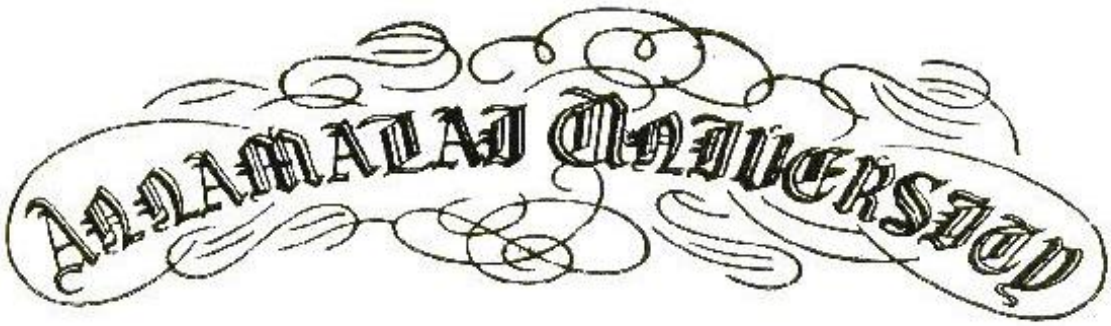


FACULTY OF ENGINEERING AND TECHNOLOGY

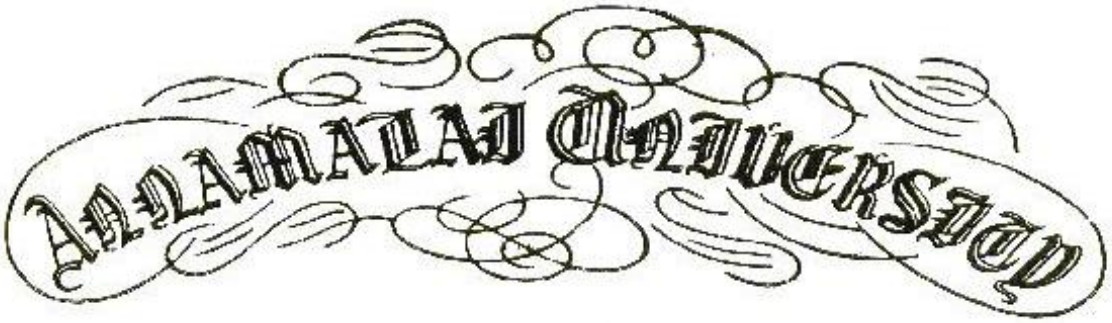
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E.(CSE) III SEMESTER

CSCP 309 - OBJECT ORIENTED PROGRAMMING LAB

Name : .....

Reg. No. : .....



FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E.(CSE) III SEMESTER

CSCP 309 -OBJECT ORIENTED PROGRAMMING LAB

Certified that this Bonafide Record of work done by

Mr. /Ms.....

Reg. No. ..... of B.E. (CSE) in the CSCP **- 309**

OBJECT ORIENTED PROGRAMMING LAB *during the year 2019 – 2020.*

Staff-in-Charge Internal Examiner

Annamalainagar External Examiner

Date:

Cycle 1: C++

S. No	Date	Program	Page No	Signature
1		Class and Objects		
2		Constructors and Destructors		
3		Passing/Returning objects to/ from a function		
4 a		Unary Operator Overloading		
4 b		Binary Operator Overloading		
5		Multiple Inheritance		
6a		Friend Function		
6b		THIS pointer		
7		Virtual Function		
8		Data conversion between objects of different classes		
9		File operations		

Cycle 2: JAVA

S.No	Date	Program	Page No	Signature
10		Classes and Objects		
1		String functions		
12		Creation of Package		
13		Creation of Interface		
14		Multithreading		
15		Exception Handling		

Ex. No:1 **Class and Object**
Date:

Aim: To develop a C++ program to show how to create a class and how to create objects.

C++ Classes and Objects:
The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Program:

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

class Distance
{
private:
int feet;
float inches;
public:
void setdist(int ft, float in)
{ feet = ft; inches = in; }
void getdist()
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist()
{
 cout << feet << "\'-" << inches << '\"';
}
};

int main()
{
Distance dist1, dist2;
```

```
dist1.setdist(11, 6.25);
dist2.getdist();
cout << "\ndist1 = "; dist1.showdist();
cout << "\ndist2 = "; dist2.showdist();
cout << endl;
return 0;
}
```

Sample data:

Enter feet: 10
Enter inches: 4.75
dist1 = 11'-6.25"
dist2 = 10'-4.75"

Result: Thus a C++ program was written to create a class and objects.

Ex No:2 **Constructors and Destructor**
Date:

Aim: To develop a C++ program to add constructors and destructor for Initializing and destroying objects.

Constructors and Destructors

C++Constructors are special class functions which performs initialization of every object. Constructors initialize values to object members after storage is allocated to the object. Whereas, Destructor on the other hand is used to destroy the class object.

Program:

```
#include <iostream.h>

class Counter
{
private:
unsigned int count;
public:
Counter() : count(0)
{  }
Counter(int c)
{ count = c; }

Counter (Counter &c)
{ count = c.count; }

void inc_count()
{ count++; }

void dec()
{ count--; }

void show()
{ cout<<"Count is"<<count<<endl; }
```

```
~Counter()
{ Cout<<"Object destroyed\n"; }

};

int main()
{
Counter c1, c2(20),c3(c2);
c1.inc_count();
c2.inc_count();
c3.dec();
c1.show();
c2.show();
c3.show();

return 0;
}
```

Sample data:

Count is 1
Count is 21
Count is 1

Result: Thus a C++ program was written to add constructors and destructor to a class.

Ex No:3 **Passing/Returning objects to/from a function**
Date:

Aim: To develop a C++ program to pass and return objects to and from a function.

In C++ we can pass class's objects as arguments and also return them from a function the same way we pass and return other variables. No special keyword or header file is required to do so.

Program:

```
#include <iostream.h>

class Distance
{
private:
int feet;
float inches;
public:
Distance() : feet(0), inches(0.0)
{}
Distance(int ft, float in) : feet(ft), inches(in)
{}
void getdist()
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist()
{ cout << feet << "\'-" << inches << '\"'; }
Distance add_dist(Distance);
};
Distance Distance::add_dist(Distance d2)
{
```

```
Distance temp;
temp.inches = inches + d2.inches;
if(temp.inches >= 12.0)
{
temp.inches -= 12.0;
temp.feet = 1;
}
temp.feet += feet + d2.feet;
return temp;
}
int main()
{
Distance dist1, dist3;
Distance dist2(11, 6.25);
dist1.getdist();
dist3 = dist1.add_dist(dist2);

cout << "\ndist1 = "; dist1.showdist();
cout << "\ndist2 = "; dist2.showdist();
cout << "\ndist3 = "; dist3.showdist();
cout << endl;
return 0;
}
```

Sample data:

Enter feet: 17
Enter inches: 5.75
Dist 1= 17' 5.75'
Dist 2 = 11' 6.25
Dist 3 = 29

Result: Thus a C++ program was written to pass and return objects as arguments to and from a function.

Ex No:4a **Unary operator Overloading**
Date:

Aim: To Write a c++ program to overload unary operator.

Unary operator Overloading:
The unary operators operate on the object for which they were called and normally, this operator appears on the left side of the object, as in !obj, -obj, and ++obj but sometime they can be used as postfix as well like obj++ or obj--.

Program:

```
#include <iostream>
using namespace std;
class Counter
{
private:
unsigned int count;
public:
Counter() : count(0)
{}
unsigned int get_count()
{ return count; }
void operator ++ ()
{
++count;
}};
int main()
{
Counter c1, c2;
cout << "\nc1=" << c1.get_count();
cout << "\nc2=" << c2.get_count();
```

```
++c1;
++c2;
++c2;
cout << "\nc1=" << c1.get_count();
cout << "\nc2=" << c2.get_count() << endl;
return 0;
}
```

Sample data:

```
c1=0
c2=0
c1=1
c2=2
```

Result: Thus a C++ program to overload unary operator is written.

Ex No:4b **Binary Operator Overloading**
Date:

Aim: To write a C++ program to overload binary operator.

Binary Operator Overloading

The binary operators take two arguments. You use binary operators very frequently like addition (+)operator, subtraction (-) operator and division (/)operator. Following example explains how addition (+)operator can be overloaded.

Program:

```
#include <iostream>
using namespace std;
class Distance
{
private:
int feet;
float inches;
public:
Distance() : feet(0), inches(0.0)
{}
Distance(int ft, float in) : feet(ft), inches(in)
{}
void getdist()
```

```
{
cout << "\nEnter feet: "; cin >> feet;
cout << "Enter inches: "; cin >> inches;
}
void showdist() const
{ cout << feet << "\'-" << inches << '\"'; }
Distance operator + ( Distance ) const;
};
//--------------------------------------------------------------

Distance Distance::operator + (Distance d2) const
{
int f = feet + d2.feet;
float i = inches + d2.inches;
if(i >= 12.0)
{
i -= 12.0;
f++;
}
return Distance(f,i);
}
int main()
{
Distance dist1, dist3, dist4;
dist1.getdist();
Distance dist2(11, 6.25);
dist3 = dist1 + dist2;
dist4 = dist1 + dist2 + dist3;

cout << "dist1 = "; dist1.showdist(); cout << endl;
cout << "dist2 = "; dist2.showdist(); cout << endl;
cout << "dist3 = "; dist3.showdist(); cout << endl;
cout << "dist4 = "; dist4.showdist(); cout << endl;
return 0;
}
```

Sample data:

```
Enter feet: 10
Enter inches: 6.5
dist1 = 10'-6.5"
dist2 = 11'-6.25"
dist3 = 22'-0.75"
dist4 = 44'-1.5"
```

Result: Thus a C++ program to overload binary operator is written.

Ex No:5 **Multiple Inheritance**
Date:

Aim: To write a C++ program to inherit from multiple classes.

Multiple Inheritance:
Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. The constructors of inherited classes are called in the same order in which they are inherited.

Program:

```
#include <iostream>
using namespace std;
const int LEN = 80;
//////////////////////////////////////////////////////////////
class student
{
private:
char school[LEN];
char degree[LEN];
public:
```

```
void getedu()
{
cout << " Enter name of school or university: ";
cin >> school;
cout << " Enter highest degree earned \n";
cout << " (Highschool, Bachelor's, Master's, PhD): ";
cin >> degree;
}
void putedu() const
{
cout << "\n School or university: " << school;
cout << "\n Highest degree earned: " << degree;
}};
class employee
{
private:
char name[LEN];
unsigned long number;
public:
void getdata()
{
cout << "\n Enter last name: "; cin >> name;
cout << " Enter number: "; cin >> number;
}
void putdata() const
{
cout << "\n Name: " << name;
cout << "\n Number: " << number;
}
};
////////////////////////////////////////////////////////////
class manager : private employee, private student  //management
{
private:
char title[LEN];
double dues;
public:
void getdata()
{
employee::getdata();
cout << " Enter title: "; cin >> title;
cout << " Enter golf club dues: "; cin >> dues;
student::getedu();
}
```

```
void putdata() const
{
employee::putdata();
cout << "\n Title: " << title;
cout << "\n Golf club dues: " << dues;
student::putedu();
}};
//////////////////////////////////////////////////////////////
class scientist : private employee, private student      //scientist
{
private:
int pubs;
public:
void getdata()
{
employee::getdata();
cout << " Enter number of pubs: "; cin >> pubs;
student::getedu();
}
void putdata() const
{
employee::putdata();
cout << "\n Number of publications: " << pubs;
student::putedu();
}
};
//////////////////////////////////////////////////////////////
class laborer : public employee //laborer
{
};
//////////////////////////////////////////////////////////////
int main()
{
manager m1;
scientist s1, s2;
laborer l1;
cout << endl;
cout << "\nEnter data for manager 1";
m1.getdata();
cout << "\nEnter data for scientist 1";
s1.getdata();
cout << "\nEnter data for scientist 2";
s2.getdata();
cout << "\nEnter data for laborer 1";
```

```
l1.getdata();
cout << "\nData on manager 1";
m1.putdata();
cout << "\nData on scientist 1";
s1.putdata();
cout << "\nData on scientist 2";
s2.putdata();
cout << "\nData on laborer 1";
l1.putdata();
cout << endl;
return 0;
}
```

Sample data:

```
Enter data for manager 1
Enter last name: Bradley
Enter number: 12
Enter title: Vice-President
Enter golf club dues: 100000
Enter name of school or university: Yale
Enter highest degree earned
(Highschool, Bachelor's, Master's, PhD): Bachelor's
Enter data for scientist 1
Enter last name: Twilling
Enter number: 764
Enter number of pubs: 99
Enter name of school or university: MIT
Enter highest degree earned
(Highschool, Bachelor's, Master's, PhD): PhD
Enter data for scientist 2
Enter last name: Yang
Enter number: 845
```

Enter number of pubs: 101
Enter name of school or university: Stanford
Enter highest degree earned
(Highschool, Bachelor's, Master's, PhD): Master's
Enter data for laborer 1
Enter last name: Jones
Enter number: 48323

Result: Thus a C++ program to inherit from multiple classes is written.

Ex No:6a **Friend function**
Date:

Aim : To write a C++ program to add a friend function for more than one class.

Friend function :
A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

Program:

```
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class beta;
```

```
class alpha
{
private:
int data;
public:
alpha() : data(3) {}
friend int frifunc(alpha, beta);
};
////////////////////////////////////////////////////////////
class beta
{
private:
int data;
public:
beta() : data(7) {}
friend int frifunc(alpha, beta);
};
////////////////////////////////////////////////////////////
int frifunc(alpha a, beta b)
{
return( a.data + b.data );
}
//-------------------------------------------------------------
int main()
{
alpha aa;
beta bb;
cout << frifunc(aa, bb) << endl;
return 0;
}
```

Sample data:

10

Result: Thus a C++ program to add a friend function for more than one class is written.

Ex No.6b: **THIS Pointer**
Date:

Aim: To write a C++ program to use the THIS pointer.

THIS Pointer:
In C++, this pointer is used to represent the address of an object inside a member function. For example, consider an object obj calling one of its member function say method() as obj.method(). Then, this pointer will hold the address of object obj inside the member function method().

Program:

```
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class alpha{
private:
int data;
public:
alpha()
{}
alpha(int d)                                    //one-arg constructor
{ data = d; }
void display()                                  //display data
{ cout << data; }
alpha& operator = (alpha& a)        //overloaded = operator
{
data = a.data;                                  //not done automatically
cout << "\nAssignment operator invoked";
return *this;                                   //return copy of this alpha
}
};
////////////////////////////////////////////////////////////
int main()
{
alpha a1(37);
alpha a2, a3;
a3 = a2 = a1;
cout << "\na2="; a2.display();
cout << "\na3="; a3.display();
cout << endl;
return 0;
}
```

Sample data:

```
Assignment operator invoked
Assignment operator invoked
a2=37
a3=37
```

Result: Thus a C++ program to use the THIS pointer is written.

Ex No:7 **Virtual Function**
Date:

Aim: To write a C++ program to add a virtual function.

A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword. It is used to tell the compiler to perform dynamic linkage or late binding on the function.

Program:

```
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class Base {
public:
virtual void show()
{ cout << "Base\n"; }
};
////////////////////////////////////////////////////////////
class Derv1 : public Base
{
public:
void show()
{ cout << "Derv1\n"; }
};
////////////////////////////////////////////////////////////
class Derv2 : public Base
{
public:
void show()
{ cout << "Derv2\n"; }
};
////////////////////////////////////////////////////////////
int main(){
Derv1 dv1;
Derv2 dv2;
Base* ptr;
ptr = &dv1;
ptr->show();
ptr = &dv2;
ptr->show();
return 0;
}
```

Sample data:

```
Derv1
Derv2
```

Result: Thus a C++ program to add a virtual function is written.

Ex No: 8 Data conversion between objects of different classes
Date:

Aim: To write a C++ program to convert data between objects of different classes.

Data conversion in C++ includes conversions between basic types and user-defined types, and conversions between different user-defined types.

Program:

```
#include <iostream>
#include <string>
using namespace std;
//////////////////////////////////////////////////////////////
class time12
{
private:
bool pm;
int hrs;
int mins;
public:
time12() : pm(true), hrs(0), mins(0)
{}
time12(bool ap, int h, int m) : pm(ap), hrs(h), mins(m)
{}
void display() const
{
cout << hrs << ':';
if(mins < 10)
cout << '0';
cout << mins << ' ';
string am_pm = pm ? "p.m." : "a.m.";
cout << am_pm;
}};
//////////////////////////////////////////////////////////////
class time24
{
private:
int hours;
int minutes;
int seconds;
public:
time24() : hours(0), minutes(0), seconds(0)
{}
time24(int h, int m, int s) :
hours(h), minutes(m), seconds(s)
{}
```

```
void display() const
{
if(hours < 10) cout << '0';
cout << hours << ':';
if(minutes < 10) cout << '0';
cout << minutes << ':';
if(seconds < 10) cout << '0';
cout << seconds;
}
operator time12() const;
};          //-------------------------------------------------------------
time24::operator time12() const
{
int hrs24 = hours;
bool pm = hours < 12 ? false : true;
//round secs
int roundMins = seconds < 30 ? minutes : minutes+1;
if(roundMins == 60)
{
roundMins=0;
++hrs24;
if(hrs24 == 12 || hrs24 == 24)
pm = (pm==true) ? false : true;
}
int hrs12 = (hrs24 < 13) ? hrs24 : hrs24-12;
if(hrs12==0)
{ hrs12=12; pm=false; }
return time12(pm, hrs12, roundMins);
}
////////////////////////////////////////////////////////////////
int main()
{
int h, m, s;
while(true)
{
cout << "Enter 24-hour time: \n";
cout << " Hours (0 to 23): "; cin >> h;
if(h > 23)
return(1);
cout << " Minutes: "; cin >> m;
cout << " Seconds: "; cin >> s;
time24 t24(h, m, s);
cout << "You entered: ";
t24.display();
```

```
time12 t12 = t24;
cout << "\n12-hour time: ";
t12.display();
cout << "\n\n";
}
return 0;
}
```

Sample data:

```
Enter 24-hour time:
Hours (0 to 23): 17
Minutes: 59
Seconds: 45
You entered: 17:59:45
12-hour time: 6:00 p.m.
```

Result: Thus a C++ program to convert data between objects of different classes is written.

Ex No:9 **File operations**
Date:

Aim: To write a C++ program to perform file operations.

File represents storage medium for storing data or information. Streams refer to sequence of bytes. In Files we store data i.e. text or binary data permanently and use these data to read or write in the form of input output operations by transferring bytes of data.

Program:

```
#include <fstream>
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
class person
{
protected:
char name[80];
int age;
public:
void getData()
{
cout << "\n Enter name: "; cin >> name;
cout << " Enter age: "; cin >> age;
}
void showData()
{
cout << "\n Name: " << name;
cout << "\n Age: " << age;
}
};
////////////////////////////////////////////////////////////
int main()
{
char ch;
person pers;
fstream file;
file.open("GROUP.DAT", ios::app | ios::out |
ios::in | ios::binary );
```

```
do
{
cout << "\nEnter person's data:";
pers.getData();

file.write( reinterpret_cast<char*>(&pers), sizeof(pers) );
cout << "Enter another person (y/n)? ";
cin >> ch;
}
while(ch=='y');
file.seekg(0);

file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
while( !file.eof() )
{
cout << "\nPerson:";
pers.showData();
file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
}
cout << endl;
return 0;
}
```

Sample data:

```
Enter person's data:
Enter name: McKinley
Enter age: 22
Enter another person (y/n)? n
Person:
Name: Whitney
Age: 20
Person:
Name: Rainier
Age: 21
Person:
Name: McKinley
Age: 22
```

Result: Thus a C++ program to perform file operations is written.

Ex No:10 **Class and Objects**
Date:

Aim: To write a java program to create a class and objects of it.

Class and Objects:
Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities. A class is a user defined blueprint or prototype from which objects are created.

Program:

```
class Box {
double width;
double height;
double depth;

Box(){
width=10;
height=20;
depth=15;
}

void volume(){
double vol;
vol = width*height*depth;
System.out.println("Volume is"+vol);
}

public static void main(String args[]) {
Box mybox = new Box();
mybox.volume();
}
```

.

Sample data:

Volume is 3000.0

Result: Thus a java program to create a class and objects of it.

Ex No:11 **String Manipulation**
Date:

Aim: To create a java program to perform string manipulation.

Java Strings:
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc. The java.lang.String class implements Serializable, Comparable and CharSequence interfaces

Program:

```
class StringDemo2 {
public static void main(String args[]) {
String strOb1 = "Annamalai";
String strOb2 = "University";
String strOb3 = strOb1;
System.out.println("Length of strOb1: " +
strOb1.length());
System.out.println("Char at index 3 in strOb1: " +
strOb1.charAt(3));
if(strOb1.equals(strOb2))
System.out.println("strOb1 == strOb2");
else
System.out.println("strOb1 != strOb2");
if(strOb1.equals(strOb3))
System.out.println("strOb1 == strOb3");
else
System.out.println("strOb1 != strOb3");

String substr;
substr = strob1.substring (4,9);
System.out.println(“Susbstring”+substr);

StringBuffer str=new StringBuffer(“Nihal”);
System.out.println(“String reverse”+str.reverse());
```

```
String s3;
s3=strob1.concat(ob2);
System.out.println("Concatenated String is:"+s3);
}
}
```

Sample data:

```
Length of strOb1: 12
Char at index 3 in strOb1: s
strOb1 != strOb2
strOb1 == strOb3
substring =t str
String reverse ialamannA
Concatenated string is: Annamalai University
```

Result: Thus a java program to perform string manipulation is written.

Ex No:12 **Creation of Package**
Date:

Aim: To write a java program to create a package and to use it in a class.

Packages:
While creating a package, you should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package

Program:

// Package class

```
package MyPack;
class Balance {
String name;
double bal;
Balance(String n, double b) {
name = n;
bal = b;
}
void show() {
if(bal<0)
System.out.print("--> ");
System.out.println(name + ": $" + bal);
}
}
```

// Main class

```
import pack.Balance;
class AccountBalance {
public static void main(String args[]) {
Balance current[] = new Balance[3];
current[0] = new Balance("K. J. Fielding", 123.23);
```

```
current[1] = new Balance("Will Tell", 157.02);
current[2] = new Balance("Tom Jackson", -12.33);
for(int i=0; i<3; i++) current[i].show();
}
}
```

Sample data:
K. J. Fielding: $123.23
Will Tell: $157.02
→Tom Jackson: $-12.33

Result: Thus a java program to create a package and to use it in a class is written.

Ex No:13 **Creation of Interface**
Date:

Aim: To write a java program to create an interface and implement it in a class.

Interface:
An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Program:

```
import java.lang.*;

interface shape{
public void draw();
public double getarea();
}

class circle implements shape{
double radius;
public circle(double r){
this.radius=r;
}

public double getarea(){
return Math.PI*radius*radius;
```

```
}

public void draw()
{
System.out.println(“Drawing circle”);
}
}

Class rectangle implements shape{
double width;
double height;
public rectangle (double w, double h)
{
this.width=w;
this.height=h;
}
public double getarea(){
return width*height;
}
Public void draw()
{
System.out.println(“Drawing Rectangle”);
}
}

public class testinterface{
public static void main(String args[])
{
shape s= new circle(10);
s.draw();
System.out.println(“Area = “+s.getarea());
shape s2 = new rectangle(10,10);
s2.draw()
System.out.println(“Area =”+s2.getarea());
}
}
```

Sample data:

Drawing circle
Area=313.1592653589793
Drawing rectangle
Area=100.0

Result:
Thus a java program is written to create an interface and to implement it in a class.

EX:14 **Multithreading**
Date:

Aim: To write a Java program to display addition and multiplication table using multiple threads.

Multithreading:
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process. We create a class that extends the Thread class.

Program:

```
// Create multiple threads.
class Add extends Thread {
public void run(){

System.out.println("Addition thread started");
for(int = l; i<=5; i++)
{
int j=5;
System.out.println(i+"+"+j+"="+(i+j));

}
}
```

```
class Multi extends Thread{
public void run() {
System.out.println("\t\t\t\t Multiplucation thread started “);
for(int = l; i<=5; i++)
{
int j=5;
System.out.println(“\t\t\t\t”+  i+”+”+j+”=”+(i*j));

}
System.out.println("\t\t\t\t Multiplucation thread Terminated“);
}
}

class MultiThread{
public static void main(String args[])
{
Add a = new Add();
Multi m = new Multi();
a.start();
m.start();
}
}
```

Sample Data:

Addition thread started	Multiplication thread started
1+5=6	1*5=5
2+5=7	2*5=10
3+5=8	3*5=15
4+5=9	4*5=20
5+5=10	5*5=25
Addition Thread terminated	Multiplication thread terminated.

Result: Thus a Java program to display addition and multiplication table using multiple threads is written successfully.

Ex:15 **Exception Handling**
Date:

Aim: To write a java program to handle divide by zero exception.

Exception Handling:
Java exception handling is managed via five keywords: try, catch, throw, throws, and finally. Any exception that is thrown out of a method must be specified as such by a throws clause. Any code that absolutely must be executed after a try block completes is put in a finally block.

Program:

```
class Exc2 {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
d = 0;
a = 42 / d;
System.out.println("This will not be printed.");
} catch (ArithmeticException e) { // catch divide-by-zero error
```

```
System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

Sample data:
Division by zero.
After catch statement.

Result: Thus a java program is written to handle divide by zero exception.