Lab 8

Polymorphism

Objective:

The purpose of lab is to make students understand writing of generic code using polymorphism.

Activity Outcomes:

This lab teaches you the following topics:

- Call methods of class using polymorphism
- Upcasting and Downcasting of objects.

Instructor Note:

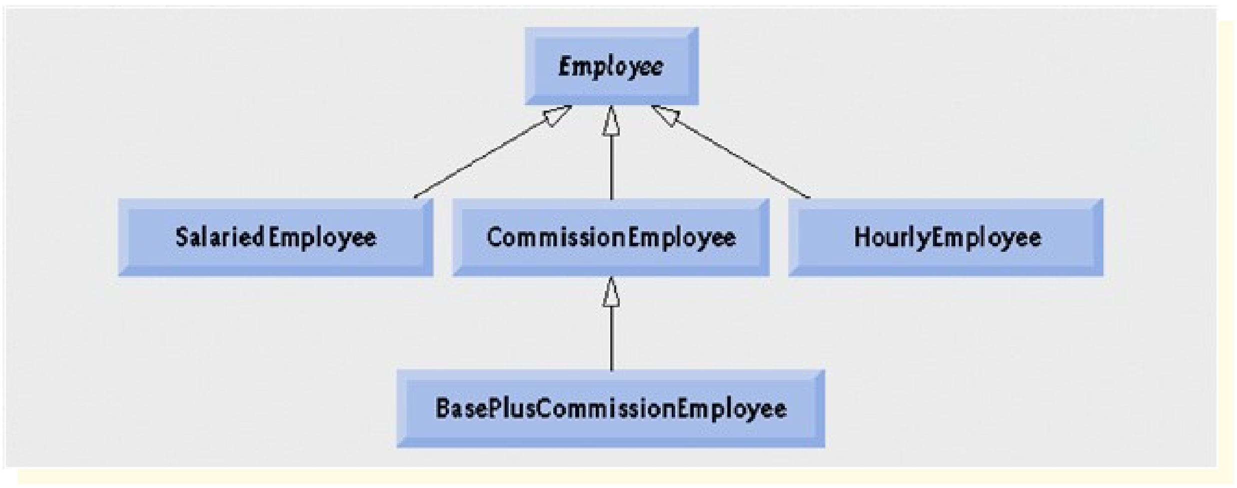
As pre-lab activity, read Chapter 11 from the text book "Introduction to Java Programming", Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

Polymorphism enables you to “program in the general” rather than “program in the specific.” In particular, polymorphism enables you to write programs that process objects that share the same superclass (either directly or indirectly) as if they’re all objects of the superclass; this can simplify programming.

Consider the following example of polymorphism. Suppose we create a program that simulates the movement of several types of animals for a biological study. Classes Fish, Frog and Bird represent the types of animals under investigation. Imagine that each class extends superclass Animal, which contains a method move and maintains an animal’s current location as x-y coordinates. Each subclass implements method move. Our program maintains an Animal array containing references to objects of the various Animal subclasses. To simulate the animals’ movements, the program sends each object the same message once per second—namely, move. Each specific type of Animal responds to a move message in its own way—a Fish might swim three feet, a Frog might jump five feet and a Bird might fly ten feet. Each object knows how to modify its x-y coordinates appropriately for its specific type of movement.

Relying on each object to know how to “do the right thing” (i.e., do what is appropriate for that type of object) in response to the same method call is the key concept of polymorphism. The same message (in this case, move) sent to a variety of objects has “many forms” of results—hence the term polymorphism



```
Employee [] employees = new Employee [4];
employees[0] = new CommisionEmployee();
employees[1] = new SalariedEmployee();
employees[2] = new HourlyEmployee();
```

```
employees[3] = new BaseCommEmployee();
for ( int i=0; i<=3;i++)
{
System.out.printf(employees[i].earnngs());
}
```

2) Solved Lab Activities (Allocated Time 50 Min.)

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***25 mins***	***Medium***	***CLO-4***
Activity 2	***25 mins***	***Medium***	***CLO-4***

Activity 1:

The following example demonstrates polymorphism for a overridden method.

Solution:

```
public abstract class Employee
{
private String firstName;
private String lastName;
private String socialSecurityNumber;

public Employee( String first, String last, String ssn )
{
firstName = first;
lastName = last;
socialSecurityNumber = ssn;
}
public String toString()
{
return String.format( "%s %s\nsocial security number: %s",
firstName, lastName, socialSecurityNumber );
} // end method toString
public abstract double earnings();
}
```

```
-----------------------------------------------------------------------
Public class SalariedEmployee extends Employee
{
private double weeklySalary;
// four-argument constructor
public SalariedEmployee( String first, String last, String ssn,double
salary )
{
super( first, last, ssn ); // pass to Employee constructor
weeklySalary = salary ;
}

@Override
public double earnings()
{
return weeklySalary;
}
}
-----------------------------------------------------------------------
public class HourlyEmployee extends Employee
{
private double wage; // wage per hour
private double hours; // hours worked for week

// five-argument constructor
public HourlyEmployee( String first, String last, String ssn,double
hourlyWage, double hoursWorked )
{
super( first, last, ssn );
wage  = hourlyWage;
hours  = hoursWorked;
}

@Override
public double earnings()
{
if (hours  <= 40 ) // no overtime
return wage * hours ;
else
return 40 * wage + (hours - 40 ) * wage * 1.5;
} }
--------------------------------------------------------------------
public class CommissionEmployee extends Employee
{
private double grossSales; // gross weekly sales
private double commissionRate; // commission percentage

// five-argument constructor
public CommissionEmployee( String first, String last, String
ssn,double sales, double rate )
{
super( first, last, ssn );
grossSales =sales ;
commissionRate  =rate;
}

@Override
public double earnings()
{
return commissionRate  * grossSales ;
}
}
--------------------------------------------------------------------
public class BasePlusCommissionEmployee extends CommissionEmployee
{
private double baseSalary; // base salary per week

public BasePlusCommissionEmployee( String first, String last,String
ssn, double sales, double rate, double salary )
{
super( first, last, ssn, sales, rate );
baseSalary = salary; // validate and store base salary
}
```

```
public void setBaseSalary(double baseSalary)
{this.baseSalary = baseSalary;}
public double getBaseSalary() {  return baseSalary;   }
```

```
@Override
public double earnings()
{
return baseSalary + super.earnings();
} }
--------------------------------------------------------------------
public class PayrollSystemTest

{

public static void main( String[] args )

{

SalariedEmployee salariedEmployee = new SalariedEmployee ("John",
"Smith", "111-11-1111", 800.00 );

HourlyEmployee hourlyEmployee= new HourlyEmployee( "Karen", "Price",
"222-22-2222", 16.75, 40 );

CommissionEmployee commissionEmployee = new CommissionEmployee(
"Sue", "Jones", "333-33-3333", 10000, .06 );

BasePlusCommissionEmployee basePlusCommissionEmployee = new
BasePlusCommissionEmployee("Bob", "Lewis", "444-44-4444", 5000, .04,
300 );

Employee[] employees = new Employee[ 4 ];

employees[ 0 ] = salariedEmployee;

employees[ 1 ] = hourlyEmployee;

employees[ 2 ] = commissionEmployee;

employees[ 3 ] = basePlusCommissionEmployee;

for (int i=0; i<4 ;i++)

{
```

```
System.out.println(employees[i].earnings()); //polymorphic call

} } }
```

Output

```
800.0

670.0

600.0

500.0
```

Activity 2:

The following example demonstrates downcasting

```
public class PayrollSystemTest

{

public static void main( String[] args )

{

SalariedEmployee salariedEmployee = new SalariedEmployee ("John",
"Smith", "111-11-1111", 800.00 );

HourlyEmployee hourlyEmployee= new HourlyEmployee( "Karen", "Price",
"222-22-2222", 16.75, 40 );

CommissionEmployee commissionEmployee = new CommissionEmployee(
"Sue", "Jones", "333-33-3333", 10000, .06 );

BasePlusCommissionEmployee basePlusCommissionEmployee = new
BasePlusCommissionEmployee("Bob", "Lewis", "444-44-4444", 5000, .04,
300 );

Employee[] employees = new Employee[ 4 ];

// initialize array with Employees

employees[ 0 ] = salariedEmployee;
employees[ 1 ] = hourlyEmployee;

employees[ 2 ] = commissionEmployee;

employees[ 3 ] = basePlusCommissionEmployee;

for (int i=0; i<4 ;i++)

{

if (employees [i] instanceof BasePlusCommissionEmployee)

{

BasePlusCommissionEmployee emp= (BasePlusCommissionEmployee )
employees[i];

emp.setBaseSalary( 1.10 * emp.getBaseSalary() );

System.out.println("New base salary with 10 percent increase is " +
emp.getBaseSalary() );

employees[i]=emp;

 }

System.out.println("Earning is " + employees[i].earnings());

} //end for

}

}
```

Output

```
Earning is 800.0

Earning is 670.0

Earning is 600.0

New base salary with 10 percent increase is 330.0

Earning is 530.0
```

3) Graded Lab Tasks(Allocated Time 2 Hr.)

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Package-delivery services, offer a number of different shipping options, each with specific costs associated.

Create an inheritance hierarchy to represent various types of packages. Use Package as the super class of the hierarchy, then include classes TwoDayPackage and OvernightPackage that derive from Package.

Super class Package should include data members representing the name and address for both the sender and the recipient of the package, in addition to data members that store the weight (in ounces) and cost per ounce to ship the package. Package's constructor should initialize these data members. Ensure that the weight and cost per ounce contain positive values.
Package should provide a public member function calculateCost() that returns a double indicating the cost associated with shipping the package. Package's calculateCost() function should determine the cost by multiplying the weight by the cost per ounce.

Derived class TwoDayPackage should inherit the functionality of base class Package, but also include a data member that represents a flat fee that the shipping company charges for two-day-delivery service. TwoDayPackage'sconstructor should receive a value to initialize this data member. TwoDayPackage should redefine member function calculateCost() so that it computes the shipping cost by adding the flat fee to the cost calculated by base class Package's calculateCost() function.

Class OvernightPackage should inherit from class Package and contain an additional data member representing an additionalfee charged for overnight-delivery service.
OvernightPackage should redefine member function calculateCost() so that it computes the shipping cost by adding the additionalfee to the cost calculated by base class Package's calculateCost() function.

Write a test program that creates objects of each type of Package and tests member function calculateCost() using polymorphism .

Lab Task 2

Create an abstract class "Person", with data member "name". Create set and get methods, and an abstract Boolean method "isOutstanding()".

Derive two classes Student and Professor. Student class has data member CGPA.

Professor Class has data member numberOfPublications. Provide setters and getters and implementation of abstract function in both classes.

In student class isOutstanding() will return true if CGPA is greater than 3.5. In the Professor class isOutstanding() will return true, if numberOfPublications> 50.

In the main class create an array of Person class and call isOutstanding() function for student and professor. isOutstanding() for professor should be called after setting the publication count to 100.

Lab Task 3

Create a class hierarchy that performs conversions from one system of units to another. Your program should perform the following conversions,
i. Liters to Gallons, ii. Fahrenheit to Celsius and iii. Feet to Meters

The Super class convert declares two variables, val1 and val2, which hold the initial and converted values, respectively. It contains an abstract function "compute()".

The function that will actually perform the conversion, compute() must be defined by the classes derived from convert. The specific nature of compute() will be determined by what type of conversion is taking place.

Three classes will be derived from convert to perform conversions of Liters to Gallons (l_to_g), Fahrenheit to Celsius (f_to_c) and Feet to Meters (f_to_m), respectively. Each derived class implements compute() in its own way to perform the desired conversion.

Test these classes from main() to demonstrate that even though the actual conversion differs between l_to_g, f_to_c, and f_to_m, the interface remains constant.