Ejercicios-ADICIONALES-PARCIAL-1...

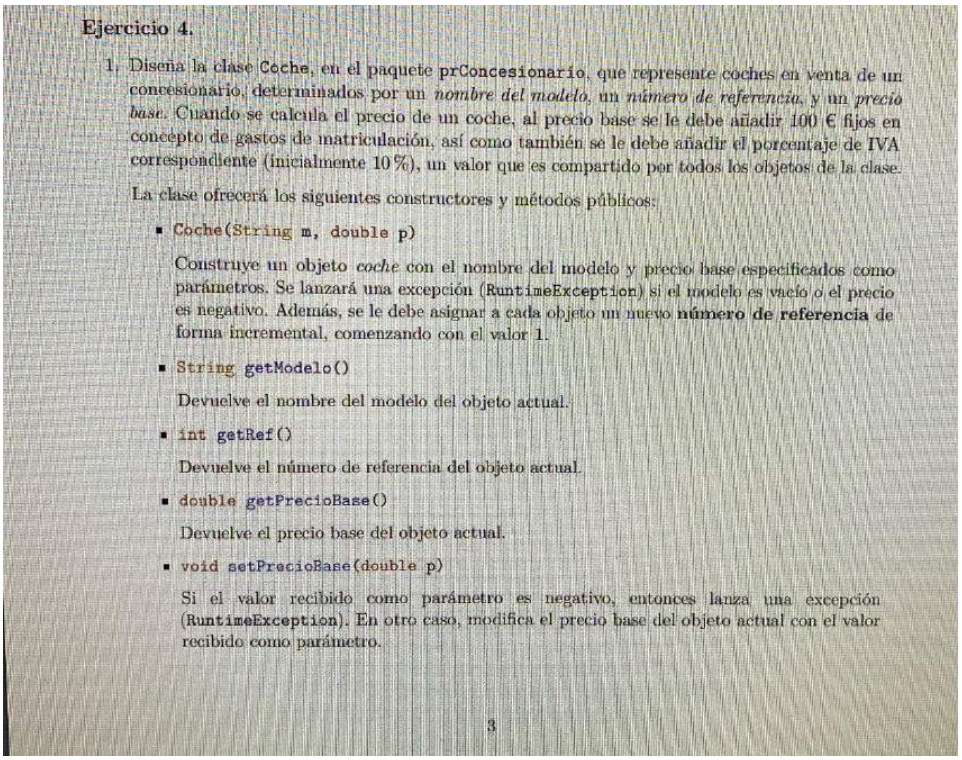
un_saltador

Programación Orientada a Objetos

1º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

ENUNCIADO DE LOS EJERCICIOS ADICIONALES (TEMA 2.2: ARRAYS Y LISTAS)



Ejercicio 4.

1. Diseña la clase `Coche`, en el paquete `prConcesionario`, que represente coches en venta de un concesionario, determinados por un *nombre del modelo*, un *número de referencia*, y un *precio base*. Cuando se calcula el precio de un coche, al precio base se le debe añadir 100 € fijos en concepto de gastos de matriculación, así como también se le debe añadir el porcentaje de IVA correspondiente (inicialmente 10 %), un valor que es compartido por todos los objetos de la clase.

   La clase ofrecerá los siguientes constructores y métodos públicos:

   - `Coche(String m, double p)`

     Construye un objeto *coche* con el nombre del modelo y precio base especificados como parámetros. Se lanzará una excepción (`RuntimeException`) si el modelo es vacío o el precio es negativo. Además, se le debe asignar a cada objeto un nuevo **número de referencia** de forma incremental, comenzando con el valor 1.

   - `String getModelo()`

     Devuelve el nombre del modelo del objeto actual.

   - `int getRef()`

     Devuelve el número de referencia del objeto actual.

   - `double getPrecioBase()`

     Devuelve el precio base del objeto actual.

   - `void setPrecioBase(double p)`

     Si el valor recibido como parámetro es negativo, entonces lanza una excepción (`RuntimeException`). En otro caso, modifica el precio base del objeto actual con el valor recibido como parámetro.

3

- `static double getGastosMatriculacion()`

  Devuelve el valor de los gastos de matriculación asociado a la clase (100 €).

- `static double getIva()`

  Devuelve el porcentaje de IVA asociado a la clase.

- `static void setIva(double d)`

  Modifica el porcentaje de IVA asociado a la clase con el valor recibido como parámetro. El valor del porcentaje de IVA asociado a la clase será inicialmente 10 %.

- `double calcPrecioFinal()`

  Calcula y devuelve el precio final del objeto actual. Para ello, añade al precio base el valor de gastos de matriculación, además, se le debe añadir el porcentaje de IVA correspondiente.

- `String toString()` *// @Redefinición*

  Devuelve la representación textual (número de referencia, modelo, y el precio final) del objeto actual, según el formato del siguiente ejemplo: `(123, Ferrari TestaRosa, 550110.0)`

2. Diseña una clase distinguida `PruebaCoches` que permita probar la clase `Coche`.

Ejercicio 5.

1. Defina la clase `Concesionario`, en el paquete `prConcesionario`, que represente un *concesionario* de coches, de tal forma que pueda contener múltiples coches de diversas clases, todas ellas derivadas de la clase `Coche` definida en ejercicios anteriores.

   La clase ofrecerá los siguientes constructores y métodos públicos:

   - `Concesionario()`

     Crea un concesionario vacío (sin coches almacenados) con una capacidad inicial de almacenamiento de tamaño `TAM` (una constante de clase `private` con valor 10).

   - `Concesionario(int)`

     Crea un concesionario vacío (sin coches almacenados) con una capacidad inicial de almacenamiento de tamaño igual al valor recibido como parámetro. Se lanzará una excepción (`RuntimeException`) si el tamaño es menor o igual a cero.

   - `void anyadirCoche(Coche)`

     Si ya existe un coche con el mismo nombre de modelo que el nuevo coche a añadir, entonces el antiguo coche sera reemplazado por el nuevo coche. En otro caso, Añade un coche al concesionario actual, considerando que si el array está lleno, entonces se doblará su capacidad y se añadirá el nuevo coche.

   - `void eliminarCoche(String)`

     Si ya existe un coche con el nombre de modelo recibido como parámetro, entonces elimina del concesionario ese coche. En otro caso, lanzará una excepción (`RuntimeException`).

   - `Coche cocheMasBarato()`

     Si no existe ningún coche en el concesionario, entonces lanza una excepción (`RuntimeException`). En otro caso, busca y devuelve el coche cuyo precio final es el más barato del concesionario.

   - `String toString()`

Devuelve la representación textual del concesionario. La representación consistirá en la secuencia de representaciones textuales de los coches que forman la colección, separados por comas y encerrados entre corchetes. P.ej:
`[ (1, seat, 11110.0), (2, renault, 13310.0), (3, peugeot, 14410.0) ]`

2. Defina la clase distinguida `PruebaConcesionario` que permita comprobar la clase `Concesionario` y realice diversas operaciones con un concesionario de coches.

Ejercicio 6.

1. Defina el **tipo enumerado** `Color`, en el paquete `prConcesionario`, con la siguiente enumeración de valores: `Negro, Rojo, Verde, Azul, Amarillo, Magenta, Cian, Blanco`.

   Nota: los tipos enumerados proporcionan automáticamente los siguientes métodos de clase: `valueOf(str)`, `values()`, y los siguientes métodos de instancia: `toString()`, `ordinal()`, `equals(o)`, `hashCode()`, `compareTo(v)`.

2. Defina la clase `CocheColor`, en el paquete `prConcesionario`, como un `Coche` con un *color* añadido, pero donde su precio base se calcula considerando el color del mismo.

   La clase ofrecerá los siguientes constructores y métodos públicos:

   - `CocheColor(String m, double p, String cn)`

     Construye un objeto coche con modelo y precio base especificados como parámetros. Se lanzará una excepción (`RuntimeException`) si el modelo es vacío, el precio es negativo, o el nombre del color es erróneo. Además, almacena el valor de la enumeración Color correspondiente al nombre del color recibido como tercer parámetro.

   - `CocheColor(String m, double p, Color col)`

     Construye un objeto coche con modelo y precio base especificados como parámetros. Se lanzará una excepción (`RuntimeException`) si el modelo es vacío o el precio es negativo. Además, almacena el valor de la enumeración Color recibido como tercer parámetro.

   - `Color getColor()`

     Devuelve el nombre del color del objeto actual.

   - `void setColor(Color c)`

     Modifica el color del objeto actual al nuevo valor recibido como parámetro.

   - `void setColor(String c)`

     Modifica el color del objeto actual al nuevo valor recibido como parámetro. Si el valor del parámetro es erróneo, entonces lanza una excepción (`RuntimeException`).

   - `String getModelo()` *// @Redefinición*

     Devuelve el nombre del modelo del objeto actual, considerando que el nombre del modelo de un coche con color es el nombre del modelo unido con el nombre del color (separados por un espacio en blanco).

   - `double getPrecioBase()` *// @Redefinición*

     Calcula y devuelve el precio base del objeto actual, considerando que el precio base de un coche con color es el valor del precio base incrementado con el precio por el color del mismo, considerando la siguiente lista de precios:

Negro	Rojo	Verde	Azul	Amarillo	Magenta	Cian	Blanco
+0	+10	+20	+30	+40	+50	+60	+70

   - `String toString()` *// @Redefinición*

     Devuelve la representación textual (referencia, modelo y precio) del objeto actual, considerando que ahora, el nombre del modelo debe llevar asociado el color del coche, y que el precio final tiene en cuenta el incremento del precio base según el color del coche.

   - `static String[] coloresDisponibles()`

     Devuelve un array con los nombres de los colores disponibles para la creación de coches.

3. Diseña una clase distinguida `PruebaConcesionarioColor` de forma similar a `PruebaConcesionario`, pero que ahora también utilice coches con color.