Metodología de la programación

Ejercicios de Programación Orientada a Objetos

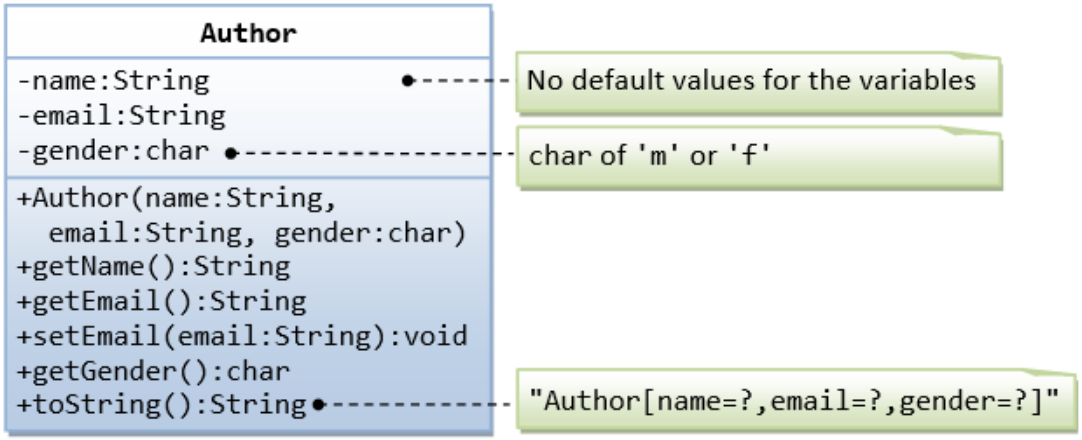
Estos ejercicios están basados en los ejercicios propuestos en la Universidad Tecnológica de Nanyang en Singapur.

2.Ejercicios de composición/agregación

2.1 Introducción a la POO por composición/agregación

Si bien la composición y la agregación son similares a nivel de código, son distintas a nivel semántico: La composición indica que un objeto está compuesto de otros, y sin esos otros este primer objeto no tiene sentido.

La agregación, por otro lado, indica que los objetos sí pueden tener sentido por sí mismos de manera independiente.



Una clase `Author` (como la del diagrama) se diseña para modelar el autor de un libro, conteniendo los atributos privados, constructores y otros métodos indicados.

- La clase dispone de un constructor para inicializar los atributos de la clase `name`, `email` y `gender` :

```
public Author (String name, String email, char gender) {......}
```

(No existe un constructor por defecto!!)

•Además existen los getters y setters correspondientes a los atributos según su utilización.

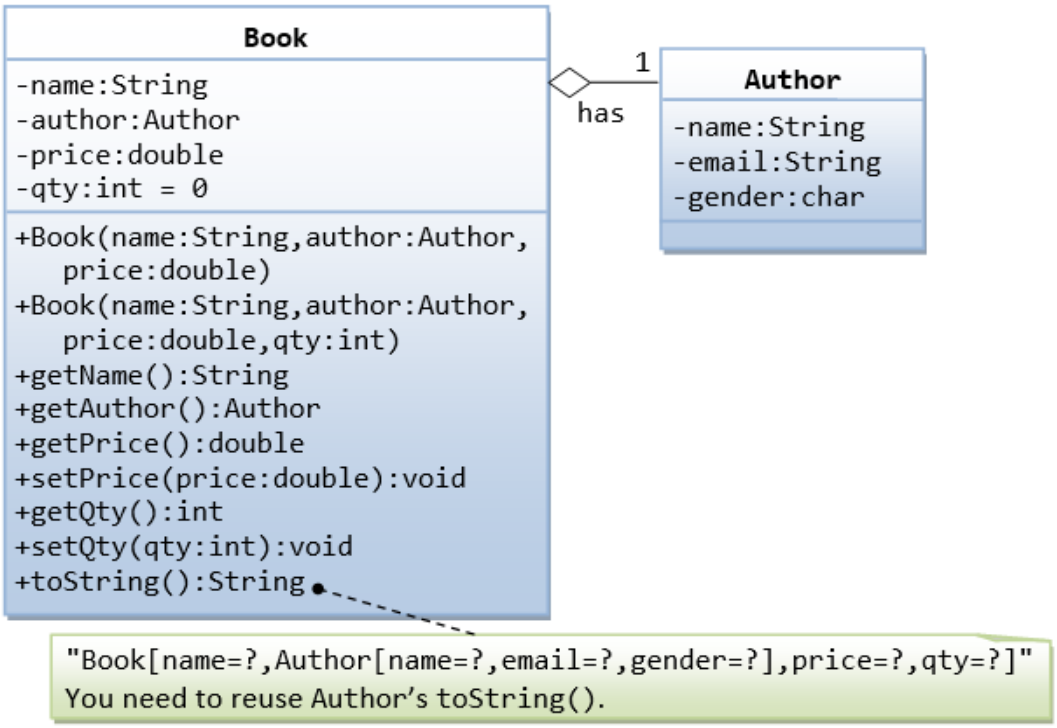
•`public` getters/setters: `getName()`, `getEmail()`, `setEmail()`,y `getGender()`;

(Recuerda: según lo que se permita hacer, ciertos atributos podrían no tener setters, o incluso getters.)

•Existe un método `toString()` que devolverá `"Author[name=?,email=?,gender=?]"`, ejemplo: `"Author[name=Tan Ah Teck,email=ahTeck@somewhere.com,gender=m]"`.

Construye un método main para probar esta clase que contenga el siguiente código:

```
Author ahTeck = new Author("Tan Ah Teck", "ahteck@nowhere.com", 'm'); // Test the constructor
System.out.println(ahTeck);  // Test toString()
ahTeck.setEmail("paulTan@nowhere.com");  // Test setter
System.out.println("name is: " + ahTeck.getName());     // Test getter
System.out.println("eamil is: " + ahTeck.getEmail());   // Test getter
System.out.println("gender is: " + ahTeck.getGender()); // Test
```



Una vez disponemos de la clase Author, es el momento de construir la clase `Book` para modelar un libro escrito por el autor, que contendrá:

- Cuatro atributos privados : `name` (`String`), `author` (de la clase `Author` anterior), `price` (`double`), y `qty` (`int`);
- Dispondrá de dos constructores:

```
public Book (String name, Author author, double price) { ..... }
public Book (String name, Author author, double price, int qty) { ..... }
```

- Dispondrá de los getters y setters correspondientes: `getName()`, `getAuthor()`, `getPrice()`, `setPrice()`, `getQty()`, `setQty()`.
- Tendrá un método `toString()` que devolverá el texto `"Book[name=?,Author[name=?,email=?,gender=?],price=?,qty=?"`. Fíjate que la parte referida al `Author` es igual que el método `toString()` de esa clase. Un código de buena calidad no repite códigos escritos anteriormente...

Escribe un programa main para probar esta nueva clase, completando todos los métodos, que tenga el siguiente código:

```
// Construct an author instance
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()

// Test Getters and Setters
dummyBook.setPrice(29.95);
dummyBook.setQty(28);
System.out.println("name is: " + dummyBook.getName());
System.out.println("price is: " + dummyBook.getPrice());
System.out.println("qty is: " + dummyBook.getQty());
System.out.println("Author is: " + dummyBook.getAuthor());  // Author's toString()
```

```
System.out.println("Author's name is: " + dummyBook.getAuthor().getName());
System.out.println("Author's email is: " + dummyBook.getAuthor().getEmail());

// Use an anonymous instance of Author to construct a Book instance
Book anotherBook = new Book("more Java",
       new Author("Paul Tan", "paul@somewhere.com", 'm'), 29.95);
System.out.println(anotherBook);  // toString()
```

Fíjate que aunque ambas clases tienen un atributo denominado "name", no se comparte entre ellas, pues se refiere a entidades distintas. Para referirnos al atributo, debemos usar la instancia de clase que está asociada al que queremos usar. Es importante que los atributos se nombren adecuadamente teniendo en cuenta la propia clase en la que se definen: haber creado un atributo nameAuthor y otro atributo nameBook habría sido una mala práctica, ya que los atributos deben ser nombrados según el contexto de la clase donde se definen. En un Author sólo hay un name, por lo que el nombre del atributo debe ser ese. Lo mismo ocurre en Book.

Intenta hacer estas modificaciones:

(1) Imprime el `name` y `email` del autor de la instancia de un libro. ¿Cómo accederías a la información? (Pista: `aBook.getAuthor().getName()`, `aBook.getAuthor().getEmail()`).

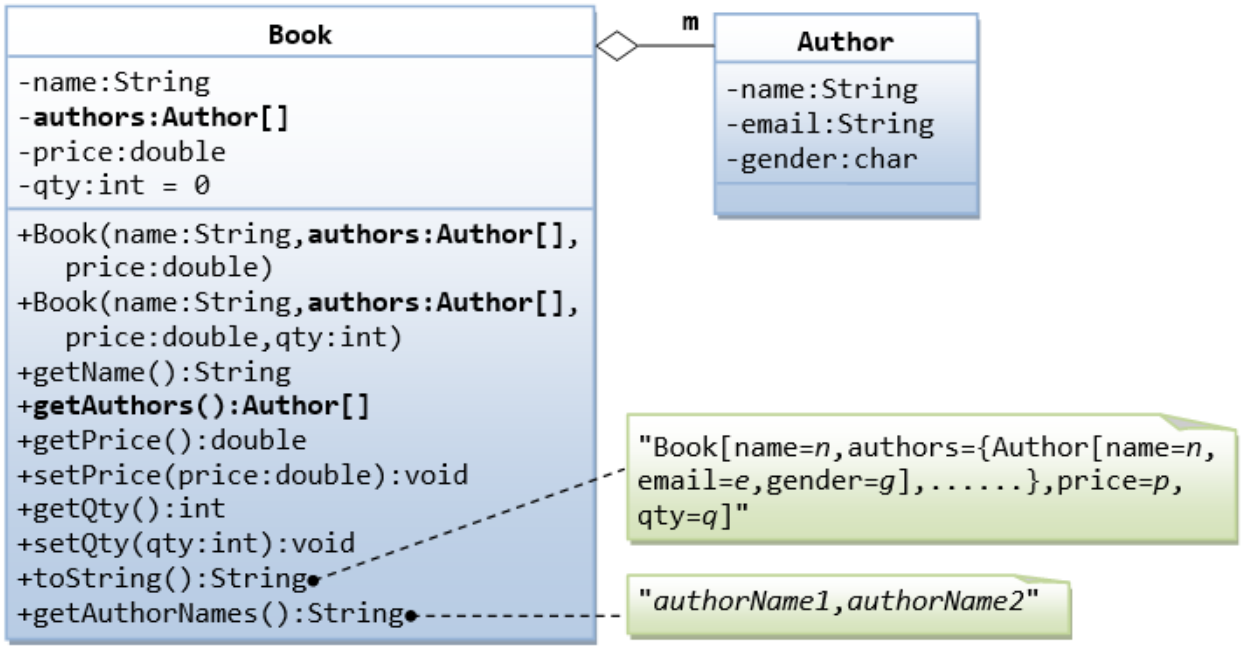
(2) Añade métodos nuevos: `getAuthorName()`, `getAuthorEmail()`, `getAuthorGender()` en la clase `Book` para poder devolver el `name`, `email` y `gender` del autor de un libro.

Como ayuda:

```
public String getAuthorName() {
    return author.getName();  // cannot use author.name as name is private in Author class
}
```

Recuerda añadir los tests para los métodos de las dos clases!!!!!!!

2.2 (Mejoramos) Un libro podría tener múltiples autores...



En las clases anteriores, el diseño estaba limitado a que un libro tuviese un autor... pero en la realidad sucede que muchos libros tienen múltiples autores. Modifiquemos la clase `Book` para permitir que uno o más autores escriban un libro. Esto lo haremos modificando el atributo author, haciendo que su nombre esté en plural, y que su tipo sea un array de Author.

Pregunta: ***¿Qué sucede al hacer estos cambios con los tests que habíamos escrito?***

Para poder añadir los autores, modificaremos el constructor, de manera que en vez de recibir un Author, recibirá un array de Authors. En este diseño, una vez que la instancia de un Book se crea, ya no se pueden cambiar los autores.

También añadiremos las modificaciones necesarias para que el método toString muestre adecuadamente los autores: `"Book[name=?,authors={Author[name=?,email=?,gender=?],.....},price=?,qty=?]"`.
También ajustaremos el método getAuthorNames.

Una vez hallas hecho las modificaciones, crea un programa de prueba main para verificar que todo funciona y que contenga, al menos, este código:

```
// 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()
```

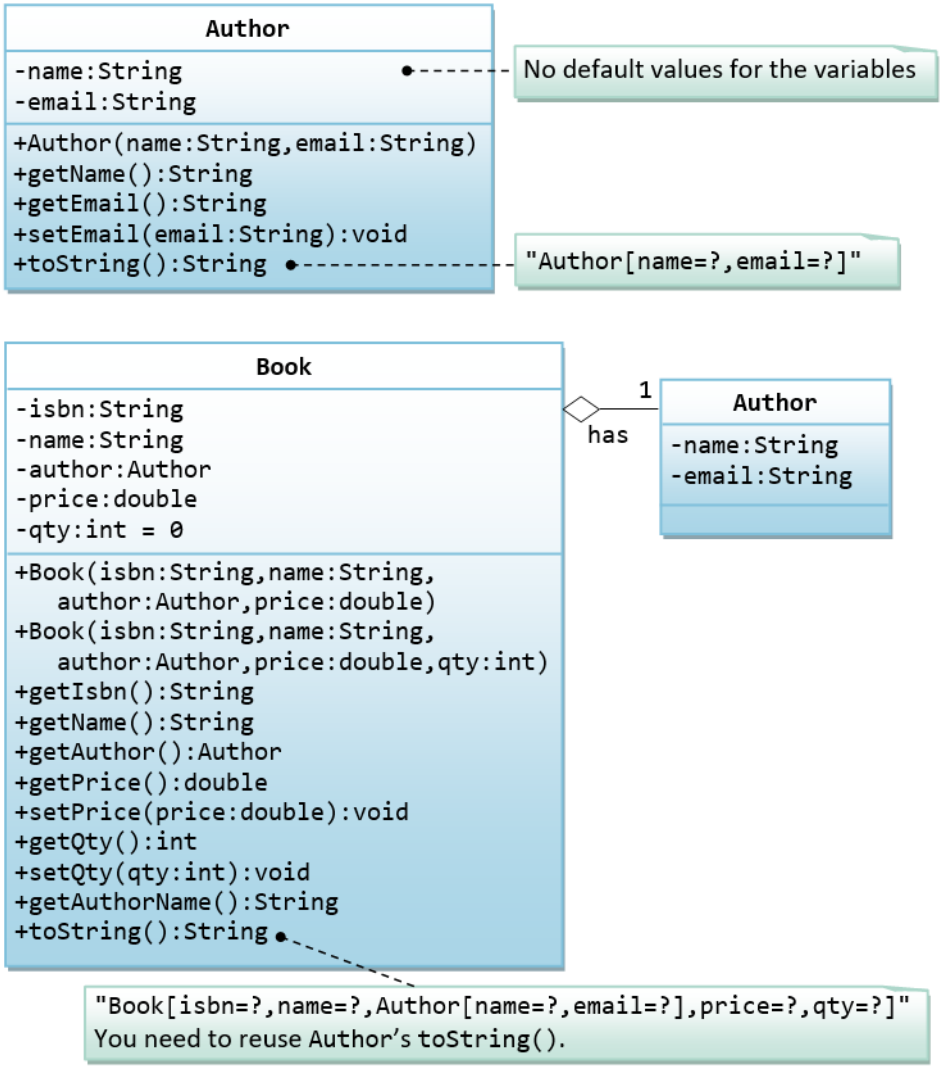
Recuerda cubrir tu código con tests!!!!

Ahora en trabajo propio:

Todas las nuevas clases debes crearlas en el paquete "**es.uah.matcomp.mp.e1.ejerciciosclases**".

2.3 Nuevas modificaciones a los Author y Book.

Dado el siguiente diagrama, programe la clase correspondiente en el paquete indicado para estos ejercicios.



Este es el código de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test Author class
        Author a1 = new Author("Tan Ah Teck", "ahteck@nowhere.com");

        System.out.println(a1);


        a1.setEmail("ahteck@somewhere.com");

        System.out.println(a1);

        System.out.println("name is: " + a1.getName());

        System.out.println("email is: " + a1.getEmail());
```

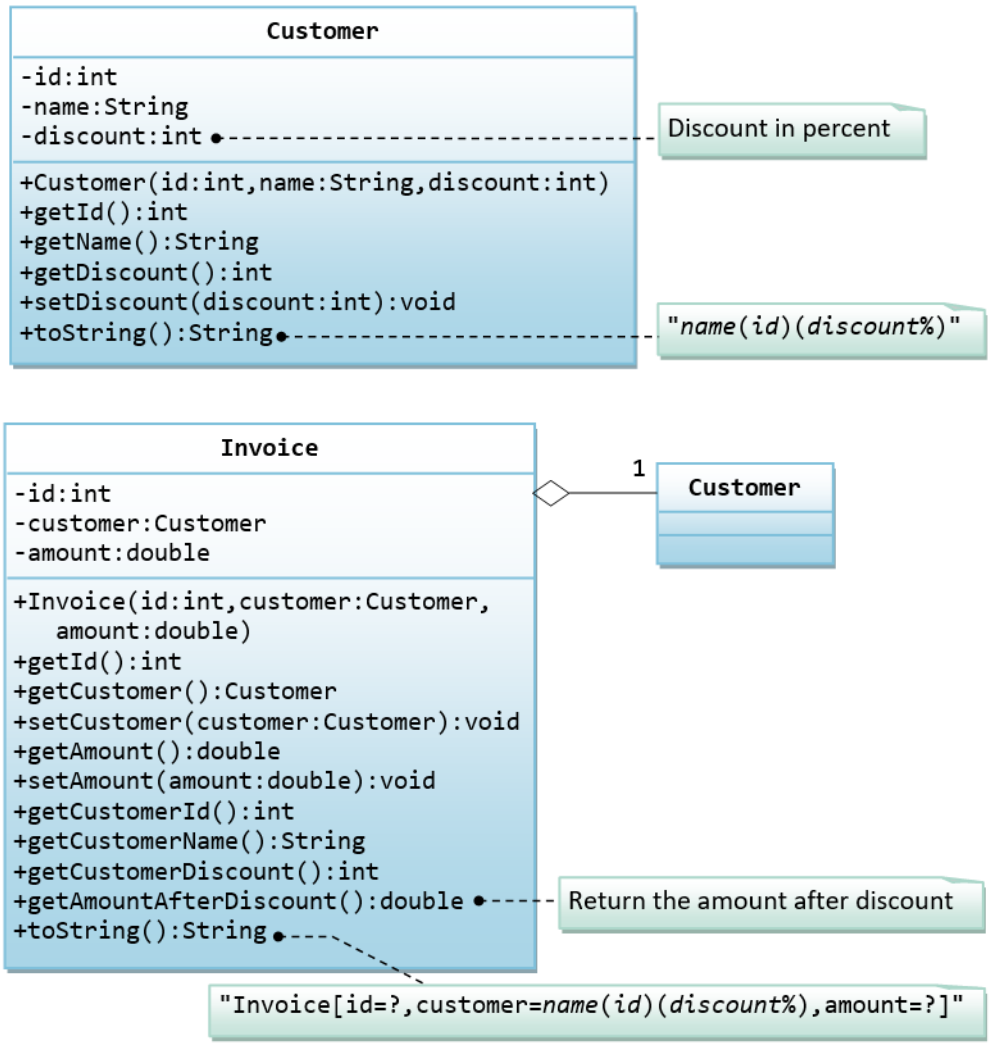
```
        // Test Book class
        Book b1 = new Book("12345", "Java for dummies", a1, 8.8, 88);
        System.out.println(b1);

        b1.setPrice(9.9);
        b1.setQty(99);
        System.out.println(b1);
        System.out.println("isbn is: " + b1.getIsbn());
        System.out.println("name is: " + b1.getName());
        System.out.println("price is: " + b1.getPrice());
        System.out.println("qty is: " + b1.getQty());
        System.out.println("author is: " + b1.getAuthor());  // Author's toString()
        System.out.println("author's name: " + b1.getAuthorName());
        System.out.println("author's name: " + b1.getAuthor().getName());
        System.out.println("author's email: " + b1.getAuthor().getEmail());
    }
}
```

La salida esperada es:

```
Author[name=Tan Ah Teck,email=ahteck@nowhere.com]
Author[name=Tan Ah Teck,email=ahteck@somewhere.com]
name is: Tan Ah Teck
email is: ahteck@somewhere.com
Book[isbn=12345,name=Java for dummies,Author[name=Tan Ah
Teck,email=ahteck@somewhere.com],price=8.8,qty=88]
Book[isbn=12345,name=Java for dummies,Author[name=Tan Ah
Teck,email=ahteck@somewhere.com],price=9.9,qty=99]
isbn is: Java for dummies
name is: Java for dummies
price is: 9.9
qty is: 99
author is: Author[name=Tan Ah Teck,email=ahteck@somewhere.com]
author's name: Tan Ah Teck
author's name: Tan Ah Teck
author's email: ahteck@somewhere.com
```

2.4 Clases de cliente y factura

Dado el siguiente diagrama, programe las clases correspondientes en el paquete indicado para estos ejercicios.



Este es el código de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test Customer class
        Customer c1 = new Customer(88, "Tan Ah Teck", 10);
        System.out.println(c1);  // Customer's toString()


        c1.setDiscount(8);

        System.out.println(c1);

        System.out.println("id is: " + c1.getId());

        System.out.println("name is: " + c1.getName());

        System.out.println("discount is: " + c1.getDiscount());


        // Test Invoice class
        Invoice inv1 = new Invoice(101, c1, 888.8);
```

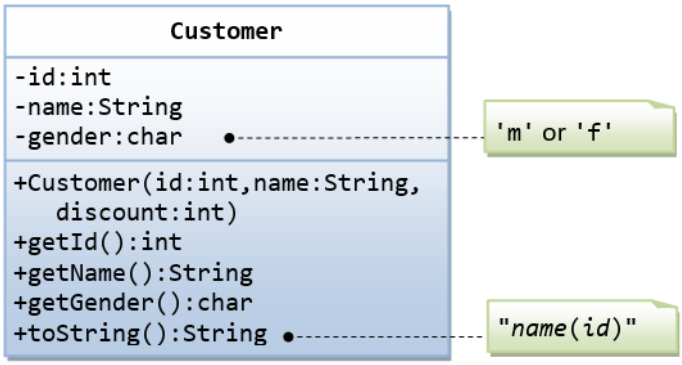
```
        System.out.println(inv1);


        inv1.setAmount(999.9);
        System.out.println(inv1);
        System.out.println("id is: " + inv1.getId());
        System.out.println("customer is: " + inv1.getCustomer());  // Customer's toString()
        System.out.println("amount is: " + inv1.getAmount());
        System.out.println("customer's id is: " + inv1.getCustomerId());
        System.out.println("customer's name is: " + inv1.getCustomerName());
        System.out.println("customer's discount is: " + inv1.getCustomerDiscount());
        System.out.printf("amount after discount is: %.2f%n", inv1.getAmountAfterDiscount());
    }
}
```

La salida esperada es:

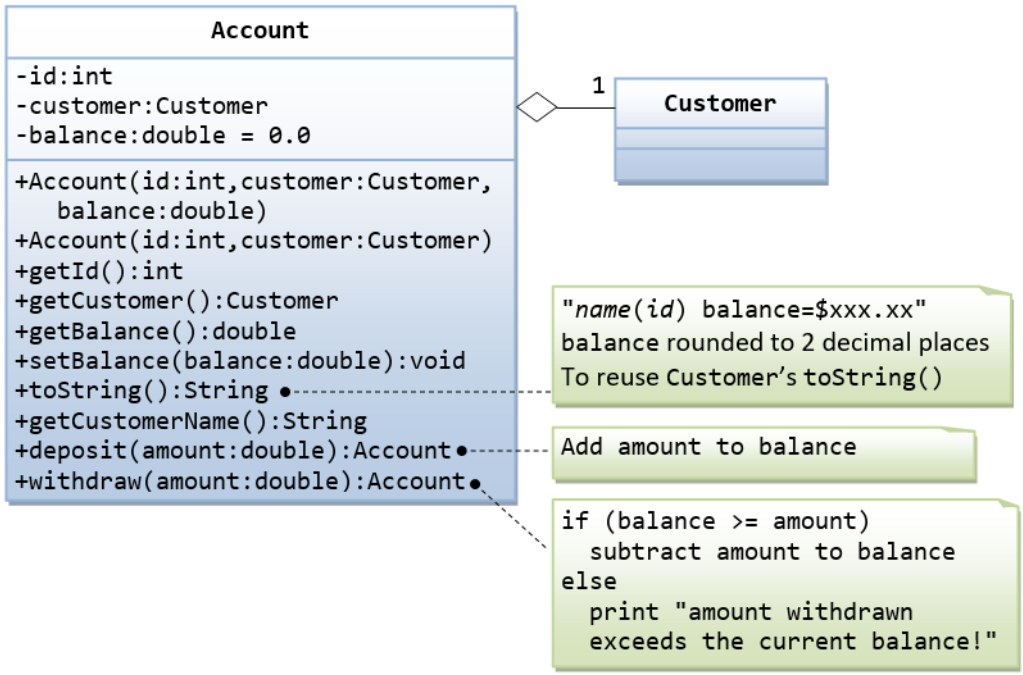
```
Tan Ah Teck(88)(10%)
Tan Ah Teck(88)(8%)
id is: 88
name is: Tan Ah Teck
discount is: 8
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=888.8]
Invoice[id=101,customer=Tan Ah Teck(88)(8%),amount=999.9]
id is: 101
customer is: Tan Ah Teck(88)(8%)
amount is: 999.9
customer's id is: 88
customer's name is: Tan Ah Teck
customer's discount is: 8
amount after discount is: 919.91
```

2.5. Las clases Cliente y Prueba

Dado el siguiente diagrama, programe las clases correspondientes en el paquete indicado para estos ejercicios.



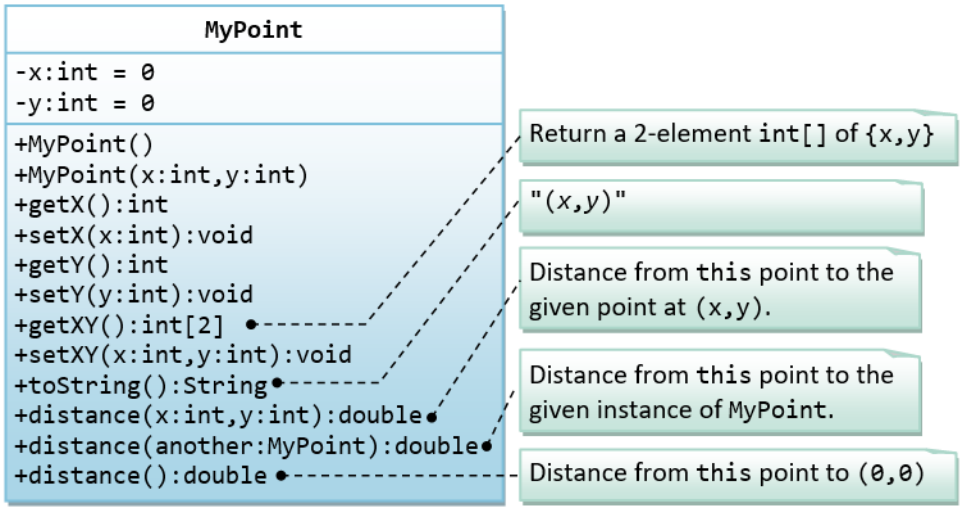
Cuidado! La clase customer la teníamos también en el ejercicio anterior… y son distintas! ¿Qué tal hacer un nuevo paquete dentro del actual? Es una solución que siempre está a mano por si se da este caso...



Programa un programa principal que pruebe los métodos públicos y sirva de ejemplo de uso.

2.6 La clase MyPoint

Dado el siguiente diagrama, programe la clase correspondiente en el paquete indicado para estos ejercicios.



En este ejercicio trabajaremos con la sobrecarga de métodos: varios métodos tendrán el mismo nombre, pero distintos parámetros, por lo que la firma de los mismos será distinta.

La clase tiene:

- dos atributos x e y de tipo int.
- Un constructor por defecto, para la localización (0,0).
- Un constructor sobrecargado que recibe las coordenadas del punto.
- Los getters y setters de los atributos, y de su conjunto (x,y).
- Un método toString para mostrar por pantalla los puntos en formato "(X, Y)"
- Un método para calcular la distancia desde el punto a las coordenadas pasadas por parámetro.
- Un método para calcular la distancia desde el punto a otro punto.
- Un método para calcular la distancia desde el punto al origen de coordenadas.

Ejemplos:

```
//Example distance(x,y)

MyPoint p1 = new MyPoint(3, 4);

System.out.println(p1.distance(5, 6));
```

```
//Example distance(another)

•MyPoint p1 = new MyPoint(3, 4);

MyPoint p2 = new MyPoint(5, 6);

System.out.println(p1.distance(p2));
```

```
//Example distance()

•MyPoint p1 = new MyPoint(3, 4);

System.out.println(p1.distance());
```

Debes escribir la clase con todos sus atributos, constructores, getters, setter y métodos sobrecargados.

Para el cálculo de distancias, que es el cálculo de una línea recta, puede usarse la raíz cuadrada llamando al método estático sqrt de la clase Math:

```
Math.sqrt(numerito);
```

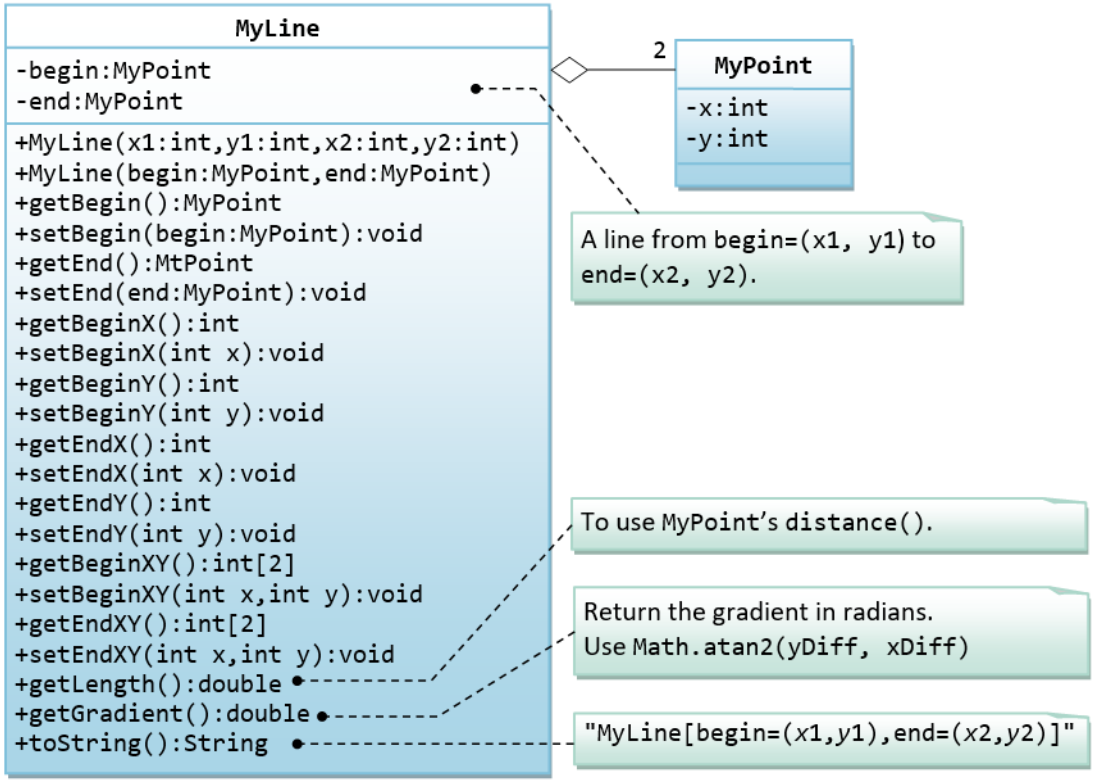
Escribe un programa principal y completa las preguntas de qué versión del método sobrecargado se está utilizando en cada mmomento

```
// Test program to test all constructors and public methods
MyPoint p1 = new MyPoint();  // Test constructor
System.out.println(p1);      // Test toString()
p1.setX(8);   // Test setters
p1.setY(6);
System.out.println("x is: " + p1.getX());  // Test getters
System.out.println("y is: " + p1.getY());
p1.setXY(3, 0);   // Test setXY()
System.out.println(p1.getXY()[0]);  // Test getXY()
System.out.println(p1.getXY()[1]);

System.out.println(p1);


MyPoint p2 = new MyPoint(0, 4);  // Test another constructor
System.out.println(p2);
// Testing the overloaded methods distance()
System.out.println(p1.distance(p2));    // which version?
System.out.println(p2.distance(p1));    // which version?
System.out.println(p1.distance(5, 6));  // which version?
System.out.println(p1.distance());      // which version?
```

Escribe ahora un programa que instancie un array de 10 puntos MyPoint, inicializados como (1,1), (2,2)...(10,10).
Muestra la matriz de distancias entre todos ellos.

2.7 Clases MyLine y MyPoint

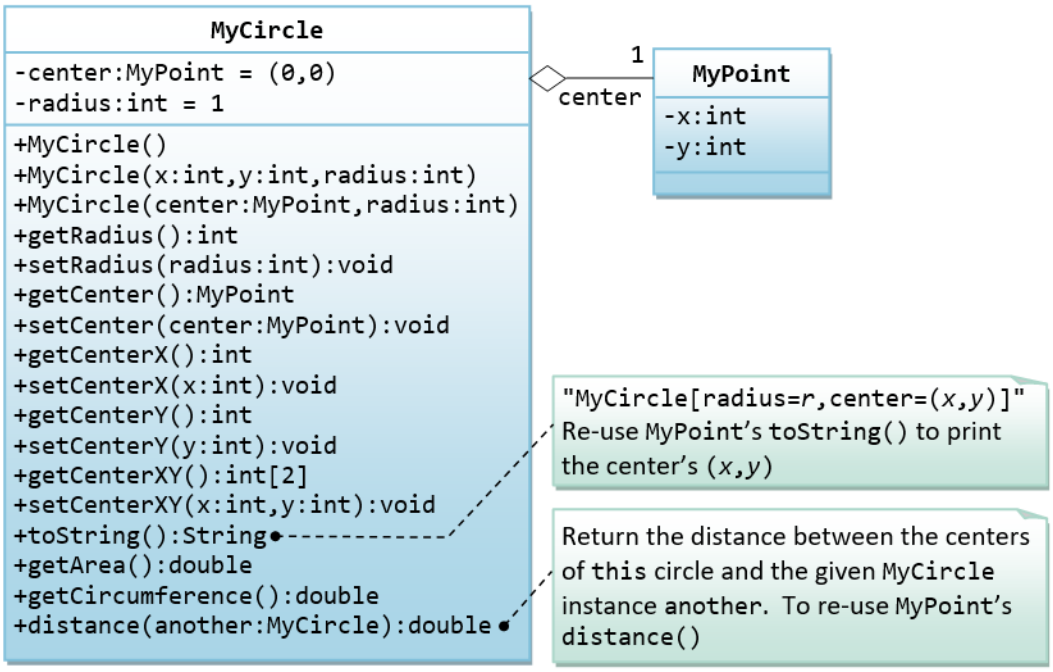
Sabemos que una línea se puede definir a partir de dos puntos: Hagámoslo. Puede usar la clase MyPoint del ejercicio anterior



Escribe un programa que pruebe las clases creadas.

2.8 MyCircle

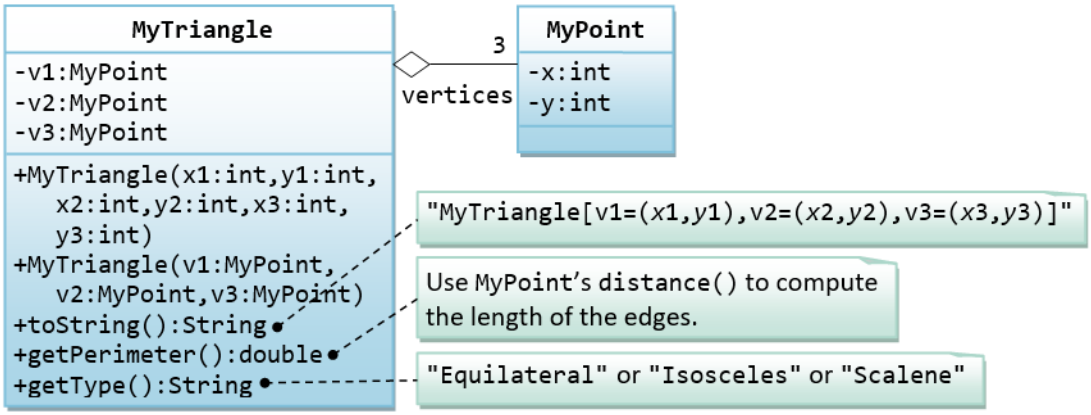
Definamos ahora MyCircle: puede seguir utilizando la clase MyPoint anterior.



Programe un método main que permita probar lo desarrollado y todas sus operaciones.

2.9 Clase MyTriangle

Supongamos ahora que queremos definir una clase MyTriangle a partir de sus 3 vértices:



Programe un método main para probar todos los métodos de la nueva clase creada.

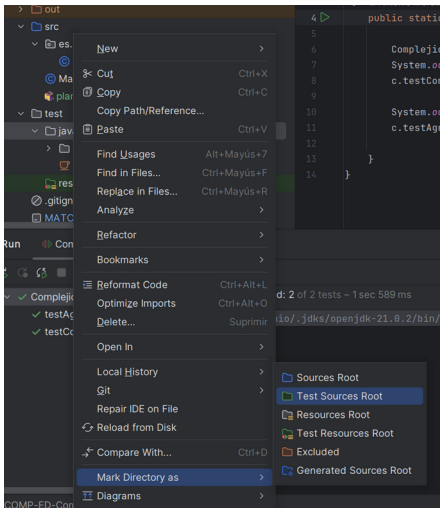
2.10 Mejorando la calidad de las clases

Dado que tenemos los programas de prueba de las clases anteriores, y en estos programas se muestra qué se espera que aparezca en pantalla según la ejecución de los ejercicios, añada para todas las clases de los ejercicios los tests necesarios para verificar la correcta ejecución de todos ellos.

Asegure que los tests tienen una cobertura del 100% del código desarrollado en las clases.

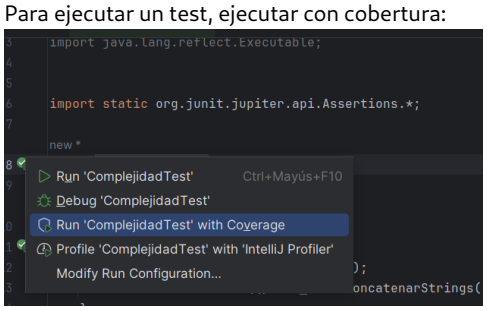
(Usamos JUnit 5)
Pasos:
Crear directorio test
Crear directorio test/java
Crear directorio test/resources

Marcar directorio test/java como “Test Source Root”
Marcar directorio test/resources como “Test Resources Root”



Sobre la clase a testear, botón derecho → “Generate” y en la ventana que aparece, elegir “Test”.

Para ejecutar un test, ejecutar con cobertura:



Referencia:
https://www.jetbrains.com/help/idea/tests-in-ide.html
https://www.jetbrains.com/help/idea/testing.html