

CO-301

Software Engineering Lab



Name: Piyush Khanna

Roll No: 2k17/CO/224

Batch: COE-A3

INDEX

[illegible]

PROGRAM -1

AIM :To count the number of lines of code in the given file.

THEORY : The lines of code measures are the most traditional measures used to quantify software complexity.They are simple, easy to count, and very easy to understand.They do not, however, take into account the intelligence content and the layout of the code.

Source lines of code (SLOC), also known as **lines of code (LOC)**, is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.

There are two major types of SLOC measures: physical SLOC and logical SLOC. Specific definitions of these terms vary depending on particular circumstances. The most common definition of physical SLOC is a count of lines in the text of the program's source code including comment lines and, sometimes, blank lines. Logical SLOC attempts to measure the number of executable expressions (such as operators, functions, etc.), but their specific definitions are tied to specific computer languages

Therefore, each approach has its own strong and weak points: physical SLOC is easier to measure, but it is very sensitive to coding style conventions and code formatting, while logical SLOC is less sensitive to these factors yet not so easy to measure.

SOURCE CODE

```
#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

    int count = 0;

    int flag = 1 ;

    string line ;

    ifstream file("ftest.cpp") ;

    while(getline(file,line)) {
```

```

        if(line.length[] == 0) {
continue ;

        }

        if(line[0] == '/' && line[1] == '/') {
continue ;

        }

        if(line[0] == '/' && line[1] == '*') {

            flag = 0 ;
continue ;

        }

        if(line[0] == '*' && line[1] == '/') {

            flag = 1 ;
continue ;

        }

        if(flag=1) {

            count++ ;

        }

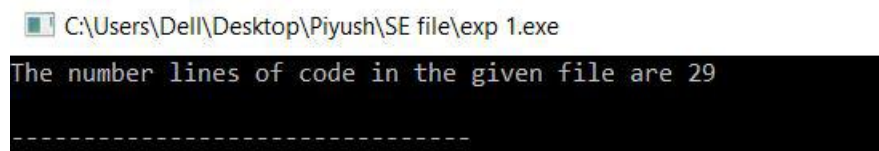
    }

    cout<< "The number lines of code in the given file are " << count <<endl ;

}

```

OUTPUT



```

C:\Users\Dell\Desktop\Piyush\SE file\exp 1.exe
The number lines of code in the given file are 29
-----

```

RESULT : The above program was successfully able to compute the lines of code in the given file excluding blank lines and comments.

Program – 2

Aim : To write a program to implement COCOMO model.

Theory : COCOMO (Constructive Cost Model) is a regression model based on LOC i.e. lines of code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

Effort :- Amount of labour that will be required to complete a task. It is measured in person month units.

Schedule :- Simply means the amount of time required for the completion of the job, which is of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Formula Used: $a(KLOC)^b$

Algorithm :

Step 1: Provide value of parameters 'a' and 'b' for different models.

Step 2: Take LOC as input and determine the type of model.

Step 3: According to the model, calculate the required estimates as per the formula.

Step 4: Output the result.

Source Code :

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<math.h>
```

```
int fround(float x)
```

```
{
```

```
    int a;
```

```
    x=x+0.5;
```

```
    a=x;
```

```
    return(a);
```

```
}
```

```
void main()
```

```

{
    float effort,time,staff,productivity;

    float table[3][4]={2.4,1.05,2.5,0.38,3.0,1.12,2.5,0.35,3.6,1.20,2.5,0.32};

    int size,model;

    char mode[][15]={"Organic","Semi-Detached","Embedded"};

    // clrscr();

    printf("\nEnter size of project (in KLOC) : ");

    scanf("%d",&size);

    if(size>=2 && size<=50)

        model=0;    //organic

    else if(size>50 && size<=300)

        model=1;    //semi-detached

    else if(size>300)

        model=2;    //embedded

    printf("\nThe mode is %s\n",mode[model]);

    effort=table[model][0]*pow(size,table[model][1]);

    time=table[model][2]*pow(effort,table[model][3]);

    staff=effort/time;

    productivity=size/effort;

    printf("\nEffort = %f Person-Month",effort);

    printf("\n\nDevelopment Time = %f Months",time);

    printf("\n\nAverage Staff Required = %d Persons",fround(staff));

    printf("\n\nProductivity = %f KLOC/Person-Month",productivity);

    getch();
}

```

Output :

 C:\Users\Dell\Desktop\Piyush\SE file\exp2.exe

```
Enter size of project (in KLOC) : 1000  
The mode is Embedded  
Effort = 14331.862305 Person-Month  
Development Time = 53.451122 Months  
Average Staff Required = 268 Persons  
Productivity = 0.069775 KLOC/Person-Month
```

Result : We are able to estimate different parameters by providing LOC as input for a particular project.

Program – 3

Aim : To write a program to calculate the function point.

Theory : FPA(Function Point Analysis) gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer. Function point (FP) is an element of software development which helps to approximate the cost of development early in process. It may measure functionality from user's point of view. It is a method or set of rules of functional size measurement. It assesses the functionality delivered to its users, based on the user's external view of the requirements. It measures the logical view of an application not the physically implemented view or the internal technical view.

Algorithm :

Step 1: Take as input the amount of each functional unit.

Step 2: Take as input the significance of all the units.

Step 3: Based on the above inputs, calculate the required estimates as per the formula.

Step 4: Output the result.

Source Code :

```
#include<stdio.h>

#include<conio.h>

int fround(float x)

{

    int a;

    x=x+0.5;

    a=x;

    return(a);

}

void main()

{

    int weights[5][3]={3,4,6,4,5,6,3,4,6,7,10,15,5,7,10};

    int UFP=0,F=0,rating,i,j;
```



```

char func_units[][30]={"External Inputs","External Outputs","External Enquiries","Internal Logical
Files","External Interface Files"};

char complexity[3][10]={"low","average","high"};

int input[5][3];

float FP,CAF;

for(i=0;i<5;i++)

{

    for(j=0;j<3;j++)

    {

        printf("\nEnter number of %s %s : ",complexity[j],func_units[i]);

        scanf("%d",&input[i][j]);}}

//calculating UFP

for(i=0;i<5;i++)

{

    for(j=0;j<3;j++)

    {

        UFP=UFP+(input[i][j]*weights[i][j]);

    } }

printf("\nUnadjusted Function Point(UFP) = %d",UFP);

printf("\nEnter Rating of 14 factors on the scale of 0-5 :\n");

printf("\n 0 - No Influence");

printf("\n 1 - Incidental");

printf("\n 2 - Moderate");

printf("\n 3 - Average");

printf("\n 4 - Significant");

printf("\n 5 - Essential");

printf("\n");

for(i=0;i<14;i++) {

```

```

scanf("%d",&rating);

F=F+rating; }

CAF=0.65+0.01*F;

FP=UFP*CAF;

printf("\nAdjusted Function Point = %f",FP);

printf("\nOR FP = %d",fround(FP));

getch();

}

```

Output :

```

C:\Users\Dell\Desktop\Piyush\SE file\exp3.exe
Enter number of low External Inputs : 4
Enter number of average External Inputs : 5
Enter number of high External Inputs : 7
Enter number of low External Outputs : 2
Enter number of average External Outputs : 2
Enter number of high External Outputs : 8
Enter number of low External Enquiries : 20
Enter number of average External Enquiries : 40
Enter number of high External Enquiries : 30
Enter number of low Internal Logical Files : 100
Enter number of average Internal Logical Files : 300
Enter number of high Internal Logical Files : 1000
Enter number of low External Interface Files : 100
Enter number of average External Interface Files : 140
Enter number of high External Interface Files : 200
Unadjusted Function Point(UFP) = 22720
Enter Rating of 14 factors on the scale of 0-5 :

0 - No Influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential
1 1 1 5 2 1 1 3 2 3 2 1 4 0

Adjusted Function Point = 20902.400391
OR FP = 20902

```

Result : We are able to calculate the Function point based on various functional units.

Program 4

Aim: Program to determine if given input is in the acceptable equivalence partitioned class

Theory: Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

Code:

```
#include <iostream>

using namespace std;

class equivalence
{
public:

    equivalence()
    {
        mini=0;
        maxi=100;
        acceptable=false;
    }

    void setEquivalence(int minArg, int maxArg, int acceptableArg)
    {
        mini=minArg;
        maxi=maxArg;
        acceptable=acceptableArg;
    }

    bool isAcceptable (int input)
```

```

{
    if ((input>=mini) && (input<=maxi)&&(acceptable==1)) return 1;
    return 0;
}
private:
    int mini;
    int maxi;
    bool acceptable;

};

int main()
{
    int input;
    bool res=0;

    int noOfEquivalentClasses;
    cin>>noOfEquivalentClasses;

    equivalence arrayOfObj[noOfEquivalentClasses];

    for (int i=0;i<noOfEquivalentClasses;i++)
    {
        int mini,maxi,acceptable;
        cin>>mini>>maxi>>acceptable;
        arrayOfObj[i].setEquivalence(mini,maxi,acceptable);
    }

    cin>>input;

```

```

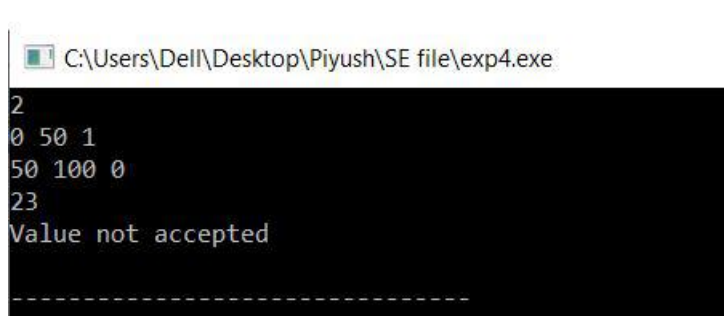
for (int i=0;i<noOfEquivalentClasses;i++)
{
    bool res = arrayOfObj[i].isAcceptable(input);
    if (res==1) break;
}

if (res==1)
    cout<<"Value accepted\n";
else
    cout<<"Value not accepted\n";

return 0;
}

```

Output:



```

C:\Users\Dell\Desktop\Piyush\SE file\exp4.exe
2
0 50 1
50 100 0
23
Value not accepted
-----

```

Conclusion : We determined by writing a program that whether the given input is in the acceptable equivalence partitioned class or not.

Program-5

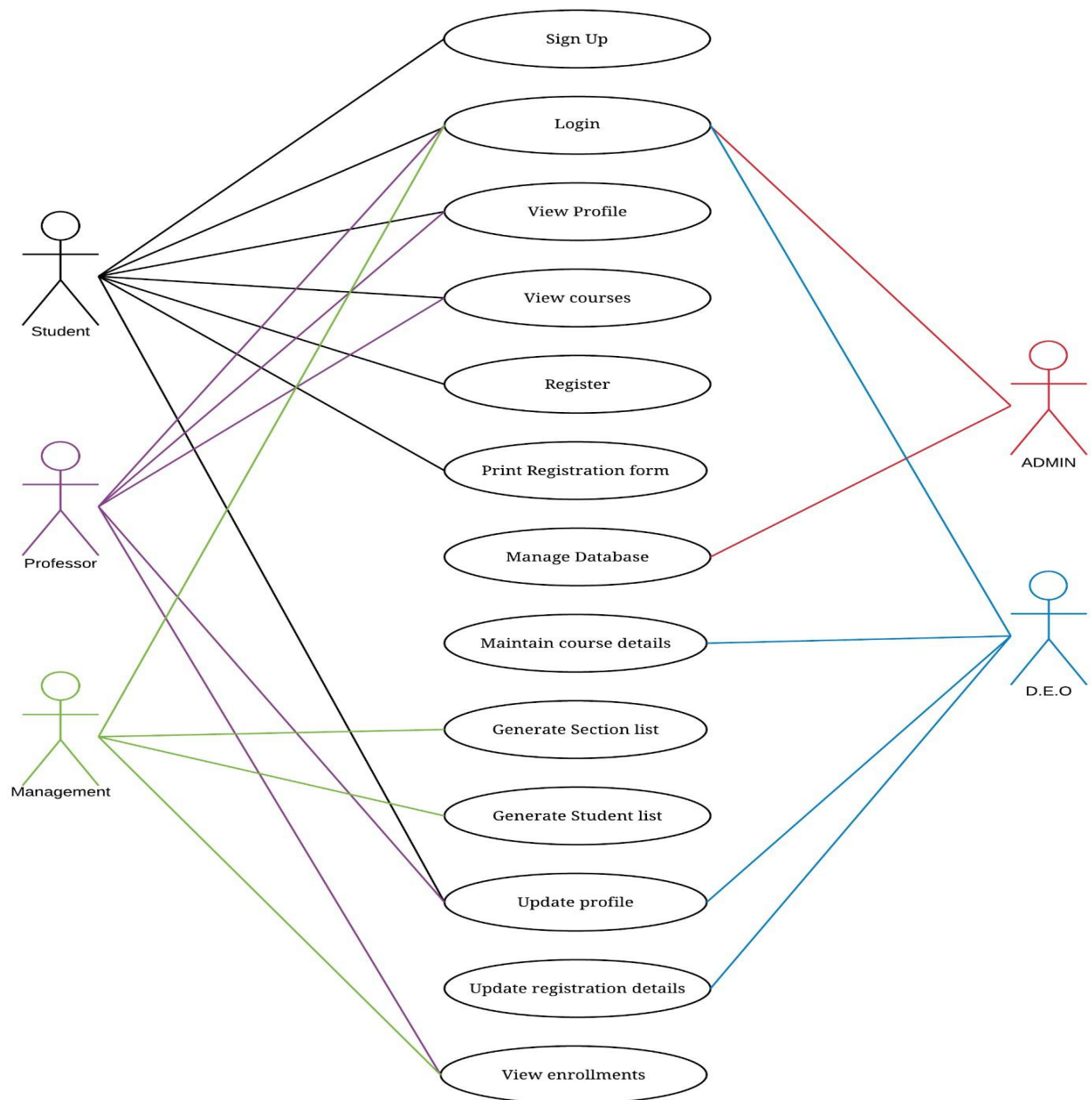
AIM: To make the use case diagram for University Registration Portal and write its use case description

THEORY:

A use case diagram visually captures what happens when an actor¹ interacts with the system. Hence, they capture the functional aspects of the system. The system is shown as a rectangle with the name of the system inside, the actors are shown as stick figures, the use cases² are shown as solid bordered ovals labeled with the name of the case, and relationships are lines or arrows between actors and the use cases.

1. Actor: An actor or external agent, lies outside the system model, but interacts with it in some way. An actor may be a person, machine, or even an information system that is external to the system.
2. Use case: A use case is initiated by the user with a goal in mind and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of the sequence as well as sequences to handle errors and failures.

A use case description describes each use case providing its flow of events, alternative flows, pre, and post-conditions, etc. This is a structured way to write the use case.



Use Case Diagram of University Registration Portal

2. Use Case Descriptions

2.1 Sign Up

2.1.1 Brief Description

This describes how the use registers to use the system.

2.1.2 Actors Involved Student

2.1.3 Pre-Condition

Student needs to meet the eligibility set by the university (new students) or have valid college IDs.

2.1.4 Post-Condition

After successful signing up the students are redirected to their profiles.

In case of failure the student is prompted to re-enter the data/ contact admin.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The student is presented a form to be filled containing basic details and results.
2. The student then sets a Login and Password.
3. The form is validated, and the student is signed up.

2.1.5.2 Alternate Flow

1. Invalid student ID/ Missing Data: student is prompted to correct the ID/ fill in data.

2.1.5 Associate Use Case N/A

2.2 Login

2.1.1 Brief Description

This describes how a use logs into the system.

2.1.2 Actors Involved

Student, Professor, Management, Admin, Data Entry Operator.

2.1.3 Pre-Condition

All Users must have a Login ID and passwords created for them in the system.

2.1.4 Post-Condition

After successfully Logging in, the student and Professors are redirected to their profiles. The management, Admin, and DEO are redirected to the management site.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user is prompted to enter their Login and Passwords.
2. The user then enters their Login and Passwords.
3. The data is the validated and the user logged in.

2.1.5.2 Alternate Flow

1. Invalid student ID/ Missing Data: student is prompted to correct the ID/ fill in data.

2.1.5 Associate Use Case N/A

2.3 View Profile

2.1.1 Brief Description

This Describes how a student or professor accesses their profile

2.1.2 Actors Involved

Student, professor

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is presented their generated profiles with their information on it.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user logs in and is the redirected to their profiles.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.4 View Courses

2.1.1 Brief Description

This describes the how a student or professor can view the list of courses and their details

2.1.2 Actors Involved Student, Professor.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is redirected to the courses list page.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user clicks on the courses menu on their profile window.
2. The user is presented with the list of courses.
3. The user then selects the course of their choice and are redirected to the course details page.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.5 Register

2.1.1 Brief Description

This describes how a student registers for the next semester.

2.1.2 Actors Involved Student

2.1.3 Pre-Condition

The student must have successfully logged into the system.

2.1.4 Post-Condition

After successfully registering for the semester the student is redirected to the confirmation windows showing the courses they've signed up for.

In case of failure the student is prompted to re-enter the data/ contact admin.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The student selected the REGISTER option from their profile.
2. The student is then presented with a list of courses they can take in the semester (Cores and electives)
3. The student selected the minimum number of courses required and additional (if they chose to do so up to a maximum of 6 courses).
4. The student then submits their selections.
5. The selections are validated, and the student is redirected to the confirmation page.

2.1.5.2 Alternate Flow

1. Invalid student ID/ Missing Data: student is prompted to correct the data.
2. Minimum courses not met: student is prompted to select more courses.
3. Maximum courses exceeded: Student is prompted to deselect some courses.

2.1.5 Associate Use Case N/A

2.6 Print Registration form

2.1.1 Brief Description

This Describes how a student prints the generated registration form.

2.1.2 Actors Involved Student.

2.1.3 Pre-Condition

The user must have successfully registered for the semester.

2.1.4 Post-Condition

The user is presented their generated registration form with their information in it.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The student completes the registration successfully.
2. The student is redirected to the confirmation page with the option to get form.
3. The student then selects get form and is presented the registration form.

2.1.5.2 Alternate Flow N/A.

2.1.5 Associate Use Case N/A.

2.7 Manage Database

2.1.1 Brief Description

This Describes how the admin maintains the database.

2.1.2 Actors Involved Admin.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition N/A

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The Admin logs in and is the redirected to the admin window.
2. The admin then executes maintenance tasks like:
 - a. Add/Remove tables
 - b. Alter tables
 - c. Reset system.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.8 Maintain Course Details

2.1.1 Brief Description

This Describes how the D.E.O maintains the course details.

2.1.2 Actors Involved

Data Entry Operator/

2.1.3 Pre-Condition

The DEO must have successfully logged into the system.

2.1.4 Post-Condition N/A/

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The DEO logs in and is the redirected to the Maintenance window.
2. The DEO then executes required tasks like:
 - a. Add course.
 - b. Remove course.
 - c. Alter existing course.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.9 Generate section lists

2.1.1 Brief Description

This Describes how the management can generate sections for courses.

2.1.2 Actors Involved Management.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is presented the generated section list of students per course.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user logs in and is the redirected to their management window.
2. The user then selects a course.
3. The user then selects the option to generate sections defining the maximum number of students per section.
4. Upon completion the user is presented the generated section list.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.10 Generate Student list

2.1.1 Brief Description

This Describes how the management can generate student list for courses/departments.

2.1.2 Actors Involved Management.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is presented the generated list of students per course/department.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user logs in and is the redirected to their management window.
2. The user then selects a course/department.
3. The user then selects the option to generate student list.
4. Upon completion the user is presented the generated student list.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.11 Update Profile

2.1.1 Brief Description

This Describes how a user can update profile details

2.1.2 Actors Involved

Student, Professor, DEO.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user details are updated in the system database.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user selects the update profile option from their window.
2. The user is then redirected to the updation form.
3. The user enters the updated information and submits it.
4. The data is then validated and on success the profile is updated.

2.1.5.2 Alternate Flow

1. Protected data: On updation of fields like name, date of birth, parent names the process fails, and the user is prompted to bring proof of documents to the administration before the updation is completed by the DEO.

2.1.5 Associate Use Case N/A

2.12 Update Registration Details

2.1.1 Brief Description

This Describes how the registration details can be altered after registration period has expired.

2.1.2 Actors Involved Data

Entry Operator.

2.1.3 Pre-Condition

The student requesting updation must have successfully registered for the semester.

2.1.4 Post-Condition

The student registration details are updated, and a new registration form is generated.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The DEO selects the student profile form the student list.
2. The DEO then updates the registration details as follows:
 - a. Add/Remove course.
 - b. Change section.
3. The DEO then submits the updates and new registration form is generated.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.13 View Enrollments

2.1.1 Brief Description

This describes how the professors and management can view the enrollment list (student lists).

2.1.2 Actors Involved

Management, Professors.

2.1.3 Pre-Condition

The user must have successfully logged into the system.

The list must've been generated by the management.

2.1.4 Post-Condition

The user is presented the enrollment list/ student list.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user selects a course.
2. The user then selects the option to view lists this can be:
3. View section list.
4. View Department list.
5. View Course list.

2.1.5.2 Alternate Flow

1. Lists not generated: In case the lists are not generated then the user is presented with the list not generated page, contact admin. The management is presented with the option to generate lists.

2.1.5 Associate Use Case N/A

LEARNINGS-

1. The use case diagram and descriptions detail out all use cases and how the actors interact with the cases to reach the desired goals.
2. We have also visually shown the said relationships.

EXPERIMENT-6

AIM: To make the use case diagram for Banking Management System and write its use case description

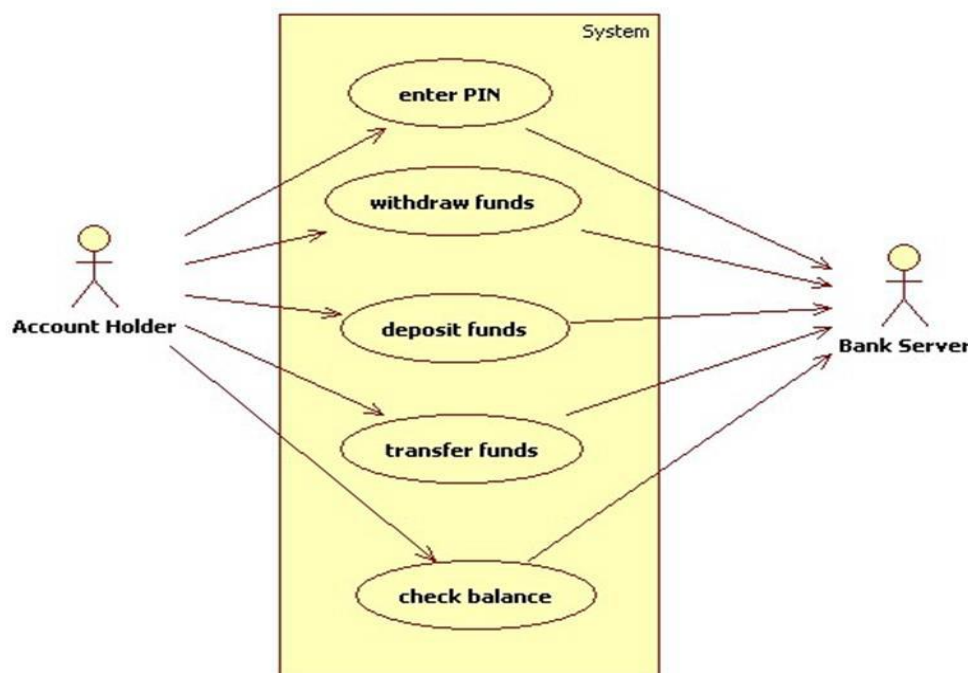
THEORY:

A use case diagram visually captures what happens when an actor¹ interacts with the system. Hence, they capture the functional aspects of the system. The system is shown as a rectangle with the name of the system inside, the actors are shown as stick figures, the use cases² are shown as solid bordered ovals labeled with the name of the case, and relationships are lines or arrows between actors and the use cases.

1. **Actor:** An actor or external agent, lies outside the system model, but interacts with it in some way. An actor may be a person, machine, or even an information system that is external to the system.

2. **Use case:** A use case is initiated by the user with a goal in mind and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of the sequence as well as sequences to handle errors and failures.

A use case description describes each use case providing its flow of events, alternative flows, pre, and post-conditions, etc. This is a structured way to write the use case.



Use Case Diagram for Banking Management System.

2. Use Case Descriptions

2.1 Sign Up

2.1.1 Brief Description

This describes how the customer logs in to use the system.

2.1.2 Actors Involved

Customer, Bank Server

2.1.3 Pre-Condition

Customers must have valid PIN and login credentials to access their accounts.

2.1.4 Post-Condition

After successful signing up the customers are redirected to their profiles.

In case of failure the customer is prompted to re-enter the data/ contact bank server.

2.1.5 Flow of events

2.1.5.1 Basic Flow

4. The customer is presented a form to be filled containing basic details and results.
5. The customer then sets Login credentials and PIN no.
6. The form is validated, and the customer is signed up.

2.1.5.2 Alternate Flow

1. Invalid PIN/Missing Customer data: customer is prompted to correct the ID/ fill in data.

2.1.5 Associate Use Case N/A

2.2 Login (Enter PIN)

2.1.1 Brief Description

This describes how a user logs into the system.

2.1.2 Actors Involved

customer, Bank Server.

2.1.3 Pre-Condition

All Users must have a Login ID and passwords created for them in the system.

2.1.4 Post-Condition

After successfully Logging in, the customer is redirected to their profile. Bank server is redirected to the management site.

2.1.5 Flow of events

2.1.5.1 Basic Flow

4. The user is prompted to enter their Login and Passwords.
5. The user then enters their Login and Passwords.
6. The data is the validated and the user logged in.

2.1.5.2 Alternate Flow

1. Invalid PIN/Missing Customer data: customer is prompted to correct the ID/ fill in data.

2.1.5 Associate Use Case N/A

2.3 View Profile

2.1.1 Brief Description

This Describes how a customer accesses their profile

2.1.2 Actors Involved

Customer

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is presented their generated profiles with their information on it.

2.1.5 Flow of events

2.1.5.1 Basic Flow

1. The user logs in and is the redirected to their profiles.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.4 Check Balance

2.1.1 Brief Description

This describes the how a customer can view the transaction details

2.1.2 Actors Involved

Cutomer, Bank Server

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

The user is redirected to 'Check Balance' page.

2.1.5 Flow of events

2.1.5.1 Basic Flow

4. The user clicks on the 'Transactions' menu on their profile window.
5. The user is presented with the list of previous transactions along with balance..
6. The user then selects the transaction of their choice and are redirected to the transaction details page.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.5 Withdraw Funds

2.1.1 Brief Description

This describes how a customer withdraws funds from his account.

2.1.2 Actors Involved

Cutomer, Bank Server

2.1.3 Pre-Condition

The customer must have successfully logged into the system.

2.1.4 Post-Condition

After successfully entering the amount to withdraw the customer is redirected to the confirmation windows showing the final transaction details.

In case of failure the customer is prompted to re-enter the data/ contact admin.

2.1.5 Flow of events

2.1.5.1 Basic Flow

6. The customer selected the Withdraw Funds option from their profile.
7. The customer is then presented with the basic bank account details.
8. The customer selected the funds to withdraw which has to be less than available balance.
9. The customer then submits the details.
10. The details are validated, and the customer is redirected to the confirmation page.

2.1.5.2 Alternate Flow

4. Invalid customer ID/ Missing Data: customer is prompted to correct the data.
5. Minimum withdraw limit not met: customer is prompted to select more courses.
6. Maximum withdraw limit exceeded: customer is prompted to reenter amount.

2.1.5 Associate Use Case N/A

2.6 Deposit Funds

2.1.1 Brief Description

This Describes how the customer deposits funds.

2.1.2 Actors Involved

Cutomer, Bank Server

2.1.3 Pre-Condition

The user must have successfully logged into the system.

2.1.4 Post-Condition

After successfully entering the amount to withdraw the customer is redirected to the confirmation windows showing the final transaction details. In case of failure the customer is prompted to re-enter the data/ contact admin.

2.1.5 Flow of events

2.1.5.1 Basic Flow

3. The Customer logs in and is the redirected to the deposit window.

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

2.8 Transfer Funds

2.1.1 Brief Description

This Describes how funds are transferred between 2 accounts.

2.1.2 Actors Involved

Customer, Bank Server

2.1.3 Pre-Condition

The Customer must have successfully logged into the system.

2.1.4 Post-Condition N/A/

2.1.5 Flow of events

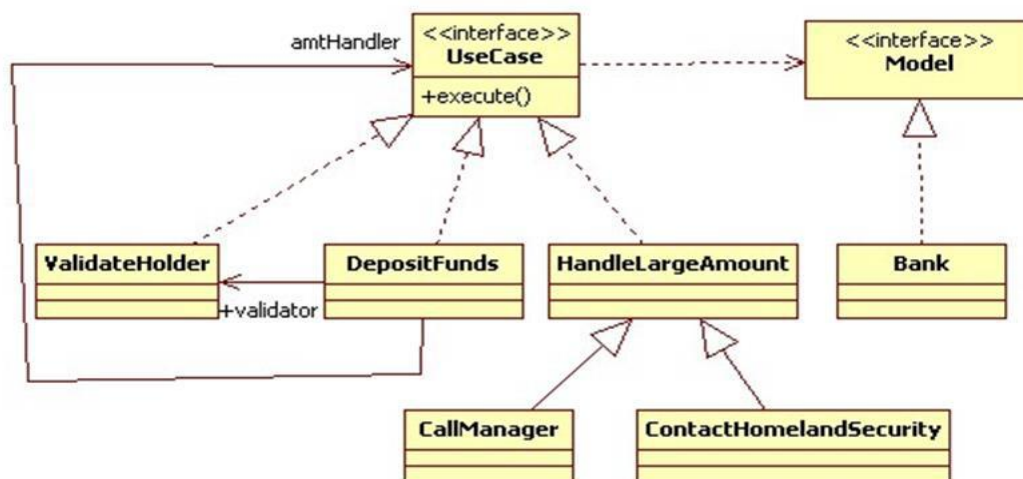
2.1.5.1 Basic Flow

3. The Customer logs in and is the redirected to the Transaction window.
4. The bank server then executes fund transfer tasks like:
 - a. Check if its a valid transaction.
 - b. Check user authenticity.
 - a. Update System. (Both the customer as well as user to whom funds are being transferred to.)

2.1.5.2 Alternate Flow N/A

2.1.5 Associate Use Case N/A

Data Flow Diagram:



LEARNINGS-

1. The use case diagram and descriptions detail out all use cases and how the actors interact with the cases to reach the desired goals.
2. We have also visually shown the said relationships.

