MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

Implementation of Polymorphism using C++ and JAVA

Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 29, 2022

Contents

1 Aim and Objectives

Aim

Implementation of Polymorphism using C++ and Java.

Objectives

1. To understand the use of pure virtual funcitons.
2. To understnad implantation of compile time and run time polymorphism.
3. To learn implementation ofmethod overriding in java.

2 Problem Statements

2.1 Problem 1 in C++

Write a C++ program with base class Employee and three derived classes namely

- SalariedEmployees
- CommissionEmployees
- HourlyEmployees

Declare calculateSalary() as a pure virtual function in base class and define it in respective derived classes to calculate salary of an employee. The company wants to implement an Object Oriented Application that performs its payroll calculations polymorphically.

2.2 Problem 2 in Java

Define a Class **Shapes** as the Base Class that can find the area of the following :

- Circle
- Square
- Rectangle

Find the area of these shapes using construtor overloading and method overloading.

2.3 Probelm 3 in Java

Create a Parent Class Hillstations with the methods location() and famousfor(). Create three subclasses by Hill Station names. These subclasses must extend the superclass and override its methods location() and famousfor(). It should refer to the base class object and the base class method overrides the superclass method, and the base class method is invoked at runtime.

3 Theory

3.1 Concept of Compile time Polymorphism

In compile-time polymorphism, a function is called at the time of program compilation. We call this type of polymorphism as early binding or Static binding.

Function overloading and operator overloading is the type of Compile time polymorphism.

It can be implemented in 2 simple ways in c++

1. Function Overloading Function overloading means one function can perform many tasks. In C++, a single function is used to perform many tasks with the same name and different types of arguments. In the function overloading function will call at the time of program compilation. It is an example of compile-time polymorphism.

```

    class Addition {
    public:
      int ADD(int X,int Y)   // Function with parameter
      {
        return X+Y;     // this function is performing addition of  two
    Integer value
      }
      int ADD() {               // Function with same name but without
    parameter
        string a= "HELLO";
        string b="SAM";   // in this function concatenation is performed
      string c= a+b;
      cout<<c<<endl;

      }
    };
    int main(void) {
      Addition obj;   // Object is created
      cout<<obj.ADD(128, 15)<<endl; //first method is called
      obj.ADD();  // second method is called
      return 0;
    }


```

2. Operator Overloading Operator overloading means defining additional tasks to operators without changing its actual meaning. We do this by using operator function.

   ‘’ The purpose of operator overloading is to provide a special meaning to the user-defined data types.

   The advantage of Operators overloading is to perform different operations on the same operand.

```
      #include <iostream>
      using namespace std;
      class A
      {

        string x;
        public:
        A(){}
        A(string i)
        {
        x=i;
```

```
12          }
13          void operator+(A);
14          void display();
15      };
16
17      void A:: operator+(A a)
18      {
19
20          string m = x+a.x;
21          cout<<"The result of the addition of two objects is : "<<m;
22
23      }
24      int main()
25      {
26          A a1("Welcome");
27          A a2("back");
28          a1+a2;
29          return 0;
30      }
31
32
```

3.2 Concept of Run Time Polymorphism

In Runtime polymorphism, functions are called at the time the program execution. Hence, it is known as late binding or dynamic binding.

Function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list.

It is achieved by using virtual functions and pointers. It provides slow execution as it is known at the run time. Thus, It is more flexible as all the things executed at the run time.

In C++ it can be implemented using these 2 methods.

1. Function overriding: In function overriding, we give the new definition to base class function in the derived class. At that time, we can say the base function has been overridden. It can be only possible in the 'derived class'. In function overriding, we have two definitions of the same function, one in the superclass and one in the derived class. The decision about which function definition requires calling happens at runtime. That is the reason we call it 'Runtime polymorphism'.

2. Virtual Functions A virtual function is declared by keyword virtual. The return type of virtual function may be int, float, void.

   A virtual function is a member function in the base class. We can redefine it in a derived class. It is part of run time polymorphism. The declaration of the virtual function must be in the base class by using the keyword virtual. A virtual function is not static.

   The virtual function helps to tell the compiler to perform dynamic binding or late binding on the function.

   If it is necessary to use a single pointer to refer to all the different classes' objects. This is because we will have to create a pointer to the base class that refers to all the derived objects.

3.3 Use of Pure Virtual Functions

When the function has no definition, we call such functions as “Do-nothing function or Pure virtual function”. The declaration of this function happens in the base class with no definition.

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++, and javac, with JDK 18.0.2 for Java

5 Input

For C++

1. Number of Each Type of Employee
2. Name, Age, Address City, and Salary of Each Employee

For Java

1. The Side of the Square
2. The Radius of the Circle
3. The Length and Breadth of the Rectangle.

6 Output

For C++

1. General Information about Each Employee
2. The Weekly, hourly and commisioned Salary for Respective Employees.

For Java

1. The Area of the Shapes
2. The Location of the Hill Stations
3. The Reason the Hill stations are Famous for.

7 Code

7.1 C++ Implementation

```cpp
// Write a C++ program with base class Employee and three derived classes namely
// 1. salaried employees
// 2. commission_employees and
// 3. hourly employees
// Declare calculate_salary() as a pure virtual function in base class and define it in respective
// derived classes to calculate salary of an employee.
// The company wants to implement an Object Oriented Application that performs its payroll
// calculations polymorphically.

// Polymorphism
// Virtual Functionss, overriding functions, and overloading functions.

#include <iostream>
using namespace std;

class Employee
{
public:
    // static int ssn;
    int emp_id = 1000;
    int age = 0;
    double basic_sal = 0, da = 0, ta = 0, gross_sal = 0, net_sal = 0;
    string address_city, position, name;

    virtual void calculate_salary() = 0;

    void display()
    {
        // ssn++;
        // cout << "Employee ssn is:" << ssn << endl;
        cout << "Employee ID is : " << emp_id << endl;
        cout << "Employee Name: " << name << endl;
        cout << "Employee Age: " << age << endl;
        cout << "Employee Address City: " << address_city << endl;
    }

    void accept()
    {
        cout << "Enter the Employee ID: " << endl;
        cin >> emp_id;
        cout << "Enter the Employee Name: " << endl;
        cin >> name;
        cout << "Enter the Employee Age: " << endl;
        cin >> age;
        cout << "Enter the Employee Address City: " << endl;
        cin >> address_city;
    }

    // Destructor
    ~Employee()
    {
        cout << "The Destructor was called" << endl;
    }
};
class SalariedEmployee : public Employee
{
public:
```

```
58      int weekly_salary;
59      int net_sal;
60      void accept()
61      {
62          Employee::accept();
63          cout << "Enter the Wage: ";
64          cin >> weekly_salary;
65      }
66
67      void calculate_salary()
68      {
69          cout << "Calculating Salary of Salaried Employee" << endl;
70          net_sal = weekly_salary * 7;
71      }
72
73      void display()
74      {
75          Employee::display();
76          cout << "Weekly Salary is: " << net_sal << endl;
77      }
78  };
79
80  class HourlyEmployee : public Employee
81  {
82  public:
83      int hours, wage;
84      int net_sal;
85
86      void accept()
87      {
88          Employee::accept();
89          cout << "Enter the basic salary: " << endl;
90          cin >> basic_sal;
91          cout << "Enter the Wage: ";
92          cin >> wage;
93          cout << "Enter the hours worked" << endl;
94          cin >> hours;
95      }
96
97      void calculate_salary()
98      {
99          cout << "Calculating Salary of Salaried Employee" << endl;
100         if (hours < 40)
101         {
102             net_sal = basic_sal + hours * wage;
103         }
104         else
105         {
106             net_sal = 40 * wage + (hours - 40) * wage * 1.5;
107         }
108     }
109
110     void display()
111     {
112         Employee::display();
113         cout << "Hourly Employee Salary is: " << net_sal << endl;
114     }
115 };
116
```

```
117 class CommissionEmployee : public Employee
118 {
119 public:
120     float gross_sales, commission_rate = 0.05;
121     float net_sal;
122 
123     void accept()
124     {
125         Employee::accept();
126         cout << "Enter the gross sales: ";
127         cin >> gross_sales;
128     }
129 
130     void calculate_salary()
131     {
132         cout << "Calculating Salary of Salaried Employee" << endl;
133         net_sal = commission_rate * gross_sales;
134     }
135 
136     void display()
137     {
138         Employee::display();
139         cout << "Commission Employee Salary is: " << net_sal << endl;
140     }
141 };
142 
143 int main()
144 {
145     cout << "Welcome to Employee Payroll Management System" << endl
146          << endl;
147 
148     int choice = 1, number = 1;
149     Employee *ptr;
150     do
151     {
152         cout << "1. Salaried Employee\n2. Commisisioned Employee\n3. Hourly
    Employee\n4. Quit\n";
153         cout << "\n\nWhose Details do you wanna enter? " << endl;
154         cin >> choice;
155 
156         if (choice == 1)
157         {
158             cout << "How many SalariedEmployees are we talking? ";
159             cin >> number;
160             SalariedEmployee pr[number];
161             for (int i = 0; i < number; i++)
162             {
163                 cout << "Enter the Information about the Salaried Employee" <<
    endl;
164                 pr[i].accept();
165             }
166             cout << "\nHere is their Information and their Pay Slips" << endl;
167             cout << endl
168                  << endl;
169 
170             cout << "Salaried Employee" << endl;
171 
172             for (int i = 0; i < number; i++)
173             {
```

```
174                 cout << "Info and Pay Slip of Salaried Employee " << i + 1 << endl
    ;
175                 ptr = &pr[i];
176                 ptr->calculate_salary();
177                 pr[i].display();
178                 cout << endl;
179             }
180         }
181         else if (choice == 2)
182         {
183             cout << "How many Commission Employees are we talking? ";
184             cin >> number;
185             CommissionEmployee tl[number];
186             for (int i = 0; i < number; i++)
187             {
188                 cout << "Enter the Information about the Commission Employee " <<
    i + 1 << endl;
189                 tl[i].accept();
190             }
191             cout << "Here is their Information and their Pay Slips" << endl;
192             cout << endl
193                  << endl;
194             for (int i = 0; i < number; i++)
195             {
196                 cout << "Info and Pay Slip of Commission Employee " << i + 1 <<
    endl;
197                 ptr = &tl[i];
198                 ptr->calculate_salary();
199                 tl[i].display();
200                 cout << endl;
201             }
202             cout << endl
203                  << endl;
204         }
205         else if (choice == 3)
206         {
207             cout << "How many Hourly Employees are we talking? ";
208             cin >> number;
209             HourlyEmployee ap[number];
210             for (int i = 0; i < number; i++)
211             {
212                 cout << "Enter the Information about the Hourly Employees " << i +
     1 << endl;
213                 ap[i].accept();
214             }
215             cout << "Here is their Information and their Pay Slips" << endl;
216             cout << endl
217                  << endl;
218             for (int i = 0; i < number; i++)
219             {
220                 cout << "Info and Pay Slip of Hourly Employees" << i + 1 << endl;
221                 ptr = &ap[i];
222                 ptr->calculate_salary();
223                 ap[i].display();
224                 cout << endl;
225             }
226             cout << endl
227                  << endl;
228         }
```

```
229
230     } while (choice != 4);
231
232     return 0;
233 }
```

Listing 1: Main.Cpp

7.1.1 C++ Input and Output

```
1  Welcome to Employee Payroll Management System
2
3  1. Salaried Employee
4  2. Commisisioned Employee
5  3. Hourly Employee
6  4. Quit
7
8
9  Whose Details do you wanna enter?
10 1
11 How many SalariedEmployees are we talking? 1
12 Enter the Information about the Salaried Employee
13 Enter the Employee ID:
14 1001
15 Enter the Employee Name:
16 Tony
17 Enter the Employee Age:
18 35
19 Enter the Employee Address City:
20 NewYork
21 Enter the Wage: 50000
22
23 Here is their Information and their Pay Slips
24
25
26 Salaried Employee
27 Info and Pay Slip of Salaried Employee 1
28 Calculating Salary of Salaried Employee
29 Employee ID is : 1001
30 Employee Name: Tony
31 Employee Age: 35
32 Employee Address City: NewYork
33 Weekly Salary is: 350000
34
35 The Destructor was called
36 1. Salaried Employee
37 2. Commisisioned Employee
38 3. Hourly Employee
39 4. Quit
40
41 Whose Details do you wanna enter?
42 2
43 How many Commission Employees are we talking? 1
44 Enter the Information about the Commission Employee 1
45 Enter the Employee ID:
46 1002
47 Enter the Employee Name:
48 Steve
49 Enter the Employee Age:
```

```
105
Enter the Employee Address City:
Queens
Enter the gross sales: 500
Here is their Information and their Pay Slips


Info and Pay Slip of Commission Employee 1
Calculating Salary of Salaried Employee
Employee ID is : 1002
Employee Name: Steve
Employee Age: 105
Employee Address City: Queens
Commission Employee Salary is: 25



The Destructor was called
1. Salaried Employee
2. Commisisioned Employee
3. Hourly Employee
4. Quit


Whose Details do you wanna enter?
3
How many Hourly Employees are we talking? 1
Enter the Information about the Hourly Employees 1
Enter the Employee ID:
1003
Enter the Employee Name:
Bruce
Enter the Employee Age:
50
Enter the Employee Address City:
Space
Enter the basic salary:
600
Enter the Wage: 200
Enter the hours worked
45
Here is their Information and their Pay Slips


Info and Pay Slip of Hourly Employees1
Calculating Salary of Salaried Employee
Employee ID is : 1003
Employee Name: Bruce
Employee Age: 50
Employee Address City: Space
Hourly Employee Salary is: 9500



The Destructor was called
1. Salaried Employee
2. Commisisioned Employee
3. Hourly Employee
4. Quit
```

```
109
110
111 Whose Details do you wanna enter?
112 4
```

Listing 2: Output for Problem 1

7.2 Java Implementation of Problem 2

```
1  package assignment_3;
2
3  public class Shapes {
4
5      public double Area;
6      public double side;
7      public double length;
8      public double breadth;
9      public int radius;
10
11     Shapes(int radius) {
12         Area = 0.0;
13         this.radius = radius;
14     }
15
16     Shapes(double length, double breadth) {
17         Area = 0.0;
18         this.length = length;
19         this.breadth = breadth;
20     }
21
22     Shapes(double side) {
23         Area = 0.0;
24         this.side = side;
25     }
26
27     double Area(int radius) {
28         Area = 3.14 * radius * radius;
29         return Area;
30     }
31
32     double Area(double length, double breadth) {
33         Area = length * breadth;
34         return Area;
35     }
36
37     double Area(double side) {
38         Area = side * side;
39         return Area;
40     }
41
42 }
```

Listing 3: Full Time Employee.java

```
1 // Define a class Shapes as
2 // 1. Circle
3 // 2. square and
4 // 3. rectangle.
5 // Find the area of these shapes using constructor overloading and method
      overloading.
```

```java

package assignment_3;

import java.lang.Math;

public class Problem_A {
    public static void main(String[] args) {
        Shapes circle = new Shapes(7);
        Shapes square = new Shapes(1.5);
        Shapes rectangle = new Shapes(1.4, 3.5);
        System.out.println("The Radius of the Circle is: " + circle.radius);
        System.out.println("The Area of the Circle is: " + circle.Area(circle.
    radius));
        System.out.println("The Side of the Square is: " + square.side);
        System.out.println("The Area of the Square is: " + square.Area(square.side
    ));
        System.out.println("The Length of the Rectangle is: " + rectangle.length);
        System.out.println("The Breadth of the Rectangle is: " + rectangle.breadth
    );
        System.out.println("The Area of the Rectangle is: "
                + String.format("%.2f", rectangle.Area(rectangle.length, rectangle
    .breadth)));
    }
}
```

Listing 4: Main.java

7.2.1 Java Output for Problem 2

```
The Radius of the Circle is: 7
The Area of the Circle is: 153.86
The Side of the Square is: 1.5
The Area of the Square is: 2.25
The Length of the Rectangle is: 1.4
The Breadth of the Rectangle is: 3.5
The Area of the Rectangle is: 4.90
```

Listing 5: Output for Problem 2

7.3 Java Implementation of Problem 3 using Interfaces

```java
package assignment_3;

abstract public class HillStation {
    abstract public void location(); // Pure Virtual Function.

    abstract public void famousfor();
}

class Manali extends HillStation {
    @Override
    public void location() {
        System.out.println("Manali in Himachal Pradesh");
    }

    @Override
    public void famousfor() {
        System.out.println(
                "Manali is a high-altitude Himalayan resort town in India's
    northern Himachal Pradesh state. It has a reputation as a backpacking center.
```

```
    Set on the Beas River, it's a gateway for skiing in the Solang Valley and
    trekking in Parvati Valley. It's also a jumping-off point for paragliding,
    rafting and mountaineering in the Pir Panjal mountains, home to 4,000m-high
    Rohtang Pass.");
    }
}

class Shimla extends HillStation {
    @Override
    public void location() {
        System.out.println("Shimla is in Himachal Pradesh");
    }

    @Override
    public void famousfor() {
        System.out.println(
                "The town is famous for pleasant walking experiences on hillsides
    surrounded by pine and oak forests. This capital city of Himachal Pradesh is
    famous for The Mall, ridge, and toy train. With colonial style buildings, the
    town has relics of ancient past that lend it a distinct look.");
    }
}

class Mahabaleshwar extends HillStation {
    @Override
    public void location() {
        System.out.println("Mahabaleshwar is in Maharashtra");
    }

    @Override
    public void famousfor() {
        System.out.println(
                "Mahabaleshwar is a hill station in India's forested Western Ghats
     range, south of Mumbai. It features several elevated viewing points, such as
    Arthur's Seat. West of here is centuries-old Pratapgad Fort, perched atop a
    mountain spur. East, Lingmala Waterfall tumbles off a sheer cliff. Colorful
    boats dot Venna Lake, while 5 rivers meet at Panch Ganga Temple to the north.")
    ;
    }
}
```

Listing 6: HillStation

```
// Create a parent class Hillstations with the methods location() and famousfor().
// Create three subclasses by hill station names e.g. Manali, Shimla etc.
// Subclasses extend the superclass and override methods location() and famousfor
    ().
// Call the methods location() and famousfor() by the Parent class, i.e.
    Hillstations
// class.
// It should refer to the base class object and the base class method overrides
    the
// superclass method, base class method is invoked at runtime.

package assignment_3;

public class Problem_B {
    public static void main(String[] args) {
        Manali obj = new Manali();
```

```
        obj.location();
        obj.famousfor();

        Shimla obj1 = new Shimla();
        obj1.location();
        obj1.famousfor();

        Mahabaleshwar obj2 = new Mahabaleshwar();
        obj2.location();
        obj2.famousfor();
    }
}
```

Listing 7: Main.java

7.3.1 Java Output

```
Manali in Himachal Pradesh
Manali is a high-altitude Himalayan resort town in India's northern Himachal
    Pradesh state. It has a reputation as a backpacking center. Set on the Beas
    River, it's a gateway for skiing in the Solang Valley and trekking in Parvati
    Valley. It's also a jumping-off point for paragliding, rafting and
    mountaineering in the Pir Panjal mountains, home to 4,000m-high Rohtang Pass.
Shimla is in Himachal Pradesh
The town is famous for pleasant walking experiences on hillsides surrounded by
    pine and oak forests. This capital city of Himachal Pradesh is famous for The
    Mall, ridge, and toy train. With colonial style buildings, the town has relics
    of ancient past that lend it a distinct look.
Mahabaleshwar is in Maharashtra
Mahabaleshwar is a hill station in India's forested Western Ghats range, south of
    Mumbai. It features several elevated viewing points, such as Arthur's Seat.
    West of here is centuries-old Pratapgad Fort, perched atop a mountain spur.
    East, Lingmala Waterfall tumbles off a sheer cliff. Colorful boats dot Venna
    Lake, while 5 rivers meet at Panch Ganga Temple to the north.
```

Listing 8: Output for Problem 3

8 Conclusion

Thus, learned to use polymorphism and implemented solution of the given problem statement using C++ and Java.

9 FAQs

1. **Discuss the use of Virtual Functions.**
   Without "virtual" you get "early binding". Which implementation of the method is used gets decided at compile time based on the type of the pointer that you call through.

   With "virtual" you get "late binding". Which implementation of the method is used gets decided at run time based on the type of the pointed-to object - what it was originally constructed as. This is not necessarily what you'd think based on the type of the pointer that points to that object.

```
    class Base
    {
      public:
          void Method1 ()  {  std::cout << "Base::Method1" << std::endl;  }
      virtual void Method2 ()  {  std::cout << "Base::Method2" << std::endl;
    }
    };

    class Derived : public Base
    {
      public:
      void Method1 ()  {  std::cout << "Derived::Method1" << std::endl;  }
      void Method2 ()  {  std::cout << "Derived::Method2" << std::endl;  }
    };

    Base* basePtr = new Derived ();
      //  Note - constructed as Derived, but pointer stored as Base*

    basePtr->Method1 ();  //  Prints "Base::Method1"
    basePtr->Method2 ();  //  Prints "Derived::Method2"

```

2. **What is the difference between early binding and late binding.**
   Early Binding: The binding which can be resolved at compile time by the compiler is known as static or early binding. Binding of all the static, private and final methods is done at compile-time.

   Example

```
    public class NewClass {
      public static class superclass {
        static void print()
        {
          System.out.println("print in superclass.");
        }
      }
      public static class subclass extends superclass {
        static void print()
        {
          System.out.println("print in subclass.");
```

```
        }
    }

    public static void main(String[] args)
    {
        superclass A = new superclass();
        superclass B = new subclass();
        A.print();
        B.print();
    }
}

// Output
// print in superclass.
// print in superclass.


```

Late Binding: In the late binding or dynamic binding, the compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method.

```


public class NewClass {
    public static class superclass {
        void print()
        {
            System.out.println("print in superclass.");
        }
    }

    public static class subclass extends superclass {
        @Override
        void print()
        {
            System.out.println("print in subclass.");
        }
    }

    public static void main(String[] args)
    {
        superclass A = new superclass();
        superclass B = new subclass();
        A.print();
        B.print();
    }
}
// Output:
// print in superclass.
// print in subclass.

```

3. **Explain the use of abstract keyword in java with examples.**
   Abstract is a non-access modifier in java applicable for classes, methods but not variables. It is used to achieve abstraction which is one of the pillar of Object Oriented Programming(OOP). Following are different contexts where abstract can be used in Java.

Due to their partial implementation, we cannot instantiate abstract classes.Any subclass of an abstract class must either implement all of the abstract methods in the super-class, or be declared abstract itself.Some of the predefined classes in java are abstract. They depend on their sub-classes to provide complete implementation. For example, java.lang.Number is a abstract class. For more on abstract classes, see abstract classes in java

```
// A java program to demonstrate
// use of abstract keyword.

// abstract with class
abstract class A
{
  // abstract with method
  // it has no body
  abstract void m1();

  // concrete methods are still allowed in abstract classes
  void m2()
  {
    System.out.println("This is a concrete method.");
  }
}

// concrete class B
class B extends A
{
  // class B must override m1() method
  // otherwise, compile-time exception will be thrown
  void m1() {
    System.out.println("B's implementation of m1.");
  }


}

// Driver class
public class AbstractDemo
{
  public static void main(String args[])
  {
    B b = new B();
    b.m1();
    b.m2();
  }
}


```

4. **State Features of abstract base classes.**
   A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

   Important Features are:

   (a) An abstract class must be declared with an abstract keyword.

   (b) It can have abstract and non-abstract methods.

(c) It cannot be instantiated.

(d) It can have constructors and static methods also.

(e) It can have final methods which will force the subclass not to change the body of the method.