**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**Ramapuram, Chennai- 600089.**

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**Department of Computer Science & Engineering**

**LAB MANUAL**

**15CSC202JJ / OBJECT ORIENTED DESIGN AND PROGRAMMING LAB**

**CLASS :** B.Tech. [U.G]

**YEAR / SEM. :** II Year / III Semester

**SOFTWARE REQUIREMENT : OOPS** inElab/C++ Compiler

**UML:** STAR UML / Lucidchart UML

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**RAMAPURAM**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Year / Sem : II / III**

**Subject Name/Code: 15CSC202JJ / OBJECT ORIENTED DESIGN AND PROGRAMMING LAB**

**LIST OF EXPERIMENTS (30 Hours)**

1. I/O Operations
2. Classes and Objects, Class Diagram
3. Methods and constructor, Use case
4. Constructor and method overloading
5. Polymorphism: Operators Overloading
6. UML Interaction Diagram
7. Inheritance and its types
8. Virtual Functions and Abstract class
9. State Chart and Activity Diagram
10. Templates
11. Exceptional Handling
12. UML Component, Deployment, Package Diagram
13. STL Containers
14. STL Associative Containers and Algorithms
15. Streams and File Handling

|  |  |
| --- | --- |
| **Ex.No:** 1 | **I/O Operations** |
| **Date:** |

**Aim:**

To Write a C Programs to

1. Find the Area of Triangle
2. Find Area & circumference of circle
3. Swapping of two values

**Algorithm 1:**

1. Start the program
2. Read the three sides of triangle a, b and c
3. Calculate the area using the formula √s(s-a)(s-b)(s-c)
4. Till the value of s=(a+b+c)/2
5. display the area
6. Stop

**Algorithm2:**

1. Start the program
2. Declare the variables rad, area,ci
3. Get the values using cin
4. Calculate the Area of circle and circumference of a circle.
5. Stop

**Algorithm 3:**

1. Start the program
2. Declare the variables a,b
3. Get the values using cin
4. Swap the values using temporary variable
5. Display the values after swapping
6. Stop

**Program 1:**

#include<iostream.h>  
#include<conio.h>

int main()  
{

  float area,s,a,b,c;

cout<<"Enter three sides of the Triangle for calculating Area”<<endl;  
 cin>>a>>b>>c;  
  s=(a+b+c)/2;  
  area=sqrt(s\*(s-a)\*(s-b)\*(s-c));

cout<<”The area of Triangle is =”<<area;

return 0;  
}

**Output 1:**

Enter the values of a, b and c

3

4

5

Area of Triangle = 6

**Program 2:**

#include<iostream.h>

 int main()

{

    int rad;

   float PI = 3.14, area, ci;

cout<<”Enter radius of circle”<<endl;

cin>>rad;

area = PI \* rad \* rad;

cout<<”Area of circle : "<<area << endl;

ci = 2 \* PI \* rad;

  cout<<”Circumference : "<< ci;

 return (0);

}

**Output2:**

Enter radius of a circle : 1

Area of circle : 3.14

Circumference  : 6.28

**Program 3:**

#include<iostream.h>

int main()  
  
{

int var1, var2, swap;

cout<<"Enter value for first integer:  ";  
 cin>>var1;  
 cout<<"Enter value for second integer:  ";  
 cin>>var2;  
 cout<<" Values Before swapping:  "<<endl;  
 cout<<"First Integer ="<<var1<<endl;  
 cout<<"Second Integer ="<<var2<<endl;  
  swap=var1;  
  var1=var2;  
  var2=swap;  
 cout<<" Values After swapping:  "<<endl;  
 cout<<"First Integer ="<<var1<<endl;  
 cout<<"Second Integer ="<<var2<<endl;  
 return 0;

}

**Output 3:**

Enter value for first Integer: 5

Enter value for second integer: 10

Values before Swapping : 5

10

Values after Swapping:

10

5

**Result:** Thus the C program for Area of Triangle, Area & circumference of circle, Swapping the values are executed successfully.

|  |  |
| --- | --- |
| **Ex.No:** 2 a | Classes and Objects |
| **Date:** |

**Aim:**

1. To illustrate the concept and use of classes inside classes
2. To create a class clock which prints the sum of all time entered by the user.
3. To illustrate the concept of nesting of member functions in classes

**Algorithm1:**

1. START

2. Include all the required header files.

3. Declare a class address.

4. declare street, pin and hno. as data members of class addresses.

5. declare two functions of the class in public visibility mode

a) void geta()-for input

b) void showa()-for output

6. Declare a class person.

7. declare name and age as data members.

8. Create two functions void getp() and void outp() for input and output of data.

9. Create an object of class address inside the class person.

10. Using the object, call the member functions of the class address.

11. Inside the main(), declare an object of class person.

12. Use that to take necessary input from the user.

13. Print the required output.

14. STOP

**Algorithm2:**

1. START

2. Create a class name called clock.

3. declare the following as data members of the class

a)h of type int(hours)

b)m of type int(mins)

4. Declare the following as member functions

a) void readtime()-to get the input

b) void showtime()- to show the output to the user

c) void addtime()-To add the time entered by the user

5. Use the following logic to calculate the total time

H=t1.h+t2.h

M=t1.m+t2.m

if(M>=60)

H=H+1

M=M-60

6. Create an object of class clock in void main()

7. Using the object, call the member functions to do the required task.

8. Call the function addtime() to add the quantities.

9. Show the required output using the function showtime().

10. STOP

**Algorithm 3:**

1. START

2. create a class lists

3. Declare the following as data members

a)int i,n lists[50];

4.Declare the following as member functions

a)void display()

b)void modify()

c) void input()

5. Create an object of class lists in void main().

6. Using the object, call the void input() to take the input from the user.

7. Using the void display() show the entered list by the user.

8. Using void modify() multiply each element in the list by 10.

9.Call the function display() inside modify() to show thw output.

10. STOP

**Program 1:**

#include<iostream.h>

#include<conio.h>

class address

{

int hno;

char street[20];

int pin;

public:

void geta()

{

cout<<"Enter the House No";

cin>>hno;

cout<<"Enter the street";

cin>>street;

cout<<"Enter the pin";

cin>>pin;

}

void showa()

{

cout<<"\nThe houseno"<<hno;

cout<<"\nThe street"<<street;

cout<<"\nThe pin"<<pin;

}

};

class person

{

address a;

char name[20];

int age;

public:

void getp()

{

cout<<"\nEnter the name";

cin>>name;

cout<<"\nEnter the age";

cin>>age;

a.geta();

}

void showp()

{

cout<<"\n Name"<<name;

cout<<"\n Age"<<age;

a.showa();

}

};

void main()

{

person p;

p.getp();

p.showp();

getch();

}

**Output1 :**

Enter the name John

Enter the age 29

Enter the houseno 678

Enter the street Park Street

Enter the pin 2345678

Name John

Age 29

Houseno 678

Street Park Street

Pin 2345678

**Program 2:**

#include<iostream.h>

#include<conio.h>

class clock

{

int h,m;

public:

void read\_time( );

void show\_time( );

void add\_time(clock t1,clock t2);

};

void read\_time()

{

cout<<"Enter hrs: ";

cin>>h;

cout<<"Enter mins: ";

cin>>m;

}

void show\_time()

{

cout<<"\nhrs: "<<h<<"\nmins: "<<m;

}

void add\_time(clock t1,clock t2)

{

int H, M;

H=t1.h+t2.h;

M=t1.m+t2.m;

if(M>=60)

{

H=H+1;

M=M-60;

}

cout<<"\ntotal hrs= "<<H<<"\ntotal mins= "<<M;

}

void main()

{

clrscr();

clock t1,t2,c;

int i;

t1.read\_time();

t2.read\_time();

t1.show\_time();

t2.show\_time();

c.add\_time(t1,t2);

getch();

}

**Output2:**

Enter hrs: 5

Enter mins: 25

Enter hrs: 6

Enter mins: 45

hrs: 5

mins: 25

hrs: 6

mins: 45

total hrs= 12

total mins= 10

**Program 3:**

#include<iostream.h>

#include<conio.h>

class lists

{

int i,n, list[50];

public:

void display();

void modify();

void input();

};

void lists::input()

{

cout<<"\n Enter the the length of the list";

cin>>n;

cout<<"\n enter the list"

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

cin>>list[i];

}

void lists::modify()

{

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

list[i]=list[i]\*10;

display();

}

void lists::display()

{

cout<<"\n the lists is:"

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

cout<<list[i]<<endl;

}

void main()

{

lists s;

s.input();

s.modify();

getch();

}

**Output 3:**

Enter the lenght of the list 6

Enter the list 1

2

3

` 4

5

6

the list is 10

20

30

40

50

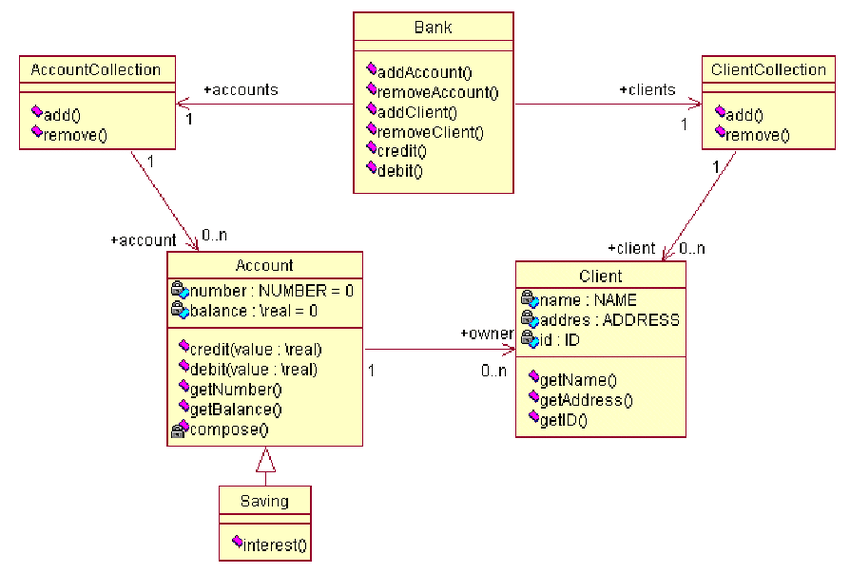
60

**Result:** Thus the concept and use of classes inside classes ,nesting of member functions in classes was illustrated successfully.

|  |  |
| --- | --- |
| **Ex.No:** 2 b | Class Diagram |
| **Date:** |

Aim:

To draw a class diagram for the banking system.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Result:**  **Thus the UML diagram for class diagrams has been drawn successfully.**   |  |  | | --- | --- | | **Ex.No:** 3a | Method and constructors | | **Date:** |   **Aim:**  To write a C++ program for implementing methods and constructors  **Algorithm1:**   1. Create the program 2. Declare the variables 3. Declare and define the method 4. Call the constructor by creating an object for it   **Algorithm2:**   1. Create the program 2. Declare the variables 3. Declare and define the constructor 4. Call the constructor by creating an object for it   **Program1:**   |  | | --- | | // C++ program to demonstrate  // accessing of data members    #include <bits/stdc++.h>  using namespace std;  class Geeks  {      // Access specifier      public:        // Data Members      string geekname;        // Member Functions()      void printname()      {         cout << "Geekname is: " << geekname;      }  };    int main() {        // Declare an object of class geeks      Geeks obj1;        // accessing data member      obj1.geekname = "Abhi";        // accessing member function      obj1.printname();      return 0;  } |   Output:  Geekname is:Abhi  **Program2:**  **# include <iostream>**  using namespace std;    class construct {  public:      int a, b;        // Default Constructor      construct()      {          a = 10;          b = 20;      }  };    int main()  {      // Default constructor called automatically      // when the object is created      construct c;      cout << "a: " << c.a << endl           << "b: " << c.b;      return 1;  } |

Output:

a:10

b:20

**Result:**

Thus the program for methods and constructors has been executed successfully

|  |  |
| --- | --- |
| **Ex.No:** 3b | **Use Case diagram** |
| **Date:** |

**Aim:**

To create a use case diagram for

1. Banking System Management
2. Online Railway Reservation System
3. ATM Machine System

**Algorithm :**

1. **Identify Actors:**

Actors are external entities that interact with your system. It can be a person, roles in another system or an organization.

1. **Identifying Use Cases :**

##### Identify what the actors need from the system.

##### Look for Common Functionality to use Include

If two or more use cases that share common functionality are present, the common functions are extracted and added to a separate use case. Then it is connected via the include relationship to show that it’s always called when the original use case is executed.

1. **Look for possibilities to Generalize Actors and Use Cases**

There may be instances where there are similar use cases but with a few attributes unique only to them. In such instances, the use cases are generalized to show the inheritance

##### Check for Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases, the extended relationship can be used and an extension rule is attached to it. Extend also includes the possibility that the use case connected by extending can supplement the base use case.

**Use case 1:**

**BANKING SYSTEM**

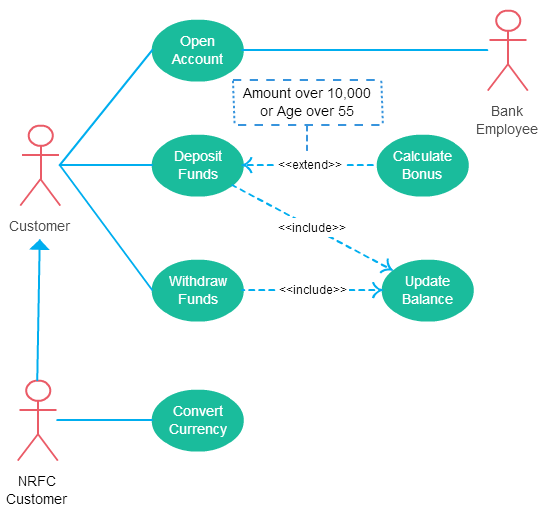
**Title:**Banking System Management

**Actors:** Customer, NRFC Customer and Bank Employee

**Preconditions:** Customer has raised a request to open a Bank account.

**Postconditions:**The Customer is given a bank account and access to his/her account and will be able to deposit, withdraw funds and convert currency.

**Description**: This use case description outlines the steps for a customer to open a bank account, deposit, withdraw funds, view balance and convert currency.



**Use case 2:**

**Railway Reservation**

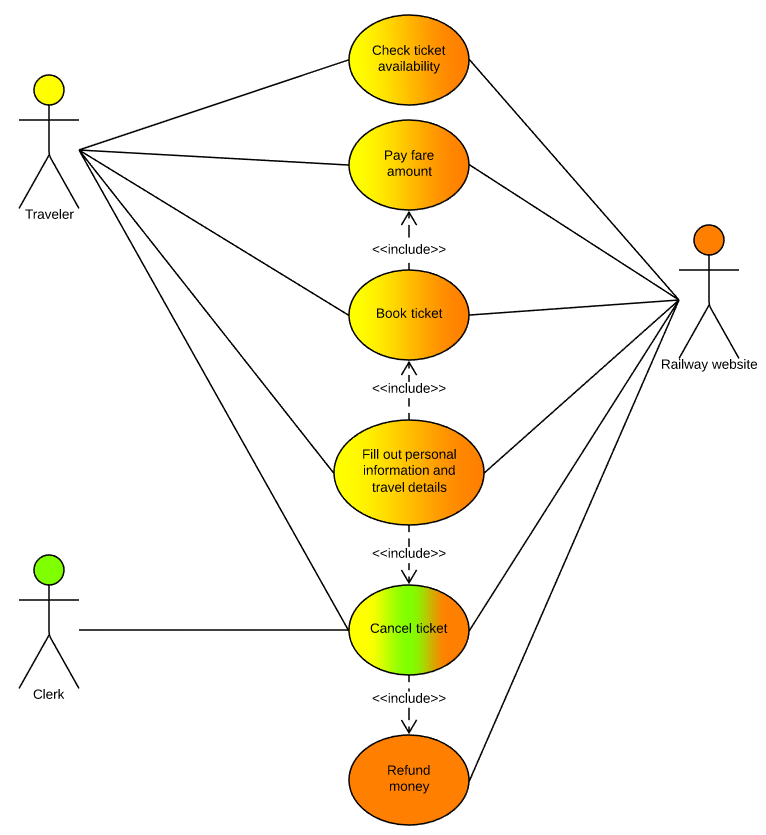
**Title:** OnlineRailway Reservation Management

**Actors:** Traveller, Clerk and Railway Website

**Preconditions:** Traveller has logged into the Railway Reservation Website using an ID.

**Postconditions:**The Customer has either booked a ticket, checked availability of tickets or has cancelled a booked ticket.

**Description**: This use case description provides the procedure that follows once a traveller logs into the railway website to reserve tickets or to cancel them.



**Use case 3:**

**ATM Machine System**

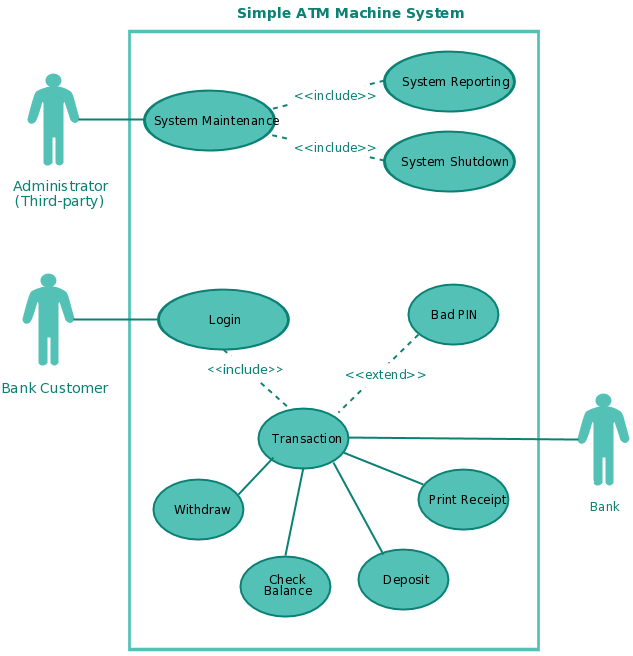
**Title:** ATM Machine System

**Actors:** Bank Customer, Bank and administrator

**Preconditions:** Customer has inserted the bank authorized ATM card.

**Postconditions:**The Customer is given options to either check balance, withdraw or deposit funds for which a receipt is also generated

**Description**: This use case description depicts the steps that are taken by the ATM Machine once the user has inserted the ATM card.



**Result:** Thus Use Case diagrams were drawn for a Banking System, Online Railway Reservation System and a simple ATM Machine System with their respective description.

|  |  |
| --- | --- |
| Ex.No: 4 | **Constructor and Method overloading** |
| Date: |

**Aim:**

.

1. Write a C++ program illustrating Constructor overloading (Both parameterized and

default).

1. To write a C++ program for matrix manipulation with dynamic memory allocation using

copy constructor and overloading of assignment operator.

**Algorithm 1:**

Step1: start

Step2: declare a class.

Step3: Declare constructors of different arguments

Sep4: define constructors.

Step5: Create objects with different arguments as that of constructors.

Step6: stop

**Algorithm 2:**

Step 1: Declare the class as **Matrix**.

Step 2: Declare the data member as **r, c and \*\*x**.

Step 3: Declare the member functions for constructor, destructor, copy constructor,

getdata, putdata and operator overloading.

Step 4: In the main, create the object for the Matrix class.

Step 5: getdata() function, is used to get the matrix elements.

Step 6: putdata() function is used to display the matrix elements.

Step 7: constructor part is used to allocate the memory space for rows and columns.

Step 8: By overloading the operator “=”, it calls the respective function and performs the

operation.

Step 9: By overloading the operators “+” and “-”, it performs the addition and subtraction

of two matrices and display the result.

Step 10: Destructor is used to destroy the memory space.

**Program1:**

/\* Program to illustrate Constructor Overloading (Both Parameterized and Default) \*/

#include<iostream>

using namespace std;

class perimeter

{

private:

int l,b,peri;

public:

perimeter() //default constructor

{

cout<<"\nEnter the values of l and b";

cin>>l>>b;

}

perimeter(int a) //Parameterized constructor with single parameter

{

l=b=a;

}

perimeter(int l1, int b1) //Parameterized constructor with two parameters

{

l=l1;

b=b1;

}

void calculate() //function to calculate the perimeter

{

peri=2\*(l+b);

cout<<peri;

}

};

int main()

{

perimeter obj1, obj2(3), obj3(2,3);

cout<<"\n Perimeter of Rectangle is ";

obj1.calculate();

cout<<"\n perimeter of Square is ";

obj2.calculate();

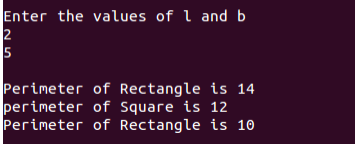
cout<<"\n Perimeter of Rectangle is ";

obj3.calculate();

return 0;

}

**Output 3:**



**Program2:**

#include <iostream.h>

#include <iomanip.h>

#include <conio.h>

class Matrix

{

int r,c;

int \*\*x;

public:

Matrix(int,int);

Matrix(Matrix &b);

Matrix operator = (Matrix b);

Matrix operator + (Matrix b);

Matrix operator - (Matrix b);

~Matrix();

friend istream & operator >>(istream &, Matrix& );

friend ostream & operator <<(ostream &, Matrix );

};

Matrix :: Matrix(int a=2,int b=2)

{

r=a;c=b;

x=new int\*[r];

for(int i=0;i<r;i++)

x[i]=new int[c];

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

for(int j=0;j<c;j++)

{

x[i][j]=0;

}

}

Matrix::Matrix(Matrix &b)

{

r=b.r;c=b.c;

x=new int\*[r];

for(int i=0;i<r;i++)

x[i]=new int[c];

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

for(int j=0;j<c;j++)

{

x[i][j]=b.x[i][j];

}

}

Matrix::~Matrix()

{

for(int i=0;i<r;i++)

delete []x[i];

}

Matrix Matrix ::operator = (Matrix b)

{

r=b.r;c=b.c;

x=new int\*[r];

for(int i=0;i<r;i++)

x[i]=new int[c];

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

for(int j=0;j<c;j++)

{

x[i][j]=b.x[i][j];

}

return b;

}

Matrix Matrix ::operator + (Matrix b)

{

Matrix t=b;

for(int i=0;i<r;i++)

for(int j=0;j<c;j++)

{

t.x[i][j]=x[i][j]+b.x[i][j];

}

return t;

}

Matrix Matrix ::operator - (Matrix b)

{

Matrix t=b;

for(int i=0;i<r;i++)

for(int j=0;j<c;j++)

{

t.x[i][j]=x[i][j]-b.x[i][j];

}

return t;

}

istream & operator >> (istream &in ,Matrix &a)

{

cout<<"\nEnter "<<a.r<<"x"<<a.c<<" Elements"<<endl;

for(int i=0;i<a.r;i++)

for(int j=0;j<a.c;j++)

{

in>>a.x[i][j];

}

return in;

}

ostream & operator << (ostream &ou ,Matrix a)

{

cout<<"\nThe Matrix is:"<<endl;

for(int i=0;i<a.r;i++,cout<<"\n"<<endl)

for(int j=0;j<a.c;j++)

{

ou<<setw(3)<<a.x[i][j]<<"\t";

}

return ou;

}

int main()

{

clrscr();

Matrix a,b,d;

int r,c;

cout<<"\nEnter the Matrix size"<<endl;cin>>r>>c;

a=Matrix(r,c);

cin>>a;

b=a;

cout<<"The a="<<a<<endl;

cout<<"The b="<<b<<endl;

d=a+b;

cout<<"\na+b"<<d<<endl;

d=a-b;

cout<<"\na+b"<<d<<endl;

getch();

return 0;

}

**Output4:**

Enter the matrix of size2x2

2 3

2 3

The matrix A:

2 3

2 3

The matrix B:

2 3

2 3

The resultant matrix(A+B):

4 6

4 6

The resultant matrix(A-B):

0 0

0 0

**Result:**

Thus the program for method and constructor overloading is implemented and executed successfully

|  |  |
| --- | --- |
| **Ex.No**: 5 | **Polymorphism: Operator Overloading** |
| **Date:** |

**Aim:**

Program to improve their skills on polymorphism

**Algorithm:**

1. Create a Class name
2. Declare the members of the class
3. Declare a operator in methods
4. Create main method
5. Call all the functions using objects.

**Program:**

#include<iostream>

using namespace std;

class Complex {

private:

    int real, imag;

public:

    Complex(int r = 0, int i =0)  {real = r;   imag = i;}

Complex operator + (Complex const &obj) {

         Complex res;

         res.real = real + obj.real;

         res.imag = imag + obj.imag;

         return res;

    }

    void print() { cout << real << " + i" << imag << endl; }

};

int main()

{

    Complex c1(10, 5), c2(2, 4);

    Complex c3 = c1 + c2; // An example call to "operator+"

    c3.print();

}

**Output:**

12 + i9

**Result:**

Thus the concept of polymorphism has been implemented and executed

|  |  |
| --- | --- |
| **Ex.No:** 6 | **Inheritance and its types** |
| **Date:** |

**AIM:**

To analyze and design an Interaction Diagram for Automated Teller Machine Using Star UML Software.

**INFRASTRUCTURE:**

HARDWARE REQUIREMENTS

* Processor: x86-based processor.
* Hard disk: 20 GB (min)
* RAM: 128 MB (min)

SOFTWARE REQUIREMENTS

* Star UML Software
* Visual Basic 6.011

**DESCRIPTION:**

UML defines two types of Interaction Diagram: The Sequence Diagram and the Collaboration Diagram. In Order to illustrate both types, the major use cases are documented using Sequence diagrams and the specific subclasses of transaction (withdrawal etc) and the invalid PIN Extension are documented using Collaboration Diagrams.

**Sequence Diagram**: Sequence diagrams typically show the flow of functionality through a use case, and consist of the following components:

1. Actors , involved in the functionality

2. Objects , that a system needs to provide the functionality

3. Messages , which represent communication between objects

**A Communication or Collaboration diagram: A** Communication or Collaboration diagram, as shown, is a directed graph that uses objects and actors as graph nodes. The focus of the collaboration diagram is on the roles of the objects as they interact to realize a system function. Directional links are used to indicate communication between objects. These links are labeled using appropriate messages. Each message is prefixed with a sequence number indicating the time ordering needed to realize the system function.

**PROCEDURE:**

**Step1:** Open a Star UML Software

**Step 2:** To create a Sequence Diagram:

* Select first an element where a new Sequence Diagram to be contained as a child.
* Select **Model | Add Diagram | Sequence Diagram** in Menu Bar or select **Add Diagram | Sequence Diagram** in Context Menu.

To create a Lifeline:

1. Select **Lifeline** in the **Toolbox**.
2. Drag on the diagram as the size of Lifeline.

To create a Lifeline from a Classifier (Class, Interface, etc.) by Drag-and-Drop:

1. Drag a Classifier from **Explorer**.
2. Drop on the diagram.

You can use **QuickEdit** for Lifeline by double-clicking or press Enter on a selected Lifeline.

* + **Visibility** : Change visibility property.
  + **Add Note** : Add a linked note.
  + **Select Type** : Select a type of the lifeline.
  + **Create Type** : Create a Class as a type of the lifeline.
  + **Add Message with Lifeline** : Add a message with a lifeline.
  + **Add Create Message with Lifeline** : Add a create message with a lifeline.
  + **Add Self Message** : Add a self message.
  + **Add Found Message** : Add a found message.
  + **Add Lost Message** : Add a lost message.
  + **Add Message from Gate** : Add a message from a gate.
  + **Add Message to Gate** : Add a message to a gate.

To create an Endpoint:

1. Select **Endpoint** in **Toolbox**.
2. Click at the position on the diagram

To create a Gate:

1. Select **Gate** in **Toolbox**.
2. Click at the position on the diagram.

To create a State Invariant:

1. Select **State Invariant** in **Toolbox**.
2. Click on a Lifeline where the State Invariant is attached.

To create a Continuation:

1. Select **Continuation** in the Toolbox.
2. Drag on the diagram as the size of Continuation.

**Step 3:**To create Collaboration Diagram

The following elements are available in a collaboration diagram.

* Object
* Link
* Self Link
* Stimulus
* Frame

In order to create Object,

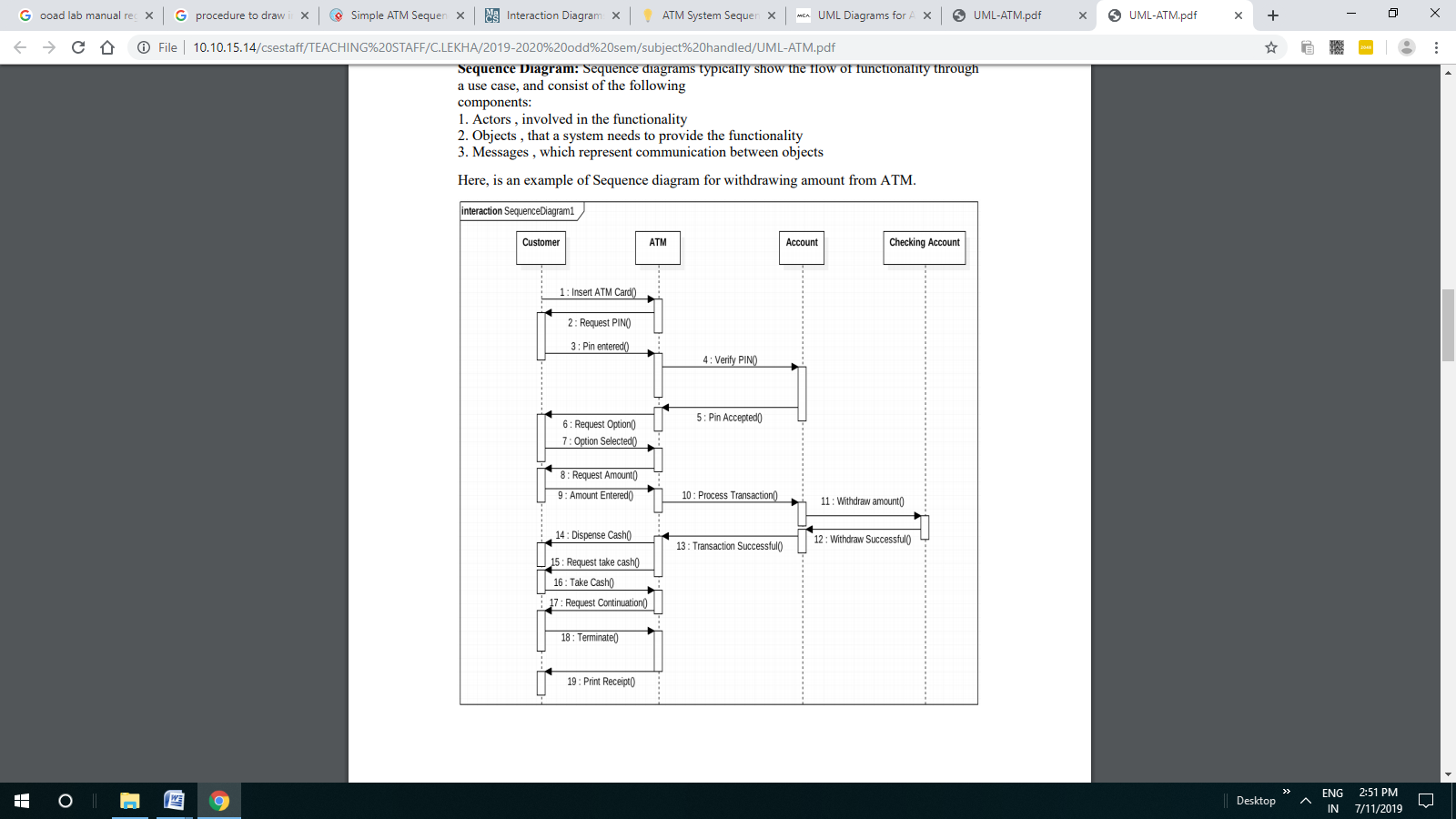
1. Click **[Toolbox] -> [Collaboration] -> [Object]** button.

In order to create outgoing stimulus from selected object to another object,

1. Double-click from-object, or select from-object and press **[Enter]** key to pop up quick dialog.
2. At the quick dialog, enter the stimulus name after the "->" string ("<-" string for incoming and "<->" for outgoing with return).

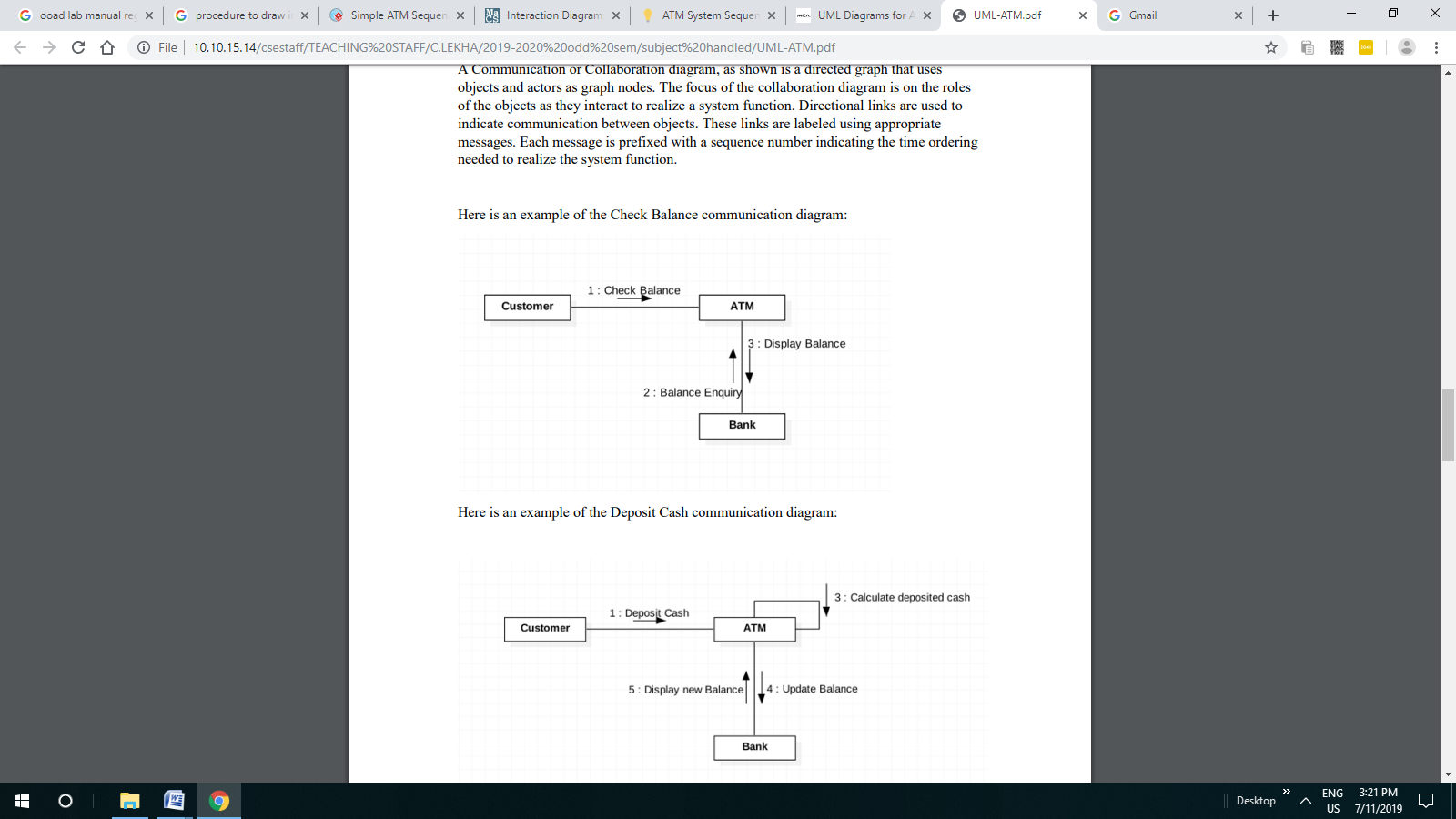
3.Press **[Enter]** key and outgoing stimulus from selected object to target object is created and placed at the last order.

**SEQUENCE DIAGRAM FOR WITHDRAWING AMOUNT FROM ATM**

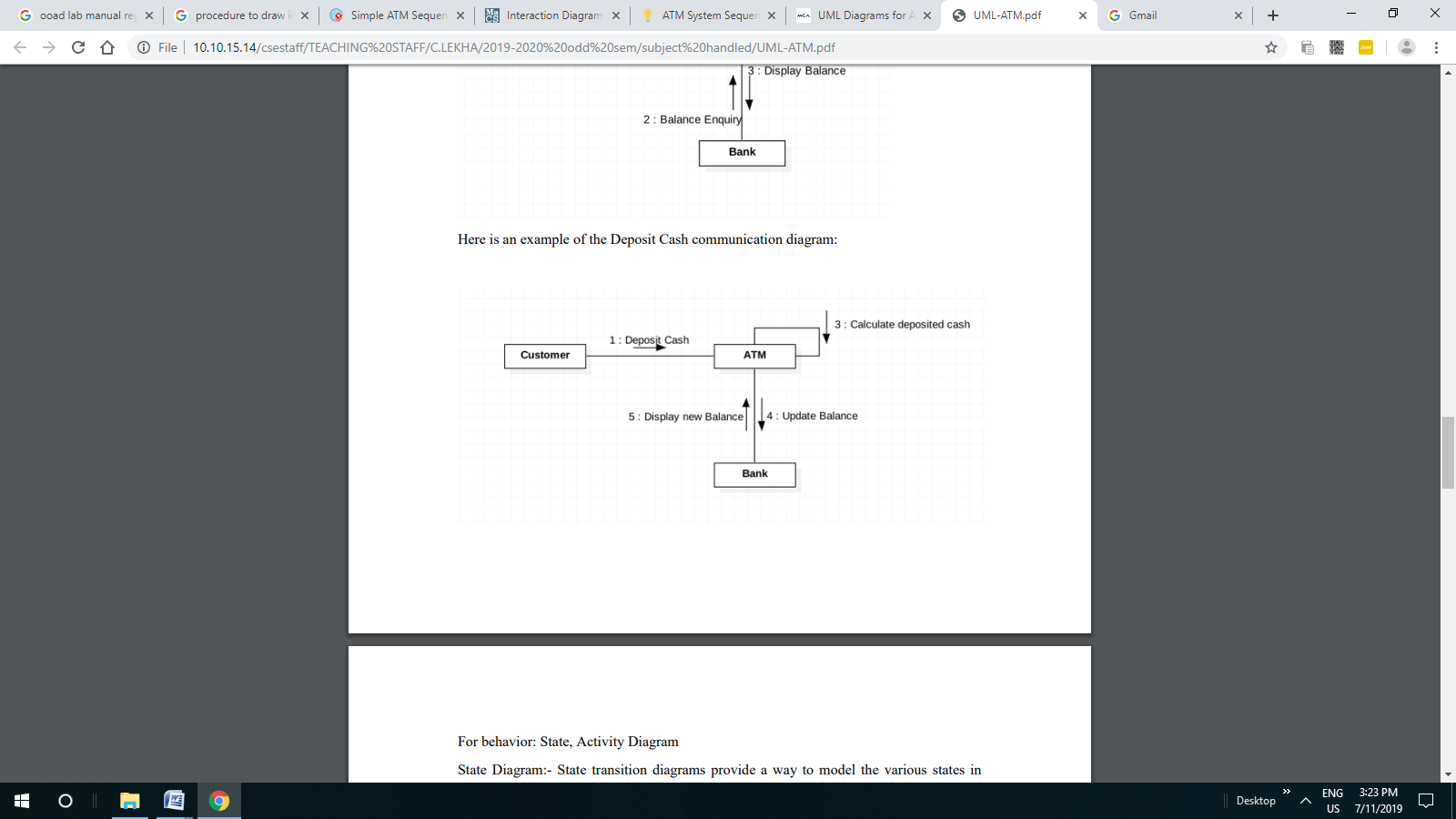
****

**COMMUNICATION/COLLABORATION DIAGRAM:**

* **To Check Balance In ATM Machine**

****

* **To Deposit Cash in ATM Machine**

****

**RESULT:** Thus the diagrams [Sequence, collaboration/Communication] for the Automated Teller Machine have been designed, executed and output is verified.

|  |  |
| --- | --- |
| **Ex.No:** 7 | **Inheritance and its types** |
| **Date:** |

**Aim:**

To write a C++ program for handling bank account of customer using inheritance concept

**Algorithm:**

1. Create a class account
2. Declare the variables like cust\_name,acc\_name,acc\_type
3. Declare and define the inline functions of get\_details() and display\_details() to get the input and output
4. Define and create another class amount and inherit the public contents of the class account
5. Use balance to get the balance value from the functions of deposit\_amt() and withdraw\_amt()
6. Call the class money by creating an object for it

**Program:**

#include<iostream.h>

#include<conio.h>

class account

{

char cust\_name[50];

int acc\_no;

char acc\_type[20];

public:

void get\_details()

{

cout<<”\n enter customer name:”;

cin>>cust\_name;

cout<<”\n enter account no:”;

cin>>acc\_no;

cout<<”\n enter type of account:”;

cin>>acc\_type;

}

void display\_details()

{

cout<<”\n customer name:”<<cust\_name;

cout<<”\n account no:”<<acc\_no;

cout<<”\n type of account:”<<acc\_type;

}

};

class money:public account

{

public:

float balance;

void get\_balance();

{

cout<<”\n enter initial deposit:”;

cin>>balance;

}

void deposit\_amt()

{

float amt;

cout<<”\n enter the amont to be deposited:”;

cin>>amt;

balance=balance+amt;

}

void withdraw\_amt()

{

float amt;

cout<<”\n enter the amount to be withdrawn:”;

cin>>amt;

balance=balance-amt;

}

void display\_balance()

{

cout<<”\n current balance in the account is:”;

cout<<balance;

}

};

void main()

{

money b;

b.get\_details();

b.get\_balance();

char c;

cout<<”\n whether you want to deposit or withdraw the amount(W/D):”;

cin>>c;

if(c==’W’)

b.withdraw\_amt();

else

b.deposit\_amt();

cout<<”the account details are:”;

b.display\_details();

b.display\_balance();

getch();

}

**Output:**

Enter Customer Name: Rahul

Enter Account Number: 50003133424

Enter Account Type: Savings

Enter initial Details: 10000

Whether you want to deposit or withdraw the amount (W/D):

W

Enter the amount to be withdrawn:

5000

The account details are:

Customer Name: Rahul

Account Number: 50003133424

Account Type: Savings

Current Balance in the account is: 5000

**Result:**

Thus the program for handling the bank account details with inheritance is implemented and executed successfully.

|  |  |
| --- | --- |
| **Ex.No:** 8 | **Virtual functions and Abstract class** |
| **Date:** |

**Aim:**

1. To write a program to implement the concept of virtual function and abstract class .

**Algorithm 1:**

1. Start the program
2. Define a base class called base and define a function called display as virtual in it.
3. Derive a new class called derived using a base class called base and define a function called display in the respective classes.
4. Now the display function shows the derived class display function.
5. Stop the program

**Program 1:**

|  |
| --- |
| #include<iostream>  using namespace std;    class Base  {     int x;  public:      virtual void fun() = 0;      int getX() { return x; }  };    // This class inherits from Base and implements fun()  class Derived: public Base  {      int y;  public:      void fun() { cout << "fun() called"; }  };    int main(void)  {      Derived d;      d.fun();      return 0;  } |

Output:

Fun() called

**Result:**

Thus the program to implement virtual function and abstract class has been written and executed successfully.

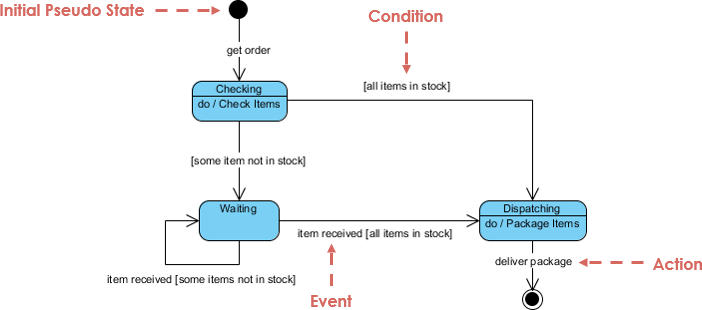
|  |  |
| --- | --- |
| Ex.No: 9 | **State chart and activity Diagram** |
| Date: |

**Aim:**

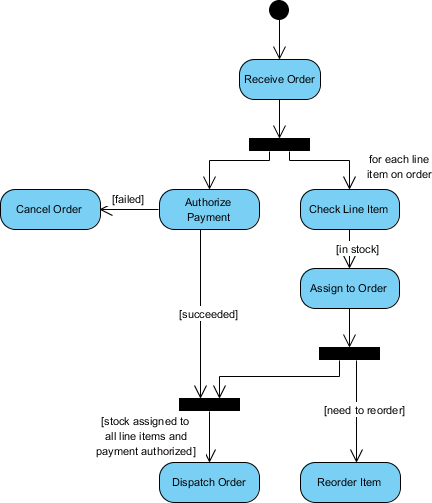
To draw state and activity diagram

**State Diagram**

.



**Activity Diagram**



**RESULT:**

Thus The state and activity diagram have been implemented successfully.

|  |  |
| --- | --- |
| Ex.No: 10 | Templates |
| Date: |

**Aim:**   
 1. To implement a C++ program to swap the numbers using the concept of function

template.

2. To implement a C++ program to display various data types using the concept of class

template.

**Algorithm 1:**

Step 1:  Include the header files

Step 2:  Declare the template class.

Step 3:  Declare and define the functions to swap the values.

Step 4:  Declare and define the functions to get the values.

Step 5:  Read the values and call the corresponding functions.

Step 6:   Display the results.

**Algorithm 2:**

Step 1:  Include the header files

Step 2:  Declare the template class.

Step 3: Create three objects s1,s2,s3

Step 4:  Declare and define the functions to get the values.

Step 5:  Read the values and call the corresponding functions.

Step 6:   Display the results.

**Program 1:**

#include<iostream.h>

#include<conio.h>

template<class t>

 void swap(t &x,t &y)

{

   t temp=x;

   x=y;

   y=temp;

}

void fun(int a,int b,float c,float d)

{

   cout<<"\na and b before swapping :"<<a<<"\t"<<b;

   swap(a,b);

   cout<<"\na and b after swapping  :"<<a<<"\t"<<b;

   cout<<"\n\nc and d before swapping :"<<c<<"\t"<<d;

   swap(c,d);

   cout<<"\nc and d after swapping  :"<<c<<"\t"<<d;

}

void main()

{

    int a,b;

    float c,d;

    clrscr();

    cout<<"Enter A,B values(integer):";

    cin>>a>>b;

    cout<<"Enter C,D values(float):";

    cin>>c>>d;

    fun(a,b,c,d);

    getch();

}

**Output 1:**

Enter A, B values (integer): 10 20

Enter C, D values (float):    2.50 10.80

A and B before swapping: 10 20

A and B after    swapping:  20 10

C and D before swapping: 2.50 10.80

C and D after    swapping: 10.80 2.50

**Program 2 :**

#include<iostream.h>

#include<conio.h>

Template <class t1,class t2>

Class sample

{

T1 a;

T2 b;

Public:

Void getdata()

{

Cout<<”enter a and b values”;

Cin>>a>>b;

}

Void display()

{

Cout<<”displaying values”<<endl;

Cout<<”a=”<<a<<endl;

Cout<<”b=”<<b<<endl;

}

};

Int main()

{

Sample<int,int>s1;

Sample<int,char>s2;

Sample<int,float>s3;

Cout<<”two integer data”<<endl;

S1.getdata();

S1.display();

Cout<<”integer and character data”<<endl;

S2.getdata();

S2.display();

Cout<<”integer and float data”<<endl;

S3.getdata();

S3.display();

Getch();

Return 0;

}

**Output 2:**

Two integer data

Enter a and b:7 11

Displaying values

A=7

B=11

Integer and character data

Enter a and b:4 s

Displaying values:

A=4

B=s

Integer and float data

Enter a and b:14 19.67

Displaying values

A=14

B=19.67

**RESULT:**

Thus a C++ program to perform swap the numbers using the concept of template is implemented successfully.

|  |  |
| --- | --- |
| Ex.No: 11 | Exception Handling |
| Date: |

**Aim:**

1. To implement a C++ program to perform exception handling for Divide by zero

Exception

2. To implement a C++ program to perform exception handling with multiple catch

statements

**Program 1:**

#include<iostream.h>

#include<conio.h>

void main()

{

int a,b,c;

float d;

clrscr();

cout<<"Enter the value of a:";

cin>>a;

cout<<"Enter the value of b:";

cin>>b;

cout<<"Enter the value of c:";

cin>>c;

try

{

if((a-b)!=0)

{

d=c/(a-b);

cout<<"Result is:"<<d;

}

else

{

throw(a-b);

}

}

catch(int i)

{

cout<<"Answer is infinite because a-b is:"<<i;

}

**getch();**

}

**Output 3:**

Enter the value for a: 20

Enter the value for b: 20

Enter the value for c: 40

Answer is infinite because a-b is: 0

**Algorithm4 :**

Step 1: Include the header files

Step 2: Declare and define the function test().

Step 3: Within the try block check whether the value is greater than zero or not.

a.    if  the value greater than zero throw the value and catch the corresponding exception.

b.    Otherwise throw the character and catch the corresponding exception.

Step 4: Read the integer and character values for the function test().

**Program 2:**

#include<iostream.h>

#include<conio.h>

void test(int x)

{

   try

   {

              if(x>0)

                 throw x;

        else

                 throw 'x';

   }

    catch(int x)

   {

              cout<<"Catch a integer and that integer is:"<<x;

   }

    catch(char x)

   {

              cout<<"Catch a character and that character is:"<<x;

   }

}

 void main()

{

   clrscr();

   cout<<"Testing multiple catches\n:";

   test(10);

   test(0);

   getch();

}

**Output4 :**

Testing multiple catches

Catch a integer and that integer is: 10

Catch a character and that character is: x

**RESULT:**

Thus a C++ program to perform exception handling for Divide by zero Exception ,exception handling with multiple catchis implemented successfully.

|  |  |
| --- | --- |
| **Ex.No:** 12 | **UML Package, Component, Deployment diagram** |
| **Date:** |

**Aim:**

To create an UML package, component and deployment diagram for Exam registration. It meets the needs of the applicant and helps them in registering for the exam ,enquiry about the registered subject ,modification in database and cancellation for the registered subject.

**Algorithm :**

The Exam Registration System is an integrated system that has four modules as part of it.the four modules are

1. **Registration for the exam**

In this module, the user can select the subject to register for the exam,Enquiry about the registered subject, Modification in the student database, canceling the registered subject

1. **Form for Registration**

In this module the user can apply for the exam by giving the details about the candidate and selecting the subject for the registration.

1. **Modification in the Database**

In this module the user can change the data like the phone number, address can be done.

1. **Cancellation for the registered subject**

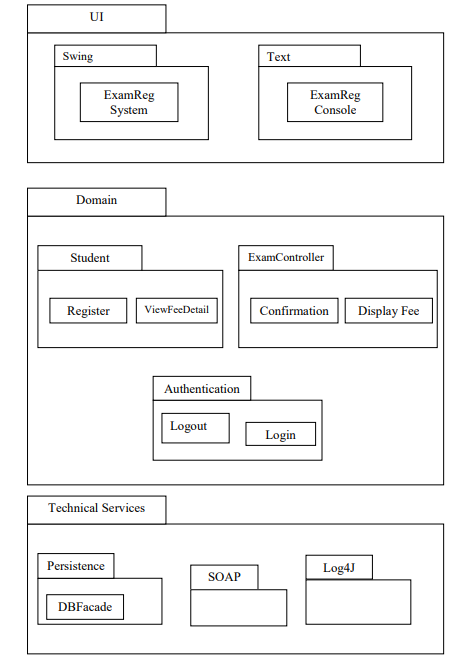
In this module the user can cancel their name which is registered for the exam.

**PACKAGE DIAGRAM**

The three layer in the Exam registration system are user interface layer, domain

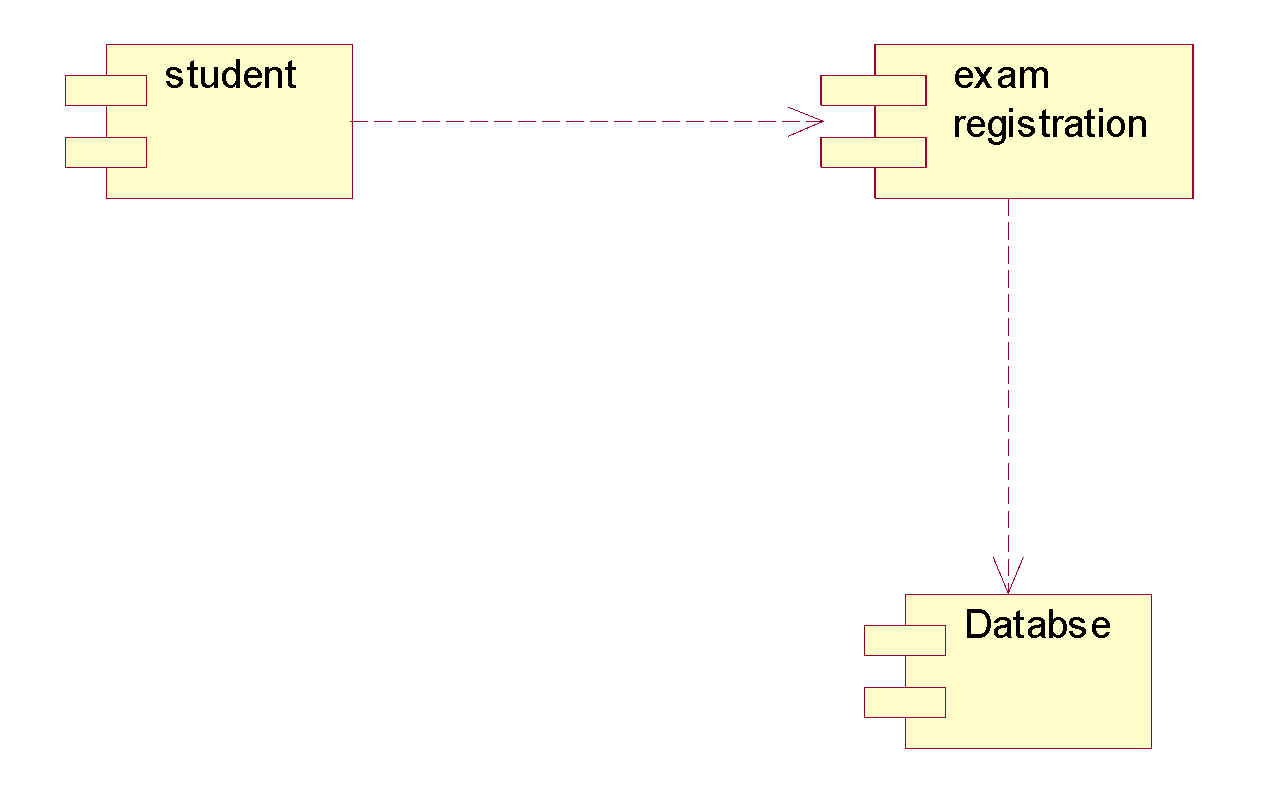
layer, technical service layer

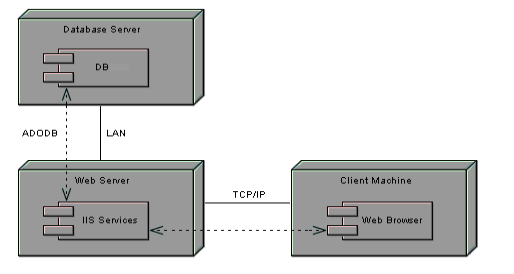
1. **The user interface layer-** represents the user interface components such asweb, applicant, passport administrator, police, and regional administrator.
2. **The domain layer-** has major actions such as give and get details, verificationand issues
3. **Technical service layer**- authenticated users only can access the technicalservices.



**COMPONENT DIAGRAM**

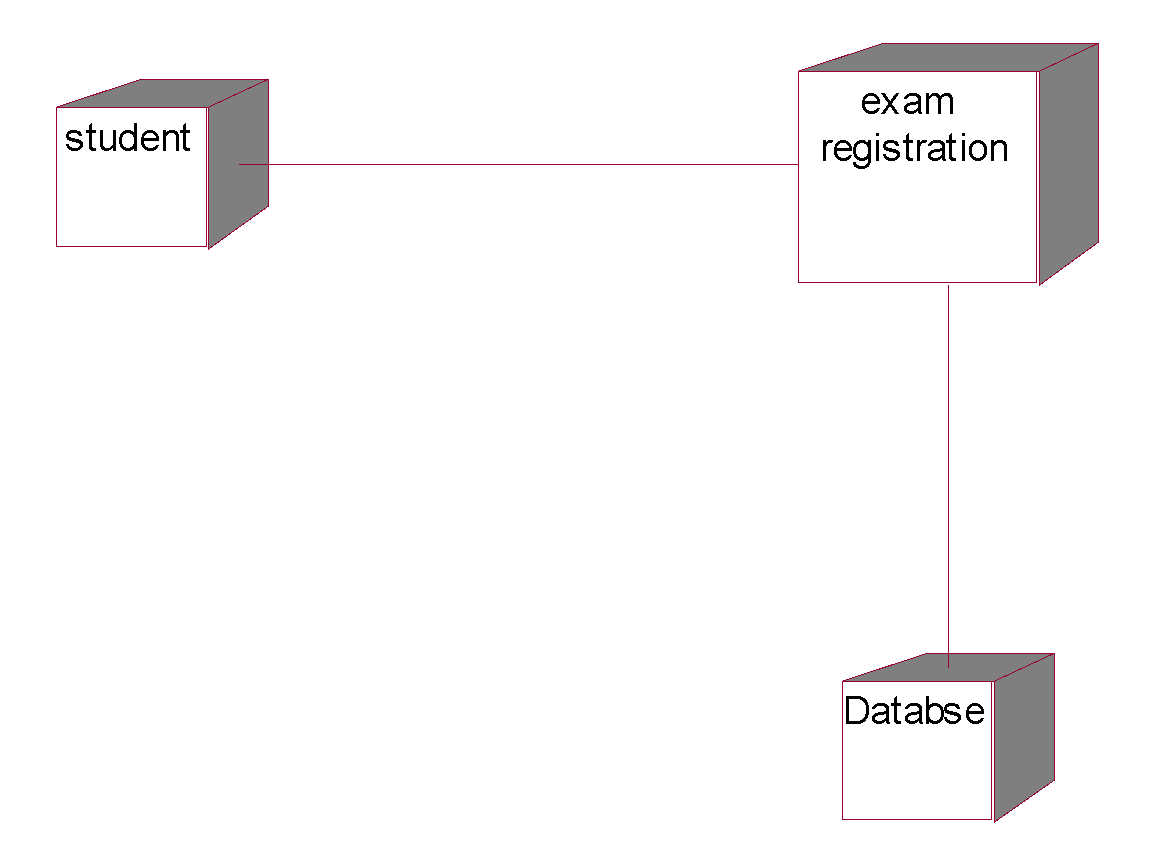
Component diagrams are used to visualize the organization and relationships among components in a system

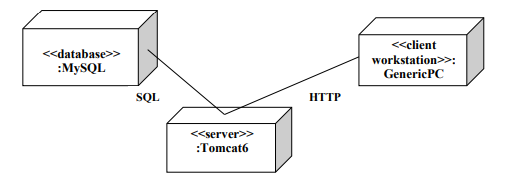




**DEPLOYMENT DIAGRAM:**

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.





**Result:** Thus Package, Component and Deployment diagrams were drawn for Exam Registration System with their respective descriptions.

|  |  |
| --- | --- |
| **Ex.No:** 13 | **STL Containers** |
| **Date:** |

**Aim:**

To construct a STL(Standard Template Library) for sequential containers and iterators

**Algorithm1:**

1. Include header files

2. Declare the map function for holding the employee details like Employee

Id and name.

3. Assign the values for Map container using array index notation

4. Display the details of an Employee using map name

5. Display all the details stored in map using iterators and Map functions.

6. End of program

**Algorithm 2:**

insert(const\_iterator q, const value\_type g) – Adds element ‘g’ before the element referenced by iterator ‘q’ and returns an iterator that points to the newly added element

insert(const\_iterator q, size\_type n, const value\_type g) – Adds ‘n’ elements each with value ‘g’ before the element currently referenced by iterator ‘q’ and returns an iterator that points to the first of the newly added elements

insert(const\_iterator q, InputIterator first, InputIterator last) – Adds a range of elements starting from first to last, the elements being inserted after the position currently referred by ‘q’

**Program1:**

#include <iostream>

#include <vector>

int main()

{

    vector <int> gquiz1;

    vector <int> gquiz2;

    vector <int> :: iterator i;

    gquiz1.push\_back(10);

    gquiz1.push\_back(20);

    gquiz2.push\_back(30);

    gquiz2.push\_back(40);

    cout << "Before Swapping, \n"

         <<"Contents of vector gquiz1 : ";

    for (i = gquiz1.begin(); i != gquiz1.end(); ++i)

        cout << \*i << '\t';

    cout << "\nContents of vector gquiz2 : ";

    for (i = gquiz2.begin(); i != gquiz2.end(); ++i)

        cout << \*i << '\t';

    swap(gquiz1, gquiz2);

    cout << "\n\nAfter Swapping, \n";

    cout << "Contents of vector gquiz1 : ";

    for (i = gquiz1.begin(); i != gquiz1.end(); ++i)

        cout << \*i << '\t';

    cout << "\nContents of vector gquiz2 : ";

    for (i = gquiz2.begin(); i != gquiz2.end(); ++i)

        cout << \*i << '\t';

    cout << "\n\nNow, we clear() and then add "

         << "an element 1000 to vector gquiz1 : ";

    gquiz1.clear();

    gquiz1.push\_back(1000);

    cout << gquiz1.front();

    return 0;

}

Output:

Before Swapping,

Contents of vector gquiz1 : 10 20

Contents of vector gquiz2 : 30 40

After Swapping,

Contents of vector gquiz1 : 30 40

Contents of vector gquiz2 : 10 20

Now, we clear() and then add an element 1000 to vector gquiz1 : 1000

Program 2:

#include <iostream>

#include <vector>

int main()

{

    vector <int> gquiz1(3, 10);

    vector <int> :: iterator it;

    it = gquiz1.begin();

    it = gquiz1.insert(it, 20);

    gquiz1.insert(it, 2, 30);

    it = gquiz1.begin();

    vector <int> gquiz2(2, 40);

    gquiz1.insert(it + 2, gquiz2.begin(), gquiz2.end());

    int gq [] = {50, 60, 70};

    gquiz1.insert(gquiz1.begin(), gq, gq + 3);

    cout << "gquiz1 contains : ";

    for (it = gquiz1.begin(); it < gquiz1.end(); it++)

        cout << \*it << '\t';

    return 0;

}

Output 2:

gquiz1 contains : 50 60 70 30 30

40 40 20 10 10 10

**Result:**

Thus the implementation of STL Has been implemented successfully.

|  |  |
| --- | --- |
| Ex.No: 14 | STL Associative Containers and Algorithms |
| Date: |

**Aim:** To write a C++ program using Associative Containers(Map).

**Algorithm:**

1. Include header files
2. Declare the map function for holding the employee details like Employee Id and name.
3. Assign the values for Map container using array index notation
4. Display the details of an Employee using map name
5. Display all the details stored in map using iterators and Map functions.
6. End of program

**Program:**

|  |  |
| --- | --- |
|  | #include <string.h> |

|  |  |
| --- | --- |
|  | #include <iostream> |

|  |  |
| --- | --- |
|  | #include <map> |

|  |  |
| --- | --- |
|  | #include <utility> |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | using namespace std; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | int main() |

|  |  |
| --- | --- |
|  | { |

|  |  |
| --- | --- |
|  | map<int, string> Employees; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | // 1) Assignment using array index notation |

|  |  |
| --- | --- |
|  | Employees[5234] = "Mike C."; |

|  |  |
| --- | --- |
|  | Employees[3374] = "Charlie M."; |

|  |  |
| --- | --- |
|  | Employees[1923] = "David D."; |

|  |  |
| --- | --- |
|  | Employees[7582] = "John A."; |

|  |  |
| --- | --- |
|  | Employees[5328] = "Peter Q."; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | cout << "Employees[3374]=" << Employees[3374] << endl << endl; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | cout << "Map size: " << Employees.size() << endl; |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  | for( map<int,string>::iterator ii=Employees.begin(); ii!=Employees.end(); ++ii) |

|  |  |
| --- | --- |
|  | { |

|  |  |
| --- | --- |
|  | cout << (\*ii).first << ": " << (\*ii).second << endl; |

|  |  |
| --- | --- |
|  | } |

|  |  |
| --- | --- |
|  | } |

**Output :**

Employees[3374]=Charlie M.

Map size: 5

1923: David D.

3374: Charlie M.

5234: Mike C.

5328: Peter Q.

7582: John A.

**Result:**

Thus the implementation of STL associative containers has been implemented successfully.

|  |  |
| --- | --- |
| Ex.No: 15 | **Streams and File Handling** |
| Date: |

**Aim:**

.

Write a C++ program illustrating streams and file handling.

**Algorithm 1:**

Step1: start

Step2: declare a class.

Step3: Open a file.

Sep4: Execute a loop if a file successfully opens.

Step5: Write in a file .

Step6: Close the file

#include <fstream>

using namespace std;

// Driver Code

int main()

{

    // Creation of ofstream class object

    ofstream fout;

    string line;

    // by default ios::out mode, automatically deletes

    // the content of the file. To append the content, open in ios:app

    // fout.open("sample.txt", ios::app)

    fout.open("sample.txt");

    // Execute a loop If file successfully opened

    while (fout) {

        // Read a Line from standard input

        getline(cin, line);

        // Press -1 to exit

        if (line == "-1")

            break;

        // Write line in file

        fout << line << endl;

    }

    // Close the File

    fout.close();

    // Creation of ifstream class object to read the file

    ifstream fin;

    // by default open mode = ios::in mode

    fin.open("sample.txt");

    // Execute a loop until EOF (End of File)

    while (fin) {

        // Read a Line from File

        getline(fin, line);

        // Print line in Console

        cout << line << endl;

    }

    // Close the file

    fin.close();

    return 0;

**Result:**

Thus the implementation of streams and files has been implemented successfully.