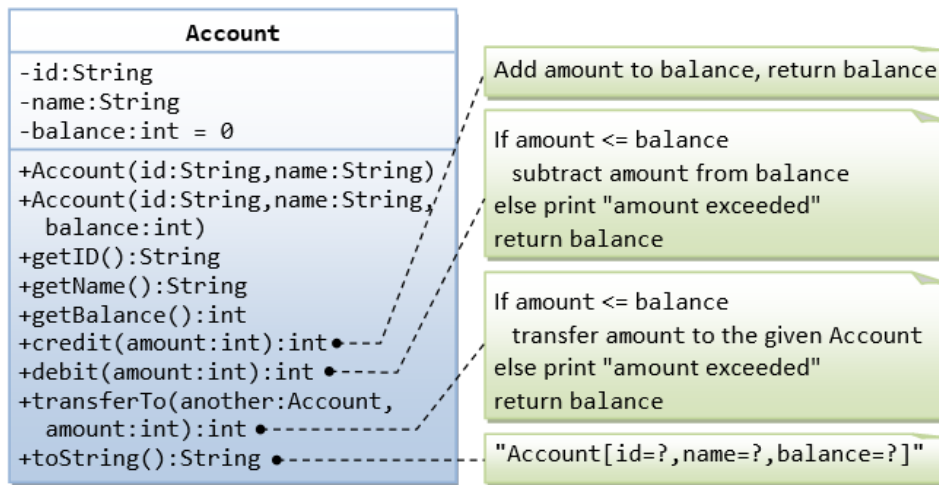
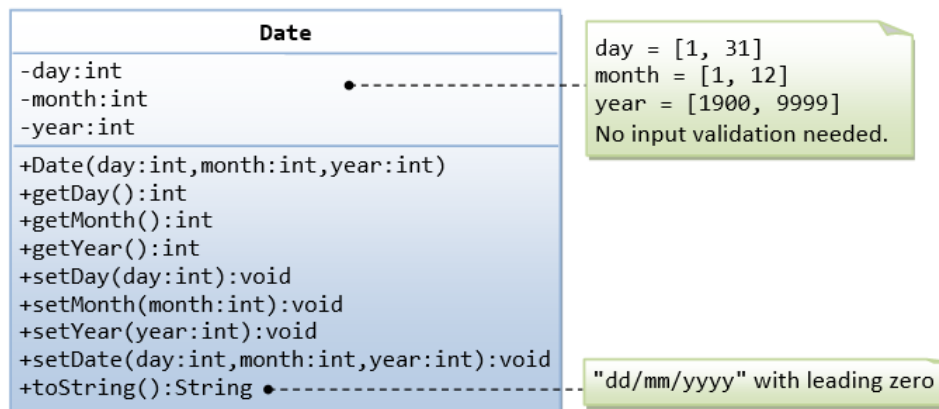


## APARTADO1. CREAR CLASES

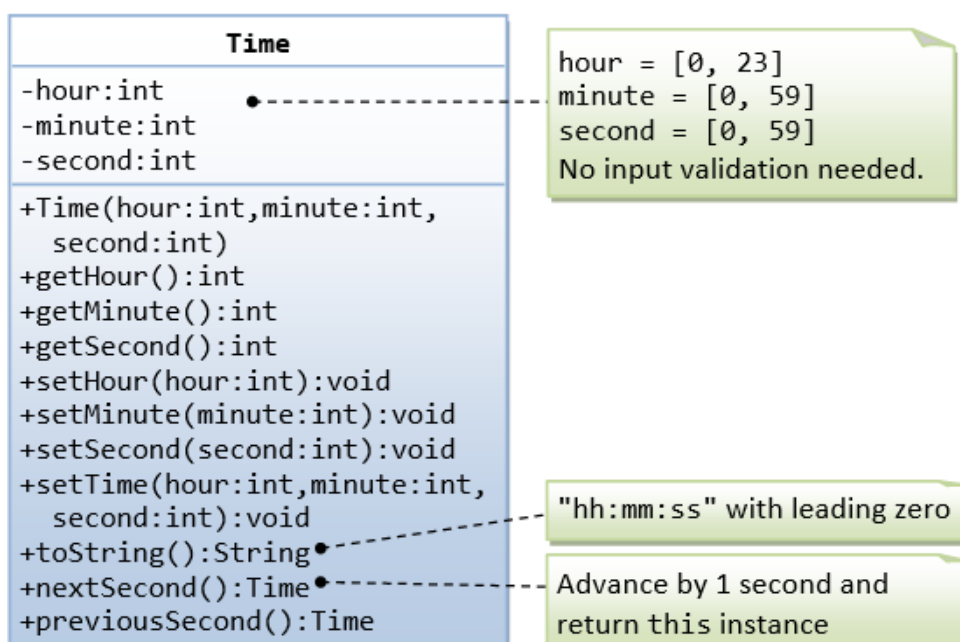
### APARTADO1.EJERCICIO1:



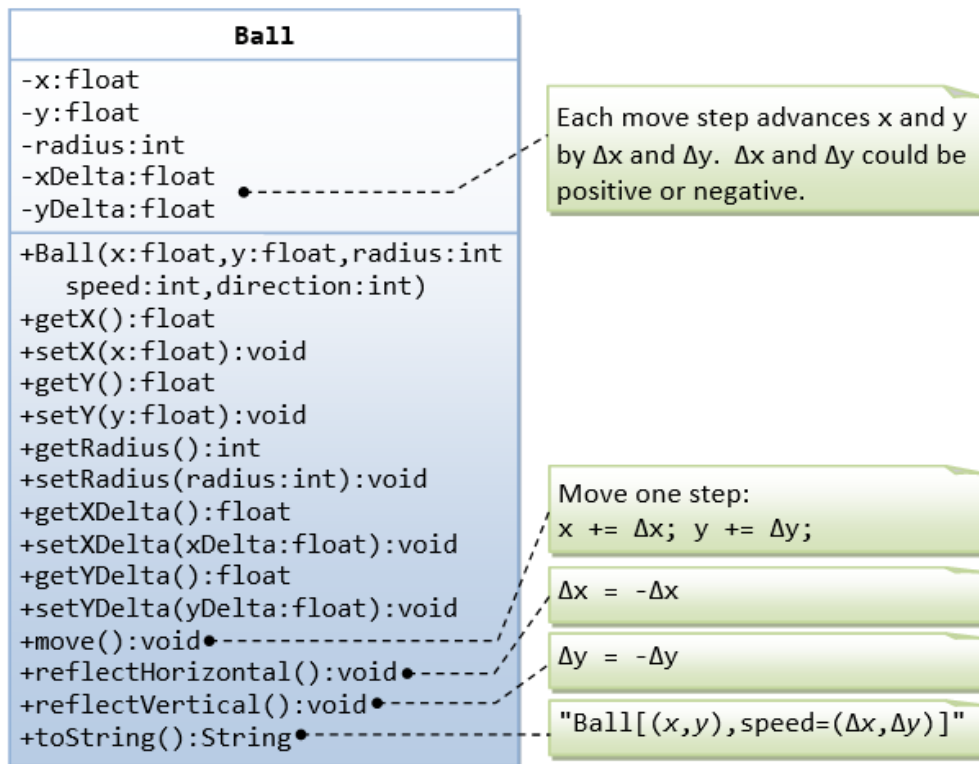
### APARTADO1.EJERCICIO2:



### APARTADO1.EJERCICIO3:



#### APARTADO1.EJERCICIO4:

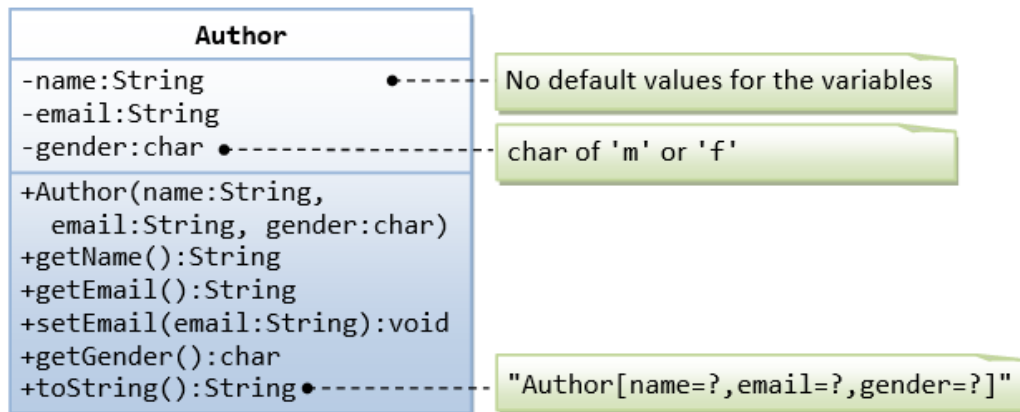


## 2. COMPOSICIÓN

### APARTADO2.EJERCICIO1:

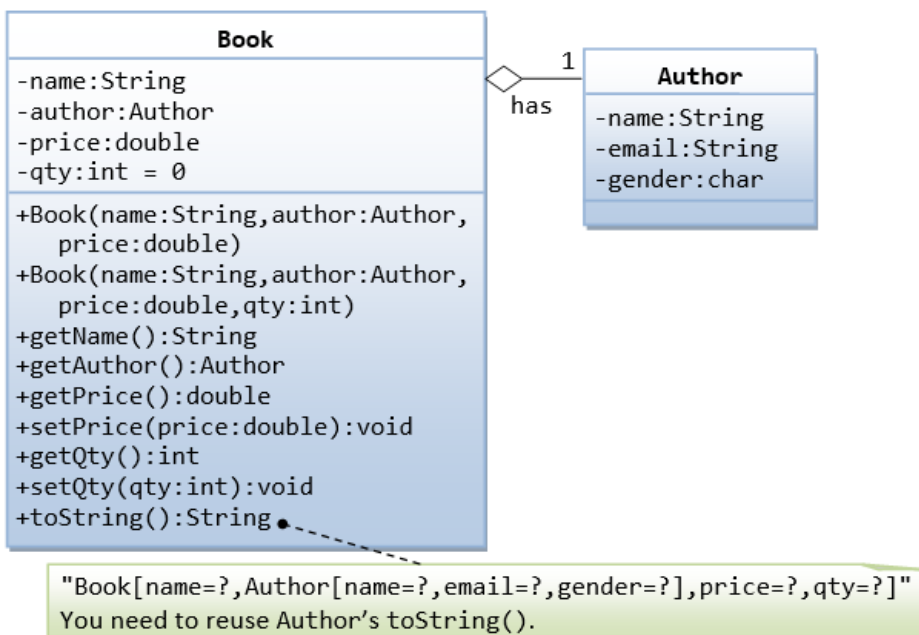
Codificación la relación entre un libro y la información de sus autores.

#### 1.1. CLASE AUTOR



```
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm'); // Test the constructor
System.out.println(ahTeck); // Test toString()
```

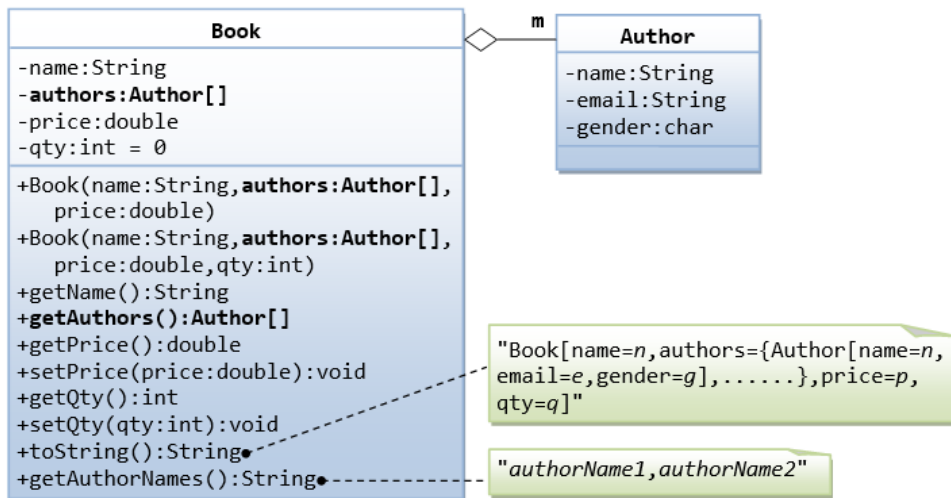
#### 1.2. CLASE LIBRO QUE SÓLO CONTIENE UN ÚNICO AUTOR.



```
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm');
System.out.println(ahTeck); // Author's toString()
```

```
Book dummyBook = new Book("Java for dummy", ahTeck, 19.95, 99); // Test Book's Constructor
System.out.println(dummyBook); // Test Book's toString()
```

### 1.3. CLASE LIBRO QUE PUEDE TENER MÁS DE UN AUTOR.



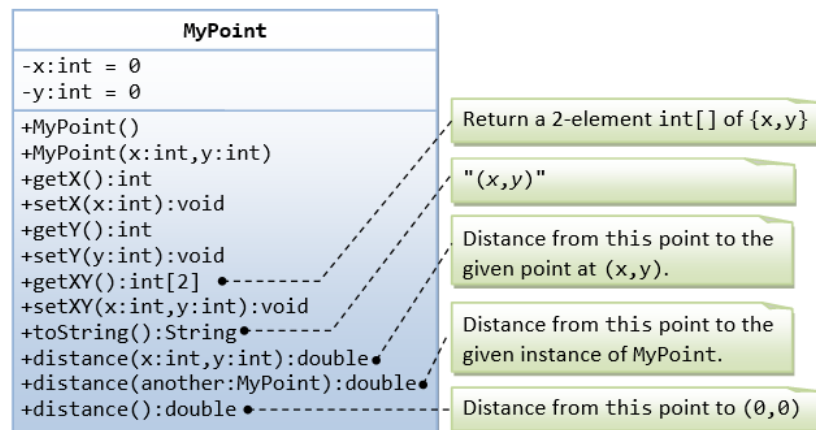
```

// Declare and allocate an array of Authors
Author[] authors = new Author[2];
authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

// Declare and allocate a Book instance
Book javaDummy = new Book("Java for Dummy", authors, 19.99, 99);
System.out.println(javaDummy); // toString()
  
```

Codificación la relación entre un punto y su uso en la definición de diferentes figuras geométricas.

La clase MyPoint que modela un punto 2D definido por las coordenadas X e Y.



```

classDiagram
    class MyCircle {
        -center: MyPoint = (0,0)
        -radius: int = 1
        +MyCircle()
        +MyCircle(x: int, y: int, radius: int)
        +MyCircle(center: MyPoint, radius: int)
        +getRadius(): int
        +setRadius(radius: int): void
        +getCenter(): MyPoint
        +setCenter(center: MyPoint): void
        +getCenterX(): int
        +setCenterX(x: int): void
        +getCenterY(): int
        +setCenterY(y: int): void
        +getCenterXY(): int[2]
        +setCenterXY(x: int, y: int): void
        +toString(): String
        +getArea(): double
        +getCircumference(): double
        +distance(another: MyCircle): double
    }
    class MyPoint {
        -x: int
        -y: int
    }
    MyCircle "1" -- "1" MyPoint : center
  
```

**MyCircle**

- center:MyPoint = (0,0)
- radius:int = 1
- +MyCircle()
- +MyCircle(x:int,y:int,radius:int)
- +MyCircle(center:MyPoint,radius:int)
- +getRadius():int
- +setRadius(radius:int):void
- +getCenter():MyPoint
- +setCenter(center:MyPoint):void
- +getCenterX():int
- +setCenterX(x:int):void
- +getCenterY():int
- +setCenterY(y:int):void
- +getCenterXY():int[2]
- +setCenterXY(x:int,y:int):void
- +toString():String
- +getArea():double
- +getCircumference():double
- +distance(another:MyCircle):double

**MyPoint**

- x:int
- y:int

1 center 1

"MyCircle[radius=r,center=(x,y)]"  
Re-use MyPoint's toString() to print the center's (x,y)

Return the distance between the centers of this circle and the given MyCircle instance another. To re-use MyPoint's distance()

```
classDiagram
    class MyTriangle {
        -v1:MyPoint
        -v2:MyPoint
        -v3:MyPoint
        +MyTriangle(x1:int,y1:int,x2:int,y2:int,x3:int,y3:int)
        +MyTriangle(v1:MyPoint,v2:MyPoint,v3:MyPoint)
        +toString():String
        +getPerimeter():double
        +getType():String
    }
    class MyPoint {
        -x:int
        -y:int
    }
    MyTriangle "3" -- "*" MyPoint : vertices
```

MyTriangle

- v1:MyPoint
- v2:MyPoint
- v3:MyPoint
- +MyTriangle(x1:int,y1:int,x2:int,y2:int,x3:int,y3:int)
- +MyTriangle(v1:MyPoint,v2:MyPoint,v3:MyPoint)
- +toString():String
- +getPerimeter():double
- +getType():String

MyPoint

- x:int
- y:int

3 vertices

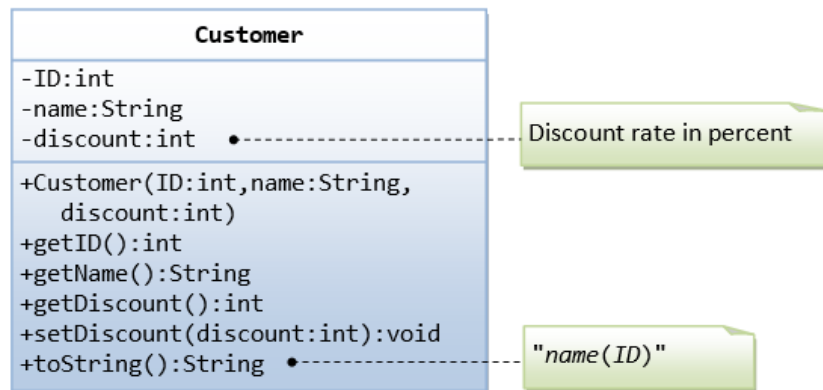
"MyTriangle[v1=(x1,y1),v2=(x2,y2),v3=(x3,y3)]"

Use MyPoint's distance() to compute the length of the edges.

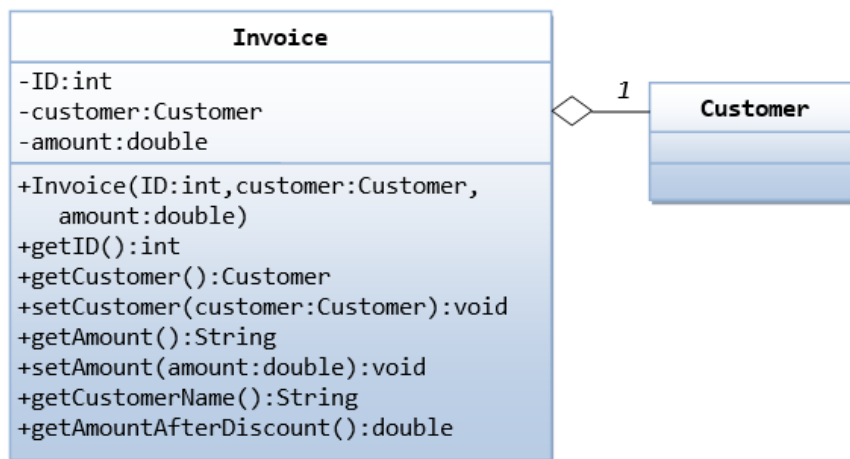
"Equilateral" or "Isosceles" or "Scalene"

## APARTADO2.EJERCICIO3:

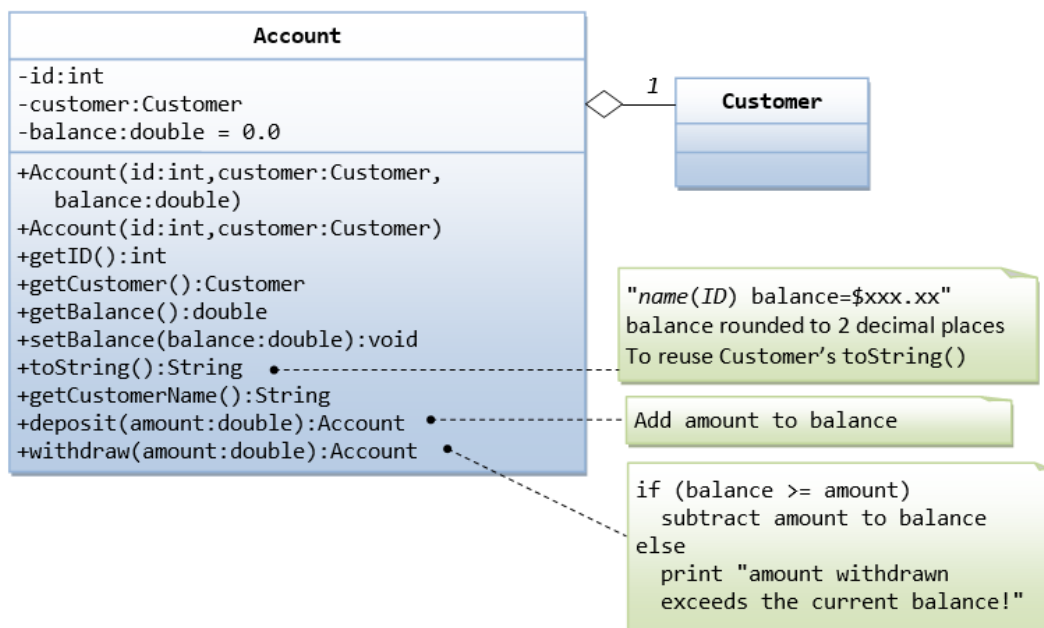
### 3.1. Clase Customer.



### 3.2. Clase Invoice



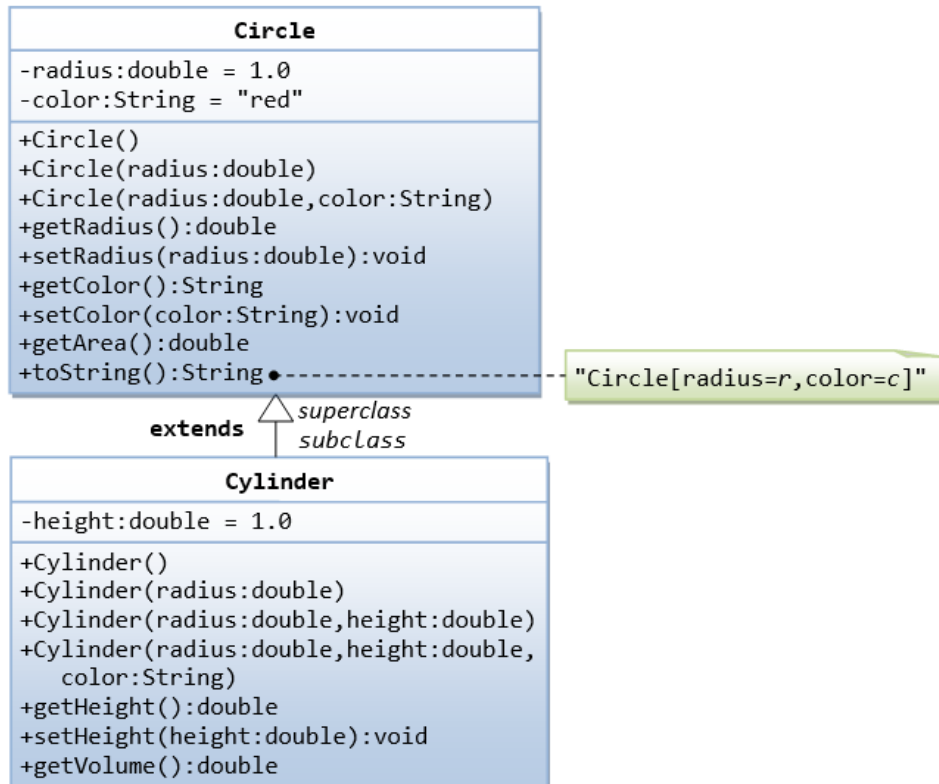
### 3.3. Clase Account



### APARTADO3. EJERCICIO DE HERENCIA.

#### APARTADO3.EJERCICIO1. Clase Circle & Cylinder.

Crear una clase Circle que actúe como superclase y una clase Cylinder que extienda de Circle introduciendo una especialización como es el atributo "height".



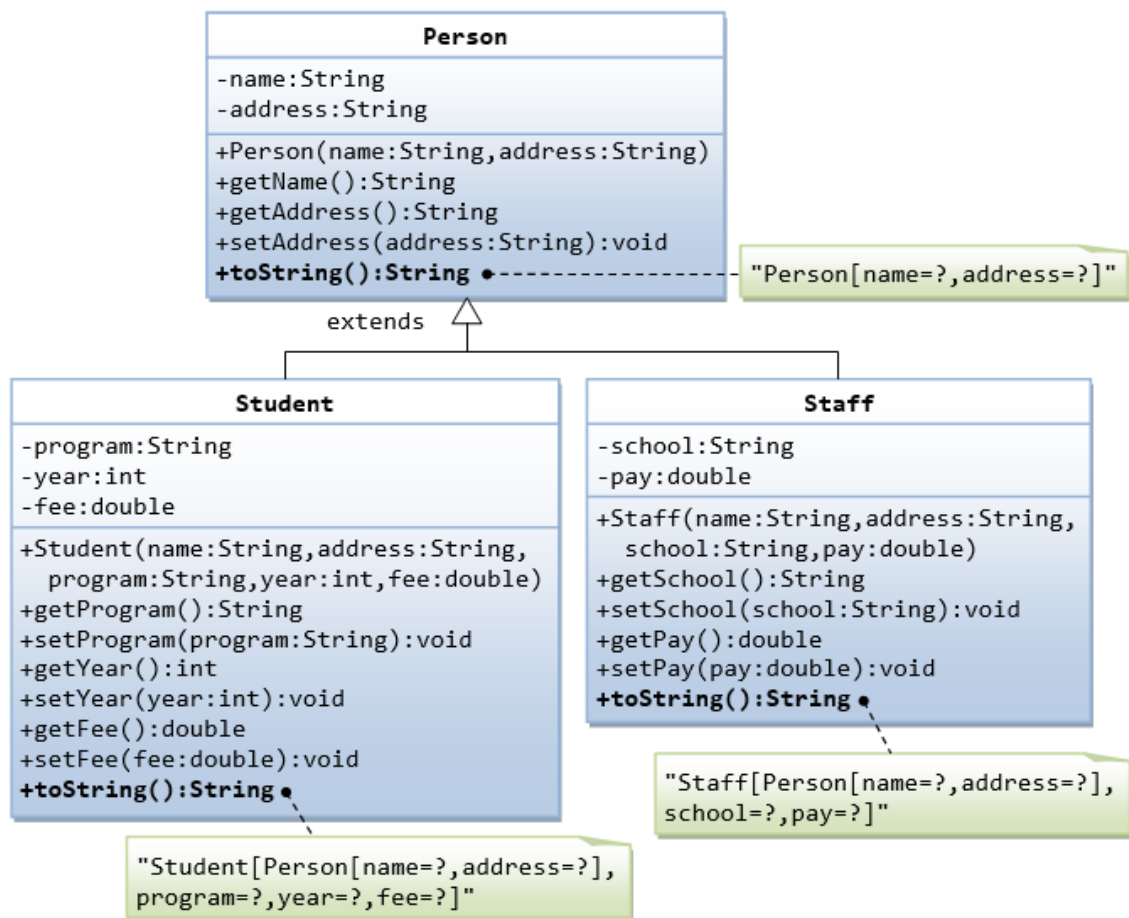
```
public class Cylinder extends Circle { // Save as "Cylinder.java"
    private double height; // private variable

    // Constructor with default color, radius and height
    public Cylinder() {
        super(); // call superclass no-arg constructor Circle()
        height = 1.0;
    }
    // Constructor with default radius, color but given height
    public Cylinder(double height) {
        super(); // call superclass no-arg constructor Circle()
        this.height = height;
    }
    // Constructor with default color, but given radius, height
    public Cylinder(double radius, double height) {
        super(radius); // call superclass constructor Circle(r)
        this.height = height;
    }

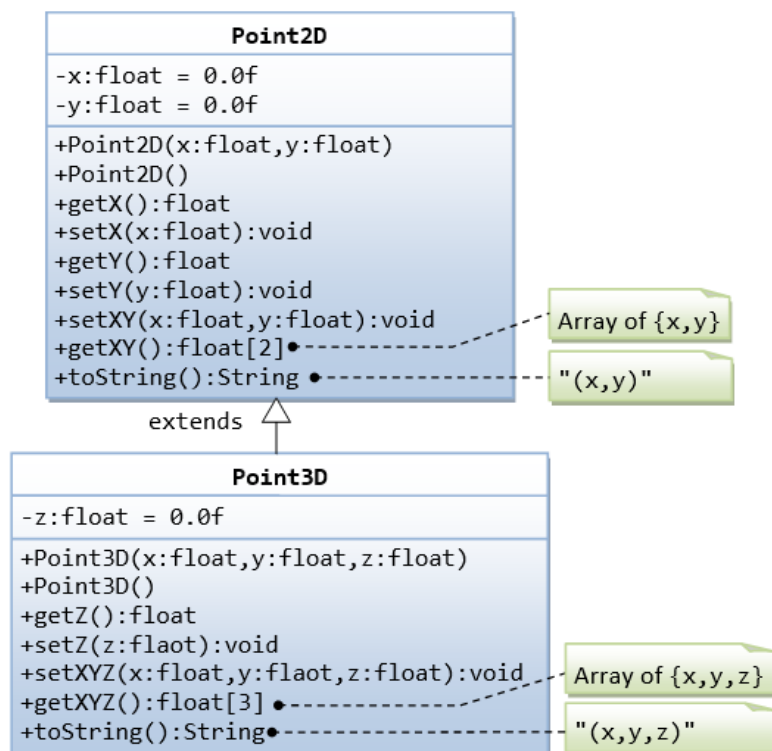
    // A public method for retrieving the height
    public double getHeight() {
        return height;
    }

    // A public method for computing the volume of cylinder
    // use superclass method getArea() to get the base area
    public double getVolume() {
        return getArea()*height;
    }
}
```

### APARTADO3.EJERCICIO2

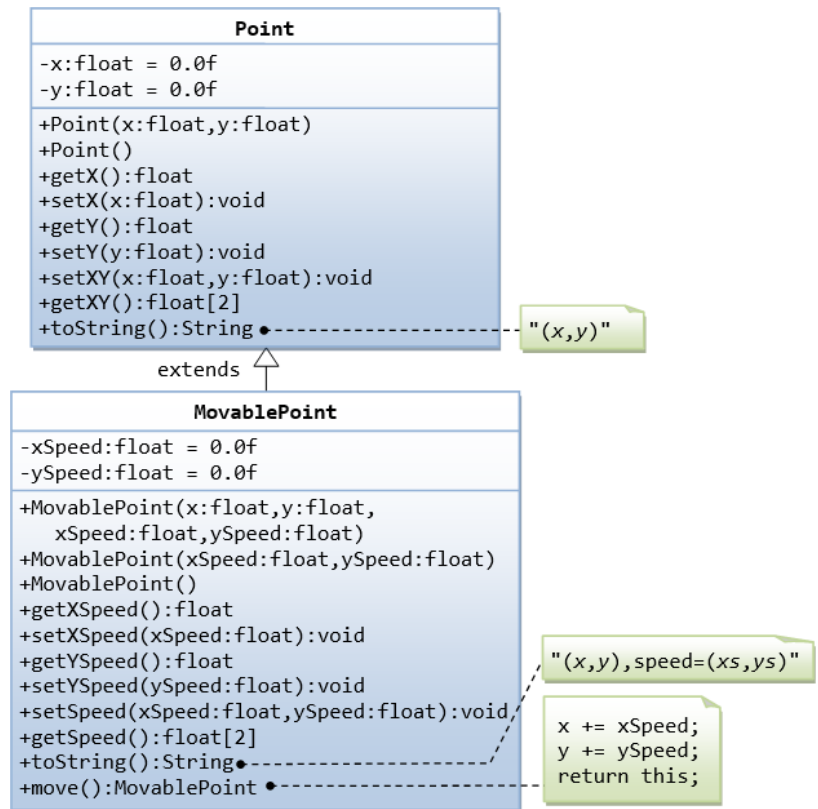


### APARTADO3.EJERCICIO3:





#### APARTADO3.EJERCICIO4:



#### APARTADO3.EJERCICIO5:

