SESIÓN 5: Clases y Objetos I

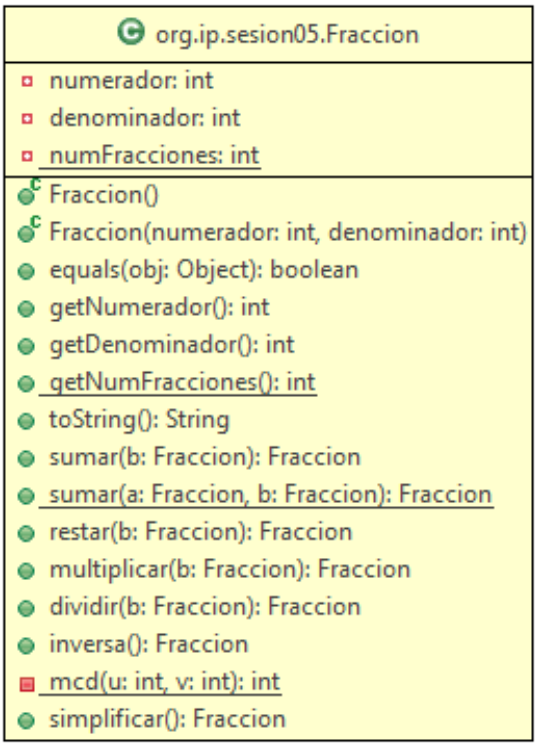
Objetivos

- 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 construir un diagrama de clases en UML para describir las mismas.

Nota importante: Siga el esquema de nombrado de paquetes que se indicó en la sesión 01 es decir: **org.ip.sesion05**. 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. Crea dicho diagrama **(fraccion.cld)**



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

- ✓ Crear las fracciones 1/5, 4/5, -11/22.
- ✓ Mostrar las tres fracciones creadas.
- ✓ Mostrar el número de fracciones creadas.
- ✓ Comprobar si la primera fracción es igual a la segunda.
- ✓ 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 inversa de la primera fracción.
- ✓ Mostrar la tercera fracción simplificada.
- ✓ Mostrar el número de fracciones creadas.

Ejemplo de ejecución:

```
LAS FRACCIONES CREADAS SON

PRIMERA FRACCIÓN => 1/5
SEGUNDA FRACCIÓN => 4/5
TERCERA FRACCIÓN => -11/22

El número de fracciones creadas es 3
La primera fracción NO ES IGUAL a la segunda
El numerador de la tercera fracción es => -11
El denominador de la primera fracción es => 5
La suma, utilizando el método de clase de 1/5 + 4/5 es 25/25
La suma, utilizando el método 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 división de 1/5 / -11/22 es 22/-55
La inversa de la primera fracción 1/5 es 5
La fraccion -11/22 simplificada es -1/2
El número de fracciones creadas es 11
```

Trabajo autónomo

2. 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 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 (*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 un círculo dado con los valores de su centro y de su radio. Una salida ejemplo, debería ser *Ecuacion Cuadratica: ax*x +bx +c*. Para ello debes tener en cuenta si los valores de los coeficientes son positivos o negativos, para poner un signo positivo o negativo al respecto.
- ✓ 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.

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

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

Crea el diagrama de clases UML correspondiente **(ecuacioncuadratica.cld)**

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 *EcuacionCuadrativa* 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.

Ejemplos de ejecuciones: 

```
Introduce los valores de los coeficientes de la
ecuación cuadrática: ax*x + b*x + c = 0
a = 0
b = 1
c = 1
EcuacionCuadratica: 1.0x + 1.0 = 0
No es una ecuación cuadrática

Introduce los valores de los coeficientes de la
ecuación cuadrática: ax*x + b*x + c = 0
a = 2
b = 1
c = 1
EcuacionCuadratica: 2.0x*x + 1.0x + 1.0 = 0
Ecuación cuadratica sin raices reales

Introduce los valores de los coeficientes de la
ecuación cuadrática: ax*x + b*x + c = 0
a = 1
b = -3
c = 2
EcuacionCuadratica: 1.0x*x - 3.0x + 2.0 = 0
Ecuación cuadratica con dos raices de valores
x1 = 2.0
x2 = 1.0

Introduce los valores de los coeficientes de la
ecuación cuadrática: ax*x + b*x + c = 0
a = 0
b = 0
c = -4
EcuacionCuadratica: -4.0 = 0
No es una ecuación cuadrática

Introduce los valores de los coeficientes de la
ecuación cuadrática: ax*x + b*x + c = 0
a = -2
b = -7
c = 5
EcuacionCuadratica: -2.0x*x - 7.0x + 5.0 = 0
Ecuación cuadratica con dos raices de valores
x1 = -4.10849528301415
x2 = 0.6084952830141508
```

3. 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 nos permitirá comparar si dos objetos Circulo son iguales o no.
- ✓ Implementa el método *compareTo* que permita comparar dos objetos Circulo en función de su área.
- ✓ Implementa tres métodos de instancia, *calcularArea*, *calcularLongitud* y *calcularDiametro*, que permitan calcular la superficie del círculo, la longitud de la circunsferencia 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 otros tres métodos de instancia, uno de ellos permita desplazar el círculo en el plano 2D, otro que permita hacer más grande o más pequeño un círculo en función de un factor mayor que cero y por último otro que permita determinar si un punto está dentro o no de un círculo dado. Los métodos deberán denominarse, *desplazar*, *cambiarTamanio* y *estaDentro*. Para este último método debemos saber que un punto *(x, y)* cualquiera del plano 2D está dentro de la circunsferencia centrada en *(xCentro, yCentro)* y con radio *r* si satisface la siguiente ecuación $(x - xCentro)^2 + (y - yCentro)^2 \leq r^2$.

Crea el diagrama de clases UML correspondiente **(circulo.cld)**

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 diseño 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 número de circulos creados es 4
Circulos 1 y 4 son iguales
Circulo 1 es mas grande que el circulo 2 en area
Circulo 1 => 19.634954084936208, 5.0, 15.707963267948966
El punto (1.0, 2.0) está dentro del circulo 1 Circulo={(xCentro=1.0, yCentro=1.0), radio=2.5}
Indica las coordenadas del punto P (x, y) centro de un nuevo circulo
2.5
4.5
Introduce el valor del radio del nuevo circulo
3.5
Circulo 5 Circulo={(xCentro=2.5, yCentro=4.5), radio=3.5}
Circulo 5 => 38.48451000647496, 7.0, 21.991148575128552
El número de circulos creados es 5
Circulo 6 Circulo={(xCentro=2.0, yCentro=2.0), radio=5.0}
Circulo 7 Circulo={(xCentro=3.0, yCentro=3.0), radio=1.0}
El número de circulos creados es 7
```