# TO SAME

# SESIÓN 6: Clases y Objetos I

# **Objetivos**

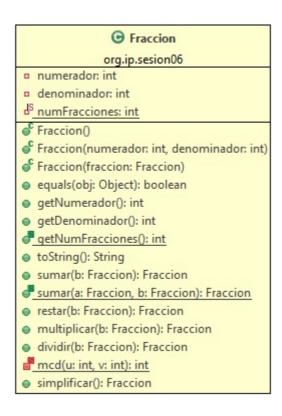
- Saber declarar una clase.
- Saber crear o instanciar objetos de una clase haciendo uso de los constructores.
- Distinguir entre variables de instancia y de clase (static) y métodos de instancia y de clase.
- Distinguir entre variables y métodos, públicos y privados.
- Saber interpretar un diagrama de clases en UML para la implementación de las mismas.

**Nota importante:** Siga el esquema de nombrado de paquetes que se indicó en la sesión 01 es decir: **org.ip.sesion06**. En ese paquete se crearán todos los programas que se proponen en la sesión dándoles un nombre al programa y que se indica en cada ejercicio entre **paréntesis y en negrita**.

Al final de la sesión, el alumno deberá cargar el trabajo realizado a su repositorio personal indicando la clave correspondiente a la sesión.

### **Ejercicios propuestos**

**1.** Implementa una clase *Fraccion* (**Fraccion**) con dos atributos: el numerador y el denominador y algunas de las operaciones habituales como la suma, resta, multiplicación, división, inversa y simplificación. Fíjate en el diagrama de clases que se muestra a continuación. El constructor por defecto deberá crear la fracción nula 0/1.



A continuación, crea un programa **(TestFraccion)** que permita probar la clase anterior. Para ello deberás:

✓ Crear las fracciones 1/5, 4/5, -11/22, 4/5 (utilizando el constructor copia).

```
Fraccion frac1 = new Fraccion(1, 5);
Fraccion frac2 = new Fraccion(4, 5);
Fraccion frac3 = new Fraccion(-11, 22);
Fraccion frac4 = new Fraccion(frac2);
```

- ✓ Mostrar las cuatro fracciones creadas.
- ✓ Mostrar el número de fracciones creadas.
- ✓ Comprobar si la primera fracción es igual a la segunda.
- ✓ Comprobar si la segunda fracción es igual a la cuarta
- ✓ Mostrar el numerador de la tercera fracción.
- ✓ Mostrar el denominador de la primera fracción.
- ✓ Mostrar la suma de la primera y segunda fracción. Comprueba que coincide el resultado utilizando los dos métodos diseñados.
- ✓ Mostrar la resta de la primera y segunda fracción.
- ✓ Mostrar el producto de la primera y segunda fracción.
- ✓ Mostrar la división de la primera y tercera fracción.
- ✓ Mostrar la tercera fracción simplificada.
- ✓ Mostrar el número de fracciones creadas.

#### Ejemplo de ejecución:



```
LAS FRACCIONES CREADAS SON
```

PRIMERA FRACCION => 1/5

```
SEGUNDA FRACCION => 4/5

TERCERA FRACCION => -11/22

CUARTA FRACCION => 4/5

El numero de fracciones creadas es 4

La primera fraccion NO ES IGUAL a la segunda

La segunda fraccion ES IGUAL a la cuarta

El numerador de la tercera fraccion es => -11

El denominador de la primera fraccion es => 5

La suma, utilizando el metodo de clase de 1/5 + 4/5 es 25/25

La suma, utilizando el metodo de objeto de 1/5 + 4/5 es 25/25 simplificada 1

La resta de 1/5 - 4/5 es -15/25 simplificada -3/5

El producto 1/5 x 4/5 es 4/25

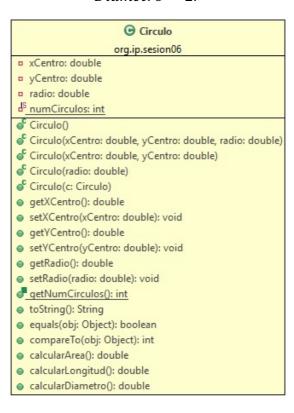
La division de 1/5 / -11/22 es 22/-55

La fraccion -11/22 simplificada es -1/2

El numero de fracciones creadas es 11
```

- **2.** Diseña e implementa una clase *Circulo* (**Circulo**) que permita representar un círculo una posición xCentro, yCentro del plano y con un valor de radio, radio. Para ello:
  - ✓ Declara como propiedades *privadas*, atributos o variables de objeto o instancia los campos xCentro, yCentro y radio.
  - ✓ Declara un atributo de clase e inicialízalo a 0, numCirculos. Este atributo permite llevar el control del número de círculos que se vayan creando durante la ejecución de un programa.
  - ✓ Implementa un constructor general que permita crear un objeto de dicha clase inicializando los atributos (xCentro, yCentro y radio) a los valores que proporciona el usuario como parámetros. Y otros constructores particulares que permitan crear un objeto círculo pasándole como parámetro sólo el centro, sólo el *radio*, y un último constructor que le pase otro objeto círculo como parámetro (constructor copia).
  - ✓ Implementa los métodos *getters* y *setters* vinculados a todos los atributos de la clase. Destacar que también hay que implementar el método de clase getNumCirculos().
  - ✓ Implementa el método toString que permita representar un círculo dado con los valores de su centro y de su radio. Una salida ejemplo, debería ser *Circulo* = {(3.0, 2.0), 1.5}.
  - ✓ Implementa el método equals que para comparar si dos objetos Circulo son iguales o no.
  - ✓ Implementa el método compareTo para comparar dos objetos Circulo en base a su *área*.
  - ✓ Implementa los tres siguientes métodos de instancia: calcularArea, calcularLongitud y calcularDiametro, que permitan calcular la superficie del círculo, la longitud de la circunferencia que lo limita y el valor del diámetro de dicha figura geométrica. Para ello es importante conocer las siguientes fórmulas, donde r representa el radio:

 $Area = \pi r^2$   $Longitud = 2\pi r$  Diametro = 2r



Implementa una clase (**TestCirculo**) que permita obtener exactamente la misma salida que se muestra a continuación. Además, debe utilizar al principio del programa (main) todos los constructores implementados para este fin en la clase Circulo.

```
Circulo circ1 = new Circulo(1.0, 1.0, 1.0);
Circulo circ2 = new Circulo();
Circulo circ3 = new Circulo(0.0, 0.0);
Circulo circ4 = new Circulo(circ3);
```

En el programa debe de ejecutar todo lo necesario para mostrar la salida que se indica en la siguiente figura, destacando que se tendrán que utilizar todos los métodos (excepto algunos *setters* o *getters*) que se han demandado en el diagrama de la clase Circulo y que han debido ser correctamente implementados.

#### Ejemplo de ejecución:



```
*** Programa que permite trabajar con circulos ***
Circulo 1 Circulo={(xCentro=1.0, yCentro=1.0), radio=1.0}
Circulo 2 Circulo={(xCentro=0.0, yCentro=0.0), radio=0.0}
Circulo 3 Circulo={(xCentro=0.0, yCentro=0.0), radio=0.0}
Circulo 4 Circulo={(xCentro=0.0, yCentro=0.0), radio=0.0}
Circulos 1 y 2 son distintos
Circulos 2, 3 y 4 son iguales
El numero de circulos creados es 4
Circulo 1 es mas grande que el circulo 2 en area
Circulo 1 => [<area>=3.141592653589793, <diametro>=2.0, <longitud>=6.283185307179586]
Indica las coordenadas del punto P (x, y) centro de un nuevo circulo
2.5
3.5
Introduce el valor del radio del nuevo circulo
Circulo 5 Circulo={(xCentro=2.5, yCentro=3.5), radio=4.0}
Circulo 5 => [<area>=50.26548245743669, <diametro>=8.0, <longitud>=25.132741228718345]
El numero de circulos creados es 5
```

# Trabajo autónomo

3. Diseña e implementa una clase Java (Reloj) que permita controlar el tiempo a nivel de hora, minuto y segundo. La clase dispondrá de tres constructores, uno sin parámetros que pone el reloj a 00:00:00, otro que se le pasan la hora, minutos y segundos (si los valores están fuera de rango los fija a 0); y un tercero que es el constructor copia. Además, tendrá los siguientes métodos: (1) Método que muestre las horas, minutos y segundo en el formato hh:mm:ss. (2) Métodos getters para obtener la hora, minutos y segundos. (3) Métodos para establecer (setters) las horas, minutos y segundos. Si los valores de horas (0 ≤ hora ≤ 23), minutos (0 ≤ minuto ≤ 59) y segundos (0 ≤ segundo ≤ 59) están fuera de rango, los fija a 0. (4) Métodos para incrementar en uno la hora, minuto y segundo, teniendo en cuenta como afecta ese incremento a los atributos relacionados. Por ejemplo, si se incrementa en uno los segundos, si el segundo alcanza el valor de 60, se incrementará en uno el minuto y se pondrá a 0 el valor del segundo. Y si ese incremento de los minutos hace que se alcance el valor de 60 de los minutos, se incrementará la hora en uno, estableciéndose a 0 el minuto. Por último, si ese incremento de la hora provoca que se alcance el valor de 24, se pondrá a 0 la hora.



Además, hay que implementar una clase (**TestReloj**) que permita obtener exactamente la misma salida que se muestra a continuación y que permita comprobar el correcto funcionamiento de la clase Reloj. Para ello se deberán crear cinco objetos reloj. Para el primero de ellos se utilizará el constructor por defecto, y después se fijará la hora, minuto y segundo a 23:59:59, se muestra el reloj, se incrementa un segundo y se vuelve a mostrar el reloj. El segundo creará un objeto reloj con parámetros de hora, minuto y segundo igual a 24:60:60, luego muestra el reloj, incrementa un segundo, un minuto y una hora, y luego vuelve a mostrar el reloj. El tercer reloj será igual que el primero, haciendo uso de constructor copia, muestra el reloj, luego en un bucle incrementa 86400 segundos, y vuelve a mostrar el reloj. El cuarto también será igual al primer reloj (constructor copia), muestra el reloj, luego en un bucle incrementa 1440 minutos, y vuelve a mostrar el reloj. Finalmente, se creará un quinto reloj igual que el primer reloj (constructor copia), muestra el reloj, luego en un bucle incrementa 24 horas, y vuelve a mostrar el reloj.

#### Ejemplo de ejecución:



23:59:59 00:00:00

00:00:00 01:01:01

00:00:00 00:00:00

00:00:00 00:00:00

00:00:00 00:00:00

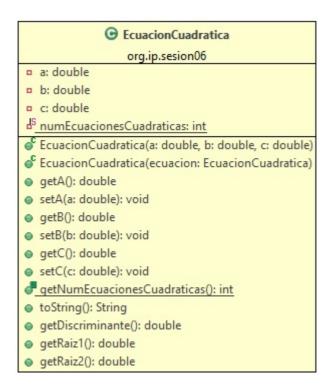
- **4.** Diseña e implementa una clase para la solución de *ecuaciones de segundo grado* o *ecuación cuadrática* (**EcuacionCuadratica**) de la forma  $ax^2 + bx + c = 0$ . Para ello
  - ✓ Declara como propiedades privadas, atributos o variables de objeto o instancia los campos a, b y c. Que representan los tres coeficientes.
  - ✓ Declara un atributo de clase e inicialízalo a 0, numEcuacionesCuadraticas. Este atributo permite llevar el control del número de ecuaciones cuadráticas que se vayan creando durante la ejecución de un programa.
  - ✓ Implementa un constructor general que permita crear un objeto de dicha clase inicializando los atributos (a, b y c) a los valores que proporciona el usuario como parámetros.
  - ✓ Implementa los métodos *getters* y *setters* vinculados a todos los atributos de la clase. Destacar que también hay que implementar el método de clase asociado al atributo de clase que hemos creado anteriormente, getNumEcuacionesCuadraticas().
  - ✓ Implementa el método toString que permita representar una ecuación de segundo grado en función de sus coeficientes. Una salida ejemplo, debería ser *Ecuacion Cuadratica*: (a = 1.0, b = -2.0, c = 1.0).
  - ✓ Implementa un método de instancia, getDiscriminante, que permitan calcular y devolver el valor del discriminante de la ecuación cuadrática, cuya fórmula es la siguiente:

$$Discriminante = b^2 - 4ac$$

✓ Implementa los métodos de instancia, getRaiz1 y getRaiz2, que permitan calcular y devolver las dos raíces de la ecuación, cuyas fórmulas son las que se indican a continuación. Sabemos que estos dos métodos tienen sentido si el discriminante es no-negativo, devolviendo 0 estos métodos si el discriminante es negativo.

$$raiz_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$raiz_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$



Implementa una clase (**TestEcuacionCuadratica**) que permita obtener exactamente la misma salida que se muestra a continuación. Para este test debes tener en cuenta que introduzca desde teclado los valores de los coeficientes (a, b y c), muestre la ecuación (toString()) y el resultado en función del valor del discriminante. Es decir, debe comprobar si *a* es cero, si el discriminante es positivo, y mostrar las dos raíces. Si el discriminante es 0, debe mostrar la raíz única. Y en cualquier otro caso, debe mostrar un mensaje como que "La ecuación no tiene raíces reales".

En el programa debe de ejecutar todo lo necesario para mostrar la salida que se indica en la siguiente figura, destacando que se tendrán que utilizar todos los métodos que se han demandado en el diseño de la clase EcuacionCuadratica y que han debido ser correctamente implementados. Note que este test debe ser igual al del ejercicio de la sesión 02 pero haciendo uso de clases, y que cada caso supone una ejecución independiente.

#### Ejemplos de ejecuciones:



```
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
a = 0
b = 1
c = 1
Ecuacion Cuadratica: (a=0.0, b=1.0, c=1.0)
No es una ecuacion cuadratica
```

```
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
a = 2
b = 1
c = 1
Ecuacion Cuadratica: (a=2.0, b=1.0, c=1.0)
Ecuacion cuadratica sin raices reales
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
b = -3
c = 2
Ecuacion Cuadratica: (a=1.0, b=-3.0, c=2.0)
Ecuacion cuadratica con dos raices de valores
x1 = 2.0
x2 = 1.0
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
a = 1
b = 0
c = -4
Ecuacion Cuadratica: (a=1.0, b=0.0, c=-4.0)
Ecuacion cuadratica con dos raices de valores
x1 = 2.0
x2 = -2.0
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
a = -2
b = -3
Ecuacion Cuadratica: (a=-2.0, b=-3.0, c=5.0)
Ecuacion cuadratica con dos raices de valores
x1 = -2.5
x2 = 1.0
Introduce los valores de los coeficientes de la
ecuacion cuadratica: ax*x + b*x + c = 0
a = 1
b = -8
c = 16
Ecuacion Cuadratica: (a=1.0, b=-8.0, c=16.0)
Ecuacion cuadratica con una unica raiz de valor doble
x = 4.0
```