Fast**National University of Computer & Emerging Sciences, Karachi  
Spring 2020 CS-Department  
CS 217 – Object-oriented Programming**

**LAB-MANUAL-10**

**Abstract Classes and Pure Virtual Functions**

**Abstract Class:**

Abstract classes are essential to providing an 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.

Consider an example of a Shape base class with sub-classes (Triangle and Rectangle)​that inherit the Shape class.

Now, suppose we need a function to return the area of a shape. The function will be declared in the Shape class; however, it cannot be defined there as the formula for the area is different for each shape. A non-specific shape does not have an area, but rectangles and triangles do. Therefore, the pure virtual function for calculating​ area will be implemented differently by each sub-class.

**Pure Virtual Function:**

Abstract class in C++ programming is a class that contains at least one pure virtual function and act as a base class. An abstract class may also contain non-virtual functions and member variables.

Pure virtual function is a function that has no implementation. All derived class who inherit the abstract base class must implement all pure virtual functions.

Main purpose of abstract class is to have all common functions that can be in derived classes also in the base class itself and have some pure virtual function to defer implementation of it to derived classes.

A ***virtual function in C++*** is nothing but a member function in a class that we declare within the base class and redefine in a derived class.

A ***pure virtual function in C++***is nothing but a virtual function that we know exists but cannot be implemented. It is simply declared, not implemented.

Its declaration is as follows:

/\* 0 does not indicate that we initialize the function to a null or zero value \*/

**virtual void function() = 0;**

***Note:*** It is important to note that a pure virtual function can only be declared, not defined.

**Properties of Abstract Class:**

* Abstract class have at least on pure virtual function and act as a base class.
* An object of this class cannot be created but a pointer can be.
* It forces all derived classes to implement all pure virtual functions.
* Abstract class cannot be instantiated, but pointers and references of Abstract class type can be created.
* Abstract class can have normal functions and variables along with a pure virtual function.
* Abstract classes are mainly used for up casting, so that its derived classes can use its interface.
* Classes inheriting an Abstract Class must implement all pure virtual functions, or else they will become Abstract too.

The following code snippet implements the abstract Shape class along with its sub-classes:

#include <iostream>

using namespace std;

class Shape {

   public:

      virtual int Area() = 0; // Pure virtual function is declared as follows.

      // Function to set width.

      void setWidth(int w) {

         width = w;

      }

      // Function to set height.

      void setHeight(int h) {

         height = h;

      }

   protected:

      int width;

      int height;

};

// A rectangle is a shape; it inherits shape.

class Rectangle: public Shape {

   public:

      // The implementation for Area is specific to a rectangle.

      int Area() {

         return (width \* height);

      }

};

// A triangle is a shape too; it inherits shape.

class Triangle: public Shape {

   public:

      // Triangle uses the same Area function but implements it to

      // return the area of a triangle.

      int Area() {

         return (width \* height)/2;

      }

};

int main() {

  Rectangle R;

  Triangle T;

  R.setWidth(5);

  R.setHeight(10);

  T.setWidth(20);

  T.setHeight(8);

  cout << "The area of the rectangle is: " << R.Area() << endl;

  cout << "The area of the triangle is: " << T.Area() << endl;

}

**Real Time Implementation of Abstract Classes:**

Below is a pseudo real time of implementation of an abstract class for a Vending Machine that will deliver Tea and Coffee. Common methods milk, hot water and sugar will be defined and provided by abstract class and an abstract method addSachets () will be  introduced in the abstract class itself that is unimplemented and will be implemented by derived classes e.g. Tea and coffee etc. **So, they can add tea or coffee depends upon their choice.**

#include <iostream>

using namespace std;

class Beverages{

public:

      //This is common behaviour to all derived classes.

      void addHotItems(){

            addHotWater();

            addMilk();

            addSugar();

      }

      //Defer / force derived classes to implement it.

      virtual void addSachets () = 0;//abstract function

private:

      // internal private functions.

      void addHotWater(){

            cout<<"\tAdd Hot Water "<< endl;

      }

      void addMilk(){

            cout<<"\tAdd Milk "<< endl;

      }

      void addSugar(){

            cout<<"\tAdd Sugar "<< endl;

      }

};

//Derived class tea

class Tea:public Beverages

{

public:

      void addSachets ()// implement specific behavior

      {

            cout<<"\tAdd Tea "<< endl;

      }

      Tea(){

            cout<<"Preparing Tea... "<< endl;

      }

};

class Coffee :public Beverages

{

public:

      void addSachets (){

            cout<<"\tAdd Coffee "<< endl;

      }

      Coffee(){

            cout<<"Preparing Coffee... "<< endl;

      }

};

//Test vinding machine

int main(){

      //create pointer of abstract base class

      //and assign an object of derived class

      Beverages\*tea = new Tea();

      tea ->addHotItems();

      tea ->addSachets();

      //Prepare Coffee

      Beverages\*coffee = new Coffee();

      coffee ->addHotItems();

      coffee ->addSachets();

      return 0;

}

**Output:**  
Preparing Tea…  
Add Hot Water  
Add Milk  
Add Sugar  
Add Tea

Preparing Coffee…  
Add Hot Water  
Add Milk  
Add Sugar  
Add Coffee

**Another Example:**

1. #include<iostream>
2. using namespace std;
3. class Apple
4. {
5. public:
6. // Pure Virtual Function declaration
7. virtual void **price**() = 0;
8. // Member functions
9. void **ringtone**()
10. {
11. cout<<"The ringtone is: Reflection"<<endl;
12. }
13. };
14. class iPhoneX: public Apple
15. {
16. public:
17. void **price**()
18. {
19. cout<<"The price is: 65,500"<<endl;
20. }
21. };
22. int **main**()
23. {
24. cout<<"Welcome to DataFlair tutorials"<<endl<<endl;
25. iPhoneX i;
26. i.**price**();
27. i.**ringtone**();
28. return 0;
29. }

**Exercises:**

* We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for eac of the two classes and print the percentage of marks for both the students.

### Create an abstract class 'Bank' with an abstract method 'getBalance'. $100, $150 and $200 are deposited in banks A, B and C respectively. 'BankA', 'BankB' and 'BankC' are subclasses of class 'Bank', each having a method named 'getBalance'. Call this method by creating an object of each of the three classes.

* We have to calculate the area of a rectangle, a square and a circle. Create an abstract class 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each. The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius. Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for printing the area of rectangle, square and circle respectively. Create an object of class 'Area' and call all the three methods.