Şafak AKINCI
152120141003

02.01.2017

Experiment 8
Polymorphism

Objectives

* How polymorphism makes programming more convenient and systems more extensible.
* The distinction between abstract and concrete classes and how to create abstract classes.
* To use runtime type information (RTTI)
* How C++ implements virtual functions and dynamic binding.
* How virtual destructors ensure that all appropriate destructors run on an object.
* How polymorphism makes programming more convenient and systems more extensible.
* The distinction between abstract and concrete classes and how to create abstract classes.

Prelab Activities

Programming Output

```
Phylum: Mollusca    Genus: Crassostrea


Phylum: Mollusca    Genus: Crassostrea
Phylum: Mollusca    Genus: Crassostrea
Mollusca
```

Correct The Code

1-	1	:	`virtual void print () const = 0;`	:	logic error
2-	6	:	`VirginiaOyster(string genusString):Oyster(genusString)`	:	compilation error
	12	:	`~VirginiaOyster()`	:	logic and compilation error
	14	:		:	logic error
3-	13	:	`class DerivedClass : public BaseClass`	:	compilation error
4-	public	:	`void printName () const ;`	:	compilation error

Lab Exercises

Lab Exercise 1 – Polymorphic Media Players

```
/*****************************************
 * Shape.h                               *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef Shape_h
#define Shape_h

#include <iostream>         //To use standart input and output functions which are cin and cout.
#include <string>           //To declare string type variables.
using namespace std;        //To use std namespace for string (std::string)


//Shape Class is an ABSTRACT CLASS because of the PURE VIRTUAL FUNCTIONS.
class Shape
{
    //Protected variables ARE ACCESSIBLE out of the class.
protected:
```

```
    string m_color;
    bool m_visibility;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of Shape Class.
    Shape(string color ="Black", bool visibility =false);

    //Virtual Destructor Function of Shape Class.
    virtual ~Shape() =0;

    //Returns the shape's color as string.
    string getColor() const;

    //Returns the m_visibility as boolean.
    bool getVisibility() const;

    //Sets the m_color.
    void setColor(string color);

    //Sets the m_visibility.
    void setVisibility(bool visibility);

    //Those are pure virtual functions, they will implement in derived classes.
    virtual double CalculateSurfaceArea() = 0;
    virtual string GetShapeDescription() = 0;

};

#endif /* Shape_h */

/*****************************************
 * Shape.cpp                             *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "Shape.h"          //To know function prototypes of Shape Class.

//Constructor Function of Shape Class.
Shape::Shape(string color, bool visibility):m_color(color),m_visibility(visibility)
{
```

```
}

//Virtual Destructor Function of Shape Class.
Shape::~Shape()
{

}

//Returns the shape's color as string.
string Shape::getColor() const
{
    return m_color;
}

//Returns the m_visibility as boolean.
bool Shape::getVisibility() const
{
    return m_visibility;
}

//Sets the m_color.
void Shape::setColor(string color)
{
    m_color = color;
}

//Sets the m_visibility.
void Shape::setVisibility(bool visibility)
{
    m_visibility = visibility;
}

/*****************************************
 * TwoDShape.h                           *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef TwoDShape_h
#define TwoDShape_h

#include "Shape.h"      //To derive TwoDShape Class from Shape Class.

//TwoDShape is ABSTRACT CLASS because of the pure virtual functions.
class TwoDShape:public Shape
```

```
{
    //Protected variables ARE ACCESSIBLE out of the class.
    //Center of mass stands for
    //                          the center for circle,
    //                          the intersection point of diagonals for rectangle.
protected:
    double m_centerofmass[2];

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of TwoDShape Class.
    TwoDShape(double* centerofmass, string color ="Black", bool visibility =false);

    //Virtual Destructor Function of TwoDShape Class.
    virtual ~TwoDShape();


    //Those are pure virtual functions, they will implement in derived classes.
    virtual double CalculateSurfaceArea() = 0;
    virtual string GetShapeDescription() = 0;
};

#endif /* TwoDShape_h */
```

```
/*****************************************
 * TwoDShape.cpp                         *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "TwoDShape.h"      //To know function prototypes of TwoDShape Class.


//Constructor Function of TwoDShape Class.
//TwoDShape is derived from Shape Class, so Shape Class' Constructor should be called.
TwoDShape::TwoDShape(double* centerofmass, string color, bool visibility):Shape(color, visibility)
{
    for(int i=0; i<2; i++)
        m_centerofmass[i] = centerofmass[i];
}

//Virtual Destructor Function of TwoDShape Class.
TwoDShape::~TwoDShape()
{
```

```
}

/*****************************************
 * ThreeDShape.h                         *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/


#ifndef ThreeDShape_h
#define ThreeDShape_h

#include "Shape.h"      //To derive ThreeDShape Class from Shape Class.

//ThreeDShape is ABSTRACT CLASS because of the pure virtual functions.
class ThreeDShape:public Shape
{
    //Protected variables ARE ACCESSIBLE out of the class.
protected:
    double m_centerofmass[3];

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of TwoDShape Class.
    ThreeDShape(double* centerofmass, string color = "Black", bool visibility =false);

    //Virtual Destructor Function of ThreeDShape Class.
    virtual ~ThreeDShape();

    //Those are pure virtual functions, they will implement in derived classes.
    virtual double CalculateSurfaceArea() = 0;
    virtual string GetShapeDescription() = 0;

};

#endif /* ThreeDShape_h */

/*****************************************
 * ThreeDShape.cpp                       *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
```

```
 ****************************************/

#include "ThreeDShape.h"        //To know function prototypes of ThreeDShape Class.

//ThreeDShape is derived from Shape Class, so Shape Class' Constructor should be called.
ThreeDShape::ThreeDShape(double* centerofmass, string color, bool visibility):Shape(color, visibility)
{
    for(int i=0; i<3; i++)
        m_centerofmass[i] = centerofmass[i];
}


//Virtual Destructor Function of ThreeDShape Class.
ThreeDShape::~ThreeDShape()
{

}

/****************************************
 * Circle.h                             *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 8: Polymorphism           *
 ****************************************/

#ifndef Circle_h
#define Circle_h

#include "TwoDShape.h"          //To derive Circle Class from TwoDShape Class.

//Circle class is derived from TwoDShape Class.
class Circle:public TwoDShape
{
    //Private variables ARE NOT ACCESSIBLE out of the class.
private:
    double radius;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of Circle Class.
    Circle(double radius, double* centerofmass, string color ="Black", bool visibility =false);

    //Destructor Function of Circle Class.
    ~Circle();
```

```
    //Returns the circle's area.
    double CalculateSurfaceArea();

    //Returns the shape's description as string.
    string GetShapeDescription();
};

#endif /* Circle_h */

/*****************************************
 * Circle.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "Circle.h"         //To know function prototypes of Circle Class.


//Constructor Function of Circle Class.
//Circle Class is derived from TwoDShape Class, so TwoDShape Class' Constructor should be called.
Circle::Circle(double radius, double* centerofmass, string color, bool
visibility):TwoDShape(centerofmass,color,visibility)
{
    (*this).radius = radius;
}


//Destructor Function of Circle Class.
Circle::~Circle()
{

}

//Returns the circle's area.
double Circle::CalculateSurfaceArea()
{
    return 3.14*radius*radius;
}

//Returns the shape's description as string.
string Circle::GetShapeDescription()
{
    //m_visibility is a boolean variable, we can not add it to string directly.
    string s_visibility;
    if(m_visibility==true)
```

```
        s_visibility = "true";
    else
        s_visibility = "false";

    string description =
    "Circle:\t"+m_color
    +"\nVisibility:\t"+s_visibility
    +"\nCenter of Mass:\t<"+to_string(m_centerofmass[0])+","+to_string(m_centerofmass[1])
    +">\nRadius:\t"+to_string(radius)
    +"\nArea:\t"+to_string((*this).CalculateSurfaceArea())+"\n";

    return description;
}

/*****************************************
 * CustomRectangle.h                     *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef CustomRectangle_h
#define CustomRectangle_h

#include "TwoDShape.h"          //To derive CustomRectangle Class from TwoDShape Class.

//CustomRectangle Class is derived from TwoDShape Class as public.
class CustomRectangle:public TwoDShape
{
    //Private variables ARE NOT ACCESSIBLE out of the class.
private:
    double width;
    double height;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of CustomRectangle Class.
    CustomRectangle(double width, double height, double* centerofmass, string ="Black", bool visibility =false);

    //Destructor Function of CustomRectangle Class.
    ~CustomRectangle();

    //Returns the rectangle's area.
    double CalculateSurfaceArea();
```

```
    //Returns the shape's description as string.
    string GetShapeDescription();
};

#endif /* CustomRectangle_h */

/*****************************************
 * CustomRectangle.cpp                   *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "CustomRectangle.h"            //To know function prototypes of CustomRectangle Class.

//Constructor Function of CustomRectangle Class.
//CustomRectangle Class is derived from TwoDShape Class, so TwoDShape Class' Constructor should be called.
CustomRectangle::CustomRectangle(double width, double height, double* centerofmass, string color, bool
visibility):TwoDShape(centerofmass,color,visibility),width(width),height(height)
{

}

//Destructor Function of CustomRectangle Class.
CustomRectangle::~CustomRectangle()
{

}

//Returns the rectangle's area.
double CustomRectangle::CalculateSurfaceArea()
{
    return width*height;
}

//Returns the shape's description as string.
string CustomRectangle::GetShapeDescription()
{
    //m_visibility is a boolean variable, we can not add it to string directly.
    string s_visibility;
    if(m_visibility==true)
        s_visibility = "true";
    else
        s_visibility = "false";

    string description =
```

```
    "CustomRectangle:\t"+m_color
    +"\nVisibility:\t"+s_visibility
    +"\nCenter of Mass:\t<"+to_string(m_centerofmass[0])+","+to_string(m_centerofmass[1])
    +">\nWidth:\t"+to_string(width)
    +"\nHeight:\t"+to_string(height)
    +"\nArea:\t"+to_string((*this).CalculateSurfaceArea())+"\n";

    return description;
}

/*****************************************
 * RectangularPrism.h                    *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef RectangularPrism_h
#define RectangularPrism_h

#include "ThreeDShape.h"        //To derive RectangularPrism Class from ThreeDShape Class.

//RectangularPrism Class is derived from ThreeDShape Class as public.
class RectangularPrism:public ThreeDShape
{
    //Private variables ARE NOT ACCESSIBLE out of the class.
private:
    double width;
    double height;
    double depth;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of RectangularPrism Class.
    RectangularPrism(double width, double height, double depth, double* centerofmass, string color ="Black", bool
visibility =false);

    //Destructor Function of RectangularPrism Class.
    ~RectangularPrism();


    //Returns the rectangle prism's area.
    double CalculateSurfaceArea();

    //Returns the shape's description as string.
```

```
    string GetShapeDescription();
};

#endif /* RectangularPrism_h */

/*****************************************
 * RectangularPrism.cpp                  *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "RectangularPrism.h"       //To know function prototypes of RectangularPrism Class.

//Constructor Function of RectangularPrism Class.
//RectangularPrism Class is derived from ThreeDShape Class, so ThreeDShape Class' Constructor should be called.
RectangularPrism::RectangularPrism(double width, double height, double depth, double* centerofmass, string color,
bool visibility):ThreeDShape(centerofmass,color,visibility),width(width),height(height),depth(depth)
{

}

//Destructor Function of RectangularPrism Class.
RectangularPrism::~RectangularPrism()
{

}

//Returns the rectangle prism's area.
double RectangularPrism::CalculateSurfaceArea()
{
    return width*height*depth;
}

//Returns the shape's description as string.
string RectangularPrism::GetShapeDescription()
{
    //m_visibility is a boolean variable, we can not add it to string directly.
    string s_visibility;
    if(m_visibility==true)
        s_visibility = "true";
    else
        s_visibility = "false";

    string description =
    "RectangularPrism:\t"+m_color
```

```
    +"\nVisibility:\t"+s_visibility
    +"\nCenter of Mass:\t<"+to_string(m_centerofmass[0])+","+to_string(m_centerofmass[1])
+","+to_string(m_centerofmass[2])
    +">\nWidth:\t"+to_string(width)
    +"\nHeight:\t"+to_string(height)
    +"\nDepth:\t"+to_string(depth)
    +"\nArea:\t"+to_string((*this).CalculateSurfaceArea())+"\n";

    return description;
}

/*****************************************
 * Sphere.h                              *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef Sphere_h
#define Sphere_h

#include "ThreeDShape.h"            //To derive Sphere Class from ThreeDShape Class.

//Sphere Class is derived from ThreeDShape Class as public.
class Sphere:public ThreeDShape
{
    //Private variable IS NOT ACCESSIBLE out of the class.
private:
    double radius;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of Sphere Class.
    Sphere(double radius, double* centerofmass, string color ="Black", bool visibility = false);

    //Destructor Function of Sphere Class.
    ~Sphere();

    //Returns the sphere's area.
    double CalculateSurfaceArea();

    //Returns the shape's description as string.
    string GetShapeDescription();

};
```

```cpp
#endif /* Sphere_h */

/*****************************************
 * Sphere.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "Sphere.h"         //To know function prototypes of Sphere Class.

//Constructor Function of Sphere Class.
//Sphere Class is derived from ThreeDShape Class, so ThreeDShape Class' Constructor should be called.
Sphere::Sphere(double radius, double* centerofmass, string color, bool
visibility):ThreeDShape(centerofmass,color,visibility),radius(radius)
{

}

//Destructor Function of Sphere Class.
Sphere::~Sphere()
{

}

//Returns the sphere's area.
double Sphere::CalculateSurfaceArea()
{
    return 4*3.14*radius*radius;
}

//Returns the shape's description as string.
string Sphere::GetShapeDescription()
{
    //m_visibility is a boolean variable, we can not add it to string directly.
    string s_visibility;
    if(m_visibility==true)
        s_visibility = "true";
    else
        s_visibility = "false";

    string description =
    "Sphere:\t"+m_color
    +"\nVisibility:\t"+s_visibility
    +"\nCenter of Mass:\t<"+to_string(m_centerofmass[0])+","+to_string(m_centerofmass[1])
```

```
+","+to_string(m_centerofmass[2])
    +">\nRadius:\t"+to_string(radius)
    +"\nArea:\t"+to_string((*this).CalculateSurfaceArea())+"\n";


    return description;
}

/*****************************************
 * ShapePlacer.h                         *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef ShapePlacer_h
#define ShapePlacer_h

#include "Shape.h"              //To use Shape type variables.
#include <vector>               //To declare vector type array.

//Shape description must be in the following form.
// <shape type>  -<color>-
// <Visibility>  -<Center of Mass> -
// <Additional Properties>

class ShapePlacer
{
    //Private variable IS NOT ACCESSIBLE out of the class.
private:

    //m_shapeList vector holds lots of shapes.
    vector <Shape* > m_shapeList;

    //Public variables ARE ACCESSIBLE out of the class.
public:

    //Constructor Function of ShapePlacer Class.
    ShapePlacer();

    //Destructor Function of ShapePlacer Class.
    ~ShapePlacer();

    //Given newShape the type of Shape* will be added to vector.
    void AddNewShape(Shape* newShape);
```

```
    //Prints all shapes which are held the vector to console.
    void VisualizeAllShapes();

    //Finds the most proper shape according to the given area and returns it as Shape*.
    Shape* FindTheMostProperShape(double area);

};


#endif /* ShapePlacer_h */

/*****************************************
 * ShapePlacer.cpp                       *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "ShapePlacer.h"        //To know function prototypes of ShapePlacer Class.


//Constructor Function of ShapePlacer Class.
ShapePlacer::ShapePlacer()
{

}

//Destructor Function of ShapePlacer Class.
ShapePlacer::~ShapePlacer()
{

}

//Given newShape the type of Shape* will be added to vector.
void ShapePlacer::AddNewShape(Shape* newShape)
{
    m_shapeList.push_back(newShape);
}

//Prints all shapes which are held the vector to console.
void ShapePlacer::VisualizeAllShapes()
{
    for(int i=0; i<m_shapeList.size(); i++)
        cout<<m_shapeList[i]->GetShapeDescription()<<endl;

}
```

```
//Finds the most proper shape according to the given area and returns it as Shape*.
Shape* ShapePlacer::FindTheMostProperShape(double area)
{
    double* differenceArray = new double [m_shapeList.size()];

    //There should be one more array that will hold the differences between shape's area and given area.
    for(int i=0; i<m_shapeList.size(); i++)
    {
        differenceArray[i] = m_shapeList[i]->CalculateSurfaceArea() - area;

        //There can be negative numbers in the differenceArray, we should take absolute of them.
        if(differenceArray[i] < 0 )
            differenceArray[i] *= -1;
    }

    //Then, we should find the minimum number in the array, actually INDEX its number.
    double minimumNumberInTheArray  = differenceArray[0];
    int index = 0;

    for(int i=0; i<m_shapeList.size(); i++)
        if( differenceArray[i]< minimumNumberInTheArray )
        {
            minimumNumberInTheArray = differenceArray[i];
            index = i;
        }

    //The minimum difference and its index are found!

    return m_shapeList[ index ];
}

/**********************************************
 * TestMain.cpp                               *
 **********************************************
 * IDE : Xcode                                *
 * Author : Şafak AKINCI                      *
 * Experiment 8: Polymorphism                 *
 **********************************************/

#include "Circle.h"                 //To create Circle type objects.
#include "ShapePlacer.h"            //To create ShapePlacer type objects.
#include "CustomRectangle.h"        //To create CustomRectangle type objects.
#include "RectangularPrism.h"       //To create RectangularPrism type objects.
#include "Sphere.h"                 //To create Sphere type objects.

//Holds the center of mass for two dimensional objects which are Circle and CustomRectangle.
```

```
double two_centerOfMass[2] = { 2,2 };

//Holds the center of mass for two dimensional objects which are Sphere and RectangularPrism.
double three_centerOfMass[3] = { 3,3,3 };

int main ()
{
    //Holds the shapes. (Container)
    ShapePlacer test_shapePlacer;

    //Shapes will be created.
    Shape* shapeForTest;

    //Two Dimensional Shapes
    shapeForTest = new Circle(3      ,two_centerOfMass,"Orange",true);
    test_shapePlacer.AddNewShape(shapeForTest);

    shapeForTest = new CustomRectangle(4,5,two_centerOfMass,"Blue",true);
    test_shapePlacer.AddNewShape(shapeForTest);


    //Three Dimensional Shapes
    shapeForTest = new Sphere(4     ,three_centerOfMass,"Red",true);
    test_shapePlacer.AddNewShape(shapeForTest);

    shapeForTest = new RectangularPrism(3,4,5,three_centerOfMass,"Dark Blue",true);
    test_shapePlacer.AddNewShape(shapeForTest);


    //Then print all of them to console.
    test_shapePlacer.VisualizeAllShapes();


    //Find the most proper shape according to the given area, in this case it is 50.
    shapeForTest = test_shapePlacer.FindTheMostProperShape(50);
    cout<<"Given area is:\t\t50\n"
        <<"The most proper shape is:\t\t"<<shapeForTest->GetShapeDescription()<<endl;

}//end main ()
```

AT QUIZ

```
/*****************************************
 * Player.h                              *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
```

```
 * Experiment 8: Polymorphism            *
 ****************************************/

#ifndef Player_h
#define Player_h

#include <iostream>         //To use standart input and output functions.
using namespace std;        //To declare string type variables under the std namespace.



//Classes that contain at least one pure virtual function are known as abstract base classes.


//It can be used to create pointers to it, and take advantage of all its polymorphic abilities.
//And can actually be dereferenced when pointing to objects of derived (non-abstract) classes.

//Player Class is an ABSTRACT CLASS because of the PURE VIRTUAL FUNCTIONS.
//Abstract base classes cannot be used to instantiate objects.
//This is a class that can only be used as base class, and thus is allowed to have virtual member functions without
definition (known as pure virtual functions). The syntax is to replace their definition by =0 (an equal sign and a
zero).
class Player
{
    //Protected variables can be accessed out of the class.
protected:
    int maxVolumeLevel;
    int volumeLevel;
    string mediaName;

    //Public variables can be accessed out of the class.
public:

    //Constructor Function of Player Class.
    Player (int maxVolumeLevel = 100);

    //Prints that player is opened.
    void Open();

    //Adds given amount to volumeLevel and prints it.
    virtual void VolumeUp(int amount);

    //Subtracts given amount from volumeLevel and prints it.
    virtual void VolumeDown(int amount);

    //Prints that media is playing.
    void Play();
```

```
    //Prints that media is pausing.
    void Pause();

    //Prints that media is stopping.
    void Stop();

    //Prints that it is closed.
    void Close();

    //Those are pure virtual functions, It means derived classes have to contain the definition of them.
    virtual void Forward() = 0;
    virtual void BackWard() = 0;
    virtual void EjectMedia() = 0;
    virtual void MounthMedia(const string& mediaName) = 0;


    //Destructor Function of Player Class.
    //Virtual destructors ensure that all appropriate destructors run on an object.
    virtual ~Player();
};

#endif /* Player_h */
```

```
/*****************************************
 * Player.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "Player.h"         //To know function prototypes of Player Class.

//Constructor Function of Player Class.
Player::Player(int maxVolumeLevel):maxVolumeLevel(maxVolumeLevel)
{
    volumeLevel = 35;
}

//Prints that player is opened.
void Player::Open()
{
    cout<<"Player:\tOpened..."<<endl;
}

//Adds given amount to volumeLevel and prints it.
```

```
void Player::VolumeUp(int amount)
{
    cout<<"Player:\tVolume Level:\t"<<(volumeLevel += amount)<<endl;
}

//Subtracts given amount from volumeLevel and prints it.
void Player::VolumeDown(int amount)
{
    cout<<"Player:\tVolume Level:\t"<<(volumeLevel -= amount)<<endl;
}

//Prints that media is playing.
void Player::Play()
{
    cout<<"Player:\tPlaying the media "<<mediaName<<endl;
}

//Prints that media is pausing.
void Player::Pause()
{
    cout<<"Player:\tPausing the media "<<mediaName<<endl;
}

//Prints that media is stopping.
void Player::Stop()
{
    cout<<"Player:\tStopping the media "<<mediaName<<endl;
}

//Prints that player is closed.
void Player::Close()
{
    cout<<"Player:\tClosed..."<<endl;
}

//Destructor Function of Player Class.
Player::~Player()
{

}

/*****************************************
 * MusicPlayer.h                         *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
```

```
 ****************************************/

#ifndef MusicPlayer_h
#define MusicPlayer_h

#include "Player.h"     //To derive MusicPlayer Class from Player Class.

//MusicPlayer Class is derived from Player Class.
class MusicPlayer:public Player
{
//Protected variables can be accessed out of the class.
protected:
    string* supportedFormats;
    int supportedFormatCount;

//Public variables can be accessed out of the class.
public:
    //Constructor Function of MusicPlayer Class.
    MusicPlayer(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);

    //Destructor Function of MusicPlayer Class.
    //Virtual destructors ensure that all appropriate destructors run on an object.
    virtual ~MusicPlayer();

/***Those are PURE virtual functions in PLAYER CLASS.***/

    //Prints that Music Player does not support the forward operation.
    void Forward();

    //Prints that Music Player does not support the backward operation.
    void BackWard();

    //Prints that media is ejecting.
    void EjectMedia();

    //Prints that Music Player supports the given mediaName and initialize mediaName which is the protected member of
Player Class.
    void MounthMedia(const string& mediaName);
};


#endif /* MusicPlayer_h */

/****************************************
 * MusicPlayer.cpp                      *
 ****************************************
```

```
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 8: Polymorphism           *
 ****************************************/

#include "MusicPlayer.h"        //To know function prototypes of MusicPlayer.h

//Constructor Function of MusicPlayer Class.
//MusicPlayer Class is derived from Player Class, so this constructor function has to call the constructor of Player
Class.
MusicPlayer::MusicPlayer(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):Player(maxVolumeLevel)
{
    (*this).supportedFormats = supportedFormats;
    (*this).supportedFormatCount = supportedFormatCount;
}

//Prints that Music Player does not support the forward operation.
void MusicPlayer::Forward()
{
    cout<<"Music Player:\tDoes not support the forward operation."<<endl;
}

//Prints that Music Player does not support the backward operation.
void MusicPlayer::BackWard()
{
    cout<<"Music Player:\tDoes not support the backward operation."<<endl;
}

//Prints that media is ejecting.
void MusicPlayer::EjectMedia()
{
    cout<<"Music Player:\tEjecting the media : "<<mediaName<<endl;
}

//Prints that Music Player supports the given mediaName and initialize mediaName which is the protected member of
Player Class.
void MusicPlayer::MounthMedia(const string &mediaName)
{
    (*this).mediaName = mediaName;
    cout<<"Music Player:\tTrying to mount the media..."<<endl;
    cout<<"Music Player:\tMedia is supported and playable..."<<endl;
}

//Destructor Function of MusicPlayer Class.
MusicPlayer::~MusicPlayer()
{
```

```
}
/*****************************************
 * VideoPlayer.h                         *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/
#ifndef VideoPlayer_h
#define VideoPlayer_h

#include "Player.h"         //To derive VideoPlayer Class from Player Class.

//VideoPlayer Class is derived from Player Class.
class VideoPlayer:public Player
{
//Public variables can be accessed out of the class.
public:
    int supportedFormatCount;
    string* supportedFormats;

/***Those are PURE virtual functions in PLAYER CLASS.***/
    //Prints that Video Player does not support the forward operation.
    void Forward();

    //Prints that Video Player does not support the backward operation.
    void BackWard();

    //Prints that media is ejecting.
    void EjectMedia();

    //Prints that Video Player supports the given mediaName and initialize mediaName which is the protected member of
Player Class.
    void MounthMedia(const string& mediaName);

    //Constructor Function of VideoPlayer Class.
    VideoPlayer(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);

    //Deconstructor Function of VideoPlayer Class.
    //Virtual destructors ensure that all appropriate destructors run on an object.
    virtual ~VideoPlayer();
};

#endif /* VideoPlayer_h */
```

```cpp
/*****************************************
 * VideoPlayer.cpp                       *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "VideoPlayer.h"        //To know function prototypes of VideoPlayer Class.

//Constructor Function of VideoPlayer Class.
//VideoPlayer Class is derived from Player Class, so this constructor function has to call the constructor of Player
Class.
VideoPlayer::VideoPlayer(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):Player(maxVolumeLevel)
{
    (*this).supportedFormatCount = supportedFormatCount;
    (*this).supportedFormats = supportedFormats;
}


//Prints that Video Player does not support the forward operation.
void VideoPlayer::Forward()
{
    cout<<"Video Player:\tDoes not support the forward operation."<<endl;
}

//Prints that Video Player does not support the backward operation.
void VideoPlayer::BackWard()
{
    cout<<"Video Player:\tDoes not support the backward operation."<<endl;
}

//Prints that media is ejecting.
void VideoPlayer::EjectMedia()
{
    cout<<"Video Player:\tEjecting the media : "<<mediaName<<endl;
}


//Prints that Video Player supports the given mediaName and initialize mediaName which is the protected member of
Player Class.
void VideoPlayer::MounthMedia(const string& mediaName)
{
    (*this).mediaName = mediaName;
    cout<<"Video Player:\tTrying to mount the media..."<<endl;
    cout<<"Video Player:\tMedia is supported and playable..."<<endl;
```

```
}

//Deconstructor Function of VideoPlayer Class.
VideoPlayer::~VideoPlayer()
{

}

/*****************************************
 * Walkman.h                             *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/
#ifndef Walkman_h
#define Walkman_h

#include "MusicPlayer.h"        //To derive Walkman Class from MusicPlayer Class.

//Walkman Class is derived from MusicPlayer Class.
//It inherits the members of MusicPlayer Class.
class Walkman:public MusicPlayer
{

//Public variables can be accessed out of the class.
public:
    //Constructor Function of Walkman Class.
    Walkman(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);

    //Destructor Function of Walkman Class.
    ~Walkman();

    //Prints that closing the carriage.
    void EjectMedia();

};

#endif /* Walkman_h */

/*****************************************
 * Walkman.cpp                           *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/
```

```cpp
#include "Walkman.h"        //To know function prototypes of Walkman Class.

//Constructor Function of Walkman Class.
//Walkman Class is derived from MusicPlayer Class, so this constructor function has to call the constructor of
MusicPlayer Class.
Walkman::Walkman(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):MusicPlayer(maxVolumeLevel,supportedFormats,supportedFormatCount)
{

}

//Destructor Function of Walkman Class.
Walkman::~Walkman()
{

}

//Prints that closing the carriage.
void Walkman::EjectMedia()
{
    MusicPlayer::EjectMedia();
    cout<<"Walkman:\tClosing the carriage"<<endl;
}
```

```cpp
/*****************************************
 * Ipod.h                                *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef IPod_h
#define IPod_h

#include "MusicPlayer.h"            //To derive IPod Class from MusicPlayer Class.

//IPod Class is derived from MusicPlayer Class.
class IPod:public MusicPlayer
{

//Public variables can be accessed out of the class.
public:
    //Constructor Function of IPod Class.
    IPod(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);
```

```
    //Destructor Function of IPod Clas.
    ~IPod();

    //Prints that music is forwarding.
    void Forward();

    //Prints that music is backwarding.
    void BackWard();

    //Prints that equalizer is adjusting.
    void VolumeUp(int amount);

    //Prints that equalizer is adjusting.
    void VolumeDown(int amount);
};


#endif /* IPod_h */

/*****************************************
 * Ipod.cpp                              *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "IPod.h"       //To know function prototypes of IPod Class.


//Constructor Function of IPod Class.
//IPod Class is derived from MusicPlayer Class, so this constructor function has to call the constructor of
MusicPlayer Class.
IPod::IPod(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):MusicPlayer(maxVolumeLevel,supportedFormats,supportedFormatCount)
{
    volumeLevel = 50;
}

//Destructor Function of IPod Clas.
IPod::~IPod()
{

}

//Prints that music is forwarding.
void IPod::Forward()
```

```
{
    cout<<"IPod:\tForward the music "<<mediaName<<endl;
}

//Prints that music is backwarding.
void IPod::BackWard()
{
    cout<<"IPod:\tBackward the music "<<mediaName<<endl;
}

//Prints that equalizer is adjusting.
void IPod::VolumeUp(int amount)
{
    Player::VolumeUp(amount);
    cout<<"IPod:\tAdjusting the equalizer"<<endl;
}

//Prints that equalizer is adjusting.
void IPod::VolumeDown(int amount)
{
    Player::VolumeDown(amount);
    cout<<"IPod:\tAdjusting the equalizer"<<endl;
}
```

```
/*****************************************
 * VLCVideoPlayer.h                      *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/
#ifndef VLCVideoPlayer_h
#define VLCVideoPlayer_h

#include "VideoPlayer.h"        //To derive VLCVideoPlayer Class from VideoPlayer Class.

//VLCVideoPlayer Class is derived from VideoPlayer Class.
class VLCVideoPlayer:public VideoPlayer
{

//Public variables can be accessed out of the class.
public:

    //Constructor Function of VLCVideoPlayer Class.
    VLCVideoPlayer(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);

    //Prints that music is forwarding.
```

```
    void Forward();

    //Prints that music is backwarding.
    void BackWard();

    //Prints that equalizer is adjusting.
    void VolumeDown(int amount);

    //Prints that equalizer is adjusting.
    void VolumeUp (int amount);

    //Destructor Function of VLCVideoPlayer Class.
    ~VLCVideoPlayer();
};

#endif /* VLCVideoPlayer_h */

/*****************************************
 * VLCVideoPlayer.cpp                    *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "VLCVideoPlayer.h"         //To know function prototypes of VLCVideoPlayer Class.

//Constructor Function of VLCVideoPlayer Class.
//VLCVideoPlayer Class is derived from VideoPlayer Class, so this constructor function has to call the constructor of
VideoPlayer Class.
VLCVideoPlayer::VLCVideoPlayer(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):VideoPlayer(maxVolumeLevel,supportedFormats,supportedFormatCount)
{
    volumeLevel = 50;
}

//Prints that music is forwarding.
void VLCVideoPlayer::Forward()
{
    cout<<"VLCVideoPlayer:\tForward the music"<<mediaName<<endl;
}

//Prints that music is backwarding.
void VLCVideoPlayer::BackWard()
{
    cout<<"VLCVideoPlayer:\tBackward the music"<<mediaName<<endl;
}
```

```cpp
//Prints that equalizer is adjusting.
void VLCVideoPlayer::VolumeUp(int amount)
{
    cout<<"VLCVideoPlayer:\tAdjusting the equalizer..."<<endl;

    //Given amount is added to the current volumeLevel.
    VideoPlayer::VolumeUp(amount);
}

//Prints that equalizer is adjusting.
void VLCVideoPlayer::VolumeDown(int amount)
{
    cout<<"VLCVideoPlayer:\tAdjusting the equalizer..."<<endl;

    //Given amount is subtracted from the current volumeLevel.
    VideoPlayer::VolumeDown(amount);
}

//Destructor Function of VLCVideoPlayer Class.
VLCVideoPlayer::~VLCVideoPlayer()
{

}
```

```cpp
/*****************************************
 * BugiVideoPlayer.h                     *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#ifndef BugiVideoPlayer_h
#define BugiVideoPlayer_h

#include "VideoPlayer.h"        //To derive BuigVideoPlayer Class from VideoPlayer Class.

//BugiVideoPlayer is derived from VideoPlayer Class.
class BugiVideoPlayer:public VideoPlayer
{

//Public variables can be accessed out of the class.
public:

    //Constructor Function of BugiVideoPlayer Class.
    BugiVideoPlayer(int maxVolumeLevel, string supportedFormats[], int supportedFormatCount);
```

```
    //Destructor Function of BugiVideoPlayer Class.
    ~BugiVideoPlayer();

    //Prints that media player crushed!
    void EjectMedia();

};

#endif /* BugiVideoPlayer_h */

/*****************************************
 * BugiVideoPlayer.cpp                   *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 8: Polymorphism            *
 *****************************************/

#include "BugiVideoPlayer.h"        //To know function prototypes of BugiVideoPlayer Class.

//Constructor Function of BugiVideoPlayer Class.
//BugiVideoPlayer Class is derived from VideoPlayer Class, so this constructor function has to call the constructor
of VideoPlayer Class.
BugiVideoPlayer::BugiVideoPlayer(int maxVolumeLevel, string supportedFormats[], int
supportedFormatCount):VideoPlayer(maxVolumeLevel,supportedFormats,supportedFormatCount)
{
    volumeLevel = 50;
}

//Prints that media player crushed!
void BugiVideoPlayer::EjectMedia()
{
    cout<<"BugiVideoPlayer:\tMedia player crushed!"<<endl;
}

//Destructor Function of BugiVideoPlayer Class.
BugiVideoPlayer::~BugiVideoPlayer()
{

}

/*****************************************
 * TestMain.cpp                          *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
```

```
 * Experiment 8: Polymorphism            *
 ****************************************/

#include "Ipod.h"                   //To create Ipod type variables.
#include "Walkman.h"                //To create Walkman type variables.
#include "BugiVideoPlayer.h"        //To create BugiVideoPlayer type variables.
#include "VLCVideoPlayer.h"         //To create VLCVideoPlayer type variables.


void TEST_ForAll(Player* player, const string& mediaName)
{
    //Open player!
    player->Open();

    //Try to mount given media.
    player->MounthMedia(mediaName);

    //Try to increase volume level.
    player->VolumeUp(20);

    //Try to decrease volume level.
    player->VolumeDown(10);

    //Play player!
    player->Play();

    //Pause player!
    player->Pause();

    //Stop player!
    player->Stop();

    //Play player!
    player->Play();

    //Try to forward!
    player->Forward();

    //Try to backward.
    player->BackWard();

    //Eject the media!
    player->EjectMedia();

    //Close player!
    player->Close();
```

```
}

int main()
{

    cout<< "+---------+" << endl
    << "| WALKMAN |" << endl
    << "+---------+" << endl;
    string walkmanFormats[] = { "mp3" };

    //walkman is a pointer that normally should point a Player type object but, Player Class is an ABSTRACT CLASS.(We
can not create any object from Player Class.)

    //A Walkman type object is created in HEAP MEMORY, and its address will be kept in walkman pointer.
    //This situation is called UPCASTING.(A base class pointer points to derived class object.)
    Player* walkman = new Walkman(70, walkmanFormats, 1);

    //Walkman's functions will test.
    TEST_ForAll(walkman, "test.mp3");


    cout<< "+------+" << endl
    << "| IPOD |" << endl
    << "+------+" << endl;
    string iPodFormats[] = { "mp3","wav" };

    //ipod is a pointer that normally should point a MusicPlayer type object but, it points an object the type of
IPod.
    //An IPod type object is created in HEAP MEMORY, and its address will be kept in ipod pointer(MusicPlayer*).
    //This situation is called UPCASTING.(A base class pointer points to derived class object.)
    MusicPlayer* ipod  = new IPod(100, iPodFormats, 2);

    //IPod's functions will test.
    TEST_ForAll(ipod, "test.mp3");


    cout << "+------------------+" << endl
    << "| BUGI VIDEO PLAYER |" << endl
    << "+------------------+" << endl;
    string bugiVideoFormats[] = { "mp4" };


    //bugiVideoPlayer is a pointer that points an object the type of BugiVideoPlayer. (Object is created in Heap
Memory)
    BugiVideoPlayer* bugiVideoPlayer = new BugiVideoPlayer(100, bugiVideoFormats, 1);

    //bugiVideoPlayer's functions will test.
```

```
    TEST_ForAll(bugiVideoPlayer, "test.mp4");


    cout << "+------------------+" << endl
    << "| VLC VIDEO PLAYER |" << endl
    << "+------------------+" << endl;
    string vlcVideoFormats[] = { "mp4","avi" };

    //vlcVideoPlayer is a pointer that points an object the type of VLCVideoPlayer. (Object is created in Heap
Memory)
    Player* vlcVideoPlayer = new VLCVideoPlayer(100, vlcVideoFormats, 2);

    //vlcVideoPlayer's functions will test.
    TEST_ForAll(vlcVideoPlayer, "test.mp4");

    //Indicates that program ended successfully.
    return 0;

}//end main()
```

CONCLUSION

→ Polymorphism is the art of taking advantage of this simple but powerful and versatile feature.
→ A virtual member is a member function that can be redefined in a derived class, while preserving its calling properties through references.
→ Non-virtual members can also be redefined in derived classes, but non-virtual members of derived classes cannot be accessed through a reference of the base class.
→ A constructor cannot be VIRTUAL.

→ Polymorphism allows dynamic binding (late binding).
→ Classes that contain at least one pure virtual function are known as ABSTRACT base classes.
→ Abstract base classes CANNOT BE USED TO INSTANTIATE OBJECTS.
→ They are classes that can only be used as base classes, and thus are allowed to have virtual member functions without definition (known as pure virtual functions). The syntax is to replace their definition by =0 (an equal sign and a zero):