National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2022, Lab Manual - 10

| Course Code: CL-1004 | Course : Object Oriented Programming Lab |
|---|---|
| Instructor(s) : | Nida Munawar, Abeeha Sattar |

# Lab # 10

## Outline:

1. Runtime polymorphism
2. Abstract Classes
3. Virtual Functions
4. Lab Tasks

## Function Call Binding with Class Objects

Connecting the function call to the function body is called Binding. When it is done before the program is run, it's called Early Binding or Static Binding or Compile-time Binding. By default, early binding takes place

When it is done after the program is run, it's called Late Binding or Dynamic Binding or Run-time Binding.it is done by using virtual function in base class

***Function Overriding using Base Class Pointer***

```
#include<iostream>
using namespace std;
class Base
{
 public:
  void show()
 {
  cout << "Base class";
 }
};
class Derived:public Base
{
 public:
 void show()
 {
  cout << "Derived Class";
 }
};

int main()
{
 Base* b;       //Base class pointer
 Derived d;     //Derived class object
 b = &d; // passing derived class address into base class pointer
 b->show();     //Early Binding Occurs
}
```

**Output: Base Class**

In the above example, although, the object is of Derived class, still Base class's method is called. This happens due to Early Binding. Compiler on seeing **Base class's pointer**, set call to Base class's **show()** function, without knowing the actual object type.

***Function Overriding using Base Class Pointer & Virtual Function:***

```
#include<iostream>
using namespace std;
class Base
{
 public:
 virtual void show() // virtual function
 {
  cout << "Base class";
 }
};
class Derived:public Base
{
 public:
 void show()
 {
  cout << "Derived Class";
 }
};

int main()
{
 Base* b;       //Base class pointer
 Derived d;     //Derived class object
 b = &d; // passing derived class address into base class pointer
 b->show();     //Late Binding Occurs
}
```

**Output**

```
Total Objects: 1
    Derived Class
```

**Program Explanation:**

In this program, we made the base class's show function as virtual. This enables us to use the base class's object to call the functions of derived class which are overridden and virtual. Thus in the main function you can see that we created a base class pointer object and made it to point derived class object by passing derived class objects address. Then using the **(->)** operator we called the derived class objects show

function which prints the output as “Derived Class”. This is known as Dynamic Polymorphism or Late binding.

## 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().

## 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.

# Lab Tasks:

**Task 1:**

Suppose interest rates of HMB, HBL & MCB banks are 4, 2 and 3 percent respectively. You need to write a program using the concept of abstract classes and virtual functions that simply displays interest rate for each bank.

**Task 2:**

Define abstract class “Shape” that provides interface (through virtual functions) to the two derived classes “Rectangle” and “Triangle” to implement the function called “getArea()”. The program will output area of rectangle and area of triangle.

**Task 3:**

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.