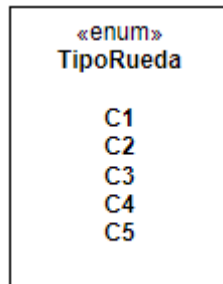
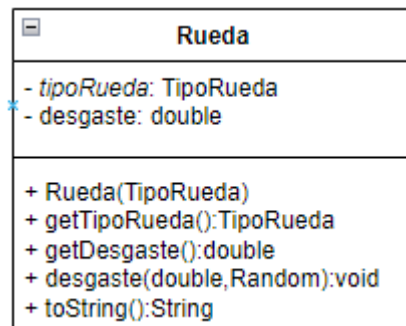


Se desea implementar un videojuego ambientado en la Formula 1. La aplicación será capaz de simular una carrera en diferentes circuitos para ello el alumno deberá implementar las siguientes clases, implementadas en el paquete `f1`



TipoRueda: Tipo enumerado que podrá tomar uno de los siguientes valores (C1, C2, C3, C4 y C5) donde C1 será el neumático más blando (más velocidad y más desgaste) y C5 será el más duro (menos velocidad y menos desgaste)



Rueda: Clase que implementa una rueda de un formula 1. Almacena dos variables de instancia: el *tipo de rueda* (`TipoRueda`), variable de instancia constante y un valor *desgaste* (`double`), comprendido entre los valores 0 y 1 que indica el desgaste porcentual de la rueda en ese momento. Ambos atributos son privados y se dispondrá de los siguientes métodos públicos que han de ser implementados por el alumno:

1. Constructor en el que se proporciona el tipo de la rueda como parámetro. Este constructor implementa la instanciación de una rueda del tipo indicado como parámetro con un desgaste inicial de 1. En caso de que

el tipo de rueda indicado sea nulo, se deberá lanzar una excepción de tipo *RuntimeException*

2. *getTipoRueda()* : *TipoRueda* y *getDesgaste()* : *double* que devuelven el tipo de rueda y el desgaste actual de la rueda que recibe el mensaje
3. La representación de un objeto Rueda debe mostrar el tipo de rueda, así como el desgaste actual con un máximo de dos decimales, por ejemplo tras la creación de una rueda de tipo C1 su representación debería ser


C1 (100.00%)

4. *desgaste(double factorDesgasteCircuito, Random rnd)* : *void* que simula el desgaste de una rueda tras una vuelta a un circuito determinado. Este método realizará un desgaste de la rueda modificando su atributo desgaste según el siguiente decremento de desgaste:

$$\begin{aligned} \text{desgaste} = & \text{desgaste} \\ & * \text{factorDesgasteCircuito} \\ & * \text{factorDesgasteRueda} \\ & * (0.99 + (0.1 * \text{rnd.nextDouble()})) \end{aligned}$$

donde el factor de desgaste de la rueda dependerá del tipo de la rueda (C1,C2,...C5) implementado con las siguientes constantes:

Constantes	factorDesgasteRueda
FD_C1	0.9
FD_C2	0.92
FD_C3	0.94
FD_C4	0.96
FD_C5	0.97

 Motor
- <i>fabricante</i> : String - <i>probRotura</i> : double - <i>potencia</i> : int
+ <i>Motor</i> (String, int, double) + <i>getFabricante</i> (): String + <i>getPotencia</i> (): int + <i>getProbRotura</i> (): double + <i>desgaste</i> (Random): void + <i>toString</i> (): String

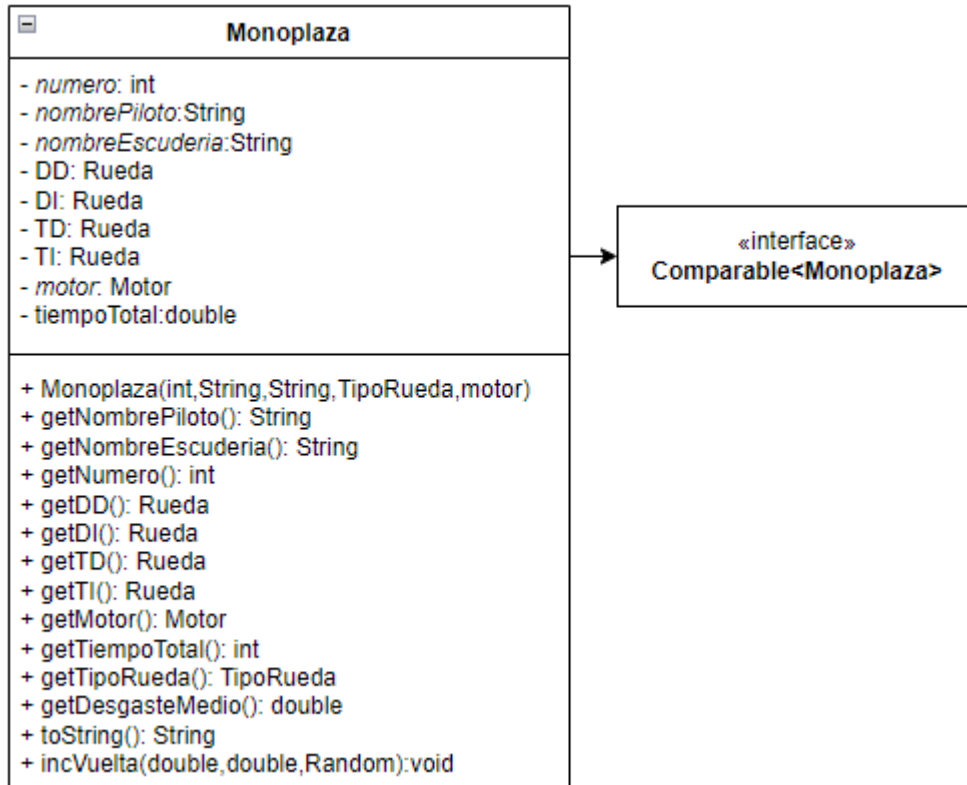
Motor: Clase que implementa un motor de formula 1. Almacena tres variables de instancia: *fabricante* (String), *probabilidad de rotura* (double), que almacena un valor entre 0 y 1 que indica la posibilidad de tener un perdida de potencia por cada vuelta que realiza durante la carrera y un valor *potencia* (int), que indica la potencia en caballos de ese motor. Todos los atributos son privados, de los cuales, el fabricante y la probabilidad de rotura son valores constantes una vez se instancien y dispondrán de los siguientes métodos publicos que han de ser implementados:

1. Constructor en el que se proporciona el nombre del fabricante (Ferrari, Mercedes, Renault, Ford, etc), la potencia del mismo como parámetros de entrada y la probRotura. Este constructor implementa la instanciación de un motor con ese fabricante, potencia y probabilidad. En caso de que el fabricante sea nulo/vacio, la potencia sea menor o igual que cero o la probabilidad de rotura no este en el intervalo [0..1] se deberá lanzar una excepción de tipo *RuntimeException*
2. *getFabricante()* : String, *getPotencia()* : int y *getProbRotura()* : double que devuelven el fabricante, la potencia y la probabilidad de rotura del objeto motor que recibe el mensaje
3. La representación de un objeto Motor debe mostrar la informacion del fabricante y la potencia del motor

Ferrari (600CV)

4. *desgaste(Random rnd)* que simula el desgaste de un motor tras una vuelta a un circuito determinado. Este método realizará un desgaste del motor modificando su atributo potencia segun en siguiente decremento de desgaste:

$$potencia = \begin{cases} potencia & rnd.nextDouble() < (1 - probRotura) \\ potencia * 0.9 & \text{en otro caso} \end{cases}$$



Monoplaza: Clase que se compone de un número de coche (int), un nombre de piloto (String), un nombre de escuderia (String), cuatro ruedas (DD,DI,TD,TI) de tipo Rueda y un motor (Motor). Además dispone de un contador de tiempo total (double) en segundos que inicialmente, cuando se crea una instancia, será cero.

1. La clase tendrá un constructor en el que se proporciona el número del coche, el nombre del piloto, el nombre de la escuderia, el tipo de rueda que va a colocar para competir en la carrera y el motor.
 - (a) Los parámetros de entrada no pueden ser nulos/vacios ni ≤ 0 , según el caso, lanzándose una excepción de tipo *RuntimeException* en caso contrario
2. Implementar
 - (a) `getNombrePiloto() : String`
 - (b) `getNombreEscuderia() : String`
 - (c) `getNumero() : int`
 - (d) `getDD() : Rueda, getDI() : Rueda, getTD() : Rueda, getTI() : Rueda`

- (e) *getMotor()* : *Motor*, *getTiempoTotal()* : *double*
que devuelven los valores de sus atributos correspondientes.
- Definir un orden natural de los monoplazas en función de su tiempo Total, y en caso de que este sea igual, su numero ordenara de menor a mayor según su número de coche
 - getTipoRueda()* : *TipoRueda* devuelve el tipo de la rueda de cualquiera de las que lleva puestas (deben ser todas iguales)
 - getDesgasteMedio()* : *double* devuelve la media aritmetica del desgaste de los cuatro neumaticos [0..1] donde 1 significa que estan al 100% y 0 que estan al 0%
 - La representación de un objeto Monoplazaza debe mostrar la informacion del automovil siguiendo el siguiente esquema
- 55 : Carlos Sainz (Ferrari/Ferrari (600CV)) C3(55.23%) 345.234 seg
- que representa al monoplaza con numero 55, pilotado por Carlos Sainz, de la escudería Ferrari con motor Ferrari (toString() de motor), con neumaticos C3 con un desgaste medio del 55,23%, y lleva 345,234 segundos de tiempo de carrera en el momento actual
- incVuelta(double tiempoBaseCircuito, double factorDesgasteCircuito, Random rnd)* : *void* simula una vuelta al circuito. Para ello hace uso de los parámetros:
 - tiempo Base Circuito: el tiempo base (en segundos) que se tarda en dar una vuelta al circuito. Sobre este tiempo se calcula el tiempo real para realizar una vuelta por el monoplaza en funcion de sus neumaticos, motor, desgaste, etc...
 - factor de desgaste del circuito: factor necesario para calcular el desgaste de los neumaticos en la siguiente vuelta
 - variable aleatoria para aleatorizar los tiempos
 El método calcula el nuevo tiempo total del monoplaza tras la simulación de la vuelta siguiendo la siguiente formula:

$$\begin{aligned}
 tiempoTotal = & \quad tiempoTotal \\
 & + tiempoBaseCircuito \\
 & + 5 * (1 - getDesgasteMedio()) \\
 & + ventajaTipoNeumatico \\
 & + 5 * ((700 - potencia)/700) \\
 & + 1 * rnd.nextDouble()
 \end{aligned}$$

donde ventajaTipoNeumatico será un valor int tal que:

Tipo de Neumático del monoplaza	ventajaTipoNeumatico
C1	1
C2	2
C3	3
C4	4
C5	5

Una vez calculado el tiempo total, el método actualiza el desgaste del motor y de las ruedas aplicando los métodos *desgaste(Random rnd)* de la clase motor y *desgaste(double factorDesgasteCircuito, Random rnd) : void* de la clase rueda para cada una de las ruedas (DD,DI,TD,TI)