National University of Computer & Emerging Sciences, Karachi
Computer Science Department

Spring 2021, Lab Manual – 12

Course Code: CL1004	**Course: Object Oriented Programming Lab**
Instructor(s):	**Shazia Paras Shaikh**

Content:

1. Abstract Classes
2. Virtual Functions
3. Lab Tasks

Abstract Classes:

Sometimes implementation of all functions cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class.

For example, let Shape be a base class, and different kinds of shapes can be children of the shape class. Therefore, we cannot provide one implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly, an Animal class doesn't have implementation of move() (assuming that all animals move, and many animals have different ways to move. Eg. Humans move differently than snakes), but all animals must know how to move.

Lastly, we cannot create objects of abstract classes, as they are "incomplete" because they lack the function definitions in their class body.

An abstract class must have **at least one** pure virtual function.

Virtual Functions:

A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword. A virtual function must be defined in the base class, even though it is not used. The prototypes of a virtual function of the base class and all the derived classes must be identical. If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions. We covered these in our previous lab!

Pure 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 functions.
- A pure virtual function is used if a function does not have any use in the base class but function must be implemented by all of its derived classes
- 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().

A complete example:

A pure virtual function is implemented by classes which are derived from a 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;
public:
  virtual void fun() = 0;
  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:

```
// C++ program to calculate the area of a square and a circle

#include <iostream>
using namespace std;

// Abstract class
class Shape {
   protected:
    float dimension;

   public:
    void getDimension() {
        cin >> dimension;
    }

    // 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.

Important things to remember about abstract classes:

1. A class is considered abstract if it has at least one pure virtual function in it.
2. If a class inherits an abstract class and does not override the pure virtual function, then that (derived) class also becomes an abstract class, and cannot be instantiated.
3. A derived class inheriting from an abstract class is called a concrete class if it overrides the pure virtual function present in the abstract base class.
4. While you can't instantiate an object of an abstract class, you can still create a pointer for it. **This pointer can point to of objects of its concrete child classes**. This is helpful when we are generalizing things, and also helps with achieving runtime polymorphism.

Tasks

Question 1:

The program defines three classes: Person, Student, and Faculty.

The Person class is an abstract base class that has two data members: a string name_ and an int age_. The class also declares a pure virtual function display(), which has no implementation.

The Student class is derived from Person and adds a data member id_ of type int. The class overrides the display() function and prints out the name_, age_, and id_ of the student.

The Faculty class is also derived from Person and adds a data member department_ of type string. The class overrides the display() function and prints out the name_, age_, and department_ of the faculty member.

The University class manages a collection of Person objects. It has an array people_ of size MAX_PEOPLE (a constant integer defined in the program) that holds pointers to Person objects. The class also has an integer num_people_ that keeps track of the number of Person objects currently in the array.

The University class defines two member functions: addPerson() and displayPeople(). The addPerson() function takes a pointer to a Person object as its parameter and adds it to the next available slot in the people_ array. If the array is already full, the function prints an error message.

The displayPeople() function iterates through the people_ array and calls the display() function on each Person object to print out their information.

In the main() function, the program creates three objects: s1 and s2 of class Student, and f1 of class Faculty. The program then adds these objects to an instance of the University class using the addPerson() function, and displays all the people using the displayPeople() function. Finally, the program deallocates the memory for the objects using delete.

Question 2:

There are four classes: Order, Starter, MainDish, and Meal. The Order class is the base class, and the Starter and MainDish classes both inherit from it using virtual inheritance.

The Order class has a function serve(), which is overridden in the Starter and MainDish classes with their specific implementations of serving the food. The Meal class overrides the serve() function as well, but it calls the serve() functions of both Starter and MainDish classes, and adds a message to let the customer know that they can enjoy their meal.

In the main() function, an object of the Meal class is created, and its serve() function is called, which triggers the overridden function in the Meal class to print out the messages of serving the starter, main dish, and enjoying the meal.

Question 3:

There are two classes Car and Bike that both inherit from the abstract base class Vehicle. The Vehicle class has a pure virtual function display(), which is overridden in the Car and Bike classes with their specific implementation to display the type of the vehicle.

There are also two classes CarFactory and BikeFactory, both of which inherit from the abstract base class VehicleFactory. The VehicleFactory class has a pure virtual function createVehicle(), which returns a pointer to a Vehicle object. The CarFactory and BikeFactory classes both implement the createVehicle() function and return a pointer to a Car or Bike object respectively.

In the main() function, the user is prompted to enter the type of vehicle they want to create, and based on their choice, either a CarFactory or a BikeFactory object is created. Then, the createVehicle() function of the factory object is called to create a new Vehicle object, and its display() function is called to print out the type of the vehicle.

Question 4:

There are two classes VeggieBurger and ChickenBurger that both inherit from the base class Burger. The Burger class has a virtual function makeBurger(), which is overridden in the VeggieBurger and ChickenBurger classes with their specific implementation to make a veggie or chicken burger.

In the main() function, the user is prompted to enter the type of burger they want to make, and based on their choice, either a VeggieBurger or a ChickenBurger object is created. Then, the makeBurger() function of the burger object is called to make the specific type of burger.

Question 5:

There are three classes CityTour, BeachTour, and AdventureTour, that all inherit from the Tour base class. The Tour class has a pure virtual function tourDetails() which is overridden in each of the derived classes with their specific implementation to provide details of the tour package.

CityTour::tourDetails() function prompt: This tour package includes a visit to famous landmarks, museums, and local markets.

BeachTour::tourDetails() function prompt: This tour package includes beach activities, water sports, and a relaxing stay at the resort.

AdventureTour::tourDetails() function prompt: This tour package includes trekking, camping, and other outdoor activities.

In the main() function, the user is prompted to enter the type of tour package they want to select, and based on their choice, either a CityTour, BeachTour, or AdventureTour object is created. Then, the tourDetails() function of the tour object is called to display the specific details of the selected tour package.