CS 6461 OBJECT ORIENTED PROGRAMMING LABORATORY

CS6461 OBJECT ORIENTED PROGRAMMING LABORATORY L T P C 0 0 3 2

OBJECTIVES:

• To get a clear understanding of object-oriented concepts.
• To understand object oriented programming through C++ & JAVA.

LIST OF EXPERIMENTS:

C++:

1. program using functions
   functions with default arguments
   implementation of call by value, address, reference
2. simple classes for understanding objects, member functions & constructors
   classes with primitive data members,
   classes with arrays as data members
   classes with pointers as data members
   classes with constant data members
   classes with static member functions
3. compile time polymorphism
   operator overloading
   function overloading
4. run time polymorphism
   inheritance
   virtual functions
   virtual base classes
   templates
5. file handling
   sequential access
   random access

JAVA:

6. simple java applications
   for understanding references to an instant of a class
   handling strings in JAVA
7. simple package creation
   developing user defined packages in java
8. interfaces
   developing user defined interfaces
   use predefined interfaces
9. threading
   creation of threading in java applications
   multi threading
10. exception handling mechanism in java
    handling predefined exceptions
    handling user defined exceptions

Ex No – 1a(1) **Default arguments in C++**

AIM:

To write a C++ program to find the sum for the given variables using function with default arguments.

ALGORITHM:

1) Start the program.
2) Declare the variables and functions.
3) Give the values for two arguments in the function declaration itself.
4) Call function sum() with three values such that it takes one default arguments.
5) Call function sum() with two values such that it takes two default arguments.
6) Call function sum() with one values such that it takes three default arguments
7) Inside the function sum(), calculate the total.
8) Return the value to the main() function.
9) Display the result.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
float sum(float a,int b=10,int c=15,int d=20);
int a=2,b=3,c=4,d=5;
clrscr();
cout<<"\nsum="<<sum(0);
cout<<"\nsum="<<sum(a,b,c,d);
cout<<"\nsum="<<sum(a,b,c);
cout<<"\nsum="<<sum(a,b);
cout<<"\nsum="<<sum(a);
cout<<"\nsum="<<sum(b,c,d);
getch();
}
float sum(float i, int j, int k, int l)
{
return(i+j+k+l);
}
```

Output:

```
sum=45
sum=14
sum=29
sum=40
sum=47
sum=32
```

RESULT:

Thus, the given program for function with default arguments has been written and executed successfully.

Ex No – 1a(2) **Default arguments in C++**

Aim:

To implement function with default arguments.

Algorithm:

1. Declare the default function.
2. Invoke the default function.

3. Display the result

Source Code:

```
#include<iostream.h>
void printLine(char ='_',int =70);
void main()
{
printLine();
printLine('/');
printLine('*',40);
printLine('R',55);
}
void printLine(char ch, int Repeatcount)
{
int i;
cout<<endl;
for(i=0;i<Repeatcount;i++)
cout<<ch;
}
```

Output:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R R
```

RESULT:

Thus, the given program for function with default arguments has been written and executed successfully

Ex No – 1b(1) **IMPLEMENTATION OF CALL BY VALUE**

AIM:

To write a C++ program to find the value of a number raised to its power that demonstrates a function using call by value.

ALGORITHM:

1) Start the program.
2) Declare the variables.
3) Get two numbers as input
4) Call the function power to which a copy of the two variables is passed .
5) Inside the function, calculate the value of x raised to power y and store it in p.
6) Return the value of p to the main function.
7) Display the result.
8) Stop the program.

SOURCE CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
int x,y;
double power(int, int);
clrscr();
cout<<"Enter x,y:"<<endl;
cin>>x>>y;
cout<<x<<" to the power "<<y <<" is "<< power(x,y);
getch();
}
```

```
double power(int x,int y)
{
double p;
p=1.0;
if(y>=0)
while(y--)
p*=x;
else
while(y++)
p/=x;
return(p);
}
```

Output:

```
ENTER X , Y:
2 3
2 TO THE POWER 3 IS 8
```

RESULT:

Thus, the given program for implementation of call by value has been written and executed successfully.

Ex No – 1b(2) **IMPLEMENTATION OF CALL BY ADDRESS**

AIM:

To write a c++ program and to implement the concept of Call by Address

ALGORITHM:

1. Start the program
2. Include suitable header file
3. Declare a function swap with two pointes variables arguments
4. Declare and initialize the value as two variable in main()
5. Print the value of two variable before swapping
6. Call the swap function by passing address of the two variable as arguments
7. Print the value of two variable after swapping
8. Stop the program

Source Code:

```
#include<iostream.h>
#include<conio.h>
void swap(int *x,int *y);
int main()
{
clrscr();
int i,j;
i=10;
j=20;
cout<<"\n the value of i before swapping is:"<<i;
cout<<"\n the value of i before swapping is:"<<j;
swap (&i,&j);
cout<<"\n the value of i after swapping is:"<<i;
cout<<"\n the value of i after swapping is:"<<j;
getch();
return(0);
}
void swap(int *x,int*y)
{
int temp=*x;
*x=*y;
*y=temp;
}
```

OUTPUT:

The value of i before swapping is: 20
The value of j before swapping is: 10
The value of i after swapping is: 10
The value of j after swapping is: 20

RESULT:

Thus to write a c++ program and to implement the concept of Call by Address was successfully completed.

Ex.No:2a **Classes with primitive data members**

Aim:

To write a program in C++ to prepare a student Record using class and object.

Algorithm:

1. Create a class record.
2. Read the name, Regno ,mark1,mark2,mark3 of the student.
3. Calculate the average of mark as Avg=mark1+mark2+mark3/3
4. Display the student record.
5. Stop the program.

Source Code:

```
#include<iostream.h>
#include<conio.h>
class record
{
public:
char name[20];
int regno;
int marks,m1,m2,m3;
float avg;
void getdata()
{
cout<<"\nenter the name: " ;
cin>>name;
cout<<"enter the regno: ";
cin>>regno;
cout<<"enter the m1,m2,m3: \n";
cin>>m1>>m2>>m3;
}
void calculate()
{
avg=(m1+m2+m3)/3;
}
void display()
{
cout<<"******************\n";
cout<<"\nName: "<<name;
cout<<"\nRegno: "<<regno;
cout<<"\nMark1: "<<m1;
cout<<"\nMark2: "<<m2;
cout<<"\nMark3: "<<m3;
cout<<"\nAvg: "<<avg;
cout<<"******************\n";
}
};
void main()
{
```

```
record r;
clrscr();
r.getdata();
r.calculate();
r.display();
getch();
}
```

Output:

```
Enter the name: Avanthika
Enter the reg no: 1
Enter the m1,m2,m3: 90,90,90
Name: Avantika
Regno: 1
Mark1: 90
Mark2: 90
Mark3: 90
Average:90
```

RESULT:

Thus the C++ program for implementation of classes and objects was created, executed and output was verified successfully.

Ex.No:2B **CLASSES WITH ARRAYS AS DATAMEMBERS**

Aim:

Algorithm:

Program:

```
#include<iostream.h>
Class product
{
    int pro_code[50];
    float pro_price[50];
    int count;
public:
 void cnt()
{count=0;}
Void getproduct();
Void display();
Void displaysum();
Void displayproducts();
};

Void product::getproduct()
{
        cout<<"Enter product Code:";
        cin>>pro_code[count];
        cout<<"Enter product Cost:";
        cin>>pro_price[count];
        count++;
}

Void product::displaysum()
{
        float sum=0;
         for(int i=0;i<count;i++)
```

```
                    sum=sum+pro_price[i];
      cout<<"Total Value:"<<sum<<"\n";
}
Void product::displayproducts()
{
   Cout<<" \nCode                Price\n";
   for(int i=0;i<count;i++)
   {
          cout<<"\n"<<pro_code[i];
         cout<<"  "<<pro_price[i];
   }
Cout<<"\n";
}

int main()
{
         product obj;
         obj.cnt();
         int x;
         do
         {
                  Cout<<"Enter choice\n";
                  Cout<<"\n1.Add a product";
                  Cout<<"\n2.Display a product total value";
                  Cout<<"\n3.Display all products";
                  Cout<<"\n4.Quit";
                  Cin>>x;
                  switch(x)
                  {
                           case1:order.getproduct();
                           case2:order.displaysum();
                           case3:order.displayproducts();
                           case4:break;
                           default:cout<<"|n Invalid choice";
                  }
         }while(x!=5);
return 0;
}
```

RESULT:

Thus the C++ program for implementing arrays as data members was created, executed and output was verified successfully.

Ex.No:2C CLASSES WITH POINTERS AS DATA MEMBERS

Aim:

To implement classes with pointers as data members

Algorithm:

1. Create a class called data
2. Define a function called print and print the variable called a
3. Declare a pointer to object in main function
4. To declare a pointer to data member dereference the pointer to what its point to.
5. display the result

Source Code:

```
#include<iostream.h>
Class data
{
```

```
        Public:
          int a;
void print()
{
        Cout<<"a:"<<a;
}
}

int main()
{
        data d,*dp;
        dp=&d;          //pointer to object
        int data::*ptr=&data::a;        //pointer to data member
       d.*ptr=10;
        d.print();
        dp->*ptr=20;
        dp->print();
}
```

RESULT:

Thus the C++ program for implementing classes with pointers as data members was created, executed and output was verified successfully.

Ex.No:2D CLASSES WITH CONSTANT DATA MEMBER

Aim:

To implement classes with constant data member.

Algorithm:

1.Create the class called test
2.Declare the integer const variable called i
3.Initialization is occurred during constructor
4.i is a const data member in every object its independent copy is present.
5.Initialize using constructor and the value of i cannot be changed
6.Display the result.

Source Code:

```
#include<iostream.h>
class test
{
        const int i;
        public:
        test(int x)
        {
                i=x;
        }
};
int main()
{
        test t(10);
    test s(20);

}
```

RESULT:

Thus the C++ program for implementing classes with constant data members was created, executed and output was verified successfully.

Ex.No:2E **CLASSES WITH STATIC MEMBER FUNCTION**

Aim:

To implement static member function in class.

Algorithm:

1. Create class test with static data member as count.
2. Create a member function to increment the count.
3. Declare the static data member using scope resolution operator.
4. Display the count value.

Source Code:

```
#include<iostream.h>
class test
{
int code;
static int count;
public:
void setcode()
{
cout<<”Object Number:”<<code<<”\n”;
}
};
int test::count;
int main()
{
test t1,t2;
t1.setcount();
t2.setcount();
test::showcount();
test t3;
t1.showcode();
t2.showcode();
t3.showcode();
return(0);
}
```

Output:

```
count 2
count 3
Object Number 1
Object Number 2
Object Number 3
```

RESULT:

Thus to write a c++ program and to implement the concept of Static Data member function was successfully completed.

EX NO 3a(1) COMPILE TIME POLYMORPHISM-UNARY OPERATOR OVERLOADING

Aim

To implement the concept of unary operator overloading by creating a C++ program.

Algorithm

1. Start the Program
2. Create a class named space and declare necessary data members and member functions and operator function as member function
3. The operator unary minus is overloaded to perform the operation of changing sign
4. Define member function getdata(), to get three values that is passed as arguments.
5. The operator function is defined where the sign of values is changed.
6. Function display() is defined where the values is displayed before and after sign change.
7. Stop the program.

Source Code:

```
#include<iostream.h>
#include<conio.h>
class space
{
        int x,y,z;
        public:
                void getdata(int a,int b,int c);
                void display(void);
                void operator-();
};
void space::getdata(int a,int b,int c)
{
        x=a;
        y=b;
        z=c;
}
void space::display(void)
{
        cout<<x<<" ";
        cout<<y<<" ";
        cout<<z<<"\n";
}
void space::operator-()
{
        x=-x;
        y=-y;
        z=-z;
}
void main()
{
        clrscr();
        space s;
        s.getdata(10,-20,30);
        cout<<"s:";
        s.display();
        -s;
        cout<<"s:";
        s.display();
        getch();

}
```

OUTPUT:

```
S = 10      -20       30
S = -10      20      -30
```

RESULT:

Thus to write a c++ program and to implement the concept of unary operator overloading was successfully completed.

EX No 3a(2) **BINARY OPERATOR OVERLAODING**

Aim :

To write a C++ program to implement the concept of Binary operator overloading.

Algorithm:

Step 1: Start the program.

Step 2: Declare the class.

Step 3: Declare the variables and its member function.

Step 4: Using the function getvalue() to get the two numbers.

Step 5: Define the function operator +() to add two complex numbers.

Step 6: Define the function operator –()to subtract two complex numbers.

Step 7: Define the display function.

Step 8: Declare the class objects obj1,obj2 and result.

Step 9: Call the function getvalue using obj1 and obj2

Step 10: Calculate the value for the object result by calling the function operator + and operator -.

Step 11: Call the display function using obj1 and obj2 and result.

Step 12: Return the values.

Step 13: Stop the program

Source Code:

```
#include<iostream.h>
#include<conio.h>
class complex
{
	float x;
	float y;
	public:
		complex(){}
		complex(float real,float imag)
		{
			x=real;y=imag;
		}
		complex operator+(complex c);
		void display(void);
};
complex complex::operator+(complex c)
{
	complex temp;
	temp.x=x+c.x;
	temp.y=y+c.y;
	return(temp);
}
void complex::display(void)
{
	cout<<x<<"+j"<<y<<"\n";
}
void main()
{
	clrscr();
	complex c1,c2,c3;
	c1=complex(2.5,3.5);
	c2=complex(1.6,2.7);
```

```
        c3=c1+c2;
        cout<<"c1=";
        c1.display();
        cout<<"c2=";
        c2.display();
        cout<<"c3=";
        c3.display();
        getch();
}
```

OUTPUT:

C1 = 2.5+j3.5
C2 = 1.6+j2.7
C3 = 4.1+j6.2

RESULT:

Thus to write a c++ program and to implement the concept of binary operator overloading was successfully completed.

Ex.No:3b **Function overloading**

Aim :

To write a C++ program to implement the concept of Function Overloading

Algorithm:

1. Start the program.
2. Create the class with variables and functions.
3. In the main(),declare the variables.
4. Use Volume() function to find volume of cylinder, cube and rectangle.
5. Define the volume() function with necessary arguments for calculation.
6. The values that are passed inside the function call will be matched with the definition part and appropriate calculations are done.
7. All the three volumes are displayed accordingly by display() function.
8. stop the program.

Source Code:

```
#include<iostream.h>
#include<conio.h>
int volume(int s)
{
        return(s*s*s);
}
double volume(double r,int h)
{
        return(3.14*r*r*h);
}
long volume(long l,int b,int h)
{
        return(l*b*h);
}
```

```
void main()
{
        clrscr();
        cout<<"!!!VOLUME!!!\n";
        cout<<volume(10)<<endl;
        cout<<volume(10,20)<<endl;
        cout<<volume(10,20,30)<<endl;
        getch();
}
```

OUTPUT:

1000
157.26
112500

RESULT:

Thus to write a c++ program and to implement the concept of function overloading was successfully completed.

Ex.No:4.a **INHERITANCE**

Aim:

To write a C++ program for implementing the inheritance concept.

Algorithm:

1. Start the process.
2. Define the base class with variables and functions.
3. Define the derived class with variables and functions.
4. Get two values in main function.
5. Define the object for derived class in main function.
6. Access member of derived class and base class using the derived class object.
7. Stop the process.

Program:

```
#include<iostream.h>
#include<conio.h>
class base
{
public:
int x;
void set_x(int n)
{x=n;
}
void show_x()
{
cout<<”\n\t Base class….”;
cout<<”\n\tx=”<<x;
}
};
class derived:public base
{
int y;
public:
void set_y(int n)
{
y=n;
}
void show_xy()
{
cout<<”\n\n\t derived class…”;
```

```
cout<<”\n\tx=”<<x;
cout<<”\n\ty=”<<y;
}
};
void main()
{
derived obj;
int x,y;
clrscr();
cout<<”\n enter the value of x: ”;
cin>>x;
cout<<”\n enter the value of y: ”;
cin>>y;
obj.set_x(x);//inherits base class
obj.set_y(y);//acess member of derived class
obj.show_x();//inherits base class
obj.show_xy();//acess member of derived class
getch();
}
```

OUTPUT:

```
enter the value of x 10
enter the value of y 20
base class….
x=10
derived class…..
x=10
y=20
```

RESULT:

Thus the C++ program for inheritance was created, executed and output was verified successfully.

Ex No 4B **VIRTUAL FUNCTION**

Aim:

To write a C++ program to implement the concept of Virtual functions

Algorithm:

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 derv1,derv2 using a base class called base and define a function called display in the respective classes.

4.Declare a base class pointer in main function.

5Declare objects for derv1 and derv2 classes respectively.

6.Assign a derv1 and derv2 obj to base pointer

7.Now display function shows the derv1 and derv2 class’ function respectively.

Source Code:

```
#include<iostream.h>
#include<conio.h>
class base
{
        public:
                virtual void display()
                {
                        cout<<"Base class display is called\n";
                }
};
class derv1 : public base
```

```
{
        public:
                void display()
                {
                        cout<<"\nDerv1's display called\n";
                }
};
class derv2:public base
{
        public:
                void display()
                {
                        cout<<"\nDerv2's display called\n";
                }
};
void main()
{
        clrscr();
        base *ptr;
        derv1 d1;
        derv2 d2;
        ptr=&d1;
        ptr->display();
        ptr=&d2;
        ptr->display();
        getch();
}
```

Output:

Derv1's display called
Derv2's display called

RESULT:

Thus the C++ program for virtual function was created, executed and output was verified successfully

EX. NO:4.C **VIRTUAL BASE CLASS**

AIM:

To write a c++ program to implement the concept of virtual base class.

ALGORITHM:

1. Start the program
2. Include suitable header files
3. Create a base class student and sports
4. In the base class student define the function void get number and put number
5. In the base class sports define the function void get score and void put score
6. Derive a class test form base student and define the function get mark and put mark
7. Derive a class result from test and sports class and define function void display
8. Get the value for get number ,get marks and get score function through main function
9. Call the display function in class result
10. Stop the program

Source Code:

```
#include<iostream.h>
class student
{
protected:
int roll_number;
public
```

```
void get_number(int a)
{
roll_number=a;
}
void put_number(void)
{
cout<<"ROLL NO:"<<roll_number<<"\n";
}
};
class test :public virtual student
{
protected:
float part1,part2;
public:
void get_marks(float x,float y)
{
part1=x;
part2=y;
}
void put_marks(void)
{
cout<<"marks obtined:"<<"\n"
<<"part1="<<part1<<"\n"
<<"part2="<<part2<<"\n";
}
};
class sports: public virtual student
{
protected:
float score;
public:
void get_score(float s)
{
score=s;
}
void put_score(void)
{
cout<<"sports wt:"<<score<<"\n\n";
}
};
class result:public test,public sports
{
float total;
public:
void display(void);
};
void result::display(void)
{
total=part1+part2+score;
put_number();
put_marks();
put_score();
cout<<"total score:"<<total<<"\n";
}
int main()
{
```

```
result student_1;
student_1.get_number(678);
student_1.get_marks(30.5,25.5);
student_1.get_score(7.0);
student_1.display();
return 0;
}
```

OUTPUT:

```
Roll no: 678
Marks obtained:
Part1=30.5
Part2=25.5
Sports wt:7
Total score: 63
```

RESULT:

Thus to write a c++ program for virtual base class was successfully completed.

EX. NO:4.D **FUNCTION TEMPLATE**

Aim:

To write a c++ program for swapping two values using function templates

Algorithm:

1.Start the program
2.Create a class with templates.
3.Create a class for sort functions.
4.Swap the values for using bubble sort
5.Display the result.

Source Code:

```
#include<iostream.h>
#include<conio.h>
template<class T>
void print(T *a,int n)
{
cout<<a[0];
for(int i=1;i<n;i++)
cout<<","<<a[i];
}
template<class T>
void bubsort(T *a, int n)
{
for(int i=1;i<n;i++)
for(int j=n-1;j>=i;j--)
if(a[j-1]>a[j])
{
T temp=a[j-1];
a[j-1]=a[j];
a[j]=temp;
}
}
template<class T>
void inssort(T *a,int n)
{
for(int i=1;i<n;i++)
{
T temp=a[i];
for(int j=i;j>0&&a[j-1]>temp;j--)
a[j]=a[j-1];
```

```
a[j]=temp;
}
}
void main()
{
int a[]={12,11,15,13,17,14,16,19,18};
char b[]={'b','h','j','c','f','e','r','a','s'};
clrscr();
cout<<"BEFORE BUBBLE SORTING\n";
print(a,9);
bubsort(a,9);
cout<<"\nAFTER BUBBLE SORTING\n";
print(a,9);
cout<<"\n\nBEFORE INSERTION SORTING\n";
print(b,9);
inssort(b,9);
cout<<"\nAFTER INSERTION SORTING\n";
print(b,9); getch(); }
```

Output:

BEFORE BUBBLE SORTING
12, 11, 15,13,17,14,16,19,18
AFTER BUBBLE SORTING
11, 12, 13,14,15,16,17,18,19

BEFORE INSERTION SORTING
b, h, j, c, f, e, r, a, s
AFTER INSERTION SORTING
a, b, c, e, f, h, j, r, s

RESULT:

Thus the C++ program for function template was created, executed and output was verified successfully

EX.NO 5A. FILE HANDLING-SEQUENTIAL ACCESS

Aim:

To implement a file handling concept using sequential access.

Algorithm:

1.Start the process
2.Get the input string
3.Write the input string char by char into a file called “Text” using put() function
4.Read the input string char by char from the file called “Text” using get() function
5.Display the result

Source Code:

```
#include<iostream.h>
#include<fstream.h>
#include<string.h>

int main()
{
        Char str[80];
Cout<<”Enter a string:”;
Cin>>str;
Int len=strlen(str);
Fstream file;
File.open(“TEXT”,ios::in|ios::out);
For(int i=0;i<len;i++)
```

```
        File.put(str[i]);
File.seekg(0);
Char ch;
While(file)
{
        File.get(ch);
        Cout<<ch;
}
return 0;
}
```

RESULT:

Thus the C++ program for file handling- sequential access concept was created, executed and output was verified successfully

EX.NO 5B. FILE HANDLING-RANDOM ACCESS

Aim:

To implement file handling concept using random access

Algorithm:

1.Start the process

2.Get the input file called ExampleFile and check the file's presence

3.Seek the input file to a particular(random) location and get the specified output

4.Display the result

Source Code:

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>

int main()
{
   ifstream IFileObject("D:\\ExampleFile.txt");
   if (!IFileObject)
   {
      cout << "File cannot be opened" << endl;
      exit(1);
   }
   string lineread;

   IFileObject.seekg(9); // move to 9th character
   // print the remaining line
   getline(IFileObject, lineread);
   cout << lineread << endl;

   getchar(); // Just To Prevent Screen from Disappearing
   cin.get(); // Just To Prevent Screen from Disappearing
}
```

Output:

ing to open a file called text

RESULT:

Thus the C++ program for file handling- random access concept was created, executed and output was verified successfully

EX.NO 6A. **SIMPLE JAVA PROGRAM**

Aim:

To write a java program to find volume of box.

Algorithm:

1. Declare the class box.
2. Declare height,width and depth in write data.
3. Declare another class example.
4. Create an object and access write data.
5. Print the output.

PROGRAM:

```
 class Box
{
 int width=2,height=2,depth=2,a;
 void writedata()
 {
 System.out.print("volume of box is:");
 System.out.println(width*height*depth);
 }
}
public class example
{
 public static void main(String args[])
 {
 Box mybox1=new Box();
 mybox1.writedata();
 }
 }
```

OUTPUT:

```
c:\jdk1.3\bin>java example
volume of box is 8
```

RESULT:

Thus a java program is executed for finding volume of box.

EX.NO 6B. **HANDLING STRINGS IN JAVA**

Aim:

To write a java program to handle string using string function.

Algorithm:

1.Start the program
2.Declare the class called stringUse
3.Initialize two strings and using the two strings manipulate string operation
4.Perform the operation like concat(),length(),chatAt(),startsWith(),endsWith(),etc
5.Display the result

Source Code:

```
import java.util.*;
public class stringuse
{
public static void main (String args [])
{
```

```
String fname="sam";
String lname="daniel";
String name= fname+lname;
int len = name.length();
System.out.println ("fname:"+fname+"lname:"+lname);
System.out.println ("length of name"+len);
String nname =name;
System.out.println ("copied name:"+ nname);
System.out.println ("lowercase name:"+name.toLowerCase()+"\n"+"upper case
name:"+name.toUpperCase());
char f = name. charAt(0);
System.out.println ("char at position 0:"+f);
boolean b=fname.endsWith ("v");
System.out.println ("checks starts with:"+b);
if(nname == name)
System.out.println ("same memory location == true");
System.out.println ("same mamory location == false");
String n1= nname.substring (0,3);
System.out.println ("n1="+n1);
boolean bool=name.equals (nname);
System.out.println ("name equality with case checking:"+bool);
boolean bool1=name.equalsIgnoreCase(nname);
System.out.println ("name equality without case checking:"+bool1);
}
}
```

OUTPUT:

```
C:\jdkl.5.0\bin> javac stringuse.java
C:\jdkl.5.0\bin> java  stringuse
Fname:sam
Lname:Daniel
Length of name:9
Copied name:samdaniel
Lowercase name:samdaniel
Uppercase name:SAMDANIEL
Char at position 0:S
Check starts with:false
Same memory location==true
N1=same
Name equality with case checking: true
Name equality without case checking:true
```

RESULT:

Thus a java program is executed for handling strings.

EX.NO 7. PACKAGES

Aim:

To write a java program to find the account balance using package.

Algorithm:

1. Declare class balance and pass parameterized constructor.
2. Initialize name and balance as n and b.
3. Use member function to check balance less than 0 and print if yes.
4. Declare another class account import package.
5. Access show function and print result.

Source Code:

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


Import pack.*;
class accountbalance
   {
   public static void main(String arg[])
   {
   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();
   }
   }
```

OUTPUT:

```
C:\jdk1.3\bin>java accountbalance
-->Tom jackson:$-12.33
```

RESULT:

Thus java program for finding the balance amount using packages is verified.

EX.NO 8A. INTERFACES-DEVELOPING USER DEFINED INTERFACES

Aim:

To write a java program using user defined interface concept.

Algorithm:

1.Start the program
2.Declare the interface called area and declare the function called compute in it.
3.Define the class called rectangle and implement the interface area.
4.Define the class called circle and implement the interface area.
5.Display the result.

Source Code:

```
Import java.io.*;
Interface area
{
final static float pi=3.14F;
float compute(float x,float y);
}
class rectangle implements area
{
public float compute(float x,float y)
{
return(x*y);
}}
class circle implements area1
{
public float compute(float x,float y)
{
return(pi*x*y);
}
}
class area
{
public static void main(String args[])
{
rectangle rect=new rectangle();
circle cir=new circle();
area1 ar;
ar=rect;
System.out.println("area of rectangle="+ar.compute (10,20));
ar=cir;
System.out.println("area of circle="+ar.compute(10,0));
}}
```

OUTPUT:

```
C:\jdk1.3\bin>javac interfacetest.java
C:\jdk1.3\bin>java interfacetest
Area of rectangle = 200.0
Area of circle=314.0
```

RESULT:

Thus a java program is executed for the concept of user defined interface

EX.NO 8B. INTERFACES-DEVELOPING PRE DEFINED INTERFACES

Aim:

To write a java program using pre defined interface concept.

Algorithm:

1.Create a class called person which implements predefined interface comparable
2.Declare data members name and lastname in a constructor.
3.Define the member function getName() and getLastName()
4.compareTo() is the method of comparable interface
5.Create a List object called mylist in order to store the objects.
6.Sort the lists by using sort()
7.Display the sorted list.

Source Code:

```
class Person implements Comparable{
public String name;
public String lastName;

public Person(String name, String lastName){
  this.name=name;
  this.lastName=lastName;
}
public String getName(){
  return name;
}
public String getLastName(){
  return lastName;
}

public int compareTo(Person p){
  return this.name.compareTo(p.getName);
}
public static void main(String arg[]){
  List myList = new ArrayList();
  myList.add(new Person("Robert","USA"));
  myList.add(new Person("Andy","UK"));
  myList.add(new Person("Harish","India"));
  Collections.sort(myList);
  for(Person person : myList){
  System.out.println("My name is "+person.getName());
  }
}
}
```

OUTPUT:

```
C:\jdk1.3\bin>javac person.java
C:\jdk1.3\bin>java person
My name is Andy
My name is Harish
My name is Robert
```

RESULT:

Thus a java program is executed for the concept of pre defined interface

EX.NO 9A. CREATION OF THREADING IN JAVA

Aim:

To implement multi threading and exception handling using java program.

Algorithm:

1.Declare class Th1 extend from thread .
2.Declare try, catch with data.
3.Declare another 2 classes Th2,Th3 define method.
4.Declare main class Th demo and create object T1, T2 and t3 for each class. Access it.
5.Display the result.

Source Code:

```
import java io.*;
class Th1 extends Thread
{
```

```
public void run()
{
try
{
thread.sleep(1000);
System.out .println(“name:elan”);
System.out println(“age:19”);
}
catch(InterruptedException i)
{
}
}
}
class Th2 extends Thread
                        {
public void run()
{
try
{
thread.sleep(2000);
System.out .println(“class:b.tech I>T”);
System.out println(“col:vec”);
}
catch(InterruptedException i)
{
}
}
}
class Th3 extends Thread
                        {
public void run()
{
try
{
thread.sleep(3000);
System.out .println(“place:chennai”);
System.out println(“area:kattankulathur”);
}
catch(InterruptedException i)
{
}
}
}
class thdemo
{
  public static void main(String args[ ])
{
Th1 t1=new Th1( );
t1.start( );
Th2 t2=new Th2( );
t2.start( );
Th3 t3=new Th3( );
t3.start( );
}
}
```

OUTPUT:

Name: elan
Age: 19
Class: b.tech I.T
Col: vec
Place: chennai
Area: kattankulathur

RESULT:

Thus threading using java program was verified and implemented.

EX.NO 9B. MULTITHREADING IN JAVA

Aim:

To write a java program on multithreading concept.

Algorithm:

1.Start the program
2.Create a main thread called ThreadDemo and starts its execution
3.Invoke the child thread class called newThread
3.newThread() invokes the superclass constructor and starts the child thread execution.
4.Main thread and child thread runs parallelly.
5.Display the result.

Source Code:

```
import java.io.*;

class Newthread extends Thread

{

Newthread()

{

super("DemoThread");

System.out.println("Child Thread:"+this);

start();

}

public void run()

{

try

{

for(int i=5;i>0;i--)

{

System.out.println("child thread:"+i);

Thread.sleep(500);

}

}

catch(InterruptedException e)

{

System.out.println("Child Interrupted");

}
```

```
System.out.println("Exiting ChildThread");
}
}
class ThreadDemo
{
public static void main(String args[])
{
new Newthread();
try
{
for(int i=5;i>0;i--)
System.out.println("main thread:"+i);
Thread.sleep(1000);
}
catch(InterruptedException e)
{
System.out.println("Main thread Interrupted");
}
System.out.println("Main thread exiting");
}
}
```

OUTPUT:

```
C:\jdk1.5.0\bin>javac ThreadDemo.java
C:\jdk1.5.0\bin>java ThreadDemo
Child Thread:Thread[DemoThread,5,main]
main thread:5
child thread:5
main thread:4
main thread:3
main thread:2
main thread:1
child thread:4
child thread:3
Main thread exiting
child thread:2
child thread:1
Exiting ChildThread
```

RESULT:

Thus multithreading using java program was verified and implemented.

EX.NO 10A HANDLING PREDEFINED EXCEPTION

Aim:

To implement the pre defined exception concept in java

Algorithm:

1.Start the program

2.Create the called error.

3.Declare and Initialize the data members.

4.Use the try and catch block to handle the exception

5.If the exception exists, corresponding catch block will be executed or else control goes out of the catch block .

6.Display the result.

Source Code:

```
class error
{
public static void main(String args[])
{
int a=10;
int b=5;
int c=5;
int x,y;
try
{
x=a/(b-c);
}
catch(ArithmeticException e)
{
System.out.println("division by zero");
}
y=a/(b+c);
System.out.println("y"+y);
}
}
```

OUTPUT

```
division by zero
y1
```

RESULT:

Thus pre defined exception using java program was verified and implemented.

EX.NO 10B HANDLING USERDEFINED EXCEPTION

Aim:

To implement the user defined exception concept in java

Algorithm:

1.Create a userdefined exception class called MyException

2.Throw an exception of user defined type as an argument in main()

3.Exception is handled using try, catch block
4.Display the user defined exception

Source Code:

```
import java.io.*;
class JavaException{
   public static void main(String args[]){
  try{
       throw new MyException(2);
       // throw is used to create a new exception and throw it.
  }
 catch(MyException e){
    System.out.println(e) ;
 }
}
}
class MyException extends Exception{
   int a;
   MyException(int b) {
     a=b;
   }
   public String toString(){
     return ("Exception Number =  "+a) ;
  }
}
```

OUTPUT

```
C:\jdk1.5.0\bin>javac JavaException.java
C:\jdk1.5.0\bin>java JavaException
Exception Number 2
```

RESULT:

Thus user defined exception using java program was verified and implemented.