**LAB 6 : Polymorphism**

**1.Ship, CruiseShip, and CargoShip Classes**

Design a Ship class that has the following members:

• A member variable for the name of the ship (a string)

• A member variable for the year that the ship was built (a string)

• A constructor and appropriate accessors and mutators

• A virtual print function that displays the ship’s name and the year it was built.

Design a CruiseShip class that is derived from the Ship class. The CruiseShip class should have the following members:

• A member variable for the maximum number of passengers (an int)

• A constructor and appropriate accessors and mutators

• A print function that overrides the print function in the base class. The CruiseShip class’s print function should

display only the ship’s name and the maximum number of passengers.

Design a CargoShip class that is derived from the Ship class. The CargoShip class should have the following members:

• A member variable for the cargo capacity in tonnage (an int).

• A constructor and appropriate accessors and mutators.

• A print function that overrides the print function in the base class. The CargoShip class’s print function should

display only the ship’s name and the ship’s cargo capacity.

Demonstrate the classes in a program that has an array of Ship pointers. The array elements should be initialized with the

addresses of dynamically allocated Ship, CruiseShip, and CargoShip objects. The program should then step through the

array, calling each object’s print function.

**Ans//**

/\*Creating objects of different classes and perform the actions according to question\*/

#include <iostream>

#include <string>

using namespace std;

//declaring the class

class Ship{

protected:

string name;

int modelYear;

public:

//default constructor

Ship()

{

name = "";

modelYear = 1990;

}

//Parameterised constructor

Ship(string n, int y)

{

name = n;

modelYear = y;

}

//accessor functions

string getName()

{

return name;

}

int getYear()

{

return modelYear;

}

//mutator functions

void setName(string n)

{

name = n;

}

void setYear(int y)

{

modelYear = y;

}

//virtual print() function

virtual void print()

{

cout <<endl<< "Name: " << name << endl;

cout << "Build year: " << modelYear<< endl;

}

};//end of class

//Declaring child class

class CruiseShip : public Ship

{

private:

int maxPass;

public:

//default constructor set maximum number of passengers to 0

CruiseShip() : Ship()

{

maxPass = 0;

}

//Parameterised constructor

CruiseShip(int maxNum, string n, int y) :

Ship(n, y)

{

maxPass = maxNum;

}

//accessor function

int getMaxPassengers()

{

return maxPass;

}

//mutator function

void setMaxPassengers(int maxNum)

{

//validate input using while loop

if(maxNum<=0){

cout << " number of passengers must be greateer than zero";

}

else

maxPass = maxNum;

}

//override print() function

virtual void print()

{

cout << "Name: " << name << endl;

cout << "Maximum Passenger(in number): " << maxPass << endl;

}

};//end of child class

//Declaring another child class

class CargoShip : public Ship{

private:

int maximum\_Tonnage;

public:

CargoShip() : Ship(){

maximum\_Tonnage = 0;

}

//overloaded constructor

CargoShip(int maxCapacity, string n, int y) :

Ship(n, y){

if(maxCapacity <= 0){

cout << "Invalid Capacity!! Maximum capacity must be greater than Zero";

exit(0);

}

else

maximum\_Tonnage = maxCapacity;

}

//accessor function

int getCapacity()

{

return maximum\_Tonnage;

}

//mutator function

void setCapacity(int maxCapcity)

{

while(maxCapcity < 0)

{

cout << "Maximum capacity cannot be negative number!";

cout << " Enter again: ";

cin >> maxCapcity;

}

maximum\_Tonnage = maxCapcity;

}

virtual void print()

{

cout << "Name: " << name << endl;

cout << "Max Capacity: " << maximum\_Tonnage << endl;

}

};//end of child class

//main program

int main()

{

cout << "Creating Array of Ship Pointers To Reference Ship, CruiseShip and CargoShip objects...\n";

Ship \*ships[3] = {

new Ship(),

new CruiseShip(),

new CargoShip()

};//declaring the array

cout << "Displaying the data for each ship...\n";

//mutator functions to store new data into the objects

ships[0]->setName("TYCON\_19");

ships[0]->setYear(2008);

//using pointer to the derived class to access

CruiseShip \*cruiseship = static\_cast<CruiseShip\*>(ships[1]);

cruiseship->setName("TITNIC");

cruiseship->setMaxPassengers(1990);

//using pointer to the derived class to access

CargoShip \*cargoship = static\_cast<CargoShip\*>(ships[2]);

cargoship->setName("HAMMERHEAD CARGO");

cargoship->setCapacity(200000);

//using while to display

int i = 0;

while(i<3){

ships[i]->print();

i++;

cout << endl;

}

return 0;

}

**2.Pure Abstract Base Class Project**

Define a pure abstract base class called BasicShape. The BasicShape class should have the following members: Private Member Variable: area, a double used to hold the shape’s area. Public Member Functions: getArea. This function should return the value in the member variable area. calcArea. This function should be a pure virtual function. Next, define a class named Circle. It should be derived from the BasicShape class. It should have the following members: Private Member Variables: centerX, a long integer used to hold the x coordinate of the circle’s center. centerY, a long integer used to hold the y coordinate of the circle’s center. radius, a double used to hold the circle’s radius. Public Member Functions: constructor—accepts values for centerX, centerY, and radius. Should call the overridden calcArea function described below. getCenterX—returns the value in centerX. getCenterY—returns the value in centerY. calcArea—calculates the area of the circle (area = 3.14159 \* radius \* radius) and stores the result in the inherited member area. Next, define a class named Rectangle. It should be derived from the BasicShape class. It should have the following members: Private Member Variables: width, a long integer used to hold the width of the rectangle. length, a long integer used to hold the length of the rectangle. Public Member Functions: constructor—accepts values for width and length. Should call the overridden calcArea function described below. getWidth—returns the value in width. getLength—returns the value in length. calcArea—calculates the area of the rectangle (area = length \* width) and stores the result in the inherited member area. After you have created these classes, create a driver program that defines a Circle object and a Rectangle object. Demonstrate that each object properly calculates and reports its area

**ANS//**

/\* program to define a pure abstract base class called BasicShape and perform actions according to question need's\*/

#include <iostream>

#define PI 3.142 //defining the constant PI

using namespace std;

//declaring class

class BasicShape{

protected:

double area;

public:

//member function to return area

double getArea()

{

return area;

}

//pure virtual function to calculate area

virtual void calcArea() = 0;

};//end of class BasciShape

//Declaring derived class

class Circle : public BasicShape

{

private:

double centerX;

double centerY;

double radius;

public:

//Parameterised constructor

Circle(double x, double y, double r)

{

//store values in member variables

centerX = x;

centerY = y;

radius = r;

//calling the overridden calcArea() function

calcArea();

}

//accessor functions

double getCenterX(){

return centerX;

}

double getCenterY(){

return centerY;

}

//override calcArea() function

virtual void calcArea(){

area = PI \* radius \* radius;

}

};//end of derived class

//Declaring another derived class

class Rectangle : public BasicShape{

private:

float width;

float length;

public:

//Parameterised constructor

Rectangle(float w, float l){

//check for invalid input

// if(b < 0 || l < 0){

// cout << "Error! Dimensions cannot be negative numbers!\n ";

// }

//store values in member variables

width = w;

length = l;

//call overridden calcArea() function

calcArea();

}

//accessor functions

float getWidth(){

return width;

}

float getLength(){

return length;

}

//overridden calcArea() function

virtual void calcArea(){

area = length \* width;

}

};//end of rectangle derived class

int main()

{

//creating a cir and rect object of circle and rectangle respectively

cout << "Circle class object, with radius of 10\n";

Circle cir(100, 150, 10);

cout << "Rectangle class object, with dimensions of 4 and 5...\n";

Rectangle rect(8, 5);

//display data

cout << "\nDisplaying data of Circle\n";

cout << "Center X: " << cir.getCenterX()<<"m" << endl;

cout << "Center Y: " << cir.getCenterY()<<"m" << endl;

cout << "Area: " << cir.getArea()<<"m2" << endl;

//display data

cout << "\nDisplaying data of Rectangle\n";

cout << "Width: " << rect.getWidth() <<"m"<< endl;

cout << "Length: " << rect.getLength() <<"m"<< endl;

cout << "Area: " << rect.getArea() <<"m2"<< endl;

return 0;

}