# ***Assignment on Inheritance***

#### Q 1: Discuss the ways using suitable examples in which inheritance promotes software reuse, saves time during program development and helps prevent errors

### Ans : Inheritance allows developer to create derived classes that reuse code declared already in a base class. Avoiding the duplication of common functionality b/w several class by building an inheritance hierarchy to contain the class can save developers a considerable amount of time .Similarly , placing common functionality in a single base class ,rather than duplicating the code in multiple unrelated classes ,helps prevent the same errors from appearing in multiple source code files.If several classes can contain duplicate code containing an error, the software developer has to send time correcting each source- code file with the errors .However ,if these classes take advantages of inheritance ,and the error occurs in common functionality of the base class ,the software developer needs to modify only the base class’s code

Ex : # include<iostream>

Using namespace std;

#### class Vehicle // Base class

#### {

#### public:

#### Vehicle()

#### {

#### cout<<"the base of vehicle...."<<endl;

#### }

#### void brakes()

#### {

#### cout<<"Breaks of vehicle"<<endl;

#### }

#### void getcapacity()

#### {

#### cout<<"Capacity of vehicle"<<endl;

#### }

#### };

#### class Bus : public Vehicle //derived class

#### {

#### public:

#### Bus()

#### {

#### Cou << "The base of bus class"<<endl;

#### }

#### };

#### class Car : public Vehicle //derived class

#### {

#### public:

#### Car()

#### {

#### cout<<"The base of car class"<<endl;

#### }

#### };

#### int main()

#### {

#### Vehicle v;

#### v.brakes();

#### v.getcapacity();

#### Car c;

#### c.brakes();

#### c.getcapacity();

#### Bus b;

#### b.brakes();

#### b.getcapacity();

#### }

#### Q 2 : Draw an inheritance hierarchy for students at a university. Use Student as the base class of the

#### hierarchy, then include classes UndergraduateStudent and GraduateStudent that derive from Student .

#### Continue to extend the hierarchy as deep (i.e., as many levels) as possible. For example, Freshman,

#### Junior and Senior might derive from UndergraduateStudent , and DoctoralStudent and MastersStudent might derive from GraduateStudent . After drawing the hierarchy, discuss the relationships that exist

#### between the classes. Also, implement the above hierarchy using C++ code.

Ans : Student

Student

# UndergraduateStudent

# GraduateStudent

#### Freshman

#### Junior

## Senior

#### MastersStudent

#### DoctoralStudent

## Sophomore

# This hierarchy contains many “is -a” (inheritance) relationships. An Ungraduate students is a student. A Graduate student is a student too.Each of the class Freshman ,Sophomore ,Junior and Senior is an UngraduateStudent and is a Student .Each of the classes DoctoralStudent and MastersStudent is a GraduateStudent and is a Student

#### MastersStudent

### DoctoralStudent

# 

#### MastersStudent

### DoctoralStudent

#### Ex : #include<iostream>

#### using namespace std;

#### class Student

#### {

#### public:

#### Student()

#### {

#### cout<<"The derived class....."<<endl;

#### }

#### };

#### class UndergraduateStudent : public Student

#### {

#### public:

#### UndergraduateStudent()

#### {

#### cout<<"The base class of Student class"<<endl;

#### }

#### };

#### class Freshman :public UndergraduateStudent

#### {

#### public:

#### Freshman()

#### {

#### cout<<"The base class of UndergraduateStudent"<<endl;

#### }

#### };

#### class Sophomore :public UndergraduateStudent

#### {

#### public:

#### Sophomore()

#### {

#### cout<<"The base class of UndergraduateStudent"<<endl;

#### }

#### };

#### class Senior :public UndergraduateStudent

#### {

#### public:

#### Senior()

#### {

#### cout<<"The base class of UndergraduateStudent"<<endl;

#### }

#### };

#### class Junior :public UndergraduateStudent

#### {

#### public:

#### Junior()

#### {

#### cout<<"The base class of UndergraduateStudent"<<endl;

#### }

#### };

#### class GraduateStudent :public Student

#### {

#### public:

#### GraduateStudent()

#### {

#### cout<<"The base class of student"<<endl;

#### }

#### };

#### class DoctoralStudent :public GraduateStudent

#### {

#### public:

#### DoctoralStudent()

#### {

#### cout<<"The base class of GraduateStudent"<<endl;

#### }

#### };

#### class MastersStudent :public GraduateStudent

#### {

#### public:

#### MastersStudent()

#### {

#### cout<<"The base class of GraduateStudent"<<endl;

#### }

#### };

#### int main()

#### {

#### Student s;

#### UndergraduateStudent u;

#### GraduateStudent g;

#### Freshman f;

#### Sophomore sp;

#### Senior se;

#### Junior j;

#### DoctoralStudent d;

#### MastersStudent m;

#### }

#### Q4 : . Create a base Shape class which is inherited by TwoDShape and ThreeDShape and these shapes are

#### further extended by their derived classes like square, rectangle, sphere etc. You should provide

#### necessary methods like calculateArea, calculateVolume in these shapes.

# Ans :

#### #include<iostream>

#### using namespace std;

#### // GENERAL (REGULAR) SHAPE

#### class Shape

#### {

#### protected:

#### double length;

#### double area;

#### double breadth;

#### double height;

#### public:

#### double getLength()

#### {

#### return length;

#### }

#### double getBreath()

#### {

#### return breadth;

#### }

#### double getArea()

#### {

#### return area;

#### }

#### double getHeight()

#### {

#### return height;

#### }

#### };

#### // 2D SHAPE

#### class Shape\_2D:public Shape

#### {

#### protected:

#### double volume;

#### public:

#### double getVolume()

#### {

#### return volume;

#### }

#### };

#### class Square:public Shape\_2D

#### {

#### public:

#### Square(double a=0)

#### {

#### length=a;

#### area=length\*length;

#### }

#### };

#### class Rectangle:public Shape\_2D

#### {

#### public:

#### Rectangle(double a=0,double b=0,double c=0)

#### {

#### length=a;

#### breadth=b;

#### height=c;

#### area=length\*breadth;

#### volume=length\*breadth\*height;

#### }

#### };

#### class Shape\_3D:public Shape

#### {

#### protected:

#### double volume;

#### public:

#### double getVolume()

#### {

#### return volume;

#### }

#### };

#### class Sphere:public Shape\_3D

#### {

#### public:

#### Sphere(double a=0)

#### {

#### length=a;

#### area=4\*3.14\*(length/2)\*(length/2);

#### volume=4/3\*3.14\*(length/2)\*(length/2)\*(length/2);

#### }

#### };

#### int main()

#### {

#### Square s(2);

#### cout<<"Length of square : " << s.getLength()<<endl;

#### cout<< "Area of square :" << s.getArea() <<endl<<endl;

#### Rectangle r(2,2,2);

#### cout << "Length of rectangle : " << s.getLength()<<endl;

#### cout << "Breadth of rectangle : " << s.getLength()<<endl;

#### cout << "Height of rectangle : " << s.getLength()<<endl;

#### cout<< "Area of rectangle :" << s.getArea() <<endl;

#### cout<< "Volume of rectangle :" << s.getVolume()<<endl<<endl;

#### Sphere sphr(2);

#### cout << "Length of sphere : " << s.getLength()<<endl;

#### cout<< "Area of sphere :" << s.getArea() <<endl;

#### cout<< "Volume of sphere :" << s.getVolume() <<endl;

#### return 0;

#### }

#### Q 3 : Draw an inheritance hierarchy for classes Quadrilateral, Trapezoid, Parallelogram, Rectangle and

#### Square. Use Quadrilateral as the base class of the hierarchy. Make the hierarchy as deep as possible.

#### Also, implement the above hierarchy using C++ code and create functions to calculate perimeter and

#### area of the shape.

#### Ans : Inheritance Hierarchy

## Quadilateral

## Trapezoid

## Square

## Parallelogram

## Rectangle

Ex : #include<iostream>

using namespace std;

class Quadilateral

{

protected:

double length1;

double length2;

double area;

double breadth1;

double breadth2;

double height;

double perimeter;

public:

double getLength1()

{

return length1;

}

double getLength2()

{

return length2;

}

double getBreadth1()

{

return breadth1;

}

double getBreadth2()

{

return breadth2;

}

double getArea()

{

return area;

}

double getHeight()

{

return height;

}

double getPerimeter()

{

return perimeter;

}

};

// 2D SHAPE

class Trapeziod:public Quadilateral

{

public:

Trapeziod(double l1=0,double b1=0,double c=0,double l2 = 0 ,double b2 =0)

{

length1=l1;

breadth1=b1;

breadth2 = b2;

height=c;

length2=l2;

area=((breadth1+breadth2)/2)\*height;

perimeter=length1+length2+breadth1+breadth2;

}

};

class Parallelogram:public Quadilateral

{

public:

Parallelogram(double a=0,double b=0,double c=0)

{

length1=a;

breadth1=b;

height=c;

area=breadth1\*height;

perimeter=2\*(length1+breadth1);

}

};

class Rectangle:public Quadilateral

{

public:

Rectangle(double a=0,double b=0,double c=0)

{

length1=a;

breadth1=b;

height=c;

area=length1 \* breadth1;

perimeter= 2\*(length1+breadth1);

}

};

class Square:public Quadilateral

{

public:

Square(double a=0)

{

length1=a;

area=length1\*length1;

perimeter=4\*length1;

}

};

int main()

{

Trapeziod t(2,2,3,3,4);

cout << "Length1 of Trapezoid : " << t.getLength1()<<endl;

cout << "Length2 of Trapezoid : " << t.getLength2()<<endl;

cout << "Breadth1 of Trapezoid : " << t.getBreadth1()<<endl;

cout << "Breadth2 of Trapezoid : " << t.getBreadth2()<<endl;

cout << " Height of Trapezoid : " <<t.getHeight()<<endl;

cout<< "Area of Trapezoid :" << t.getArea() <<endl;

cout<< "Perimeter of Trapezoid :" << t.getPerimeter()<<endl;

cout<<"---------------------------------------------------------------------"<<endl;

Parallelogram p(2,2,3);

cout << "Length1 of Parallelogram : " << p.getLength1()<<endl;

cout << "Breadth1 of Parallelogram : " << p.getBreadth1()<<endl;

cout << "Height of Parallelogram : " << p.getHeight()<<endl;

cout<< "Area of Parallelogram :" << p.getArea() <<endl;

cout<< "Perimeter of Parallelogram :" << p.getPerimeter()<<endl;

cout<<"---------------------------------------------------------------------"<<endl;

Square s(2);

cout<<"Length of square : " << s.getLength1()<<endl;

cout<< "Area of square :" << s.getArea() <<endl<<endl;

cout<< "Perimeter of Square :" << s.getPerimeter()<<endl<<endl;

cout<<"---------------------------------------------------------------------"<<endl;

Rectangle r(2,3);

cout << "Length1 of rectangle : " << r.getLength1()<<endl;

cout << "Breadth1 of rectangle : " << r.getBreadth1()<<endl;

cout<< "Area of rectangle :" << r.getArea() <<endl;

cout<< "Perimeter of rectangle :" << r.getPerimeter()<<endl<<endl;

return 0;

}

#### Q 5 : . Create any base class as per your wish and extend that into other derived class. You should be able to

#### understand the calling order or default constructors, parameterised constructors and destructors of all

#### the classes. You should implement hybrid inheritance in this example.

### Ans : #include <iostream>

### using namespace std;

### class A

### {

### public:

### int x;

### 

### };

### class B : public A

### {

### public:

### B() //default const called

### {

### x = 10;

### cout<<"the value of x : "<<x<<endl;

### }

### ~B()

### {

### cout<<"Derived class destructor"<<endl;

### }

### };

### class C

### {

### public:

### int y;

### C(int v=10) //parametrized const called

### {

### y=v;

### cout<<"The value of y :"<<y<<endl;

### }

### ~ C()

### {

### cout<<"Base class destructor "<<endl;

### }

### };

### class D : public B, public C //D is derived from class B and class C

### {

### public:

### void sum()

### {

### cout << "Sum = " << x + y<<endl;

### }

### ~D()

### {

### cout<<"Derived class destructor"<<endl;

### }

### };

### int main()

### {

### D obj1; //object of derived class D

### obj1.sum();

### return 0;

### }

### 

#### OUTPUT : the value of x : 10

#### The value of y :10

#### Sum = 20

#### Derived class destructor

#### Base class destructor

#### Derived class destructor