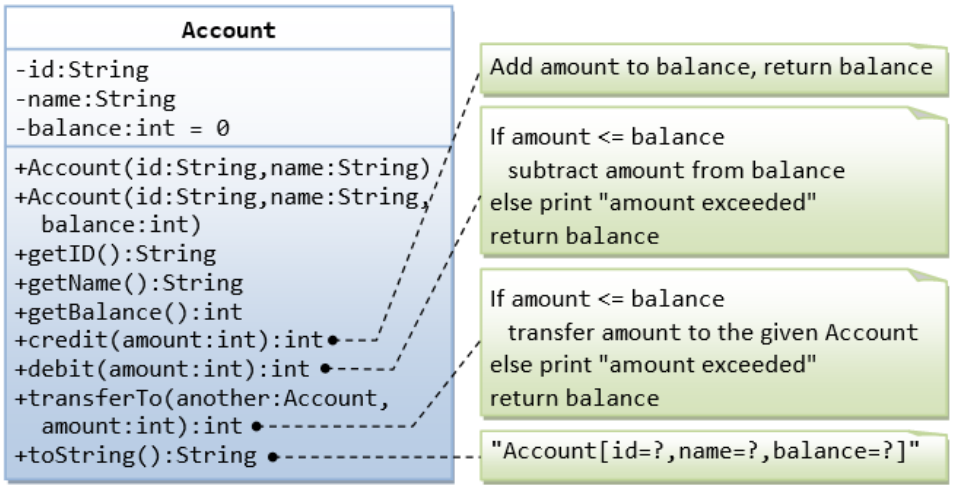
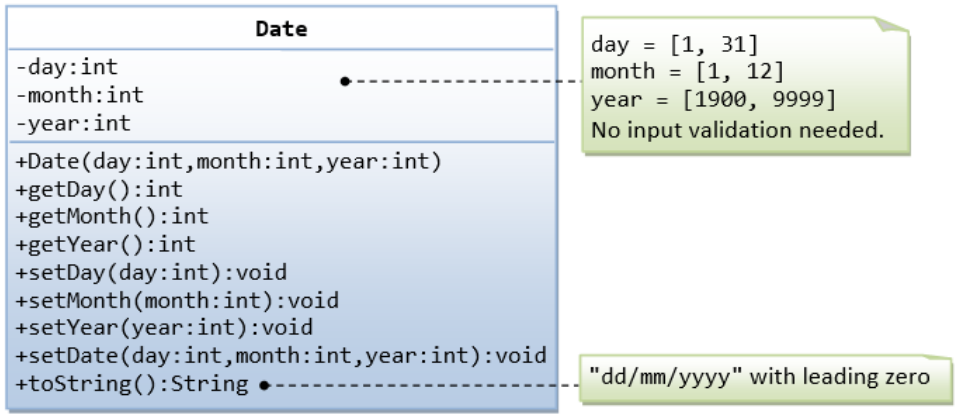
APARTADO1. CREAR CLASES

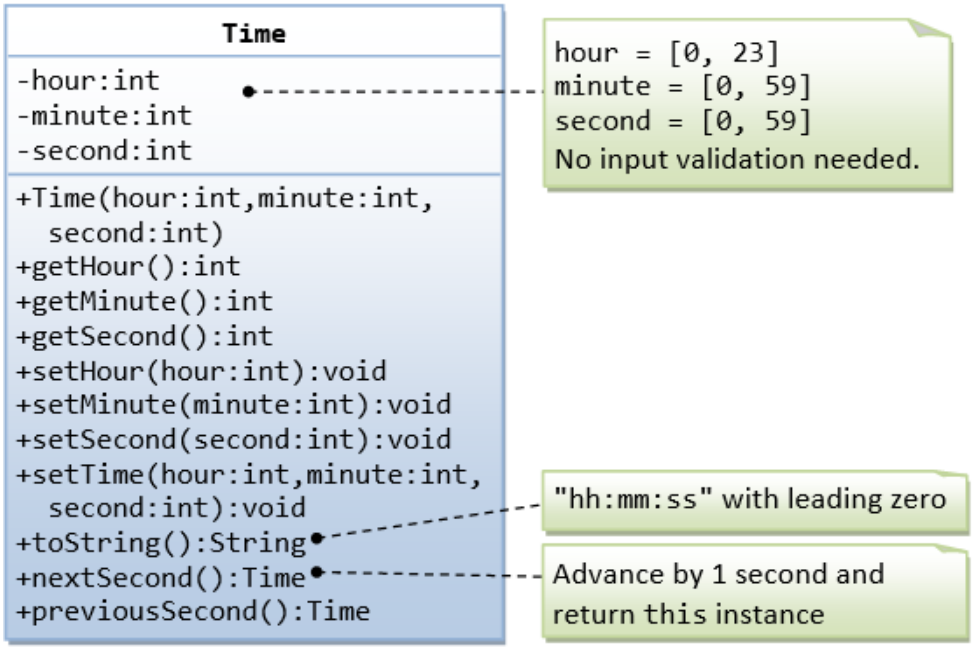
APARTADO1.EJERCICIO1:



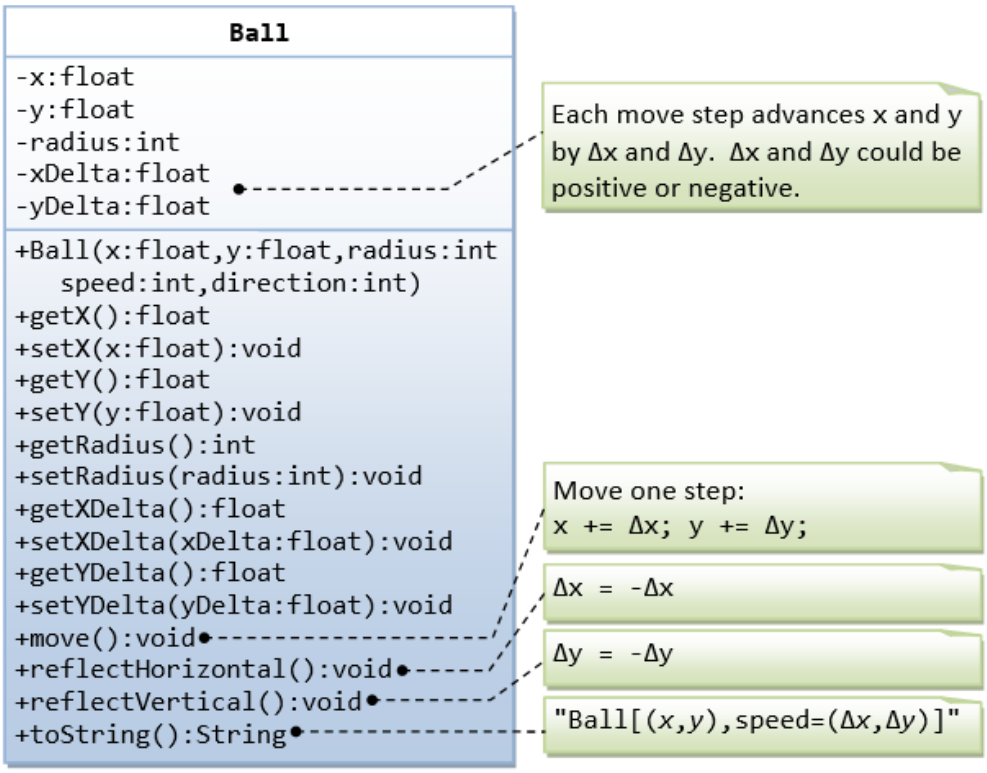
APARTADO1.EJERCICIO2:



APARTADO1.EJERCICIO3:



APARTADO1.EJERCICIO4:

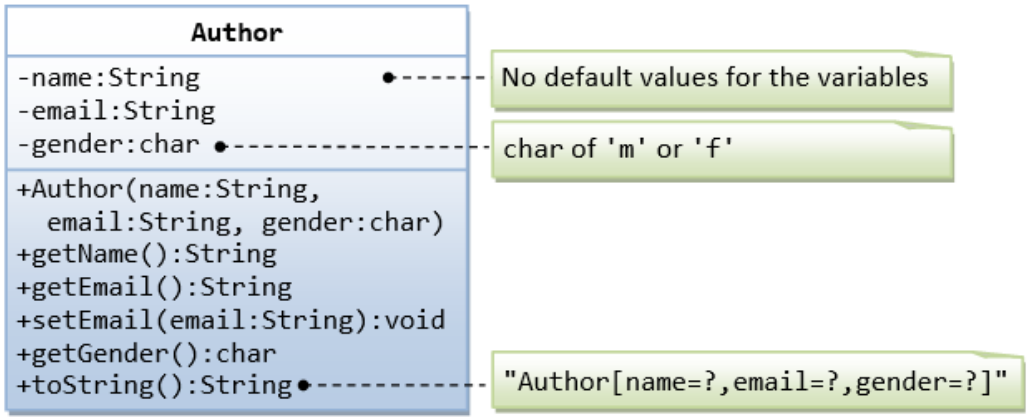


2. COMPOSICIÓN

APARTADO2.EJERCICIO1:

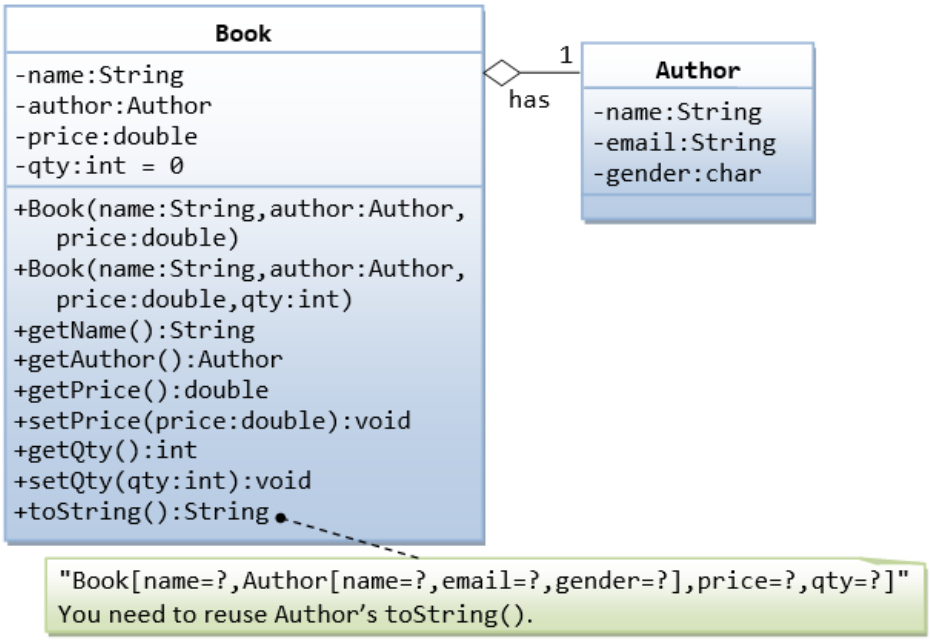
Codificación la relación entre un libro y la información de sus autores.

1.1. CLASE AUTOR



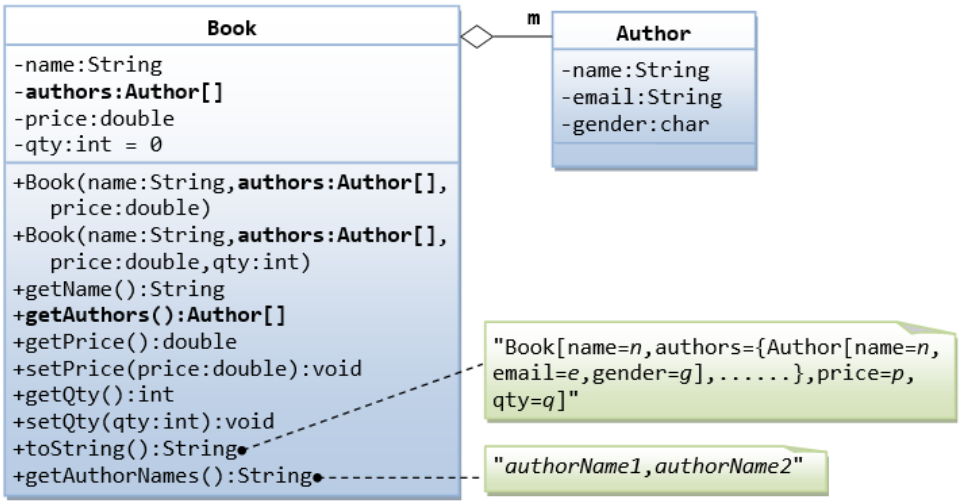

```
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm'); // Test the constructor
System.out.println(ahTeck);  // Test toString()
```

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.



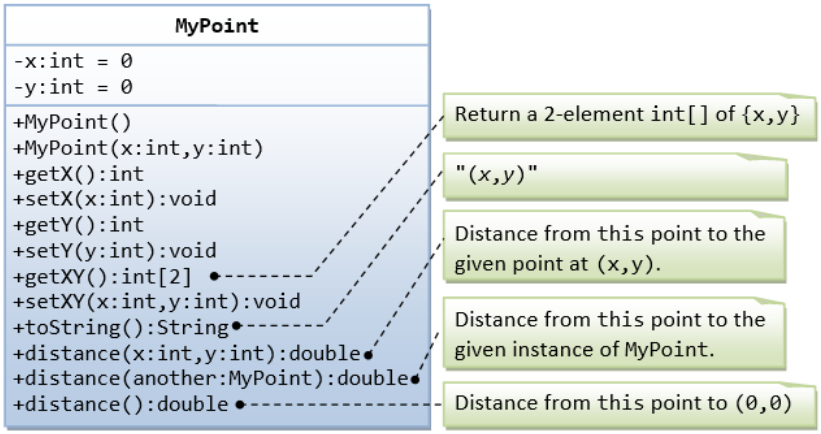

```
// Declare and allocate an array of Authors
Author[] authors = new Author[2];
authors[0] = new Author("Tan Ah Teck", "AhTeck@somewhere.com", 'm');
authors[1] = new Author("Paul Tan", "Paul@nowhere.com", 'm');

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

APARTADO2.EJERCICIO2:

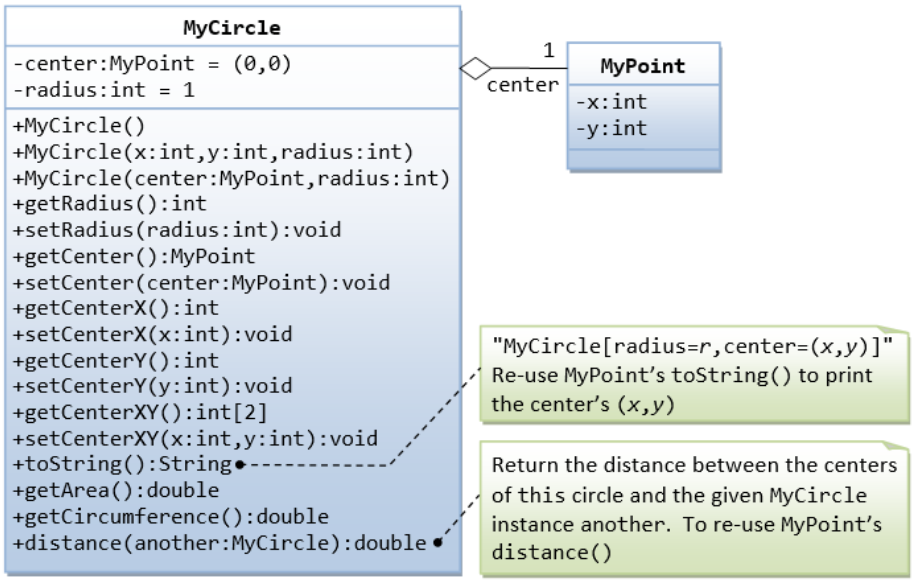
Codificación la relación entre un punto y su uso en la definición de diferentes figuras geométricas.

2.1. Clase MyPoint

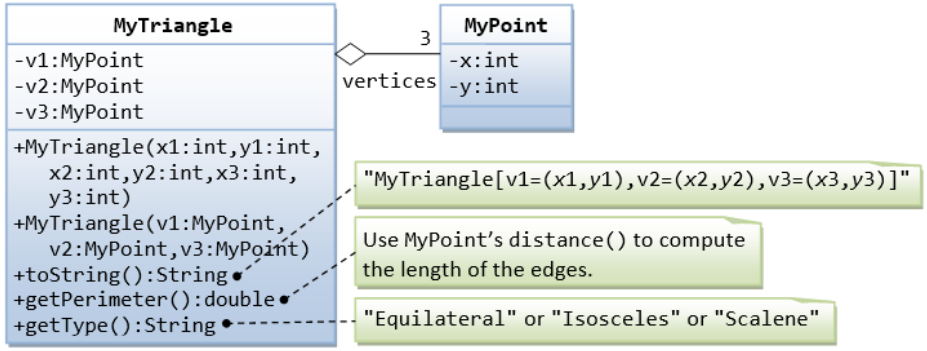
La clase MyPoint que modela un punto 2D definido por las coordenadas X e Y.



2.1. Clase MyCircle. Modela un círculo indicando el punto central y un radio.

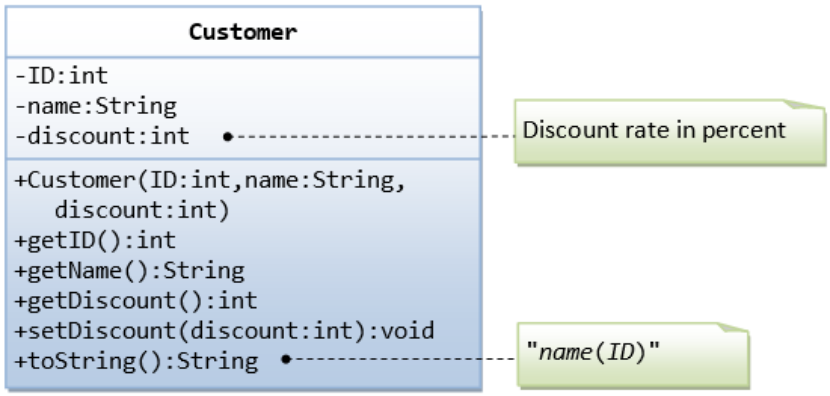


2.2. Clase MyTriangle. Se define mediante sus tres vertices.

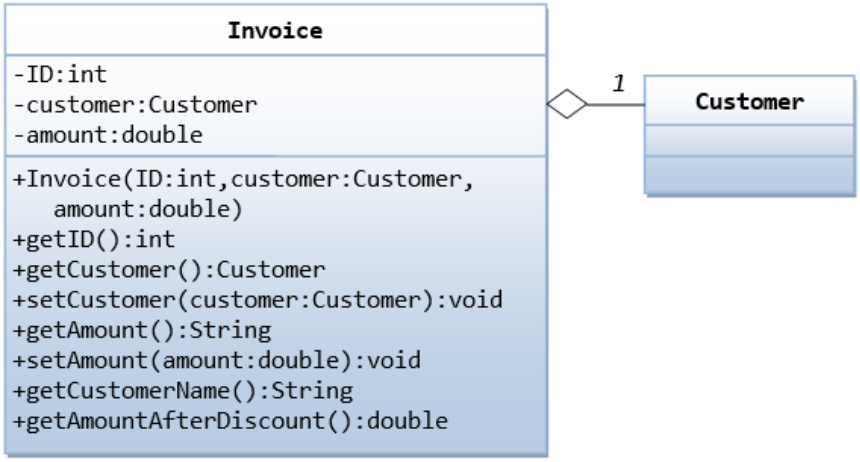


APARTADO2.EJERCICIO3:

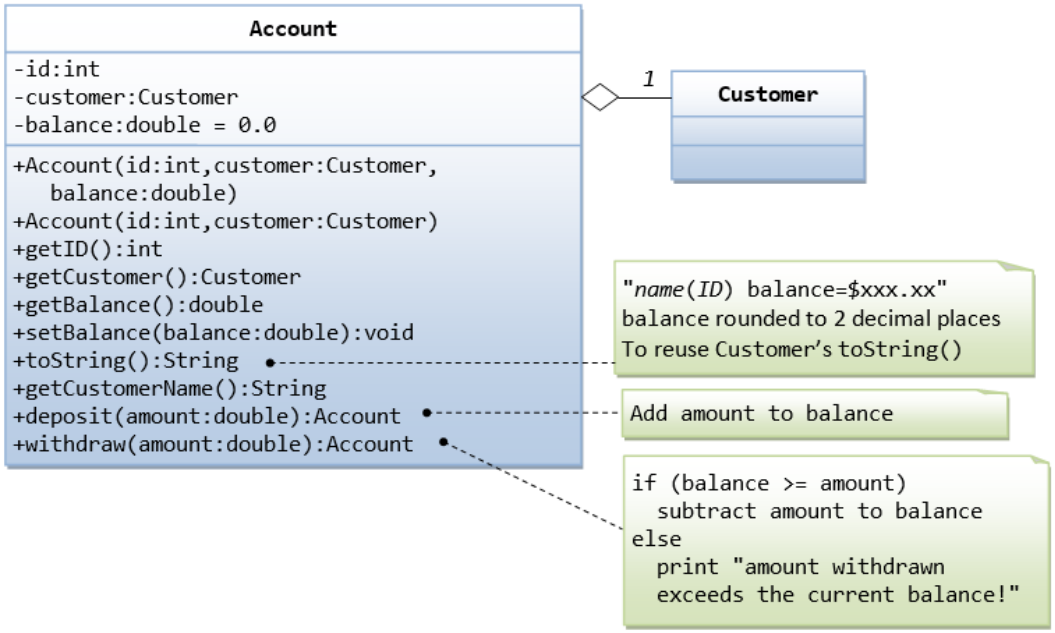
3.1. Clase Customer.



3.2. Clase Invoice



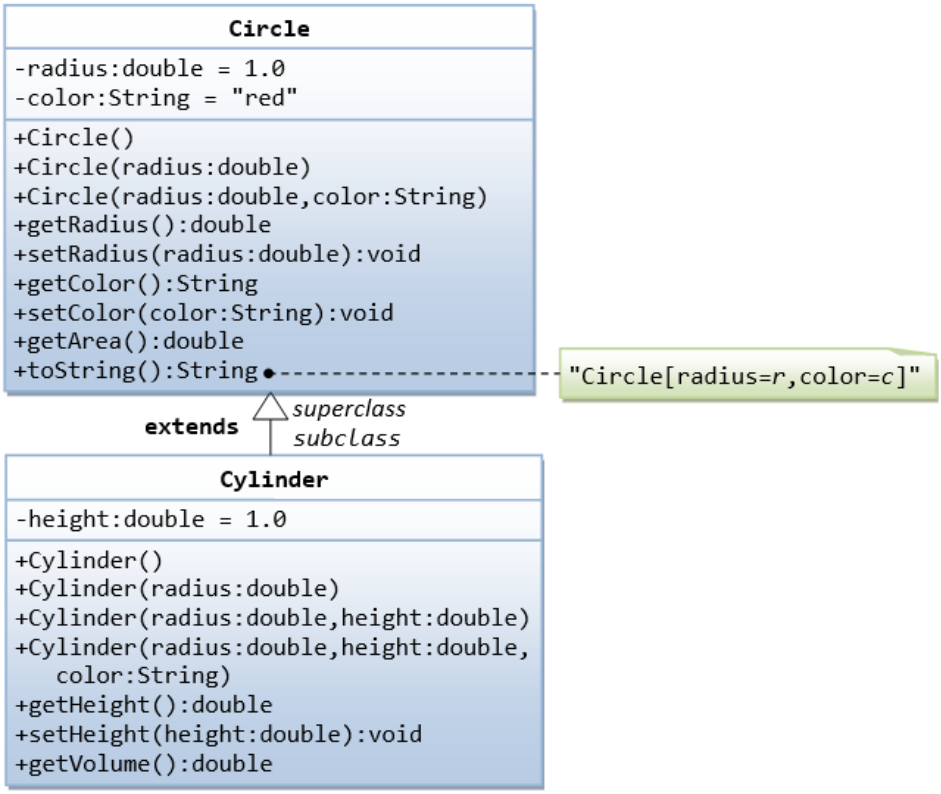
3.3. Clase Account



APARTADO3. EJERCICIO DE HERENCIA.

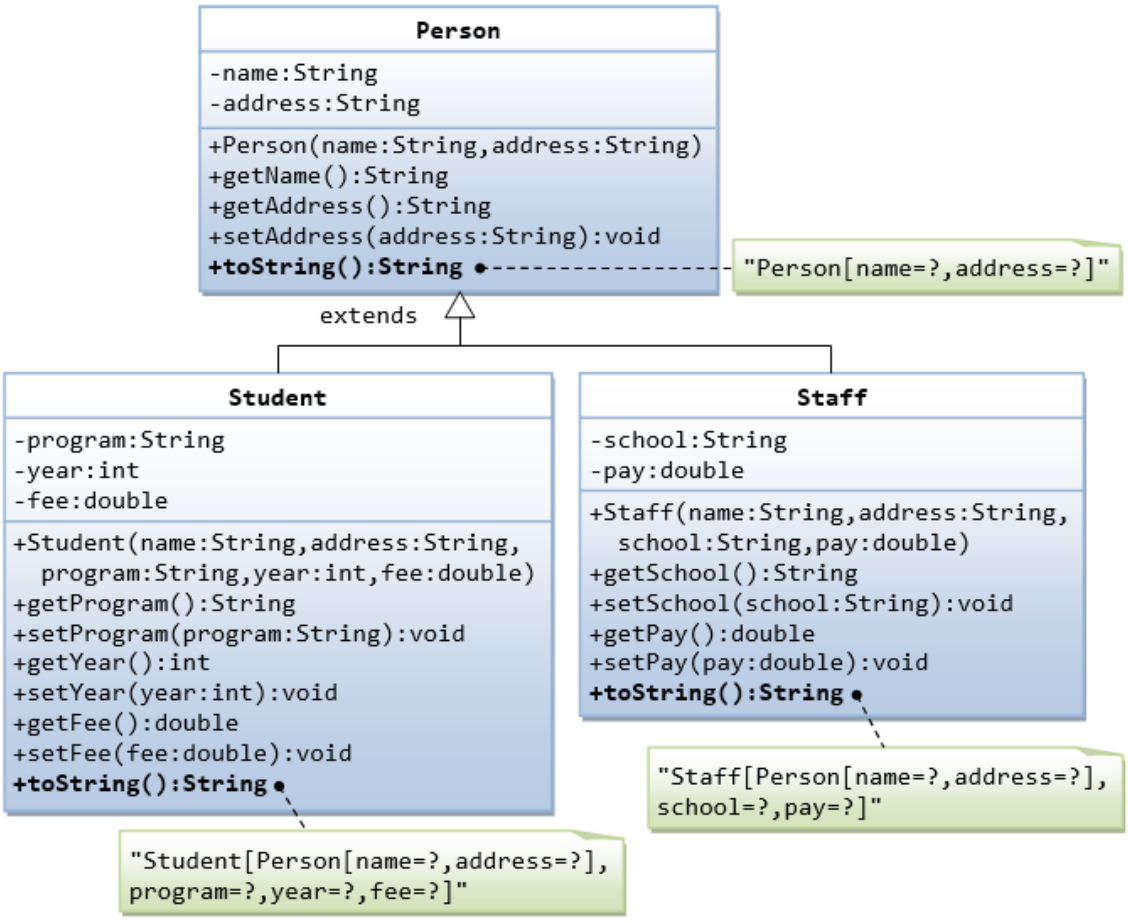
APARTADO3.EJERCICIO1. Clase Circle & Cylinder.

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

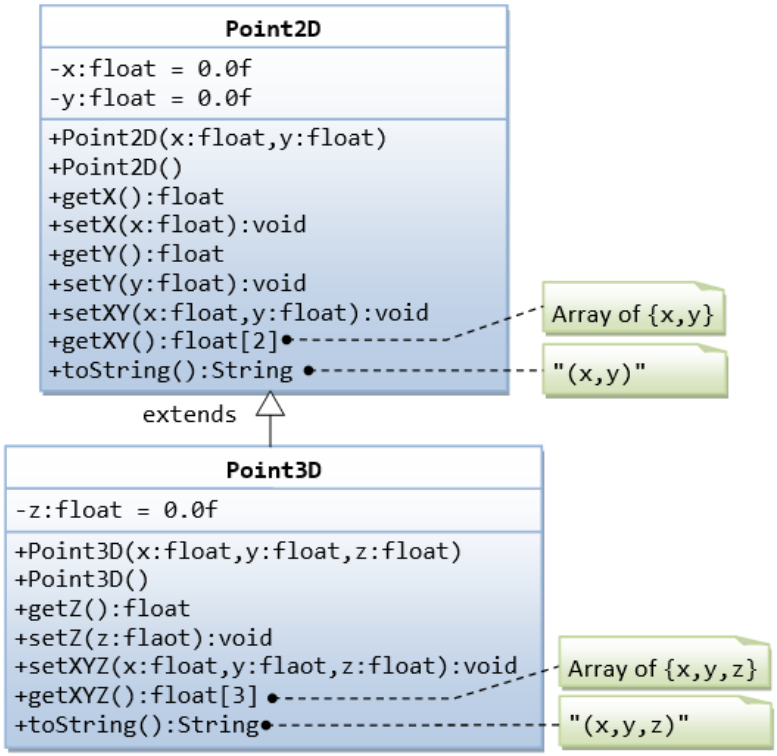



```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // 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;
   }
}
```

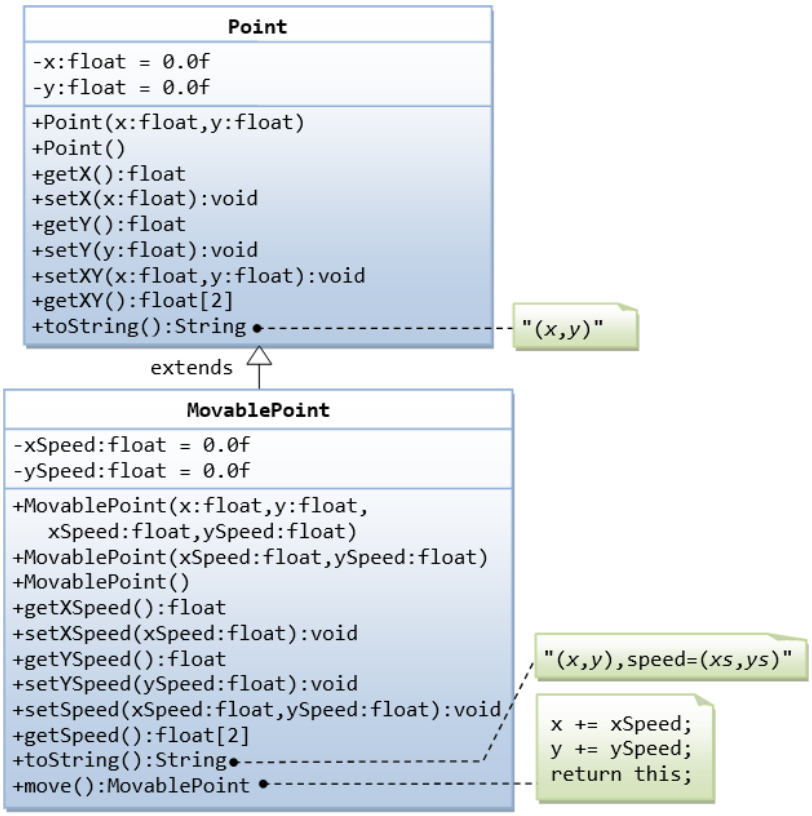
APARTADO3.EJERCICIO2



APARTADO3.EJERCICIO3:



APARTADO3.EJERCICIO4:



APARTADO3.EJERCICIO5:

APARTADO3.EJERCICIO6:

Programa el siguiente diagrama UML conformado por 3 clases y que hacen uso de la herencia.

Por una parte:

- Person: Gestiona los atributos “name” y “address” común tanto para los estudiantes como los profesores.
- Student:

  Atributos:

  - numCourses: int. Nº de cursos en los que está matriculado un alumno.
  - courses: String[]. Vector con los nombres de los cursos en los que está matriculado. Máximo posiciones 20.
  - grades: int[]. Notas obtenidas en cada uno de los cursos matriculado. Máximo posiciones 20.

  Métodos:

  - addCourseGrade(course: String, grade: int): void. Añadir un curso y nota.
  - printGrades(): void. Imprimir todas las notas obtenidas.
  - getAverageGrade():double. Calcular todas las medias de todas las notas devolviendo ese valor.
- Teacher:

  Atributos:

  - numCourses: int. Nº de cursos en los que está impartiendo docencia.
  - courses: String[]. Vector con los nombres de los cursos en los que da clases. Máximo posiciones 20.

  Métodos:

  - addCourse(course: String): boolean. Añadir asignándole un curso una única vez. Si ya está en el curso devuelve false y no añade un nuevo curso.
  - removeCourse(course: String): boolean. Elimina un curso al que previamente estaba registrado/asignado. Sino no estuviese en dicho curso devuelve false.