**PRACTICAL # 05**

**OBJECT:**

**Polymorphism in Java**

**THEORY:**

The only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed. A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

example.

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

When we apply the reference variable facts to a Deer object reference, the following declarations are legal:

Deer d = new Deer();

Animal a = d;

Vegetarian v = d;

Object o = d;

All the reference variables d,a,v,o refer to the same Deer object in the heap.

## Virtual Methods:

The behavior of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

In method overriding, a child class can override a method in its parent. An overridden method is essentially hidden in the parent class, and is not invoked unless the child class uses the super keyword within the overriding method.

public class Employee

{

private String name;

private String address;

private int number;

public Employee(String name, String address, int number)

{

this.name = name;

this.address = address;

this.number = number;

}

public String toString()

{

return name + " " + address + " " + number;

}

public String getName()

{

return name;

}

public String getAddress()

{

return address;

}

public void setAddress(String newAddress)

{

address = newAddress;

}

public int getNumber()

{

return number;

}

}

extend Employee class as follows:

/\* Salary.java \*/

public class Salary extends Employee

{

private double salary; //Annual salary

public Salary(String name, String address, int number, double

salary)

{

super(name, address, number);

setSalary(salary);

}

public double getSalary()

{

return salary;

}

public void setSalary(double newSalary)

{

if(newSalary >= 0.0)

{

salary = newSalary;

}

}

public double computePay()

{

System.out.println("Computing salary pay for " + getName());

return salary/52;

}

}

/\* File name : VirtualDemo.java \*/

public class VirtualDemo

{

public static void main(String [] args)

{

Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);

Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);

System.out.println("Call mailCheck using Salary reference --");

s.mailCheck();

System.out.println("\n Call mailCheck using Employee reference--");

e.mailCheck();

}

}

This would produce the following result:

Here, we instantiate two Salary objects . one using a Salary reference s, and the other using an Employee reference e.

While invoking *s.mailCheck()* the compiler sees mailCheck() in the Salary class at compile time, and the JVM invokes mailCheck() in the Salary class at run time.

Invoking mailCheck() on e is quite different because e is an Employee reference. When the compiler sees*e.mailCheck()*, the compiler sees the mailCheck() method in the Employee class.

Here, at compile time, the compiler used mailCheck() in Employee to validate this statement. At run time, however, the JVM invokes mailCheck() in the Salary class.

This behavior is referred to as virtual method invocation, and the methods are referred to as virtual methods. All methods in Java behave in this manner, whereby an overridden method is invoked at run time, no matter what data type the reference is that was used in the source code at compile time.

**ACTIVITIES**

**Activity 1**

Build (Assemble and link) and run the above program, but take the two numbers input from user. You can build the program using MASM or Emu8086.

**Activity 2**

Write a program that finds greater of three numbers and prints that number. You may hardcode the value for three numbers. Note each of the numbers should be single digit.

**REVIEW QUESTIONS**

1. How a conditional jump makes jump decision?
2. Why is flags register important in the conditional jumping?
3. What does a label represent in program and why is it necessary in a jump statement?
4. If/else in assembly is accomplished using a combination of which types of jumps?