Object Oriented Programming (CT-260)
Lab 09

Abstract Base Class and Pure Virtual Function

Objectives

The objective of this lab is to familiarize students with the abstract base class and pure virtual function.

Tools Required

DevC++ IDE

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Theory:

A base class which does not have the implementation of one or more of its virtual member functions is said to be an abstract base class. It serves as a framework in a hierarchy and the derived classes will be more specific and detailed.

Further, a virtual member function which does not have its implementation/definition is called a "pure" virtual member function. An abstract base class is said to be incomplete – missing some of the definition of its virtual member functions– and cannot be used to instantiate objects. However, it still can be used to instantiate pointer that will be used to send messages to different objects in the inheritance hierarchy to affect polymorphism.

Pure virtual member function declaration:

```
virtual return_type functionName( argument_list ) = 0;
```

	Abstract base class	Concrete base class
Data member	Yes	Yes
Virtual function	Yes	Yes
Pure virtual function	Yes	No
Object instantiation	No	Yes
Pointer instantiation	Yes	Yes

Abstract Classes:

An abstract class is a class that has at least one pure virtual function (i.e., a function that has no function body). The classes inheriting the abstract class must provide a definition for the pure virtual function; otherwise, the subclass would become an abstract class itself.

Abstract classes are essential for providing abstraction to the code to make it reusable and extendable. For Example, a Vehicle parent class with Truck and Motorbike inheriting from it is an abstraction that easily allows more vehicles to be added. However, even though all vehicles have wheels, not all vehicles have the same number of wheels – this is where a pure virtual function is needed.

Virtual Functions:

A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation (we do not have function body), we only declare it. A pure virtual function is declared by assigning 0 in declaration.

See the following example.

```
// An abstract class
class Test
{
    // Data members of class
public:
    // Pure Virtual Function
    virtual void show() = 0;

   /* Other members */
};
```

Note

- The = 0 syntax doesn't mean we are assigning 0 to the function. It's just the way we define pure virtual function.
- A pure virtual function is implemented by classes which are derived from Abstract class.
- A class is abstract if it has at least one pure virtual function. In example given above, Test is abstract class as it has virtual function show().

Example

A pure virtual function is implemented by classes which are derived from an Abstract class. Following is a simple example to demonstrate the same.

```
#include<iostream>
using namespace std;

class Base
{
  int x;
public:
   virtual void fun() = 0;
   int getX() { return x; }
};

// This class inherits from Base and implements fun()
class Derived: public Base
{
   int y;
public:
   void fun() { cout << "fun() called"; }
};

int main(void)
{
   Derived d;
   d.fun();
   return 0;
}
```

Output

```
fun( ) called
```

Object of abstract class cannot be created.

```
// pure virtual functions make a class abstract
#include <iostream>
using namespace std;
class Test{
   int x;
public:
   virtual void show( ) = 0;
   int getX( ){
      return x;
```

```
    }
};

int main(void){
    Test t;
    return 0;
}
```

Output

```
Compiler Error: cannot declare variable 't' to be of abstract type 'Test' because the following virtual functions are pure within 'Test': note: virtual void Test::show( )
```

An abstract class can have constructors. Consider this code that compiles and runs fine.

```
#include <iostream>
using namespace std;
// An abstract class with constructor
class Base{
protected:
    int x;
    virtual void fun( ) = 0;
public:
    Base(int i) { x = i; }
};

class Derived : public Base{
    int y;
public:
    Derived(int i, int j) : Base(i) { y = j; }
    void fun( ) { cout << "x = " << x << ", y = " << y; }
};

int main(void){
    Derived d(4, 5);
    d.fun( );
    return 0;
}
```

Output

```
x = 4, y = 5
```

Example: Abstract class and Virtual Function in calculating Area of Square and Circle:

```
#include <iostream>
using namespace std;

class Shape{
protected:
    float dimension;
```

```
public:
   void getDimension( ){
      cin >> dimension;
   }
private:
   // pure virtual Function
   virtual float calculateArea() = 0;
};
// Derived class
class Square : public Shape{
public:
   float calculateArea( ){
      return dimension * dimension;
   }
};

// Derived class
class Circle : public Shape{
public:
   float calculateArea( )   {
      return 3.14 * dimension * dimension;
   }
};

int main( ){
   Square square;
   Circle circle;
   cout << "Enter the length of the square: ";
   square.getDimension( );
   cout << "Area of square: " << square.calculateArea( ) << endl;
   cout << "\nEnter radius of the circle: ";
   circle.getDimension( );
   cout << "Area of circle: " << circle.calculateArea( ) << endl;
   return 0;
}
```

Output

```
Enter length to calculate the area of a square: 4
Area of square: 16

Enter radius to calculate the area of a circle: 5
Area of circle: 78.5
```

In this program, virtual float calculateArea( ) = 0; inside the Shape class is a pure virtual function. That's why we must provide the implementation of calculateArea( ) in both of our derived classes, or else we will get an error.

Exercise

1. Define an abstract class ***ArrayMultiplier*** with a pure virtual function **calculate()**. Then, create two derived classes: ***ArrayMultiplier1D*** for 1D array multiplication and ***ArrayMultiplier2D*** for 2D array multiplication. In the calculate() function of each derived class, perform the multiplication operation based on the specific array dimension.
2. Write a program to calculate final bill after discount. "ImtiazStore" gives 7 percent discount on total_bill while "BinHashimStore" gives 5 percent discount on total_bill. You have to initialize value of total_bill through a constructor and then calculate final bill after discount for both stores using the concept of abstract class and virtual functions.
3. You are tasked with developing a car rental system for a company. The system should allow customers to rent cars based on availability and manage the inventory of cars. To implement this system, you decide to use object-oriented programming and apply the concept of abstraction.
   - Abstract Base Class: Vehicle - You create an abstract base class called Vehicle, which represents a generic vehicle in the rental system. The Vehicle class contains common attributes like carId, brand, and model. It also defines pure virtual functions such as isAvailable( ) and rent( ), which represent the availability of the vehicle and the process of renting it. Since these functions are pure virtual, they have no implementation in the base class.
   - Derived Class: Car - You derive a Car class from the Vehicle class, representing a car in the rental system. The Car class provides concrete implementations for the pure virtual functions isAvailable( ) and rent( ). In the isAvailable( ) function, you check the availability of the car by querying its specific availability attribute. In the rent( ) function, you update the availability status of the car after it has been rented.
   - Rental System Class: RentalSystem - You create a RentalSystem class responsible for managing the rental operations. This class interacts with the Vehicle objects through the base class interface. It has functions like rentVehicle( ) and returnVehicle( ), which take a Vehicle object as a parameter and call the corresponding isAvailable( ) and rent( ) functions. The RentalSystem class can handle rental operations for any type of Vehicle, as long as it inherits from the Vehicle base class.
   - Customer Class: Customer - You create a Customer class to represent the customers of the rental system. The Customer class has methods like rentVehicle( ) and returnVehicle( ), which take a Vehicle object as a parameter. These methods interact with the RentalSystem class to perform the rental operations without knowing the specific type of the vehicle. This allows for flexibility and extensibility if new types of vehicles are added to the system in the future.

   By utilizing pure virtual functions, you create an abstraction that defines the common interface and behavior for all vehicles in the system. The derived classes provide concrete implementations specific to their type of vehicle. The RentalSystem class and the Customer class can interact with vehicles through the abstract base class interface, allowing for polymorphism and decoupling from the

specific implementations. This abstraction simplifies the codebase, promotes modularity, and allows for easy addition of new vehicle types without modifying existing classes. Write a test program by defining an array of pointers of the Base class and dynamic memory for allocating memory to objects created.

4. Suppose you have to send an encrypted message over the network. There are two techniques that you can use to encrypt your message. Suppose your message is "Hello". It will be encrypted as follows according to those 2 techniques.

Technique 1: This technique simply converts alphabets into their respective ASCII codes.	H=72 E=69 L=76 L=76 O=79	Now the string will become, 7269767679
Technique 2: This technique simply converts alphabets into their respective ASCII code and then adding 2.	H=72+2 E=69+2 L=76+2 L=76+2 O=79+2	Now the string will become, 7471787881

Make use of an abstract class ***EncryptionTecnique*** and pure virtual function ***encrypt*** and implement this scenario. Make derived classes ***EncryptionTecnique1*** and ***EncryptionTecnique2.*** Make a test program that takes an input string and encrypts it using both encryption techniques calling their respective encrypt functions.

5. Design the corresponding decryption techniques and repeat Q3.