Lab No.5 Implementing the Concept of Inheritance

5.1 Introduction

Reusability is an important characteristic of an object oriented programming (OOP). Features of a class i.e. both attributes (data members) and behaviors (member functions) can be reused. This can be done using inheritance among classes. Usually, a class with certain data members and/or member functions is defined and then another class or more is identified that can share these data members and functions. The first one is known as parent /base/super class while the later one is known as child/derived/sub class. Base class is more general while derived class is specialized version of base class and due to this reason inheritance maintains generalization and specialization. With inheritance, a programmer can both save the time and lessen the effort to write duplicate code. This lab covers inheritance in detail.

5.2 Objectives of the lab:

1 Understand the concept of inheritance and its different forms.

2 Write derived class from base class using inheritance.

3 Understand the constructor and destructor chaining in inheritance hierarchy.

4 Know how to use base class constructors within derived class.

5.3 Pre-Lab

5.3.1 Inheritance

1 New classes created from existing classes

2 Absorb attributes and behaviors from base class

3 Classes are often closely related
- "Factor out" common attributes and behaviors and place these in a base class
- Use inheritance to form derived classes

4 Derived class
- Class that inherits data members and member functions from a previously defined base class
- Derived class extends the behavior of the base class.

5 The class from which another class is derived is called ***base class*** or ***parent class*** or ***super-class***

6 The class which is derived from another class is called a ***derived class*** or ***child class*** or ***subclass***

7 The derived class is a superset of the base class

5.3.2 C++ Syntax for Inheritance

```
class ChildClass: access_specifier BaseClass
{
...
};
```

As discussed in earlier lab, there are three types of access specifiers i.e. public, protected, and private. Based on these specifiers, there are three types of inheritance: public inheritance, protected inheritance, and private inheritance in C++.

5.3.3 Protected access Specifier

1. Intermediate level of protection between **public** and **private** inheritance
2. Protected member can't be accessed from outside the base & derived classes
3. Derived-class members can refer to **public** and **protected** members of the base class simply by using the member names

5.3.4 Example: simpleInheritance.cpp

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

class A{
    private:
        int     num1;
    protected:
        int     num2;
    public:
        int     num3;
        A(){
            num1 = 1;
            num2 = 2;
            num3 = 3;
        }
        void display(){
            cout << "Number 1: " << num1 << endl;
            cout << "Number 2: " << num2 << endl;
            cout << "Number 3: " << num3 << endl;
        }
};

class B: public A{
    private:
        int     num4;
```

```
    public:
        B(){
            num4 = 4;
        }
        void display(){
            //cout << "Number 1: " << num1 << endl;      // error: can't access private data
            cout << "Number 2: " << num2 << endl;       // protected data member of A
            cout << "Number 3: " << num3 << endl;       // public data member of A
            cout << "Number 4: " << num4 << endl;       // private data member of B
        }
};

int main(){
    B       obj;
    obj.display();

    cout << "\nUpdating public data member of A using B's object..." << endl;
    cout << "\n\nCalling B's display() function: " << endl;
    obj.num3 = 5;
    obj.display();

    return 0;
}
```

Output:

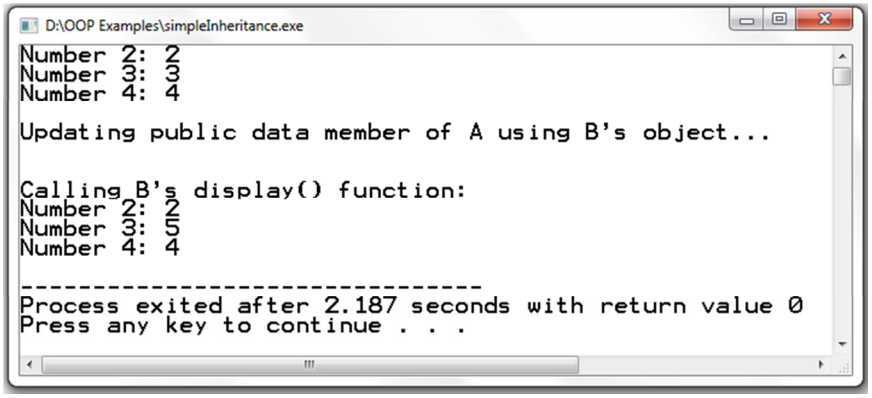

Figure 5.1: Output simpleInheritance.cpp

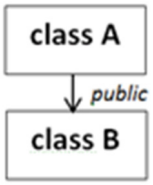

Figure 5.2: Example of Single Inheritance (simpleInheritance.cpp)

In this example, class B is derived publically from base class A as shown in Figure 5.2. Output is shown in Figure 5.1. Both protected and public data members of A are inherited in B and there outputs can be viewed using B's display() function. Private data members are only accessible within own class. They are not accessible in derived class and main function. num3 is a public data member of A and publically inherited in B. It can be modified using B's object, obj. In A, num3 is initialized with 3 and then modified in main() to 5 using obj.

With public inheritance, public features remain public, protected features remain protected, and private features remain private. That is parent class features are absorbed completely in child class. It is noted that a given behavior can be extended. For instance, display() function is present in both parent class A and child class B. But in B, it is extended to incorporate B's feature. If display() function is removed from B, still B's object can use display() function of A but in that case B's data members and updates done in B can't be accessed. Thus, display() function in class B is extendable version showing contents of both A and B.

Q. In case of private and protected inheritance, would the public member num3 of class A be accessible to B's obj in main i.e. can we write obj.num3=2;? If not, why?

5.3.5 Example: shapesInheritance.cpp

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

class Polygon {
    protected:
        int height;
        int length;
    public:
        void set_values(int a, int b){
            height = a;
            length = b;
        }
};

class Rectangle: public Polygon{
    public:
        double area() {
            return (height*length);
        }
};
```

```
class Triangle: public Polygon{
    public:
        double area(){
            return (height*length)/2;
        }
};

int main(){
    Rectangle r1;
    Triangle t1;

    r1.set_values(10, 20);
    cout << "Area of Rectangle: " << r1.area() << endl;

    t1.set_values(10, 20);
    cout << "Area of Triangle: " << t1.area() << endl;

    return 0;
}
```

Output:

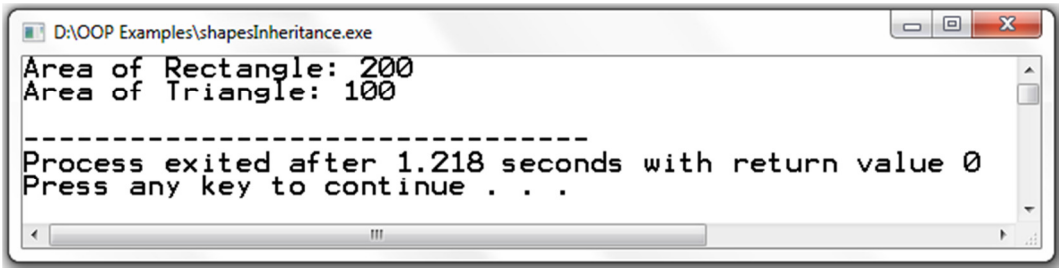

Figure 5.3: Output shapesInheritance.cpp

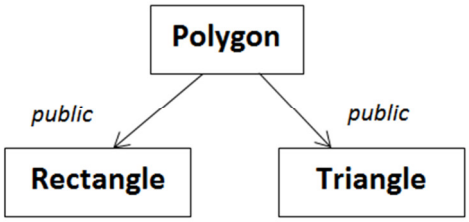

Figure 5.4: Example of Hierarchical Inheritance (shapesInheritance.cpp)

In this example, inheritance is depicted more practically using different shapes as shown in Figure 5.4. Base class, Polygon is created and then two derived classes i.e. Rectangle and Triangle are created using base class. Output is shown in Figure 5.3.

5.3.6 Example in Java

1 Java supports inheritance as it is an important characteristic of OOP.

2 The parent class is termed as super class and the inherited class is termed as sub class.
3 The keyword extends is used by the sub class to inherit features of super class. Subsequent sub class resembles public inheritance in C++.
4 Java doesn't support protected/private inheritance like C++. The developers of the Java language evidently felt that the cost in complexity needed to support private/protected inheritance (including multiple inheritance) outweighed the benefit that it would provide.
5 Next, an example of hierarchical inheritance (Java version of shapesInheritance.cpp) is demonstrated.

Polygon.java

```
package lab7s1;

public class Polygon {
    protected int height;
    protected int length;

    public void set_values(int a,int b){
            height=a;
            length=b;
    }
}
```

Rectangle.java

```
package lab7s1;

public class Rectangle extends Polygon{
    public double area(){
            return (height*length);
    }
}
```

Inheritance

Triangle.java

```
package lab7s1;

public class Triangle extends Polygon{
    public double area(){
            return (height*length)/2;
    }
}
```

Inheritance

Test.java

```
package lab7s1;

public class Main {
    public static void main(String[] args) {
            Rectangle r1 = new Rectangle();
            Triangle t1 = new Triangle();

            r1.set_values(10, 20);
            System.out.println("Area of Rectangle: "+r1.area());

            t1.set_values(10, 20);
            System.out.println("Area of Triangle: "+t1.area());
    }
}
```

5.3.7 Example in Python

1. Python supports inheritance. To do so, child class uses parent class name in creation. Subsequent inheritance acts like public inheritance of C++.
2. Just like Java, there is no protected/private inheritance in Python.
3. Next, an example of hierarchical inheritance (Python version of shapesInheritance.cpp) is demonstrated.

```
class Poly:
   def __init__(self):
      self.height = 0
      self.length = 0
   def set_values(self,h,l):
      self.height = h
      self.length = l

class Rect(Poly):
   def area(self):
      return self.height*self.length

class Tri(Poly):
   def area(self):
      return (self.height*self.length)/2

r1 = Rect()
t1 = Tri()
```

Inheritance

Inheritance

```
r1.set_values(10,20)
print "Area of Rectangle:", r1.area()

t1.set_values(10,20)
print "Area of Triangle:", t1.area()
```

Output:

```
In [3]: runfile('C:/Users/sumayya/shapesInheritance.py', wdir='C:/Users/sumayya')
Area of Rectangle: 200
Area of Triangle: 100
```

Figure 5.5: shapesInheritance.py

5.4 Activities

Perform these activities in C++, Java, and Python.

5.4.1 Activity

Reuse Point class of Lab 1 (Activity 1.4.2) as base class. Make the member data protected. Write all class member functions outside Point class.

Derive a class Circle from this Point class that has an additional data member: radius of the circle. The point from which this circle is derived represents the center of circle. Provide a no-argument constructor to initialize the radius and center coordinates to 0. Provide a 2-argument constructor: one argument to initialize the radius of circle and the other argument to initialize the center of circle (provide an object of point class in the second argument). Provide a 3-argument constructor that takes three floats to initialize the radius, x-, and y-coordinates of the circle. Provide setter and getter functions for radius of the circle. Provide two functions to determine the radius and circumference of the circle. Write all class member functions outside Circle class.

Also derive another class Cylinder from the Point class. This class contains an additional data member: radius and height of cylinder. Provide appropriate constructors to initialize the center, radius, and height of the cylinder. Provide functions to determine the area and volume of the cylinder. Area of a cylinder is 2πr*(r + h). Volume of cylinder is 2πr*r*h. Use the findCircum() of circle class where required. Write a main function to test these classes.

5.5 Testing

Test Cases for Activity 5.4.1

Sample Inputs	Sample Outputs
Declare Point object p and provide coordinates 2 and 3. Declare Circle object c using point p and radius 4.	

Display point p using c. Display circle c's radius. Compute and display circumference of circle c. Declare Cylinder object cy using circle c and height 6.4. Compute and display area of cylinder cy. Compute and display volume of cylinder cy.	Point: (2,3) Radius of circle: 4 Circumference of circle: 25.1328 Area of cylinder: 261.381 Volume of cylinder: 643.4

5.6 References:

1. **Class notes**
2. **Object-Oriented Programming in C++ by *Robert Lafore***
3. **How to Program C++ by *Deitel & Deitel***
4. **Programming and Problem Solving with Java by *Nell Dale & Chip Weems***
5. **Murach's Python Programming by *Micheal Urban & Joel Murach***