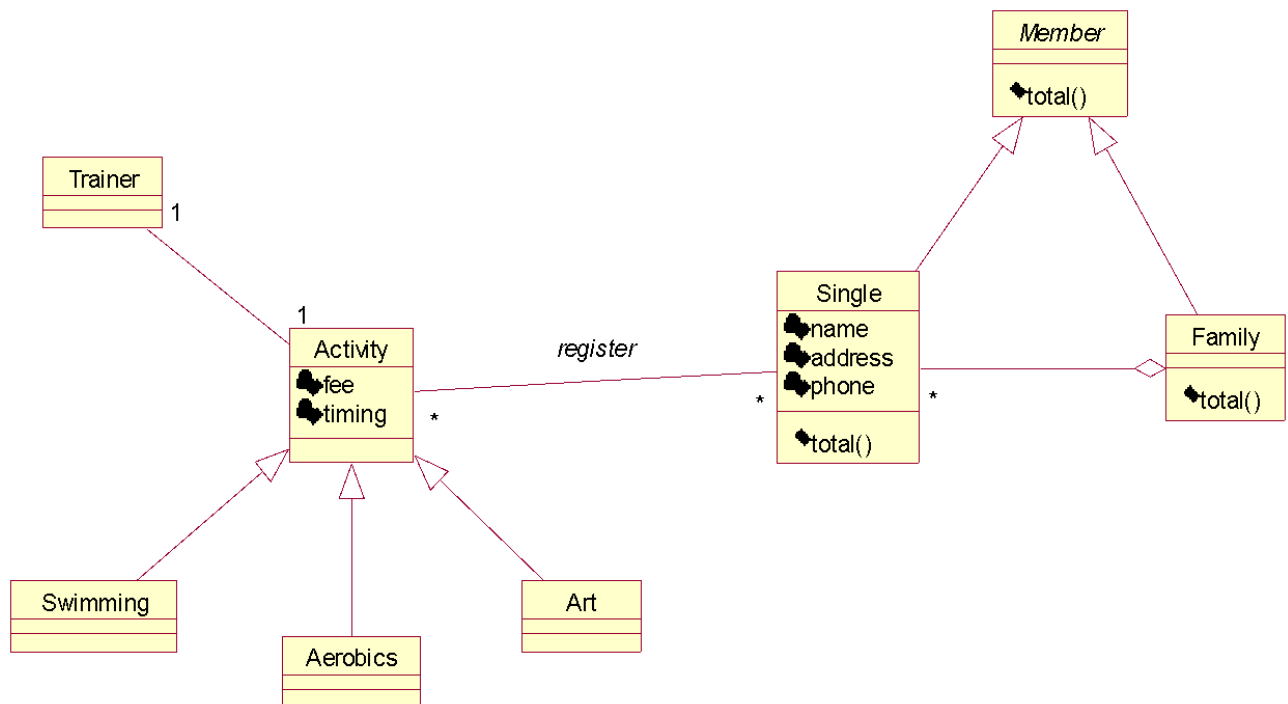
A club offers different activities such as aerobics (exercise), art & craft, and swimming. A member can register in one or more activities. Each activity has a timing, a fee, and a trainer. There are two types of memberships: single or family. A family membership provides 50% discount. For example if fee of an activity is 10,000 per month for a single person, then a family of four shall pay 20,000 only (4 x 5,000).

Q2) Give a class diagram for the above system. Show any important data members. You do not need to show the functions.

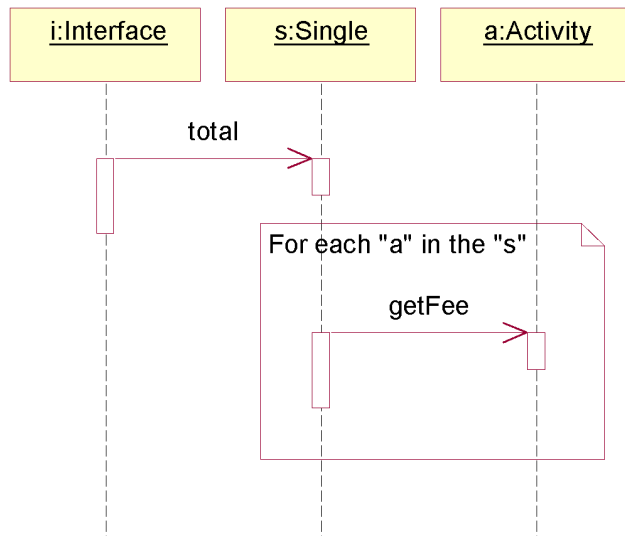
Q3) Answer the following questions:

a) Draw a sequence diagram showing the computation of total fee for a member (single person).

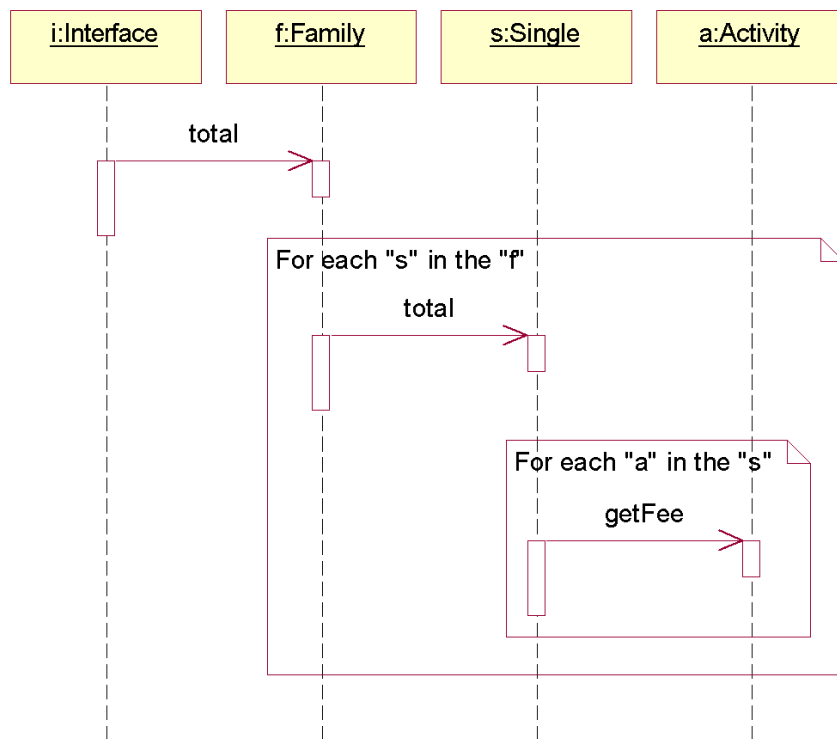
b) Draw a sequence diagram showing the computation of total fee for a family (multiple persons). Note that each individual can register in one or more different activities.



a)

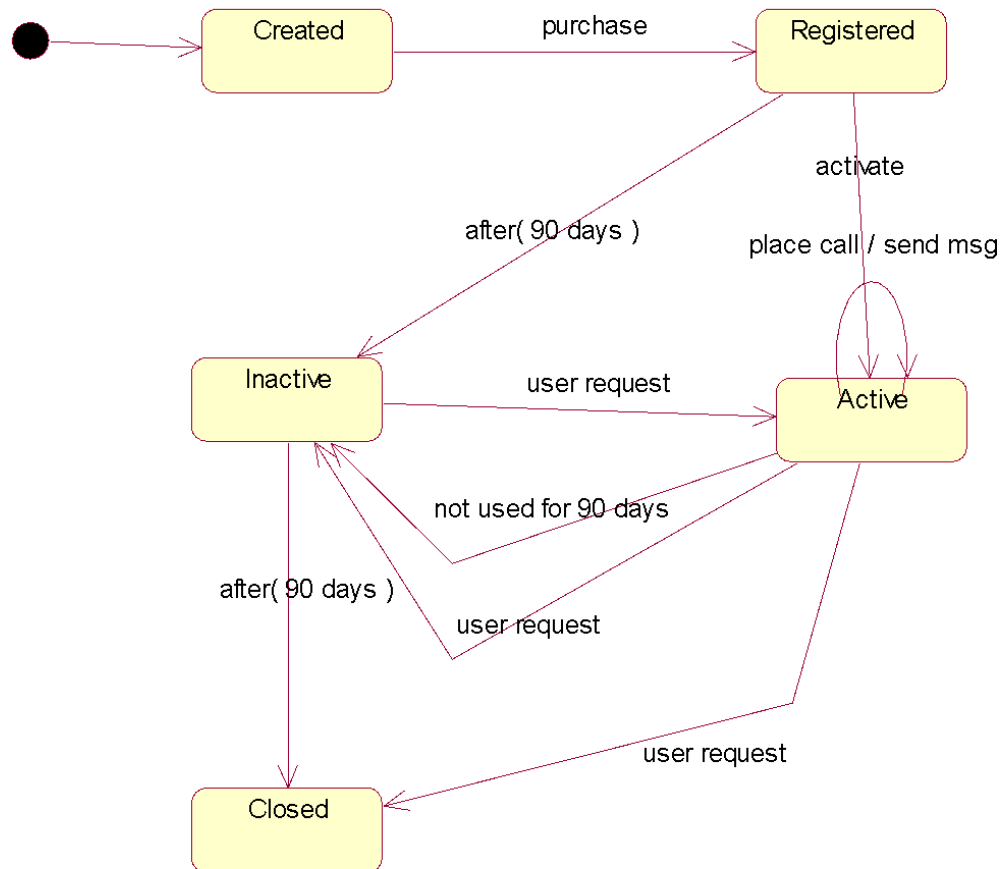


b)



Question 4 (10 marks) [CLO 3 & 4]

You want a software module to be developed that manages mobile SIM cards. The software module will be developed by a vendor to which you have to communicate your business rules. The business rules must capture details of all the states of the SIM card and on what events/actions the state of the SIM card can be changed. Develop a state diagram showing all the states of the SIM card and transitions between them. You must show at least 5 states [Created, Registered, Active, Inactive, Closed] The SIM card can transition between these states based on certain rules that you need to define and capture in the state diagram. The vendor will use this state diagram to correctly implement the module. Consider using events/actions like SIM-purchase, online-activation, not-used-for-90-days, not-used-for-180-days, request for activation, request for block, request for closure etc.



Question 5 (10 marks) [CLO 4 & 5]

Write C++ code for a binary-tree iterator. You can consider the following classes in this context:

```
class Container {
    virtual Iterator* iterator() = 0;
};

class BnTr : public Container {
private:
    Node* root;
    // ...
public:
    Iterator* iterator() {
        return new BnTrIt(this);
    }
    // ...
    friend class BnTrIt;
};

struct Node {
    int data;
    Node* left;
    Node* right;
};

class Iterator {
    Iterator(Container* con);
    virtual bool hasNext() = 0;
    virtual int next() = 0;
};
```

Hint:

```
// level-order traversal
void print(Node* root) {
    Queue que;
    que.add(root);
    while(!que.empty()) {
        Node* nod = que.rem();
        cout << nod->data;
        if (nod->left != null)
            que.add(nod->left);
        if (nod->right != null)
            que.add(nod->right);
    }
}
```

Solution

```
class BnTrIt : public Iterator {
private:
    BnTr* tree;
    Queue que;
public:
    Iterator(Container* con) {
        tree = (BnTr*) con;
        que.add(tree->root);
    }
    virtual bool hasNext() {
        return !que.empty();
    }
    virtual int next() {
        Node* nod = que.rem();
        if (nod->left)
            que.add(nod->left);
        if (nod->right)
            que.add(nod->right);
        return nod->data;
    }
};
```