Metodología de la programación

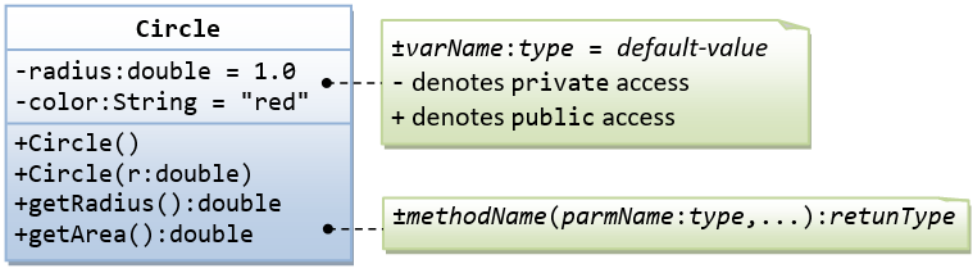
Ejercicios de Programación Orientada a Objetos

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

1.Clases

1.1 Introducción – La clase `Circle`

Este primer ejercicio mostará los conceptos básicos de la programación orientada a objetos:



La clase con nombre `circle` se muestra según el diagrama de clases anterior. Contiene:

- Dos atributos privados : `radius` (de tipo `double`) y `color` (de tipo `String`), con valores inicializados por defecto a 1.0 y "red" respectivamente.
- Dos constructores sobrecargados: Un constructor sin argumentos, también conocido como constructor por defecto, y otro constructor personalizado que tiene el argumento r para el radio del círculo.
- Dos métodos públicos: `getRadius()` y `getArea()`, que devuelven el radio y el área respectivamente.

La clase se programa en un fichero java denominado: `Circle.java` :

```
/**
 * The Circle class models a circle with a radius and color.
 */
public class Circle {  // Save as "Circle.java"
    // private instance variable, not accessible from outside this class
    private double radius;

    private String color;


    // Constructors (overloaded)
    /** Constructs a Circle instance with default value for radius and color */
    public Circle() {  // 1st (default) constructor
        radius = 1.0;

        color = "red";

    }


    /** Constructs a Circle instance with the given radius and default color */
    public Circle(double r) {  // 2nd constructor
        radius = r;

        color = "red";

    }
```

```
    /** Returns the radius */
    public double getRadius() {
       return radius;
    }

    /** Returns the area of this Circle instance */
    public double getArea() {
        return radius*radius*Math.PI;
    }
}
```

Pregunta: ¿Puede ejecutarse la clase creada?
La clase creada no puede ejecutarse directamente. Necesita de un método que la llame. Para ello se crea un fichero nuevo, con un método especial, denominado `main()`, que es el usado por java para poder arrancar un programa. Es el programa principal.

Ahora se va a escribir un programa de prueba, llamado `TestCircle` (en un fichero java llamado `TestCircle.java`) que se encarga de llamar a la clase `Circle`. Como notarás, los ficheros donde se guardan las clases deben llamarse exactamente igual que la clase, respetando las mayúsculas y minúsculas:

```
/**
 *  A Test Driver for the Circle class
 */
public class TestCircle {  // Save as "TestCircle.java"
    public static void main(String[] args) {
        // Declare an instance of Circle class called c1.
        // Construct the instance c1 by invoking the "default" constructor
        // which sets its radius and color to their default value.
        Circle c1 = new Circle();
        // Invoke public methods on instance c1, via dot operator.
        System.out.println("The circle has radius of "
            + c1.getRadius() + " and area of " + c1.getArea());
        //The circle has radius of 1.0 and area of 3.141592653589793

        // Declare an instance of class circle called c2.
        // Construct the instance c2 by invoking the second constructor
        // with the given radius and default color.
        Circle c2 = new Circle(2.0);
        // Invoke public methods on instance c2, via dot operator.
        System.out.println("The circle has radius of "
            + c2.getRadius() + " and area of " + c2.getArea());
        //The circle has radius of 2.0 and area of 12.566370614359172
    }
}
```

Ejecuta el programa usando el botón “run” (triangulo verde) de IntelliJ IDEA para probar su salida. Comprueba los resultados.

Conceptos básicos en POO:

1.**Constructor:** Modifica la clase `Circle` para añadir un tercer constructor, que reciba dos argumentos: radio color:

```
// 3rd constructor to construct a new instance of Circle with the given radius and color
public Circle (double r, String c) { ..... }
```

Modifica el programa principal para poder usar este nuevo constructor, añadiendo un nuevo círculo c3.

2.**Getter:** Un getter es un método especial que sirve para retornar el valor de un atributo. Los atributos deben ser privados salvo excepciones muy justificadas, por lo que no pueden ser consultados desde el exterior. El método getter se encarga de esta funcionalidad sólo para aquellos atributos que deseemos que sean leídos. Añade un getter para el atributo `color`.

```
// Getter for instance variable color
public String getColor() { ..... }
```

Modifica el programa principal para poder probar este nuevo método.

3.**`public` vs. `private`:** Intenta acceder al atributo radio desde el programa principal. ¿Es posible? ¿Por qué? Indica los mensajes recibidos.

4.**Setter:** De la misma manera que un getter permite retornar el valor de un atributo, un setter permite establecerlo. Los setter actúan como puerta para poder recibir datos nuevos en los atributos de una clase, controlando las operaciones para establecer las modificaciones de los mismos según los valores recibidos, de manera que no se puedan hacer modificaciones que violen la lógica interna de la clase: ¿Podemos poner un radio negativo? El setter se encargará de comprobar que los valores que se quieren establecer para su atributo son los adecuados. Añade dos métodos setters para los atributos de la clase.

```
// Setter for instance variable radius
public void setRadius(double newRadius) {
    radius = newRadius;
}

// Setter for instance variable color
public void setColor(String newColor) { ..... }
```

Comprueba las modificaciones realizadas.

```
Circle c4 = new Circle();   // construct an instance of Circle
c4.setRadius(5.5);          // change radius
System.out.println("radius is: " + c4.getRadius()); // Print radius via getter
c4.setColor("green");       // Change color
System.out.println("color is: " + c4.getColor());   // Print color via getter

// You cannot do the following because setRadius() returns void, which cannot be printed
System.out.println(c4.setRadius(4.4));
```

5.**Keyword `"this"`:** Las keywords de un lenguaje, también denominadas palabras clave, son palabras reservadas del lenguaje que tienen una funcionalidad especial. “this” se usa para referirse al objeto sobre el que se está ejecutando la operación del método en el que se usa. Esto es útil cuando un método puede trabajar con varias instancias de una misma clase (varios objetos), de manera que siempre se puede referir al objeto que está ejecutando el método con la keyword “this”. Al acceder a los métodos y atributos de un objeto desde sus propios métodos, es bueno usar esta keyword para explicitar a qué ámbito de variable se está refiriendo el programador. Por ejemplo:

```
// Instance variable
private double radius;

/** Constructs a Circle instance with the given radius and default color */
public Circle(double radius) {
    this.radius = radius;   // "this.radius" refers to the instance variable
```

```
                                    // "radius" refers to the method's parameter
    color = "red";
}


/** Sets the radius to the given value */
public void setRadius(double radius) {
    this.radius = radius;   // "this.radius" refers to the instance variable
                                    // "radius" refers to the method's argument
}
```

Modifica todos los constructores para que usen “this”.

6.**Método `toString()`:** En java es aconsejable que las clases tengan un método público denominado `toString()` que se encarga de devolver una cadena de texto con la descripción de la instancia en ejecución sobre la que se llama al dicho método. Esto permite enviar a consola o a logs los objetos para posterior inspección y depuración. El método `toString()` puede llamarse de manera explícita o de manera implícita.
Crea un método toString para la clase de trabajo.

```
/** Return a self-descriptive string of this instance in the form of Circle[radius=?,color=?] */
public String toString() {
    return "Circle[radius=" + radius + " color=" + color + "]";
}
```

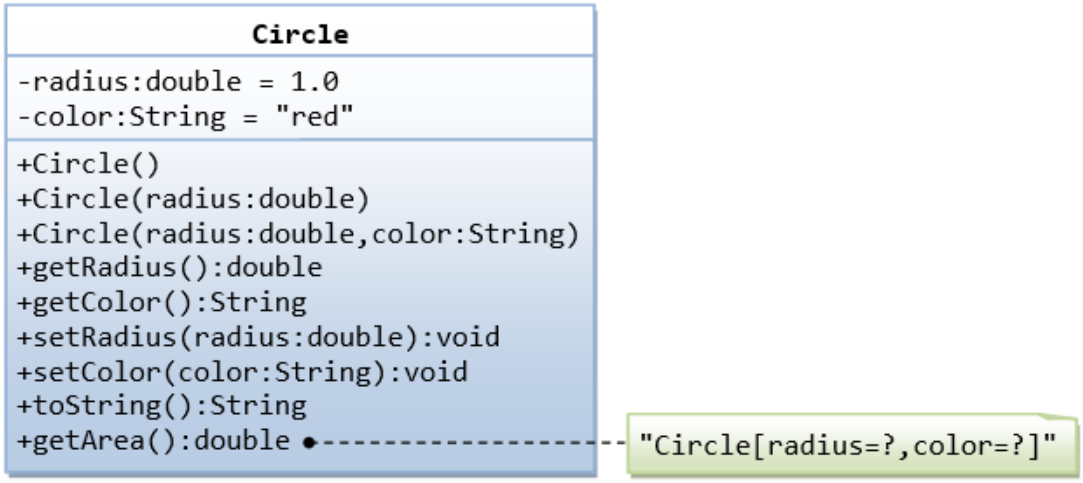
Llama de manera explícita al nuevo método:

```
Circle c5 = new Circle(5.5);
System.out.println(c5.toString());   // explicit call
```

Ahora hazlo de manera implícita:

```
Circle c6 = new Circle(6.6);
System.out.println(c6.toString());  // explicit call
System.out.println(c6);             // println() calls toString() implicitly, same as above
System.out.println("Operator '+' invokes toString() too: " + c6);  // '+' invokes toString() too
```

Una vez hechas todas las modificaciones, el diagrama UML de clase quedaría así:

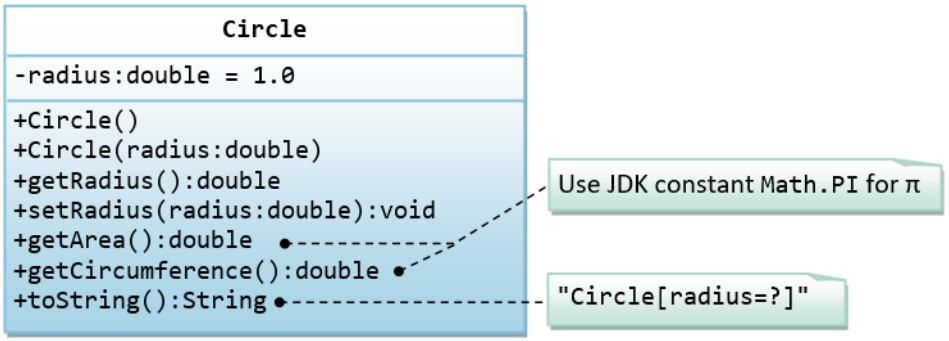


Ahora en trabajo propio:

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

1.2 Nueva clase `Circle`

Crea una nueva clase círculo, pero esta vez asígnala al paquete (harás lo mismo para el resto de clases de los ejercicios).



Este es el código del programa de prueba para esta clase:

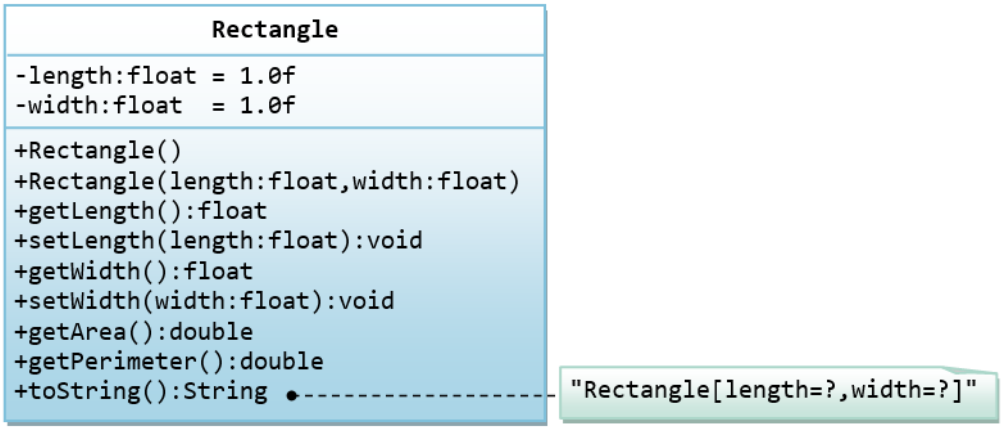
```
public class TestMain {

    public static void main(String[] args) {
        // Test Constructors and toString()
        Circle c1 = new Circle(1.1);
        System.out.println(c1);   // toString()
        Circle c2 = new Circle(); // default constructor
        System.out.println(c2);


        // Test setter and getter
        c1.setRadius(2.2);
        System.out.println(c1);      // toString()
        System.out.println("radius is: " + c1.getRadius());


        // Test getArea() and getCircumference()
        System.out.printf("area is: %.2f%n", c1.getArea());
        System.out.printf("circumference is: %.2f%n", c1.getCircumference());
    }
}
```

La salida esperada es:

```
Circle[radius=1.1]
Circle[radius=1.0]
Circle[radius=2.2]
radius is: 2.2
area is: 15.21
circumference is: 13.82
```

1.3 Nueva clase `Rectangle`

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



Este es el código del programa de prueba para esta clase:

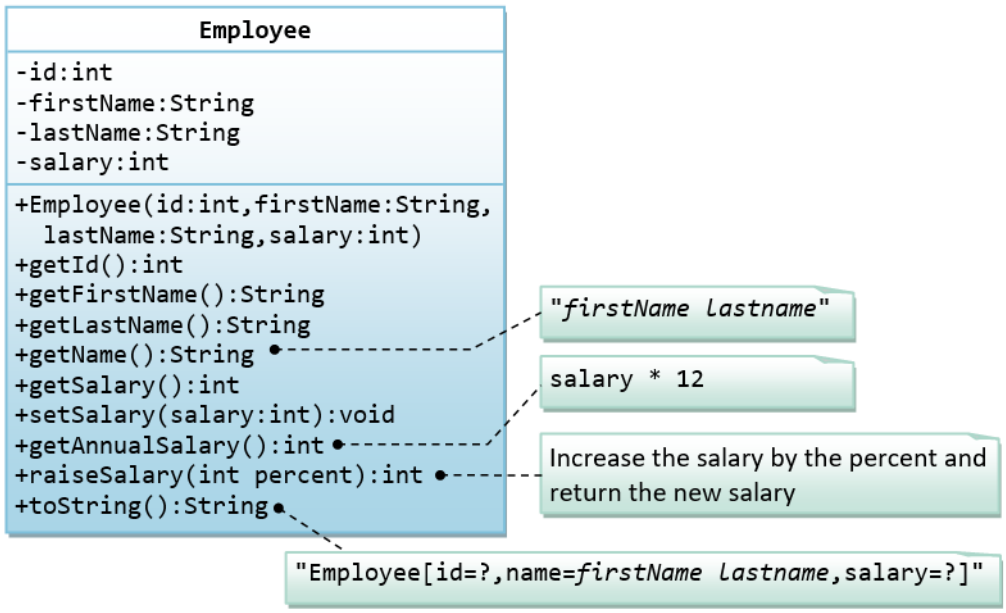
```
public class TestMain {

    public static void main(String[] args) {

        // Test constructors and toString()
        // You need to append a 'f' or 'F' to a float literal

        Rectangle r1 = new Rectangle(1.2f, 3.4f);

        System.out.println(r1);  // toString()
        Rectangle r2 = new Rectangle();  // default constructor

        System.out.println(r2);


        // Test setters and getters

        r1.setLength(5.6f);

        r1.setWidth(7.8f);

        System.out.println(r1);  // toString()

        System.out.println("length is: " + r1.getLength());

        System.out.println("width is: " + r1.getWidth());


        // Test getArea() and getPerimeter()

        System.out.printf("area is: %.2f%n", r1.getArea());

        System.out.printf("perimeter is: %.2f%n", r1.getPerimeter());

    }

}
```

La salida esperada es:

```
Rectangle[length=1.2,width=3.4]
Rectangle[length=1.0,width=1.0]
Rectangle[length=5.6,width=7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

1.4 Nueva clase `Employee`

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



Este es el código del programa de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test constructor and toString()
        Employee e1 = new Employee(8, "Peter", "Tan", 2500);
        System.out.println(e1);  // toString();

        // Test Setters and Getters
        e1.setSalary(999);
        System.out.println(e1);  // toString();
        System.out.println("id is: " + e1.getId());
        System.out.println("firstname is: " + e1.getFirstName());
        System.out.println("lastname is: " + e1.getLastName());
        System.out.println("salary is: " + e1.getSalary());

        System.out.println("name is: " + e1.getName());
        System.out.println("annual salary is: " + e1.getAnnualSalary()); // Test method

        // Test raiseSalary()
        System.out.println(e1.raiseSalary(10));
        System.out.println(e1);
    }
}
```

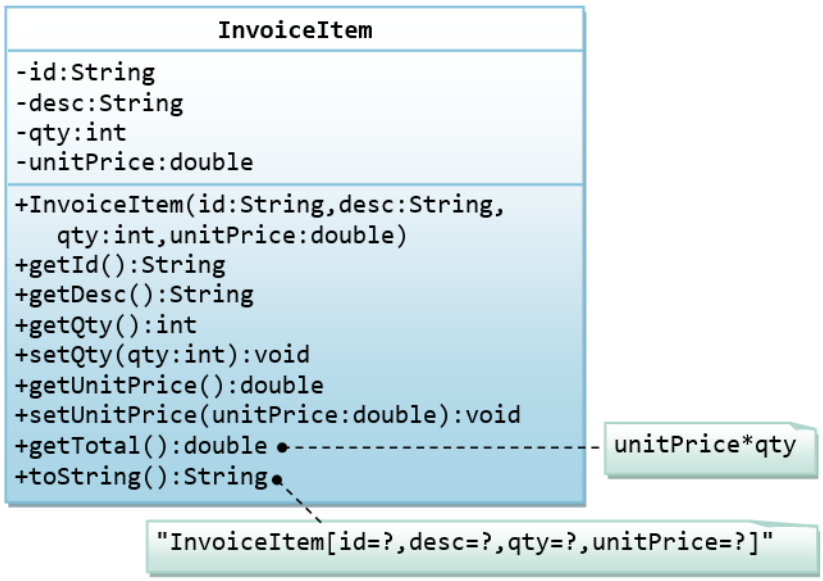
La salida esperada es:

```
Employee[id=8,name=Peter Tan,salary=2500]
Employee[id=8,name=Peter Tan,salary=999]
id is: 8
```

```
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id=8,name=Peter Tan,salary=1098]
```

1.5 Clase `InvoiceItem`

Dado el siguiente diagrama, programe la clase correspondiente a una línea de factura en el paquete indicado para estos ejercicios.



Este es el código del programa de prueba para esta clase:

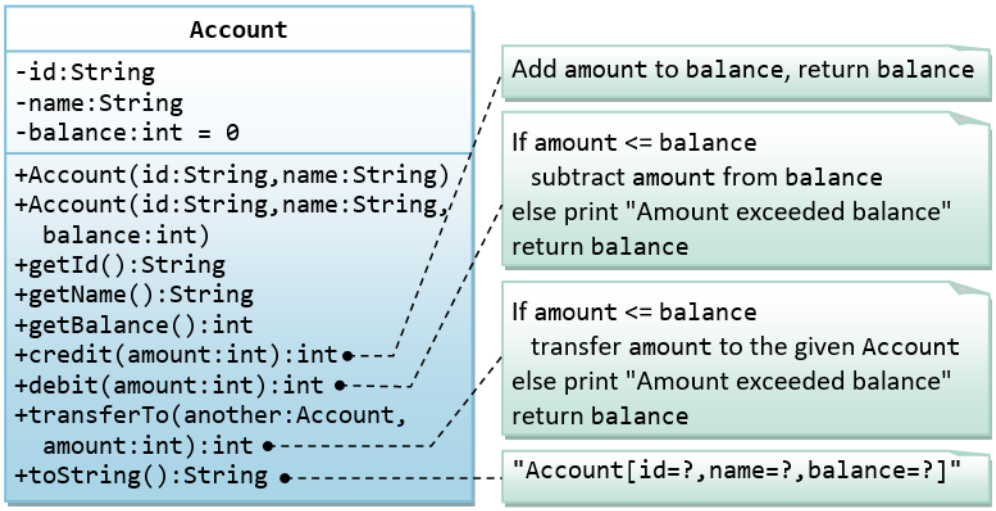
```
public class TestMain {

    public static void main(String[] args) {
        // Test constructor and toString()
        InvoiceItem inv1 = new InvoiceItem("A101", "Pen Red", 888, 0.08);
        System.out.println(inv1);  // toString();


        // Test Setters and Getters
        inv1.setQty(999);
        inv1.setUnitPrice(0.99);
        System.out.println(inv1);  // toString();
        System.out.println("id is: " + inv1.getId());
        System.out.println("desc is: " + inv1.getDesc());
        System.out.println("qty is: " + inv1.getQty());
        System.out.println("unitPrice is: " + inv1.getUnitPrice());


        // Test getTotal()
        System.out.println("The total is: " + inv1.getTotal());
    }
}
```

La salida esperada es:

```
InvoiceItem[id=A101,desc=Pen Red,qty=888,unitPrice=0.08]
InvoiceItem[id=A101,desc=Pen Red,qty=999,unitPrice=0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

1.6 Clase Account

Dado el siguiente diagrama, programe la clase correspondiente a una cuenta de un banco en el paquete indicado para estos ejercicios.
En este ejemplo es importante el uso de "this", ya que algunos métodos, como "transferTo" trabajarán con dos instancias de la misma clase, al recibir por parámetro otra instancia "Account".



Este es el código del programa de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test constructor and toString()
        Account a1 = new Account("A101", "Tan Ah Teck", 88);
        System.out.println(a1);  // toString();
        Account a2 = new Account("A102", "Kumar"); // default balance
        System.out.println(a2);


        // Test Getters
        System.out.println("ID: " + a1.getID());
        System.out.println("Name: " + a1.getName());
        System.out.println("Balance: " + a1.getBalance());


        // Test credit() and debit()
        a1.credit(100);
        System.out.println(a1);
        a1.debit(50);
        System.out.println(a1);
        a1.debit(500);  // debit() error
        System.out.println(a1);


        // Test transfer()
        a1.transferTo(a2, 100);  // toString()
        System.out.println(a1);
        System.out.println(a2);

    }
```

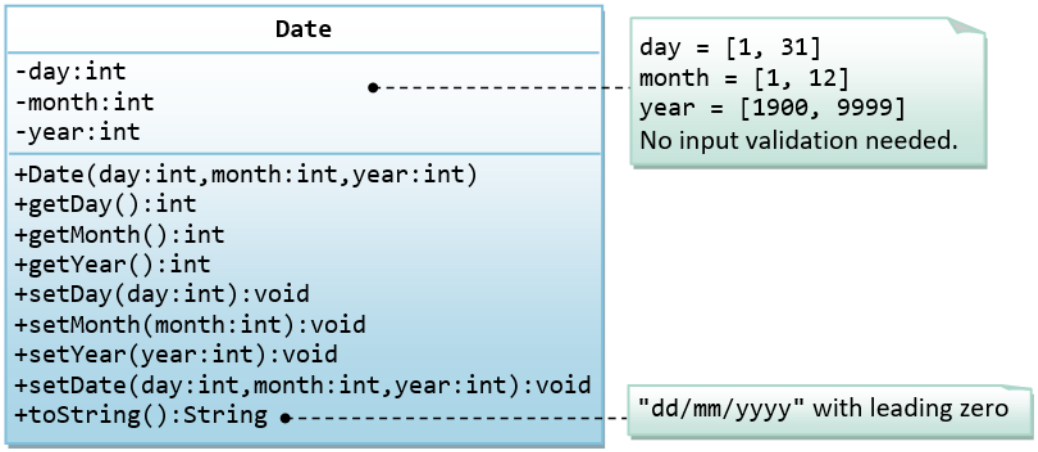
```
}
```

La salida esperada es:

```
Account[id=A101,name=Tan Ah Teck,balance=88]
Account[id=A102,name=Kumar,balance=0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id=A101,name=Tan Ah Teck,balance=188]
Account[id=A101,name=Tan Ah Teck,balance=138]
Amount exceeded balance
Account[id=A101,name=Tan Ah Teck,balance=138]
Account[id=A101,name=Tan Ah Teck,balance=38]
Account[id=A102,name=Kumar,balance=100]
```

1.7 Clase Date

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



Este es el código del programa de prueba para esta clase:

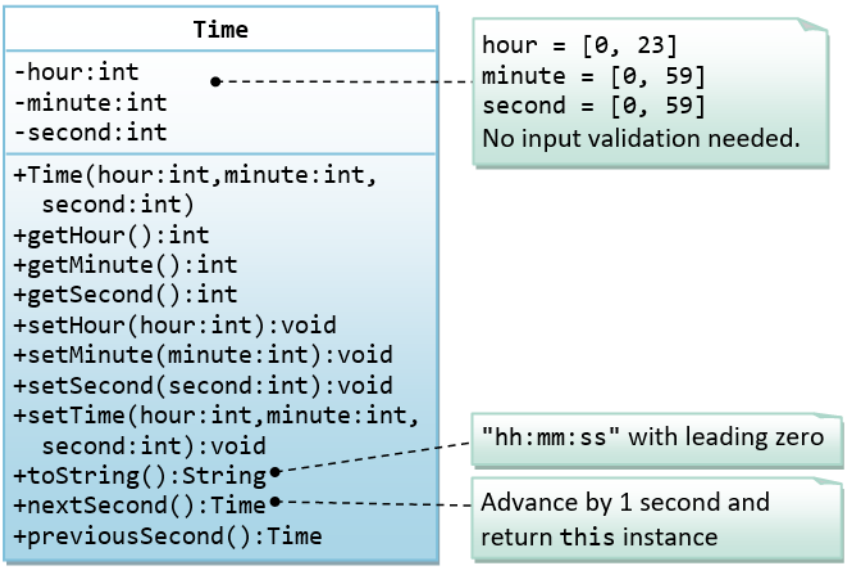
```
public class TestMain {

    public static void main(String[] args) {
        // Test constructor and toString()
        Date d1 = new Date(1, 2, 2014);
        System.out.println(d1);  // toString()

        // Test Setters and Getters
        d1.setMonth(12);
        d1.setDay(9);
        d1.setYear(2099);
        System.out.println(d1);  // toString()
        System.out.println("Month: " + d1.getMonth());
        System.out.println("Day: " + d1.getDay());
        System.out.println("Year: " + d1.getYear());

        // Test setDate()
        d1.setDate(3, 4, 2016);
        System.out.println(d1);  // toString()
    }

}
```

La salida esperada es:

```
01/02/2014
09/12/2099
Month: 12
Day: 9
Year: 2099
03/04/2016
```

1.8 Clase Time

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



Este es el código del programa de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test constructors and toString()
        Time t1 = new Time(1, 2, 3);
        System.out.println(t1);  // toString()

        // Test Setters and Getters
        t1.setHour(4);
        t1.setMinute(5);
        t1.setSecond(6);
        System.out.println(t1);  // toString()
        System.out.println("Hour: " + t1.getHour());
        System.out.println("Minute: " + t1.getMinute());
        System.out.println("Second: " + t1.getSecond());

        // Test setTime()
        t1.setTime(23, 59, 58);
        System.out.println(t1);  // toString()

        // Test nextSecond();
        System.out.println(t1.nextSecond());
        System.out.println(t1.nextSecond().nextSecond());

        // Test previousSecond()
        System.out.println(t1.previousSecond());
        System.out.println(t1.previousSecond().previousSecond());
    }
```

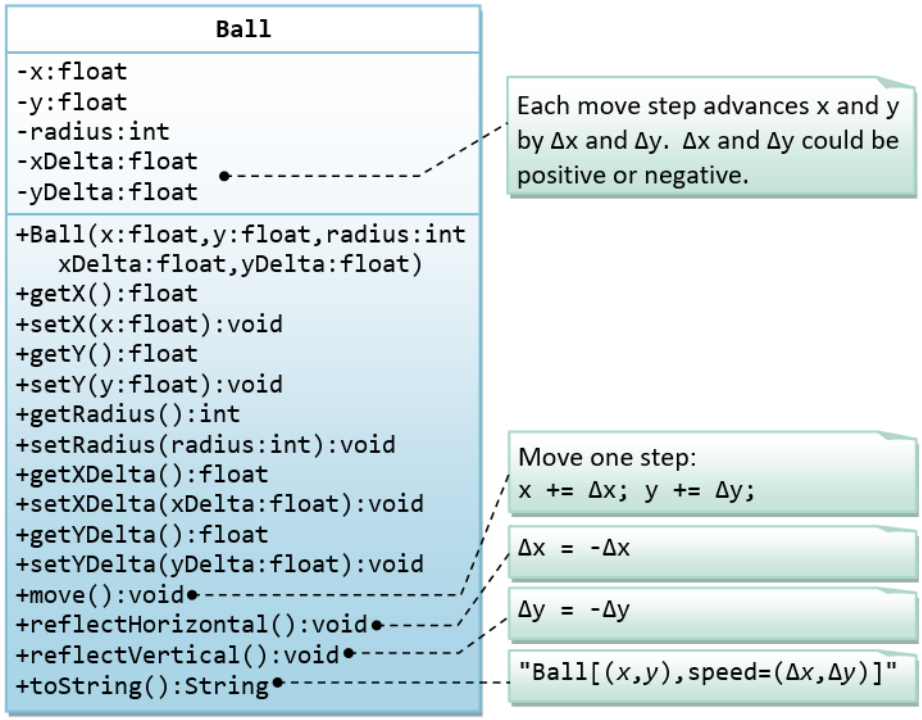
```
}
```

La salida esperada es:

```
01:02:03
04:05:06
Hour: 4
Minute: 5
Second: 6
23:59:58
23:59:59
00:00:01
00:00:00
23:59:58
```

1.9 Clase Ball

En este ejercicio se propone modelar una pelota que se va moviendo según un incremento o decremento establecido en los Delta correspondientes. La clase se inicializa, y una vez lista, cada vez que se llame al método "move" actualizará la posición de la pelota en función de los valores de los atributos. Los métodos "reflect*" indican que se cambia el sentido del delta correspondiente. Antes de programar nada, estudia detenidamente el código del programa de prueba y su salida esperada.



Este es el código del programa de prueba para esta clase:

```
public class TestMain {

    public static void main(String[] args) {
        // Test constructor and toString()
        Ball ball = new Ball(1.1f, 2.2f, 10, 3.3f, 4.4f);
        System.out.println(ball);  // toString()

        // Test Setters and Getters
        ball.setX(80.0f);
        ball.setY(35.0f);
        ball.setRadius(5);
        ball.setXDelta(4.0f);
        ball.setYDelta(6.0f);
        System.out.println(ball);  // toString()
        System.out.println("x is: " + ball.getX());
        System.out.println("y is: " + ball.getY());
        System.out.println("radius is: " + ball.getRadius());
        System.out.println("xDelta is: " + ball.getXDelta());
        System.out.println("yDelta is: " + ball.getYDelta());

        // Bounce the ball within the boundary
```

```
        float xMin = 0.0f;
        float xMax = 100.0f;
        float yMin = 0.0f;
        float yMax = 50.0f;
        for (int i = 0; i < 15; i++) {
            ball.move();
            System.out.println(ball);
            float xNew = ball.getX();
            float yNew = ball.getY();
            int radius = ball.getRadius();
            // Check boundary value to bounce back
            if ((xNew + radius) > xMax || (xNew - radius) < xMin) {
                ball.reflectHorizontal();
            }
            if ((yNew + radius) > yMax || (yNew - radius) < yMin) {
                ball.reflectVertical();
            }
        }
    }
}
```

La salida esperada es:

```
Ball[(1.1,2.2),speed=(3.3,4.4)]
Ball[(80.0,35.0),speed=(4.0,6.0)]
x is: 80.0
y is: 35.0
radius is: 5
xDelta is: 4.0
yDelta is: 6.0
Ball[(84.0,41.0),speed=(4.0,6.0)]
Ball[(88.0,47.0),speed=(4.0,6.0)]
Ball[(92.0,41.0),speed=(4.0,-6.0)]
Ball[(96.0,35.0),speed=(4.0,-6.0)]
Ball[(92.0,29.0),speed=(-4.0,-6.0)]
Ball[(88.0,23.0),speed=(-4.0,-6.0)]
Ball[(84.0,17.0),speed=(-4.0,-6.0)]
Ball[(80.0,11.0),speed=(-4.0,-6.0)]
Ball[(76.0,5.0),speed=(-4.0,-6.0)]
Ball[(72.0,-1.0),speed=(-4.0,-6.0)]
Ball[(68.0,5.0),speed=(-4.0,6.0)]
Ball[(64.0,11.0),speed=(-4.0,6.0)]
Ball[(60.0,17.0),speed=(-4.0,6.0)]
Ball[(56.0,23.0),speed=(-4.0,6.0)]
Ball[(52.0,29.0),speed=(-4.0,6.0)]
```

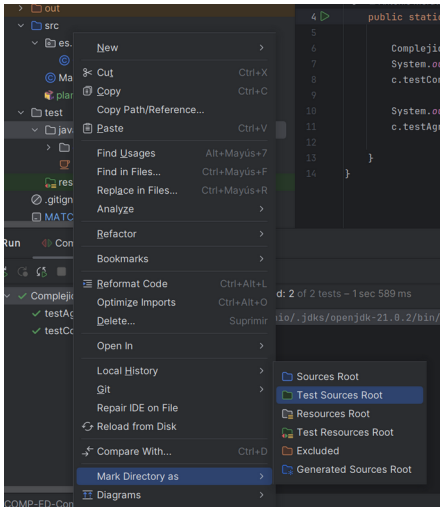
1.10 Mejorando la calidad de las clases

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