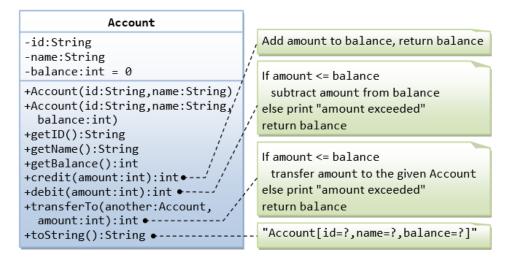
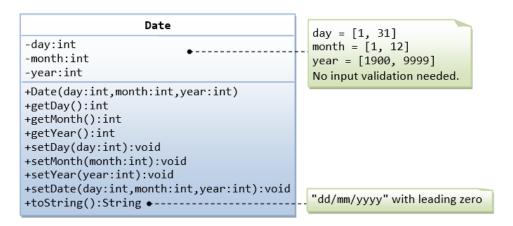
APARTADO1. CREAR CLASES

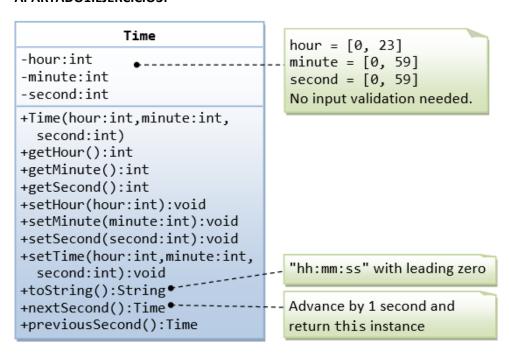
APARTADO1.EJERCICIO1:



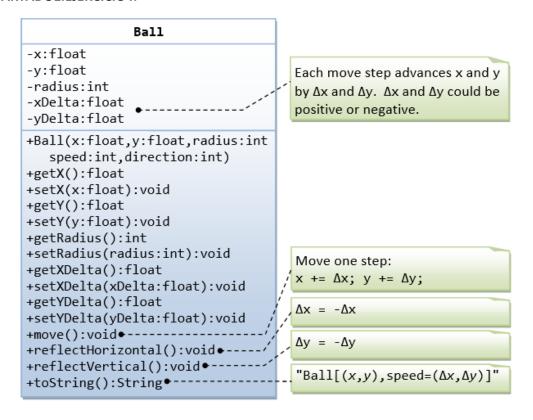
APARTADO1.EJERCICIO2:



APARTADO1.EJERCICIO3:



APARTADO1.EJERCICIO4:

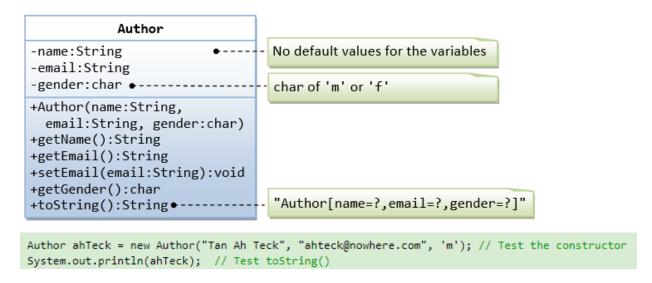


2. COMPOSICIÓN

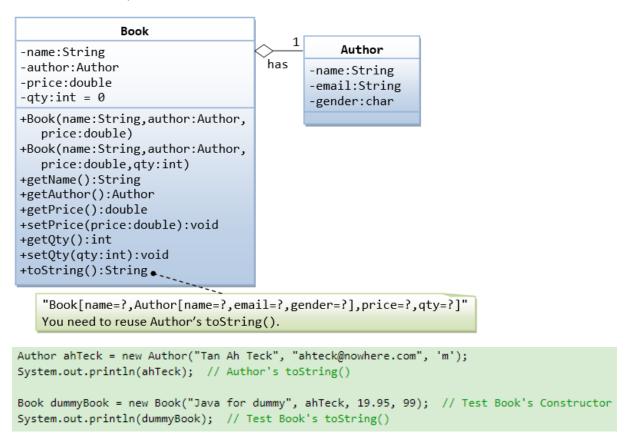
APARTADO2.EJERCICIO1:

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

1.1. CLASE AUTOR



1.2. CLASE LIBRO QUE SÓLO CONTIENE UN ÚNICO AUTOR.



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

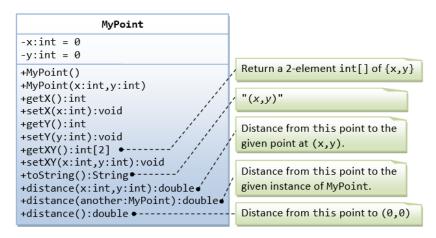
```
Author
-name:String
                                          -name:String
-authors:Author[]
                                          -email:String
-price:double
                                          -gender:char
-qty:int = 0
+Book(name:String,authors:Author[],
   price:double)
+Book(name:String,authors:Author[],
   price:double,qty:int)
+getName():String
+getAuthors():Author[]
+getPrice():double
                                     "Book[name=n,authors={Author[name=n,
+setPrice(price:double):void
                                     email=e,gender=g],.....},price=p,
+getQty():int
                                     qty=q]"
+setQty(qty:int):void
+toString():String•
                                     "authorName1,authorName2"
+getAuthorNames():String◆
// 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:

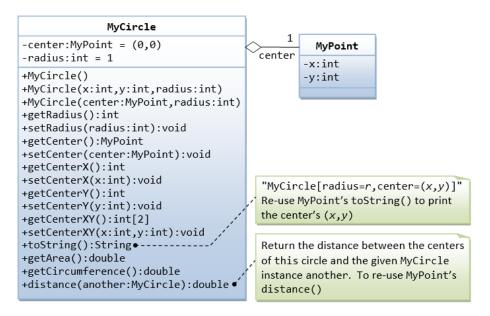
Codificación la relación entre un punto y su uso en la definición de diferentes figuras geométricas.

2.1. Clase MyPoint

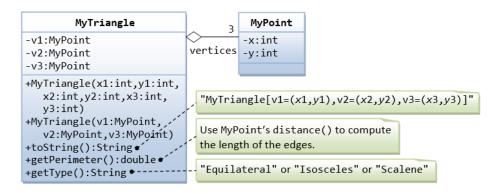
La clase MyPoint que modela un punto 2D definido por las coordenadas X e Y.



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

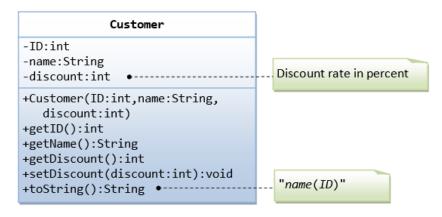


2.2. Clase MyTriangle. Se define mediante sus tres vertices.

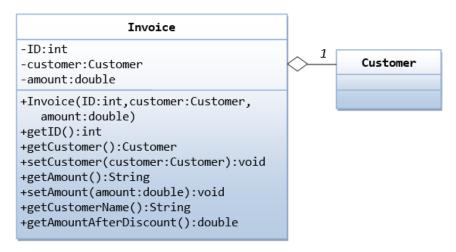


APARTADO2.EJERCICIO3:

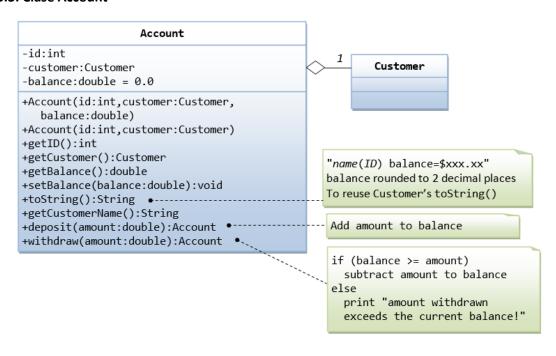
3.1. Clase Customer.



3.2. Clase Invoice



3.3. Clase Account



APARTADO3. EJERCICIO DE HERENCIA.

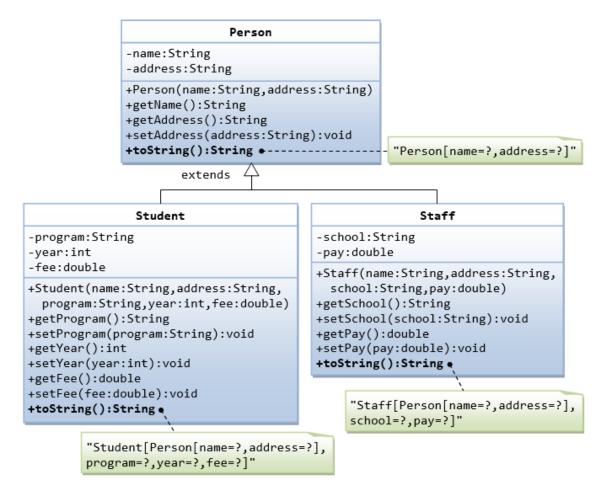
APARTADO3.EJERCICIO1. Clase Circle & Cylinder.

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

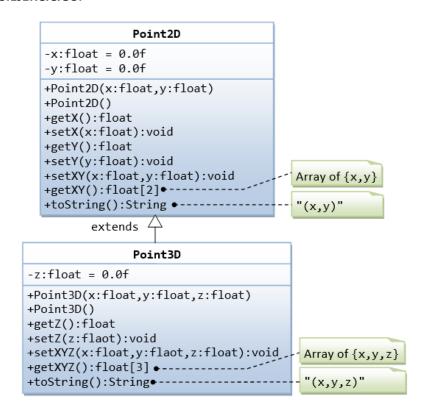
```
Circle
-radius:double = 1.0
-color:String = "red"
+Circle()
+Circle(radius:double)
+Circle(radius:double,color:String)
+getRadius():double
+setRadius(radius:double):void
+getColor():String
+setColor(color:String):void
+getArea():double
+toString():String ◆-----
                                           "Circle[radius=r,color=c]"
          extends \(\begin{array}{c} & superclass \\ & & \end{array}
                     subclass
                Cylinder
-height:double = 1.0
+Cylinder()
+Cylinder(radius:double)
+Cylinder(radius:double,height:double)
+Cylinder(radius:double,height:double,
   color:String)
+getHeight():double
+setHeight(height:double):void
+getVolume():double
```

```
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:

```
Point
 -x:float = 0.0f
 -y:float = 0.0f
 +Point(x:float,y:float)
 +Point()
 +getX():float
 +setX(x:float):void
 +getY():float
 +setY(y:float):void
 +setXY(x:float,y:float):void
+getXY():float[2]
 +toString():String
                                             (x,y)
           extends
               MovablePoint
-xSpeed:float = 0.0f
-ySpeed:float = 0.0f
+MovablePoint(x:float,y:float,
  xSpeed:float,ySpeed:float)
+MovablePoint(xSpeed:float,ySpeed:float)
+MovablePoint()
+getXSpeed():float
+setXSpeed(xSpeed:float):void
+getYSpeed():float
                                               "(x,y),speed=(xs,ys)"
+setYSpeed(ySpeed:float):void
+setSpeed(xSpeed:float,ySpeed:float):void,
                                               x += xSpeed;
+getSpeed():float[2]
                                               y += ySpeed;
+toString():String∙
                                               return this;
+move():MovablePoint
```

APARTADO3.EJERCICIO5:

```
Shape
     -color:String = "red"
     -filled:boolean = true
     +Shape()
     +Shape(color:String,filled:boolean)
     +getColor():String
     +setColor(color:String):void
     +isFilled():boolean
     +setFilled(filled:boolean):void
     +toString():String .
                                            "Shape[color=?,filled=?]"
                   extends
             Circle
                                                  Rectangle
-radius:double = 1.0
                                       -width:double = 1.0
                                       -length:double = 1.0
+Circle()
+Circle(radius:double)
                                       +Rectangle()
                                       +Rectangle(width:double,
+Circle(radius:double,
  color:String,filled:boolean)
                                          length:double)
                                       +Rectangle(width:double,
+getRadius():double
                                          length:double, color:String,
+setRadius(radius:double):void
+getArea():double
                                          filled:boolean)
+getPerimeter():double
                                       +getWidth():double
+toString():String.
                                       +setWidth(width:double):void
                                       +getLength():double
                                       +setLength(legnth:double):void
           "Circle[Shape[color=?,
                                       +getArea():double
           filled=?],radius=?]"
                                       +getPerimeter():double
                                      +toString():String
     "Rectangle[Shape[color=?,
                                                    Square
     filled=?],width=?,length=?]"
                                       +Square()
                                       +Square(side:double)
                                       +Square(side:double,
     The length and width shall be
                                          color:String,filled:boolean)
     set to the same value.
                                       +getSide():double
                                       +setSide(side:double):void
                                       +setWidth(side:double):void
 "Square[Rectangle[Shape[color=?,
                                       +setLength(side:double):void
 filled=?],width=?,length=?]]"
                                       +toString():String
```

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:

- o numCourses: int. № 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:

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

Atributos:

- o 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:

- o 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.
- o removeCourse(course: String): boolean. Elimina un curso al que previamente estaba registrado/asignado. Sino no estuviese en dicho curso devuelve false.

