Proyecto prGasolinera (Leer primero las NORMAS para el Examen)

Se desea desarrollar una aplicación para que los clientes de una gasolinera no tengan que pagar en el momento de repostar el combustible. En su lugar, la aplicación generará y almacenará un ticket que será facturado posteriormente cuando el cliente lo solicite. **Se deben crear** las clases `GasolineraException`, `Ticket`, `TicketOrdenAlternativo`, `Gasolinera`, `TicketPromocion` y `GasolineraPromocion` en el paquete `prGasolinera`. También **se debe crear**, en el paquete "por defecto", la clase `PruebaTicket`. Todas estas clases aparecen en el ***diagrama de clases*** suministrado en el campus virtual. Por otro lado, en el campus virtual **se facilitan** las clases `Main1` y `Main2`, que deben guardarse en el paquete "por defecto". Además**, se suministra** el fichero de texto `surtidores.txt`. Este fichero debe almacenarse en la carpeta del proyecto `prGasolinera`. Por último, también **se suministran** dos ficheros: `salidaConsolaMain1.txt` y `salidaConsolaMain2.txt` (con la salida de las clases `Main1` y `Main2`). A continuación, se describen cada una de las clases pedidas:

1) (**0.25 *ptos.***) La clase `GasolineraException` se utilizará para tratar las diferentes situaciones excepcionales. Se trata de una excepción *no comprobada*. En general, se lanzará esta excepción si los parámetros de entrada de los métodos no contienen valores adecuados.

2) (**0.75 *ptos.***) La clase `Ticket` mantendrá información de un ticket. En concreto, el número de ticket (`int`), el nombre de la gasolinera (`String`), la matrícula del vehículo (`String`), el número de litros que se ha repostado (`double`), el precio del litro (`double`) y un indicador para saber si el ticket ha sido facturado o aún no (`boolean`). Se definirán los constructores y los métodos necesarios para:
   a. Construir un objeto de la clase dados los valores de los cinco primeros datos descritos anteriormente. Para el último dato, un ticket siempre se creará 'no facturado' (valor inicial `false`). Si el precio o el número de litros son cero o negativos, se debe lanzar una excepción del tipo `GasolineraException` indicando que no se puede crear el ticket. También se lanzará la excepción si el número de matrícula o el nombre de la estación son `null` o de longitud cero.
   b. Obtener el número de ticket (`int getNumTicket()`), el número de litros repostados (`double getNumLitros()`), el nombre de la estación (`String getGasolinera()`) y si el ticket ha sido facturado o no (`boolean getFacturado()`). Modificar si el ticket ha sido facturado o no (`void setFacturado(boolean)`). Calcular el precio total (`double precioTotal()`).
   c. Dos objetos de la clase `Ticket` son iguales si coinciden los nombres de la gasolinera (ignorando mayúsculas y minúsculas) y sus números de ticket.
   d. El *orden natural* de dos objetos de la clase `Ticket` se define en primer lugar en función del orden natural de los nombres de las gasolineras (ignorando mayúsculas y minúsculas). En caso de igualdad, se considera menor el ticket con un número de ticket menor.
   e. La representación de un ticket viene dada por una cadena de caracteres con el siguiente formato, donde el precio mostrado se corresponde con el precio total del ticket (representa un ticket con número de ticket 100, de la gasolinera con nombre Teatinos, para un vehículo con matrícula 1111AAA, que reposta 20 litros a un precio de 1.47 euros por litro, con un precio total de 29.4):

      ```
      Ticket: 100 (gasolinera: Teatinos, matricula: 1111AAA, litros: 20.0, PRECIO = 29.4)
      ```

3) (***0.25 ptos.***) La clase `TicketOrdenAlternativo` definirá un *orden alternativo* para la clase `Ticket`, donde los objetos de la clase `Ticket` se ordenarán primero por su número de ticket (siendo menor el ticket con un número de ticket mayor). Para dos tickets con el mismo número de ticket, se aplicará el orden natural de los nombres de las gasolineras (sin distinguir mayúsculas y minúsculas).

4) (***0.75 ptos.***) Crea una aplicación, `PruebaTicket`, para probar las clases anteriores. Se debe capturar y tratar las posibles excepciones que se lancen. En esta aplicación se crearán cuatro objetos de la clase `Ticket`. El primero de la gasolinera "Teatinos", con número de ticket 1, matrícula "1111aaa", donde se han repostado 50 litros a un precio de 1.40 euros. El segundo de la gasolinera "TEATINOS",

con número de ticket 1, matrícula "1111AAA", donde se han repostado 45 litros a un precio de 2.40 euros. El tercero de la gasolinera "Teatinos", con número de ticket 2, matrícula "2222BBB", donde se han repostado 50 litros a un precio de 1.40 euros. El cuarto de la gasolinera "Ampliacion", con número de ticket 1, matrícula "3333CCC", donde se han repostado 40 litros a un precio de 1.30 euros. Después se mostrarán los datos de los dos primeros objetos por pantalla y se comprobará si el objeto primero y segundo son iguales o no, indicándolo también por pantalla. Después se creará un conjunto ordenado de objetos de la clase `Ticket` (la ordenación seguirá el *orden natural*), al que se añadirán los cuatro tickets anteriores y se mostrará por pantalla el contenido de este conjunto. A continuación se creará un nuevo conjunto ordenado de objetos de la clase `Ticket` (la ordenación seguirá en este caso el *orden alternativo* definido en 3) al que se añadirán los cuatro tickets anteriores y se mostrará por pantalla el contenido de este conjunto. Finalmente, se creará un quinto ticket con número de litros -10, eligiendo el alumno el valor del resto de datos. La salida por pantalla será la siguiente, donde el mensaje de error puede aparecer en una posición diferente:

```
Ticket: 1 (gasolinera:Teatinos, matricula:1111aaa, litros:50.0, PRECIO = 70.0)
Ticket: 1 (gasolinera:TEATINOS, matricula:1111AAA, litros:45.0, PRECIO = 108.0)
Son iguales

Tickets ordenados por orden natural:
[Ticket: 1 (gasolinera:Ampliacion, matricula:3333CCC, litros:40.0, PRECIO = 52.0), Ticket:
1 (gasolinera:Teatinos, matricula:1111aaa, litros:50.0, PRECIO = 70.0), Ticket: 2
(gasolinera:Teatinos, matricula:2222BBB, litros:50.0, PRECIO = 70.0)]

Tickets ordenados por orden alternativo:
[Ticket: 2 (gasolinera:Teatinos, matricula:2222BBB, litros:50.0, PRECIO = 70.0), Ticket: 1
(gasolinera:Ampliacion, matricula:3333CCC, litros:40.0, PRECIO = 52.0), Ticket: 1
(gasolinera:Teatinos, matricula:1111aaa, litros:50.0, PRECIO = 70.0)]
ERROR: Valores incorrectos para crear un ticket
```

5) (***6 ptos.***) La clase `Gasolinera` almacenará la información relativa a una determinada gasolinera. En concreto, su nombre (`String`), un contador (`int`) con el número del siguiente ticket a emitir, una asociación o correspondencia `surtidores` que asocia a cada tipo de combustible una lista con el contenido de cada uno de los surtidores que contienen ese tipo de combustible (`Map<String,List<Double>>`), una asociación o correspondencia `repostajes` que asocia a una matrícula de vehículo un conjunto ordenado de objetos de tipo `Ticket` (`Map<String,SortedSet<Ticket>>`) y un objeto de tipo `TicketOrdenAlternativo` que almacenará un orden alternativo para los objetos de tipo `Ticket`. El nombre de la gasolinera y el contador de tickets deben ser *protected*. La clase definirá también cuatro constantes de clase de tipo `String` que almacenarán los cuatro tipos de combustibles disponibles "`gasolina95`", "`gasolina98`", "`diesel`" y "`dieselPlus`". El nombre de cada constante coincide con el valor que almacenan pero con todas las letras en mayúsculas. Por último, definirá la constante de instancia `precios` (`Map<String,Double>`), que para cada tipo de combustible lo asocia con el precio por litro de ese combustible (esta constante se inicializará con la información proporcionada en el constructor de la clase), y la constante de clase `NUM_SURT` (`int`), con el número de surtidores de cada tipo. Habrá 4 surtidores de cada tipo. Se definirán los constructores y métodos de instancia públicos necesarios para:

a. Construir un objeto de la clase mediante dos constructores distintos. En el primero, se construye el objeto dados el nombre de la gasolinera, la asociación o correspondencia con los precios, el nombre de un fichero de datos y un orden alternativo para los tickets. En el segundo constructor se proporcionan los mismos parámetros del primero excepto el orden alternativo. En su lugar el orden alternativo se inicializará a `null`, indicando que se debe utilizar el orden natural para ordenar los tickets. El resto de la descripción es común a los dos constructores: El contador con el número del siguiente ticket a emitir se inicializará a 1. La correspondencia o asociación con los repostajes se crea vacía y se deja así. Por el contrario, la correspondencia o asociación con los surtidores se creará de la siguiente forma: (1) primero se crea la correspondencia vacía; (2) luego se crean los surtidores añadiendo a la correspondencia tantas entradas como tipos de combustible hay. Para cada entrada la clave es uno de los tipos de combustible y el valor asociado a esa entrada será una lista de tamaño `NUM_SURT`, con todas sus posiciones inicializadas a `0.0` (es decir, inicialmente habrá `NUM_SURT` de

cada tipo, todos con un contenido de `0.0`); (3) se completa la información de los surtidores con los datos del fichero de texto proporcionado como parámetro, cuyo formato es el del fichero suministrado como ejemplo en el campus virtual (`surtidores.txt`). Cada línea del fichero tiene la información de un surtidor: número de surtidor, tipo de combustible y litros en el surtidor. Todos estos datos están separados por un espacio en blanco ' '. Un ejemplo sería: `1 gasolina95 56.0`[1]. Con los datos leídos en cada línea del fichero se actualizará la asociación `surtidores` de la siguiente forma: en la lista de valores de tipo real asociada al tipo de combustible leído en esa línea del fichero (`gasolina95` en el ejemplo), se incrementará el valor almacenado en la posición indicada por el número de surtidor leído del fichero (valor `1` en el ejemplo) en el número de litros leído (valor `56.0` en el ejemplo). Hay que tener en cuenta que la posición 0 de la lista representa el primer surtidor, la posición 1 el segundo surtidor, etc. y que los valores almacenados en cada posición de la lista representan el contenido en litros de cada surtidor). Si varias líneas del fichero hacen referencia al mismo surtidor (es decir, mismo tipo de combustible y mismo número de surtidor), se sumará el nuevo contenido al ya existente (se considerará la capacidad del surtidor ilimitada).

   Si el fichero no puede abrirse, se debe lanzar una excepción del tipo `GasolineraException`, informando de ello. Si algún dato es incorrecto (el número de surtidor no es correcto, el tipo de combustible no es correcto o el número de litros es un número negativo) esa entrada se descartará y se seguirá leyendo el resto del fichero. *AYUDA: Definir los métodos auxiliares que se consideren necesarios. Una sugerencia es definir métodos auxiliares para crear e inicializar los surtidores, para leer los datos del fichero y para añadir los datos de un surtidor a la asociación* `surtidores`*.*

b. Definir el método `void repostar(String, String, int, double)` que le permita a un vehículo con la matrícula indicada como primer parámetro repostar combustible del tipo indicado como segundo parámetro y del surtidor indicado como tercer parámetro. La cantidad a repostar se proporciona como último parámetro. Si el número de surtidor o el tipo de combustible es incorrecto o la cantidad es cero o negativa se lanzará una excepción del tipo `GasolineraException` indicando el tipo de error. En otro caso se repostará la cantidad solicitada actualizando la asociación `surtidores` de forma apropiada. Si la cantidad disponible en el surtidor fuera menor que la cantidad solicitada se repostará solo la cantidad que está disponible. Si la cantidad en el surtidor fuera cero no se hace nada. Por último, si se ha podido repostar se generará un ticket que se almacenará en la asociación `repostajes`, asociándolo a la matrícula del vehículo (es decir, añadiéndolo al conjunto ordenado de tickets asociado a esa matrícula). Al crear el conjunto de tickets se utilizará el orden natural o el orden alternativo, según el constructor con el que se haya creado el objeto en a). Habrá que tener en cuenta que podría ser la primera vez que ese vehículo reposte en la gasolinera, por lo que habría que añadir una nueva entrada a la asociación. También habrá que tener en cuenta que si ya existe ese ticket en el conjunto de tickets asociado a ese vehículo se lanzará una excepción de tipo `GasolineraException`. Para crear el ticket se debe utilizar el método definido en c).

c. El método `protected Ticket crearTicket(String, double, double)` crea y devuelve un nuevo ticket dada la matrícula indicada como primer parámetro y la cantidad y el precio por litro indicados como segundo y tercer parámetros respectivamente. El número de ticket viene dado por el atributo que contiene el siguiente número de ticket, que deberá ser actualizado apropiadamente.

d. El método `void facturar (String)` factura todos los tickets pendientes de facturar (aquellos cuyo atributo `facturado` de la clase `Ticket` es `false`) del vehículo cuya matrícula se indica como parámetro. Facturar significa crear un fichero de texto cuyo nombre será una combinación del nombre de la gasolinera y de la matrícula del vehículo para el que se está generando la factura separados por un guión bajo (ej. Teatinos_1111AAA.txt"). Este fichero contendrá una línea por cada ticket pendiente de facturar, más una línea adicional con el total. Un ejemplo del formato de este fichero (que se llamaría Teatinos_1111AAA.txt) sería:

```
Ticket: 1 (gasolinera:Teatinos, matricula:1111AAA, litros:20.0, PRECIO = 25.6)
Ticket: 2 (gasolinera:Teatinos, matricula:1111AAA, litros:60.0, PRECIO = 76.8)
Ticket: 3 (gasolinera:Teatinos, matricula:1111AAA, litros:60.0, PRECIO = 76.8)
TOTAL = 179.2
```

[1] Usar `sc.useLocale(Locale.ENGLISH)` para leer los números reales en el formato solicitado (`sc` es de tipo Scanner)

Además de generar este fichero, este método actualiza los tickets facturados para indicar que ya no están pendientes de facturar.

e. El método `double obtenerConsumoFacturado(String)` calcula y devuelve la cantidad total ya facturada para el vehículo cuya matrícula se indica como parámetro.

f. La representación de los objetos de esta clase muestra toda la información contenida en la misma en el formato indicado. Observad que en los repostajes solo se muestran los "valores" de la asociación o correspondencia y no las "claves", por lo que habrá que iterar sobre la asociación o correspondencia de la forma apropiada para conseguir esta salida. Se debe utilizar *StringBuilder* o *StringJoiner*.

```
<nombre_gasolinera> =
    Gasolina95: [<cant1>, <cant2>, <cant3>, <cant4>]
    Gasolina98: [<cant1>, <cant2>, <cant3>, <cant4>]
    Diesel:     [<cant1>, <cant2>, <cant3>, <cant4>]
    DieselPlus: [<cant1>, <cant2>, <cant3>, <cant4>]
    Repostajes: [[<ticket1>,<ticket2>],[<ticket3>,<ticket4>]]
```

6) Utiliza la clase distinguida `Main1` proporcionada para probar todas las clases anteriores. La salida por pantalla debe ser igual al contenido del fichero `salidaConsolaMain1.txt` proporcionado (para visualizar este fichero se recomienda abrirlo con el propio editor de Eclipse). Por cada llamada al método `facturar` de la gasolinera se debe generar un fichero, en este caso serían 4 ficheros: `Ampliacion_1111AAA.txt`, `Ampliacion_2222BBB.txt`, `Teatinos_1111AAA.txt` y `Teatinos_2222BBB.txt`

7) (***0.75 ptos.***) La clase `TicketPromocion` *se comporta como la clase* `Ticket`, con la diferencia de que en este caso se aplica un descuento a la hora de calcular el precio total del ticket, almacenado en la variable de instancia `descuento` como un número real (ej. un 10% de descuento se almacenará como 0.1). La clase dispondrá de un constructor y los siguientes métodos:

a. El constructor `TicketPromocion(int, String, String, double, double, double)` que recibe como parámetros el número de ticket, el nombre de la gasolinera, el número de matrícula, el número de litros, el precio por litro y el descuento, en este orden.

b. El método `void precioTotal()` que será una redefinición del método correspondiente de la clase `Ticket`. En este caso, el precio total se calculará aplicando el descuento indicado en `descuento`.

c. El método `String toString()` será una redefinición del método correspondiente de la clase `Ticket` para representar un ticket mediante la misma cadena generada por el método redefinido pero anteponiendo la cadena "`PROMOCION <descuento>%:`", como en el siguiente ejemplo:

```
PROMOCION 20%: Ticket: 1 (gasolinera:Teatinos, matricula:1111aaa, litros:50.0, PRECIO = 70.0)
```

8) (**1.25 ptos.**) La clase `GasolineraPromocion` *se comporta como la clase* `Gasolinera`, con la diferencia de que a la hora de generar los tickets se aplicarán distintos descuentos dependiendo del consumo previamente facturado. La clase define cuatro constantes de clase: `CONSUMO_MINIMO1` con un valor de 100, `CONSUMO_MINIMO2` con un valor de 300, `DESCUENTO1` con un valor de 0.10 y `DESCUENTO2` con un valor de 0.30. Se definirán los siguientes constructores y métodos:

a. Dos constructores con los mismos parámetros y el mismo comportamiento que los constructores definidos para la clase `Gasolinera`.

b. El método protegido `Ticket crearTicket(String, double, double)` que será una redefinición del método correspondiente de la clase `Ticket`. En este caso, a la hora de crear un nuevo ticket se tendrá en cuenta lo siguiente: (1) se calculará el consumo total previamente facturado para la matrícula indicada como primer parámetro (método e) del apartado 5)); (2) Si el consumo facturado es inferior a `CONSUMO_MINIMO1` no se aplicará ningún descuento y se creará un objeto de tipo Ticket. Por el contrario, si el consumo acumulado se encuentra en el rango [`CONSUMO_MINIMO1`, `CONSUMO_MINIMO2`) se aplicará el descuento `DESCUENTO1`. Si el consumo es mayor o igual que `CONSUMO_MINIMO2` entonces se aplicará el descuento `DESCUENTO2`. En estos dos casos se creará un ticket de tipo `TicketPromocion`.

9) Utiliza la clase distinguida `Main2` proporcionada para probar las clases anteriores. La salida por pantalla debe ser igual al contenido del fichero `salidaConsolaMain2.txt` proporcionado. Se generarán además los ficheros `TeatinosPROMO_1111AAA.txt` y `TeatinosPROMO_2222BBB.txt`.