Lab07: Polymorphism

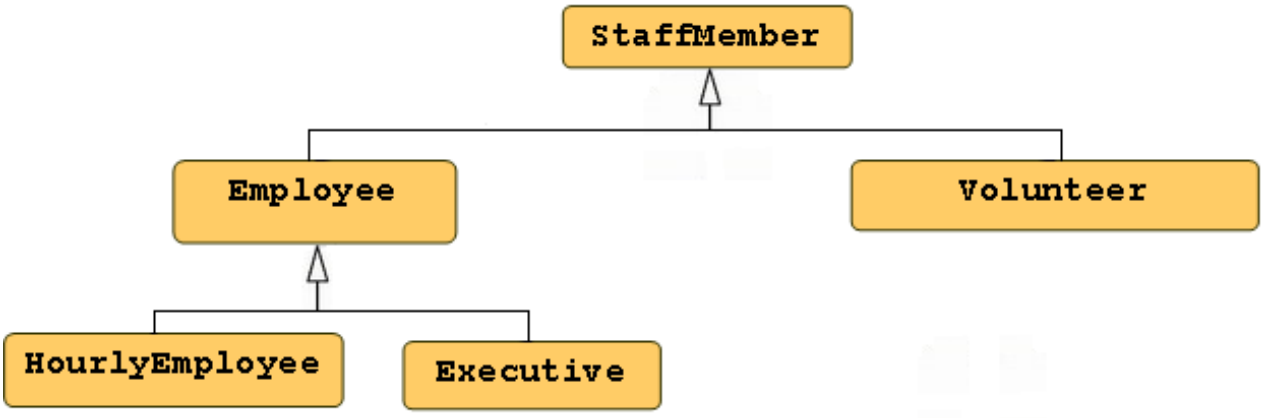
In this lab, the following topics will be covered:

1. **Polymorphism**
2. **upcasting and downcasting**
3. **instanceof operator**
4. **Exercise for practice**

1. Polymorphism (in Java)

The meaning of the word polymorphism is something like *one name, many forms*.

Polymorphism is especially useful when we want to create a heterogeneous collection of objects i.e. making one array containing different objects. Such an example is represented below. Consider a class StaffMember that represents all workers in a certain organization , some of whom are employees and some are volunteers. Among the employees: some are regular employees paid monthly, some are temporary and are paid by the hour, while some are managers (executives) and get bonuses.



The parent class StaffMember.

```
//  StaffMember Class

class StaffMember {
   private String name;
   private String phone;
   public StaffMember (String name, String phone) {
      this.name = name;
      this.phone = phone;
   }
   public double pay(){
            return 0.0;
   }
 } // end of class StaffMember
```

The class Volunteer **extends** the StaffMember.

```
// Volunteer Class

class Volunteer extends StaffMember {

   public Volunteer (String name, String phone){
      super (name, phone);
   }
   // No need to override pay, used as inherited

}//end of class Volunteer
```

The class Employee **extends** the StaffMember.

```
//   Employee class

class Employee extends StaffMember
{
   private double payRate;

    public Employee (String name,String phone, double payRate)    {
      super (name, phone);
      this.payRate = payRate;
   }
    public double getPayRate() {return payRate;}
   // override pay method
   public double pay(){return payRate;}
}
```

The class HourlyEmployee **extends** the Employee.

```
// HourlyEmployee.java
class HourlyEmployee extends Employee{
   private int hoursWorked;
   // constructor
   public HourlyEmployee (String name,String phone,double payRate)  {
      super (name,phone, payRate);
      hoursWorked = 0;
   }
  // added method
   public void addHours (int moreHours)   {
      hoursWorked += moreHours;
   }
//override method pay of Employee: Compute and return the pay for this
//    HourlyEmployee
    public double pay ()    {
      double payment = getPayRate() * hoursWorked;
      hoursWorked = 0;  // once pay computed set hoursWorked to 0
      return payment;
   }
}
```

The class Executive **extends** the Employee

```
// Executive Class
class Executive extends Employee {
   private double bonus;

   // constructor
   public Executive (String name,String phone, double payRate){
      super (name,phone, payRate);
      bonus = 0;  // bonus has yet to be awarded
   }

   // unique method
   public void awardBonus (double execBonus){
      bonus = execBonus;
   }
   // override method pay  of Employee: which is the
   // regular employee payment plus a one-time bonus
   //-----------------------------------------------
   public double pay (){
      double payment = super.pay() + bonus;
      bonus = 0; // once bonus added reset it to 0
      return payment;
   }
}
```

Suppose we would like to store personnel of all kinds in a single array, so we can easily write code that takes care of all the workers.

For example, suppose we want to implement a method getTotalCost() in class Staff that will compute how much money is needed to pay all personnel at the end of the month

```
/*
 * Staff is driver Class
*/
//  Staff.java
class Staff {
     public static void main (String[] args) {
     StaffMember[] staffList;
      staffList = new StaffMember[4];
      staffList[0] = new Executive ("Ahmad","016-1234567", 2000.00);
      staffList[1] = new Employee ("Ali","017-1234567", 800.50);
      staffList[2] = new HourlyEmployee ("Othman","012-1234567", 8.00);
      staffList[3] = new Volunteer ("Farooq","019-1234567");

      for (int i=0;i< staffList.length;i++) {
           if (staffList[i] instanceof Executive) {
                Executive e=(Executive)staffList[i];
                // downcasting to access awardBonus method
                e.awardBonus (500.00);
           }
           else if (staffList[i] instanceof HourlyEmployee) {
                HourlyEmployee h=(HourlyEmployee)staffList[i];
                // downcasting to access addHours method
                h.addHours (40);
           }
      }
      System.out.println("The total amount to pay is "+getTotalCost
(staffList));
      }
//  compute payday costs
   public static double getTotalCost (StaffMember[] stm){
      double amount = 0.0;

      for (int count=0; count < stm.length; count++){
         amount += stm[count].pay(); // polymorphism
      }
      return amount;
   }

}
```

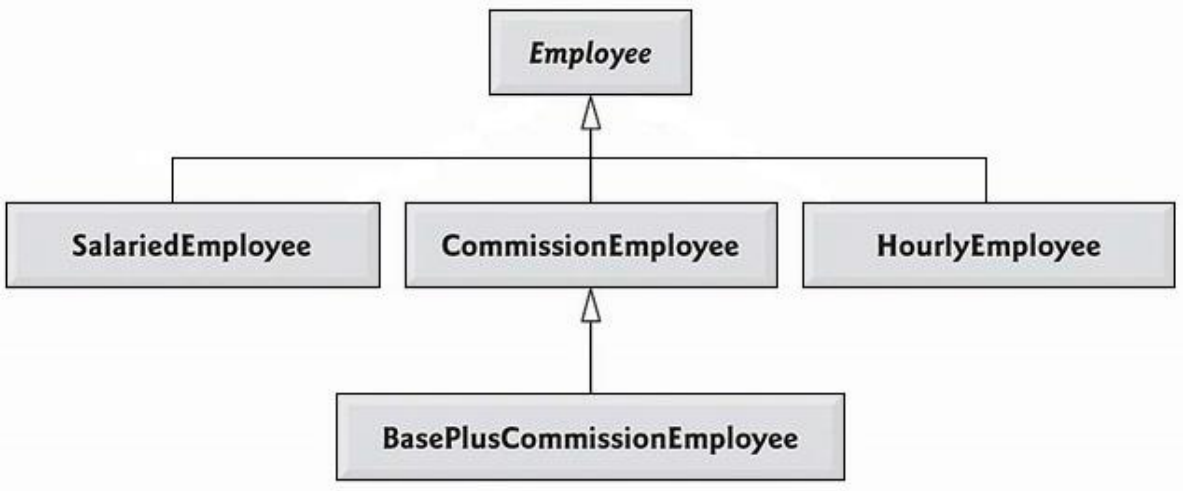
Exercises

Exercise 1 (a)

Create a payroll system using **classes**, **inheritance** and **polymorphism**
Four types of employees paid weekly

1. Salaried employees: fixed salary irrespective of hours
2. Hourly employees: 40 hours salary and overtime (> 40 hours)
3. Commission employees: paid by a percentage of sales
4. Base-plus-commission employees: base salary and a percentage of sales

The information know about each employee is his/her first name, last name and national identity card number. The reset depends on the type of employee.



Step by Step Guidelines

Step 1: Define Employee Class

- Being the base class, Employee class contains the common behavior. Add firstName, lastName and CNIC as attributes of type String
- Provide getter & setters for each attribute
- Write default & parameterized constructors
- Override **toString**() method as shown below

```
public String toString( ) {
return firstName + " " + lastName + " CNIC# " + CNIC ;
}
```

- Define **earning()** method as shown below

```
   public double earnings( ) {
   return 0.00;
}
```

Step 2: Define SalariedEmployee Class

- Extend this class from Employee class.
- Add **weeklySalary** as an attribute of type double
- Provide **getter** & **setters** for this attribute. Make sure that **weeklySalary** never sets to **negative** value. (use if)
- Write **default** & **parameterize** constructor. Don't forget to call default & parameterize constructors of Employee class.
- Override **toString**() method as shown below

```
public String toString( ) {
   return "\nSalaried employee: " + super.toString();
}
```

- Override **earning**() method to implement class specific behavior as shown below

```
public double earnings( ) {
   return weeklySalary;
}
```

Step 3: Define HourlyEmployee Class

- Extend this class from Employee class.
- Add **wage** and **hours** as attributes of type double
- Provide **getter** & **setters** for these attributes. Make sure that **wage** and **hours** never set to a negative value.
- Write default & parameterize constructor. Don't forget to call default & parameterize constructors of Employee class.
- Override **toString**() method as shown below

```
public String toString( ) {
return "\nHourly employee: " + super.toString();
}
```

- Override **earning**() method to implement class specific behaviour as shown below

```
public double earnings( ) {
   if (hours <= 40){
         return wage * hours;
   }
   else{
         return 40*wage + (hours-40)*wage*1.5;
   }
}
```

Step 4: Define CommissionEmployee Class

- Extend this class form Employee class.
- Add **grossSales** and **commissionRate** as attributes of type double
- Provide **getter** & setters for these attributes. Make sure that grossSales and commissionRate never set to a negative value.

- Write default & parameterize constructor. Don't forget to call default & parameterize constructors of Employee class.
- Override **toString**() method as shown below

```
public String toString( ) {
return "\nCommission employee: " + super.toString();
}
```

- Override **earning**() method to implement class specific behaviour as shown below

```
public double earnings( ) {
return grossSales * commisionRate;
}
```

Step 5: Define BasePlusCommissionEmployee Class

- Extend this class form **CommissionEmployee** class not from Employee class. Why? Think on it by yourself
- Add **baseSalary** as an attribute of type double
- Provide **getter** & **setters** for these attributes. Make sure that **baseSalary** never sets to negative value.
- Write default & parameterize constructor. Don't forget to call default & parameterize constructors of Employee class.
- Override **toString**() method as shown below

```
public String toString( ) {
   return "\nBase plus Commission employee: " + super.toString();
}
```

- Override **earning**() method to implement class specific behaviour as shown below

```
public double earnings( ) {
   return baseSalary + super.earning();
}
```

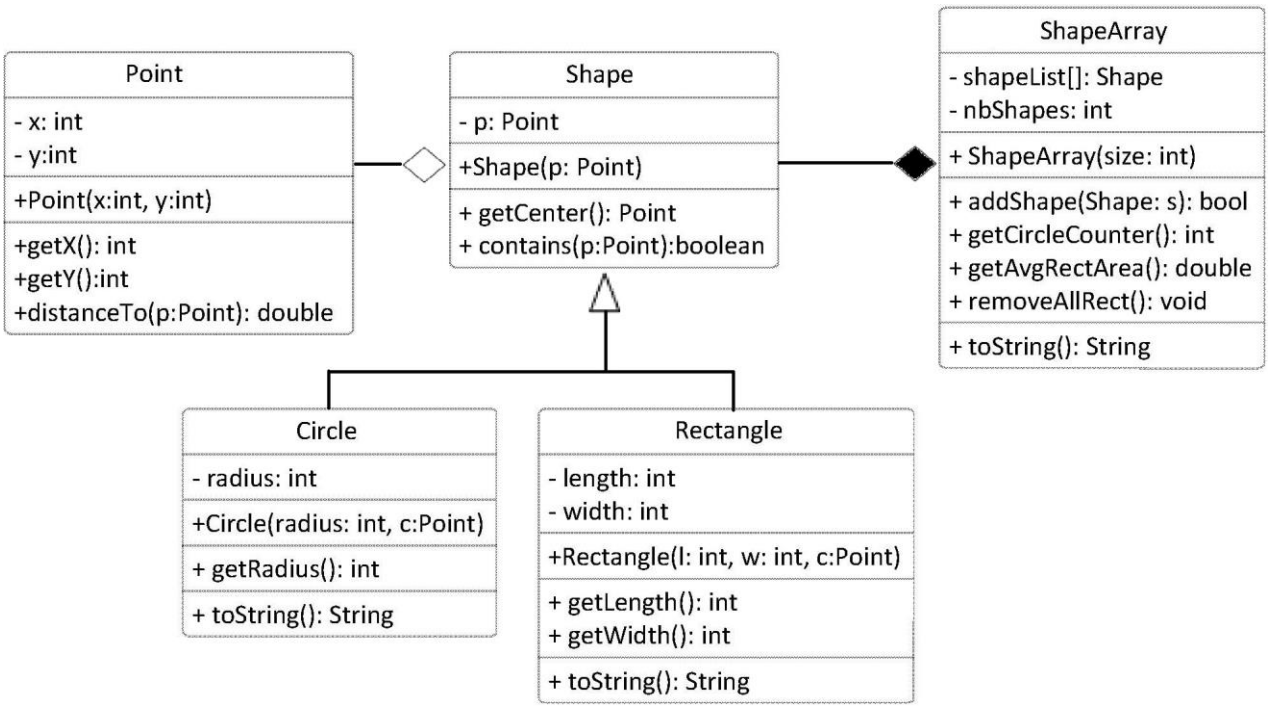
Exercise 1 (b)

Step 6: Putting it all Together

```
public class PayRollSystemTest {
    public static void main (String [] args) {

    Employee firstEmployee = new SalariedEmployee("Usman" ,"Ali","111-
11-1111", 800.00 );

    Employee secondEmployee = new CommissionEmployee("Atif" ,"Aslam",
"222-22-2222", 10000, 0.06 );

    Employee thirdEmployee = new BasePlusCommissionEmployee("Rana",
"Naseeb", "333-33-3333", 5000 , 0.04 , 300 );

    Employee fourthEmployee = new HourlyEmployee( "Renson" , "Isaac",
"444-44-4444" , 16.75 , 40 );
```

```
    // polymorphism: calling toString() and earning() on Employee's
reference
    System.out.println(firstEmployee);
    System.out.println(firstEmployee.earnings());
    System.out.println(secondEmployee);
    System.out.println(secondEmployee.earnings());

    System.out.println(thirdEmployee);
    // performing downcasting to access & raise base salary
    BasePlusCommissionEmployee currentEmployee =
    (BasePlusCommissionEmployee) thirdEmployee;

    double oldBaseSalary = currentEmployee.getBaseSalary();
    System.out.println( "old base salary: " + oldBaseSalary) ;

    currentEmployee.setBaseSalary(1.10 * oldBaseSalary);
    System.out.println("new base salary with 10% increase is:"+
currentEmployee.getBaseSalary());

    System.out.println(thirdEmployee.earnings() );

    System.out.println(fourthEmployee);
    System.out.println(fourthEmployee.earnings() );

    } // end main

} // end class
```

Exercise 2 (a)



Implement classes: Shape, Circle and Rectangle based on the class diagram and description below:

Class Point implementation is given as follow:

```
class Point {
      private int x;
      private int y;
      public Point(int x, int y) {
            this.x = x;
            this.y = y;
      }
      public int getX() { return x;}
      public int getY() { return y;}
      public double distanceTo(Point p) {
            return Math.sqrt((x-p.getX())*(x-p.getX())+
            (y-p.getY())*(y-p.getY()));
      }
      public String toString() {
            return "("+x+", "+y+")";
      }
}
```

Class **Shape** has:

- An attributes of type Point, specifies the center of the shape object.
- A constructor that allows to initialize the center attribute with the value of the passed parameter
- A method that takes an object of type Point as a parameter and returns true if the point resides within the shape's area, and false otherwise.

Class **Circle** has:

- An attribute of type integer specifies the radius measure of the circle
- A constructor that takes a Point parameter to initialize the center and an integer parameter to initialize the radius
- A getRadius method to return the value of the attribute radius
- An overriding version of toString method to return the attribute values of a Circle object as String

Class **Rectangle** has:

- Two integer attributes represents the length and width of the Rectangle object
- A constructor to initialize the center, length and width attribute for a new Rectangle object
- Methods getLength and getWidth returns the values of attributes length and width respectively
- An overriding version of toString method to return the attribute values of a Rectangle object as a String

Class **ShapesArray**

- displayrectsinfo() →display all rectangles information
- getCirclecounter():int →return the number of circles
- getAvgAreas():double →return the average area of all shapes
- removeallrect() →delete all rectangles

Exercise 2 (b)

Step 6: Putting it all Together

Implementation TestShape as given.

create ShapesArray object with size=20
display these options

1. add new shape
    a. for rectangle (ask for details)
    b. for circle (ask for details)
2. display all rectangles
3. display the average shapes area
4. display the number of circles
5. remove all rectangles
6. exit

Project

- From last week you have found the Inheritance between the classes and now find the Polymorphism in your classes.