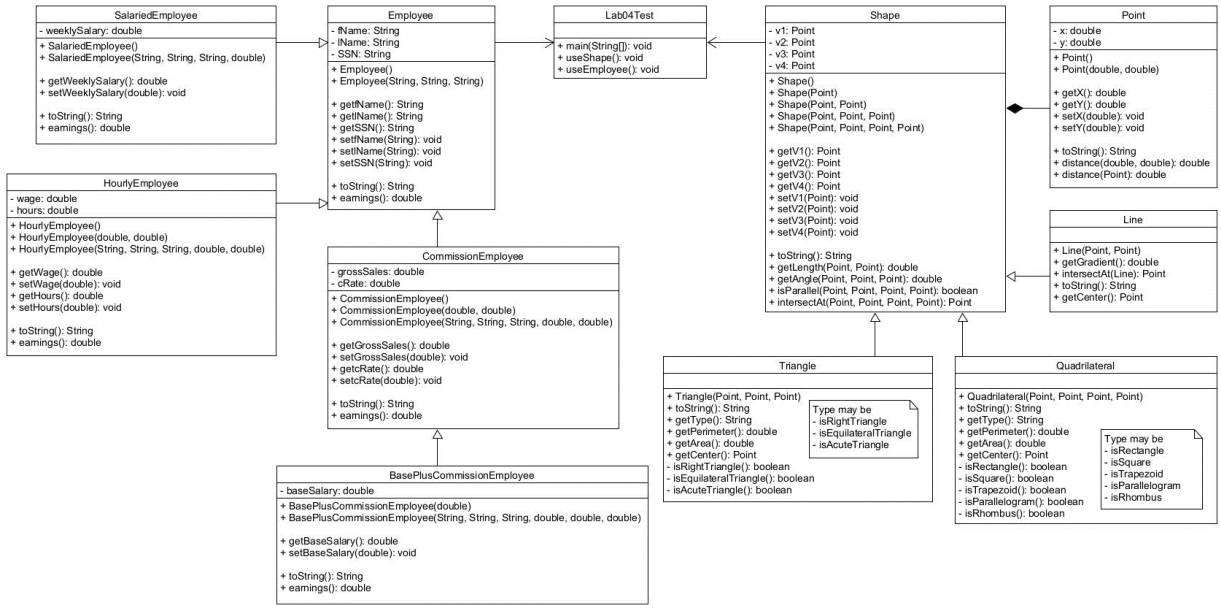
Java Programming (CSE220)

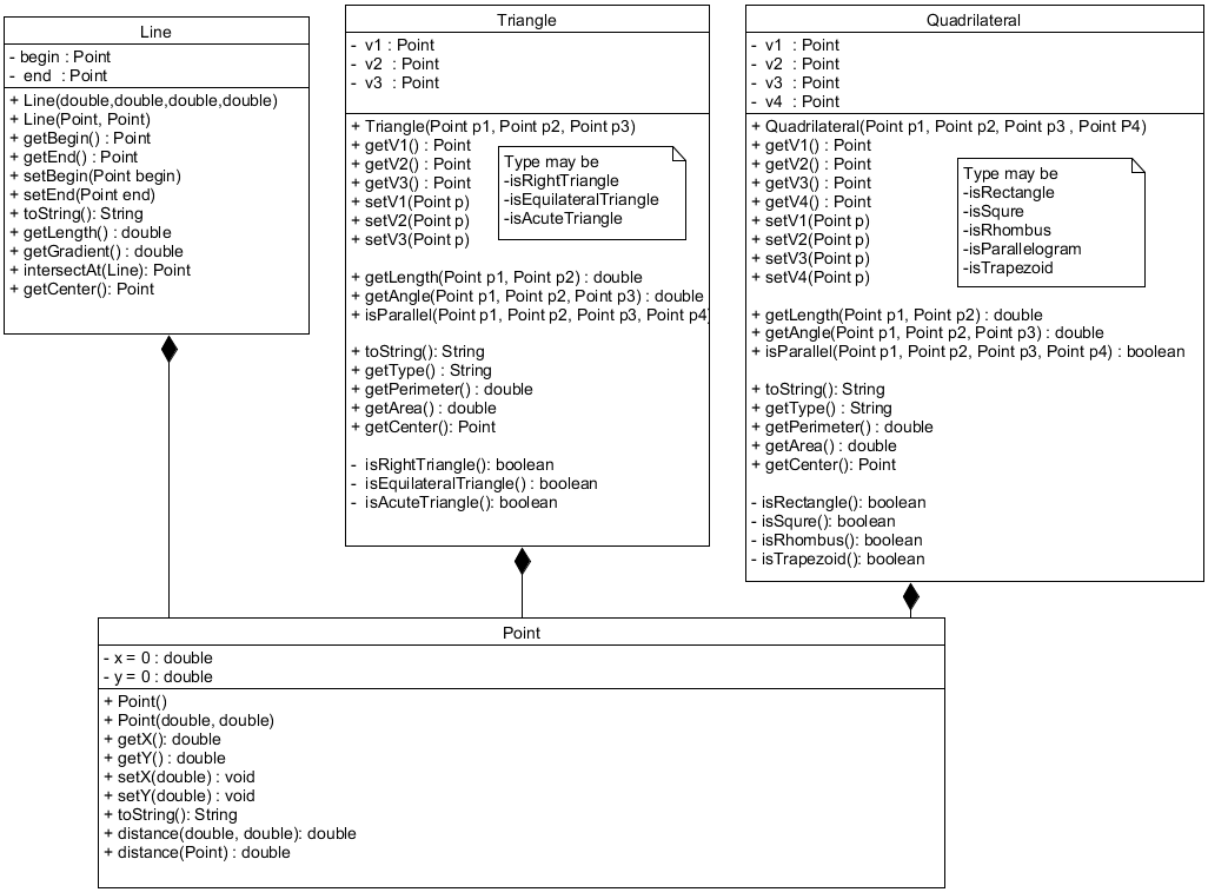
Lab 04

ID: 2020136129 Name: 최수연

Class diagram



Task-1: Consider the following class for 2D shapes. This design uses composition as Line, Triangle, and Quadrilateral shapes are composed of Points. Modify the design using Inheritance and write Java code for each class.



Code with Explanation

Lab04Test class

```
public class Lab04Test {

	public static void main(String[] args) {
		useShape();
	}

	public static void useShape() {
		// The four points
		Point v1 = new Point(0, 0);
		Point v2 = new Point(1, 1);
		Point v3 = new Point(2, 0);
		Point v4 = new Point(1, -1);

		// Line
		System.out.println("[Line]");
		Line l1 = new Line(v1, v2);
```

```
            System.out.println("1. " + l1);
            System.out.println("Gradient: " + l1.getGradient());
            System.out.println("Center: " + l1.getCenter());
            System.out.println();

            Line l2 = new Line(new Point(3, 3), new Point(1, 5));
            Point intersection1 = l1.intersectAt(l2);
            System.out.println("2. " + l2);

            if (intersection1 != null)
                  System.out.println("Intersection between Line 1 and Line 2: " +
intersection1);
            else
                  System.out.println("Line 1 and Line 2 are parallel.");
            System.out.println();

            Line l3 = new Line(new Point(3, 3), new Point(4, 4));
            Point intersection2 = l1.intersectAt(l3);
            System.out.println("3. " + l3);

            if (intersection2 != null)
                  System.out.println("Intersection between Line 1 and Line 3: " +
intersection2);
            else
                  System.out.println("Line 1 and Line 3 are parallel.");
            System.out.println();

            // Triangle
            System.out.println("[Triangle]");
            Triangle t1 = new Triangle(v1, v2, v3);
            System.out.println(t1);
            System.out.println("Type: " + t1.getType());
            System.out.println("Perimeter: " + t1.getPerimeter());
            System.out.println("Area: " + t1.getArea());
            System.out.println("Center: " + t1.getCenter());
            System.out.println();

            // Quadrilateral
            System.out.println("[Quadrilateral]");
            Quadrilateral q1 = new Quadrilateral(v1, v2, v3, v4);
            System.out.println(q1);
            System.out.println("Type: " + q1.getType());
            System.out.println("Perimeter: " + q1.getPerimeter());
            System.out.println("Area: " + q1.getArea());
            System.out.println("Center: " + q1.getCenter());
            System.out.println();

            // Distance between two points
            System.out.println("[Distance between two points]");
            System.out.println("v1 and v2 = " + v1.distance(v2));
            System.out.println("Line 1's v2 and Line 2's v2 = " +
l1.getV2().distance(l2.getV2()));
      }
}
```

Point class

```java
public class Point {
	private double x, y;

	public Point() {
		this.x = 0.0;
		this.y = 0.0;
	}

	public Point(double x, double y) {
		this.x = x;
		this.y = y;
	}

	public double getX() {
		return x;
	}

	public void setX(double x) {
		this.x = x;
	}

	public double getY() {
		return y;
	}

	public void setY(double y) {
		this.y = y;
	}

	@Override
	public String toString() {
		return "(" + x + ", " + y + ")";
	}

	public double distance(double x, double y) {
		// The distance between the current point and the point received by
the factor
		return Math.sqrt(Math.pow((this.x - x), 2) + Math.pow((this.y - y),
2));
	}

	public double distance(Point p) {
		// The distance between the current point and the point received by
the factor
		return Math.sqrt(Math.pow((this.x - p.x), 2) + Math.pow((this.y -
p.y), 2));
	}
}
```

Shape class

```java
public class Shape {
	private Point v1, v2, v3, v4;
```

```
	// Definition Shape constructors
	public Shape() {
		this.v1 = new Point();
		this.v2 = new Point();
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1) {
		this.v1 = v1;
		this.v2 = new Point();
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2) {
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = new Point();
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2, Point v3) {
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = v3;
		this.v4 = new Point();
	}

	public Shape(Point v1, Point v2, Point v3, Point v4) {
		this.v1 = v1;
		this.v2 = v2;
		this.v3 = v3;
		this.v4 = v4;
	}

	public Point getV1() {
		return v1;
	}

	public void setV1(Point v1) {
		this.v1 = v1;
	}

	public Point getV2() {
		return v2;
	}

	public void setV2(Point v2) {
		this.v2 = v2;
	}

	public Point getV3() {
		return v3;
```

```
	}

	public void setV3(Point v3) {
		this.v3 = v3;
	}

	public Point getV4() {
		return v4;
	}

	public void setV4(Point v4) {
		this.v4 = v4;
	}

	@Override
	public String toString() {
		return "Shape [v1 = " + v1 + ", v2 = " + v2 + ", v3 = " + v3 + ", v4
= " + v4 + "]";
	}

	public double getLength(Point v1, Point v2) {
		// distance between two points
		return Math.sqrt(Math.pow((v2.getX() - v1.getX()), 2) +
Math.pow((v2.getY() - v1.getY()), 2));
	}

	public double getAngle(Point v1, Point v2, Point v3) {
		// compute angle between two lines
		double m1 = (v2.getY() - v1.getY()) / (v2.getX() - v1.getX());
		double m2 = (v3.getY() - v2.getY()) / (v3.getX() - v2.getX());
		return Math.atan(Math.abs((m2 - m1) / (1 + m1 * m2))); // find Point
v2's angle
	}

	public boolean isParallel(Point v1, Point v2, Point v3, Point v4) {
		// whether the two lines are parallel
		// Same slope and different y-intercept
		double x1 = v1.getX(), y1 = v1.getY();
		double x2 = v2.getX(), y2 = v2.getY();
		double x3 = v3.getX(), y3 = v3.getY();
		double x4 = v4.getX(), y4 = v4.getY();

		double m12 = (y2 - y1) / (x2 - x1);
		double m34 = (y4 - y3) / (x4 - x3);

		double y12 = y1 - (m12 * x1);
		double y34 = y3 - (m34 * x3);

		double p = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);

		if (m12 == m34 && y12 != y34 && p == 0)
			return true;
		else
			return false;
	}
```

```
	public Point intersectAt(Point v1, Point v2, Point v3, Point v4) {
		// calculate intersection
		double x1 = v1.getX(), y1 = v1.getY();
		double x2 = v2.getX(), y2 = v2.getY();
		double x3 = v3.getX(), y3 = v3.getY();
		double x4 = v4.getX(), y4 = v4.getY();

		double p = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
		double px = ((x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 -
y3 * x4)) / p;
		double py = ((x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 -
y3 * x4)) / p;

		if (p == 0) {
			System.out.println("parallel");
			return null;
		} else
			return new Point(px, py);
	}
}
```

Line class

```
public class Line extends Shape {

	public Line(Point v1, Point v2) {
		super(v1, v2);
	}

	public double getGradient() {
		// calculate the gradient : y2 - y1 / x2 - x1
		return (this.getV2().getY() - this.getV1().getY()) /
(this.getV2().getX() - this.getV1().getX());
	}

	public Point intersectAt(Line l1) {
		// calculate intersection
		double x1 = this.getV1().getX(), y1 = this.getV1().getY();
		double x2 = this.getV2().getX(), y2 = this.getV2().getY();
		double lx1 = l1.getV1().getX(), ly1 = l1.getV1().getY();
		double lx2 = l1.getV2().getX(), ly2 = l1.getV2().getY();

		double p = (x1 - x2) * (ly1 - ly2) - (y1 - y2) * (lx1 - lx2);
		double px = ((x1 * y2 - y1 * x2) * (lx1 - lx2) - (x1 - x2) * (lx1 *
ly2 - ly1 * lx2)) / p;
		double py = ((x1 * y2 - y1 * x2) * (ly1 - ly2) - (y1 - y2) * (lx1 *
ly2 - ly1 * lx2)) / p;
		if (p == 0)
			return null;
		else
			return new Point(px, py);
	}
```

```
	@Override
	public String toString() {
		return "Line [v1 = " + this.getV1() + ", v2 = " + this.getV2() + "]";
	}

	public Point getCenter() {
		// get the center of a line
		double centerX = (this.getV1().getX() + this.getV2().getX()) / 2;
		double centerY = (this.getV1().getY() + this.getV2().getY()) / 2;
		return new Point(centerX, centerY);
	}
}
```

Triangle class

```
public class Triangle extends Shape {

	public Triangle(Point v1, Point v2, Point v3) {
		super(v1, v2, v3);
	}

	@Override
	public String toString() {
		return "Triangle [v1 = " + this.getV1() + ", v2 = " + this.getV2() +
", v3 = " + this.getV3() + "]";
	}

	public String getType() {
		// output the type of a triangle
		if (isRightTriangle())
			return "It is a right-angled triangle.";
		else if (isEquilateralTriangle())
			return "It is a equilateral triangle.";
		else if (isAcuteTriangle())
			return "It is a acute-angled triangle.";
		else
			return "Nothing.";
	}

	public double getPerimeter() {
		// get the perimeter of a triangle
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV1());
		return a + b + c;
	}

	public double getArea() {
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV1());
		// Triangle by Heron's formula
		double s = (a + b + c) / 2.0;
		return Math.sqrt(s * (s - a) * (s - b) * (s - c));
	}
```

```
	public Point getCenter() {
		// get the center of a triangle
		double centerX = (this.getV1().getX() + this.getV2().getX() +
this.getV3().getX()) / 3;
		double centerY = (this.getV1().getY() + this.getV2().getY() +
this.getV3().getY()) / 3;
		return new Point(centerX, centerY);
	}

	private boolean isRightTriangle() {
		// one angle of 90 degrees
		if ((getAngle(this.getV1(), this.getV2(), this.getV3()) == 90)
				|| (getAngle(this.getV2(), this.getV3(), this.getV1())
== 90)
				|| (getAngle(this.getV3(), this.getV1(), this.getV2())
== 90))
			return true;
		else
			return false;
	}

	private boolean isEquilateralTriangle() {
		// 3 equal sides & 3 equal angles
		if ((getAngle(this.getV1(), this.getV2(), this.getV3()) == 60)
				&& (getAngle(this.getV2(), this.getV3(), this.getV1())
== 60)
				&& (getAngle(this.getV3(), this.getV1(), this.getV2())
== 60)
				&& (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV2(), this.getV3()))
				&& (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV3(), this.getV1())))
			return true;
		else
			return false;
	}

	private boolean isAcuteTriangle() {
		// 3 angles all less than 90 degrees
		if ((getAngle(this.getV1(), this.getV2(), this.getV3()) < 90)
				&& (getAngle(this.getV2(), this.getV3(), this.getV1()) <
90)
				&& (getAngle(this.getV3(), this.getV1(), this.getV2()) <
90))
			return true;
		else
			return false;
	}
}
```

Quadrilateral class

```
public class Quadrilateral extends Shape {
```

```
	public Quadrilateral(Point v1, Point v2, Point v3, Point v4) {
		super(v1, v2, v3, v4);
	}

	@Override
	public String toString() {
		return "Quadrilateral [v1 = " + this.getV1() + ", v2 = " +
this.getV2() + ", v3 = " + this.getV3() + ", v4 = "
				+ this.getV4() + "]";
	}

	public String getType() {
		// output the type of a quadrilateral
		if (isRectangle())
			return "It is a rectangle.";
		else if (isSquare())
			return "It is a square.";
		else if (isRhombus())
			return "It is a rhombus.";
		else if (isTrapezoid())
			return "It is a trapezium.";
		else if (isParallelogram())
			return "It is a parallelogram.";
		else
			return "Nothing.";
	}

	public double getPerimeter() {
		// get the perimeter of a quadrilateral
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV4());
		double d = getLength(this.getV4(), this.getV1());
		return a + b + c + d;
	}

	public double getArea() {
		double a = getLength(this.getV1(), this.getV2());
		double b = getLength(this.getV2(), this.getV3());
		double c = getLength(this.getV3(), this.getV4());
		double d = getLength(this.getV4(), this.getV1());
		// Bretschneider's formula
		double s = (a + b + c + d) / 2.0;
		double theta = getAngle(this.getV1(), this.getV2(), this.getV3())
				+ getAngle(this.getV1(), this.getV4(), this.getV3());
		return Math.sqrt((s - a) * (s - b) * (s - c) * (s - d) - ((a * b * c
* d) * Math.pow(Math.cos(theta / 2), 2)));
	}

	public Point getCenter() {
		// get the center of a quadrilateral
		double centerX = (this.getV1().getX() + this.getV2().getX() +
this.getV3().getX() + this.getV4().getX()) / 4;
		double centerY = (this.getV1().getY() + this.getV2().getY() +
this.getV3().getY() + this.getV4().getY()) / 4;
```

```
            return new Point(centerX, centerY);
      }

      private boolean isRectangle() {
            // opposite sides equal and parallel, 4 right angles
            if (isParallel(this.getV1(), this.getV2(), this.getV3(),
this.getV4()) == true
                        && isParallel(this.getV1(), this.getV4(), this.getV3(),
this.getV2()) == true
                        && (getAngle(this.getV1(), this.getV2(), this.getV3())
== 90)
                        && (getAngle(this.getV2(), this.getV3(), this.getV4())
== 90)
                        && (getAngle(this.getV3(), this.getV4(), this.getV1())
== 90)
                        && (getAngle(this.getV4(), this.getV1(), this.getV2())
== 90)
                        && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV3(), this.getV4()))
                        && (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV4(), this.getV1())))
                  return true;
            else
                  return false;
      }

      private boolean isSquare() {
            // 4 equal sides, 4 right angles, opposite sides parallel
            if (isRectangle() == true && (getLength(this.getV1(), this.getV2())
== getLength(this.getV2(), this.getV3()))
                        && (getLength(this.getV3(), this.getV4()) ==
getLength(this.getV4(), this.getV1())))
                  return true;
            else
                  return false;
      }

      private boolean isTrapezoid() {
            // one pair of parallel sides
            if (isParallel(this.getV1(), this.getV2(), this.getV3(),
this.getV4()) == true
                        || isParallel(this.getV1(), this.getV4(), this.getV3(),
this.getV2()) == true)
                  return true;
            else
                  return false;
      }

      private boolean isParallelogram() {
            // opposite sides equal and parallel, opposite angles equal
            if (isTrapezoid() == true
                        && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV3(), this.getV4()))
                        && (getLength(this.getV2(), this.getV3()) ==
getLength(this.getV4(), this.getV1())))
```

```
                        return true;
                else
                        return false;
        }

        private boolean isRhombus() {
                // 4 equal sides, opposite sides parallel, opposite angles equal
                if (isParallelogram() == true
                                && (getLength(this.getV1(), this.getV2()) ==
getLength(this.getV2(), this.getV3()))
                                && (getLength(this.getV3(), this.getV4()) ==
getLength(this.getV4(), this.getV1())))
                        return true;
                else
                        return false;
        }
}
```

Results/Output

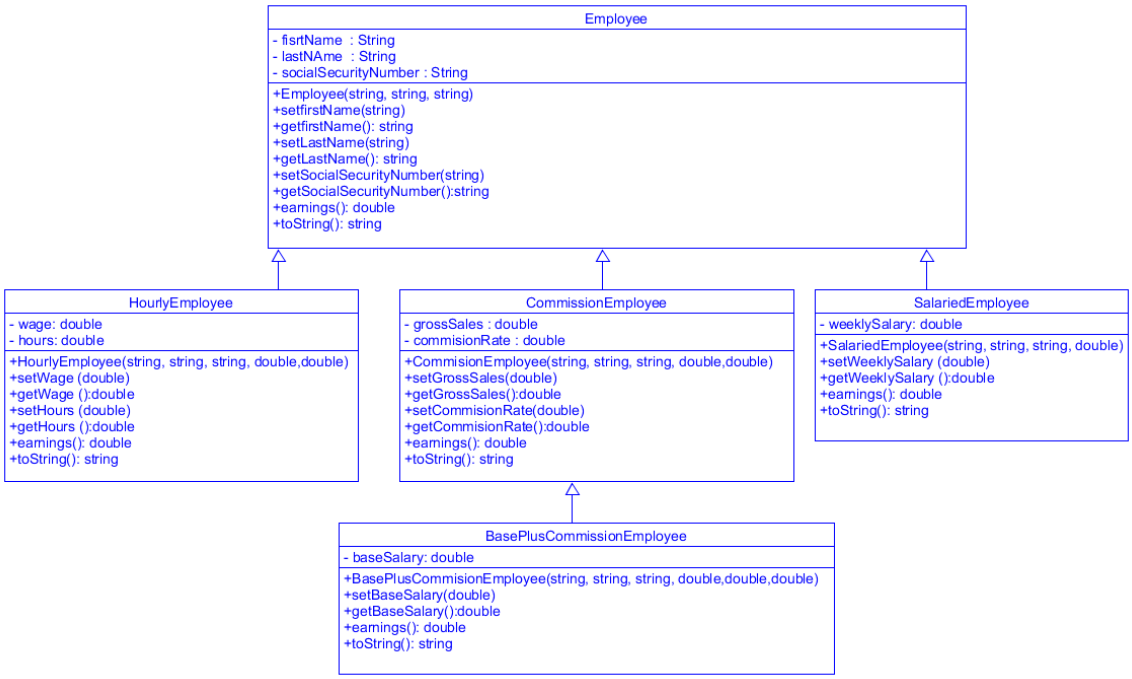
```
[Line]
1. Line [v1 = (0.0, 0.0), v2 = (1.0, 1.0)]
Gradient: 1.0
Center: (0.5, 0.5)

2. Line [v1 = (3.0, 3.0), v2 = (1.0, 5.0)]
Intersection between Line 1 and Line 2: (3.0, 3.0)

3. Line [v1 = (3.0, 3.0), v2 = (4.0, 4.0)]
Line 1 and Line 3 are parallel.
```

```
[Triangle]
Triangle [v1 = (0.0, 0.0), v2 = (1.0, 1.0), v3 = (2.0, 0.0)]
Type: It is a acute-angled triangle.
Perimeter: 4.82842712474619
Area: 0.9999999999999996
Center: (1.0, 0.3333333333333333)
```

```
[Quadrilateral]
Quadrilateral [v1 = (0.0, 0.0), v2 = (1.0, 1.0), v3 = (2.0, 0.0), v4 = (1.0, -1.0)]
Type: It is a rhombus.
Perimeter: 5.656854249492381
Area: 2.0
Center: (1.0, 0.0)
```

```
[Distance between two points]
v1 and v2 = 1.4142135623730951
Line 1's v2 and Line 2's v2 = 4.0
```

Task-2: Consider the payroll system and write code for the following classes (Inheritance)



Code with Explanation

Lab04Test class

```
public class Lab04Test {

	public static void main(String[] args) {
		useEmployee();
	}

	public static void useEmployee() {
		// Employee
		System.out.println("[Employee]");
		Employee e1 = new Employee();
		System.out.println(e1);
		System.out.println();
		System.out.println("Set employee's info!");
		e1.setfName("GILDONG");
		e1.setlName("HONG");
		e1.setSSN("2020123456");
		System.out.println(e1);
		System.out.println();

		// SalariedEmployee
		System.out.println("[SalariedEmployee]");
		SalariedEmployee se = new SalariedEmployee("SOOYEON", "CHOI",
"2020136129", 200.00);
		System.out.println(se);
		System.out.println();
```

```
            System.out.println("Changed weekly salary!");
            se.setWeeklySalary(500.00);
            System.out.println(se);
            System.out.println("Earnings: " + se.earnings());
            System.out.println();

            // HourlyEmployee
            System.out.println("[HourlyEmployee]");
            HourlyEmployee he = new HourlyEmployee("SOOYEON", "CHOI",
"2020136129", 9620 , 20);
            System.out.println(he);
            System.out.println("Earnings: " + he.earnings());
            System.out.println();
            System.out.println("Changed hours!");
            he.setHours(40);
            System.out.println(he);
            System.out.println("Earnings: " + he.earnings());
            System.out.println();

            // CommissionEmployee
            System.out.println("[CommissionEmployee]");
            CommissionEmployee ce = new CommissionEmployee("SOOYEON", "CHOI",
"2020136129", 100000 , 10);
            System.out.println(ce);
            System.out.println("Earnings: " + ce.earnings());
            System.out.println();
            System.out.println("Changed grossSales and cRate!");
            ce.setGrossSales(200000);
            ce.setcRate(15);
            System.out.println(ce);
            System.out.println("Earnings: " + ce.earnings());
            System.out.println();

            // BasePlusCommissionEmployee
            System.out.println("[BasePlusCommissionEmployee]");
            BasePlusCommissionEmployee be = new
BasePlusCommissionEmployee("SOOYEON", "CHOI", "2020136129", 100000 , 10, 300000);
            System.out.println(be);
            System.out.println("Earnings: " + be.earnings());
            System.out.println();
            System.out.println("Changed grossSales and cRate!");
            be.setGrossSales(200000);
            be.setcRate(15);
            System.out.println(be);
            System.out.println("Earnings: " + be.earnings());
            System.out.println();
            System.out.println("Changed base salary!");
            be.setBaseSalary(1000000);
            System.out.println(be);
            System.out.println("Earnings: " + be.earnings());
            System.out.println();
      }
}
```

Employee class

```
```

```java
public class Employee {
	private String fName;
	private String lName;
	private String SSN;

	public Employee() {
		this.fName = "";
		this.lName = "";
		SSN = "";
	}

	public Employee(String fName, String lName, String sSN) {
		this.fName = fName;
		this.lName = lName;
		SSN = sSN;
	}

	public String getfName() {
		return fName;
	}

	public void setfName(String fName) {
		this.fName = fName;
	}

	public String getlName() {
		return lName;
	}

	public void setlName(String lName) {
		this.lName = lName;
	}

	public String getSSN() {
		return SSN;
	}

	public void setSSN(String sSN) {
		SSN = sSN;
	}

	@Override
	public String toString() {
		return "Fisrt Name: " + fName + " , Last Name: " + lName + " , SSN: "
+ SSN;
	}

	public double earnings() {
		double ic = 0.0;
		return ic;
	}
}
```

HourlyEmployee class

```java
public class HourlyEmployee extends Employee {
	private double wage;
	private double hours;

	public double getWage() {
		return wage;
	}

	public void setWage(double wage) {
		this.wage = wage;
	}

	public double getHours() {
		return hours;
	}

	public void setHours(double hours) {
		this.hours = hours;
	}

	public HourlyEmployee() {
		super();
		this.wage = 0.0;
		this.hours = 0.0;
	}

	public HourlyEmployee(String fName, String lName, String sSN, double wage,
double hours) {
		super(fName, lName, sSN);
		this.wage = wage;
		this.hours = hours;
	}

	public HourlyEmployee(double wage, double hours) {
		super();
		this.wage = wage;
		this.hours = hours;
	}

	@Override
	public String toString() {
		return "HourlyEmployee [Wage: " + wage + " , hours: " + hours + " , "
+ super.toString() + "]";
	}

	@Override
	public double earnings() {
		// Hourly employees are paid by the hour and receive overtime pay for
all hours worked in excess of 40 hours
		// i.e., 1.5 times their hourly salary rate
		double ic = 0.0;
		if (hours <= 40)
			ic = hours * wage;
		else
```

```
                    ic = (40 * wage) + ((hours - 40) * (wage * 1.5));
            return ic;
        }
}
```

SalariedEmployee class

```
public class SalariedEmployee extends Employee {
        private double weeklySalary;

        public double getWeeklySalary() {
                return weeklySalary;
        }

        public void setWeeklySalary(double weeklySalary) {
                this.weeklySalary = weeklySalary;
        }

        public SalariedEmployee() {
                super();
                this.weeklySalary = 0.0;
        }

        public SalariedEmployee(String fName, String lName, String sSN, double
weeklySalary) {
                super(fName, lName, sSN);
                this.weeklySalary = weeklySalary;
        }

        @Override
        public String toString() {
                return "SalariedEmployee [weeklySalary: " + weeklySalary + " , " +
super.toString() + "]";
        }

        @Override
        public double earnings() {
                // Salaried employees are paid a fixed weekly salary regardless of
the number of hours worked
                return weeklySalary;
        }
}
```

CommissionEmployee class

```
public class CommissionEmployee extends Employee {
        private double grossSales;
        private double cRate;

        public double getGrossSales() {
                return grossSales;
        }

        public void setGrossSales(double grossSales) {
                this.grossSales = grossSales;
```

```
	}

	public double getcRate() {
		return cRate;
	}

	public void setcRate(double cRate) {
		this.cRate = cRate;
	}

	public CommissionEmployee() {
		super();
		this.grossSales = 0.0;
		this.cRate = 0.0;
	}

	public CommissionEmployee(double grossSales, double cRate) {
		super();
		this.grossSales = grossSales;
		this.cRate = cRate;
	}

	public CommissionEmployee(String fName, String lName, String sSN, double
grossSales, double cRate) {
		super(fName, lName, sSN);
		this.grossSales = grossSales;
		this.cRate = cRate;
	}

	@Override
	public String toString() {
		return "CommissionEmployee [grossSales: " + grossSales + " , cRate: "
+ cRate + " , " + super.toString() + "]";
	}

	@Override
	public double earnings() {
		// Commission employees are paid a percentage of their sales
		return grossSales * cRate;
	}
}
```

BasePlusCommissionEmployee class

```
public class BasePlusCommissionEmployee extends CommissionEmployee {
	private double baseSalary;

	public double getBaseSalary() {
		return baseSalary;
	}

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

	public BasePlusCommissionEmployee(String fName, String lName, String sSN,
```

```
double grossSales, double cRate,
                    double baseSalary) {
              super(fName, lName, sSN, grossSales, cRate);
              this.baseSalary = baseSalary;
       }

       public BasePlusCommissionEmployee(double baseSalary) {
              super();
              this.baseSalary = baseSalary;
       }

       @Override
       public String toString() {
              return "BasePlusCommissionEmployee [baseSalary = " + baseSalary + " ,
" + super.toString() + "]";
       }

       @Override
       public double earnings() {
              // Base-salaried commission employees receive a base salary plus a
percentage of their sales
              return baseSalary + super.earnings();
       }
}
```

Results/Output

```
[Employee]
Fisrt Name:  , Last Name:  , SSN:

Set employee's info!
Fisrt Name: GILDONG , Last Name: HONG , SSN: 2020123456

[SalariedEmployee]
SalariedEmployee [weeklySalary: 200.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]

Changed weekly salary!
SalariedEmployee [weeklySalary: 500.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 500.0

[HourlyEmployee]
HourlyEmployee [Wage: 9620.0 , hours: 20.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 192400.0

Changed hours!
HourlyEmployee [Wage: 9620.0 , hours: 40.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 384800.0

[CommissionEmployee]
CommissionEmployee [grossSales: 100000.0 , cRate: 10.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 1000000.0

Changed grossSales and cRate!
CommissionEmployee [grossSales: 200000.0 , cRate: 15.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 3000000.0

[BasePlusCommissionEmployee]
BasePlusCommissionEmployee [baseSalary = 300000.0 , CommissionEmployee [grossSales: 100000.0 , cRate: 10.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 1300000.0

Changed grossSales and cRate!
BasePlusCommissionEmployee [baseSalary = 300000.0 , CommissionEmployee [grossSales: 200000.0 , cRate: 15.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 3300000.0

Changed base salary!
BasePlusCommissionEmployee [baseSalary = 1000000.0 , CommissionEmployee [grossSales: 200000.0 , cRate: 15.0 , Fisrt Name: SOOYEON , Last Name: CHOI , SSN: 2020136129]
Earnings: 4000000.0
```

Conclusion

It was good that I was able to learn about inheritance accurately through this task, and I felt better in my head by visualizing it by representing it as a class diagram.

And while doing Task1, I thought that each class had more similar functions than I thought, so I could organize them more neatly.