DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN	**PROGRAMACIÓN ORIENTADA A OBJETOS** (Prueba realizada el 10 de Junio de 2019)	
APELLIDOS, Nombre	TITULACIÓN Y GRUPO	
	MÁQUINA	

Proyecto prPatinetes (Leer primero las NORMAS para el Examen)

Se desea crear una aplicación para gestionar la información relacionada con empresas de alquiler de patinetes eléctricos distribuidos en una determinada ciudad. **Se han de crear**, dentro de un paquete denominado `prPatinetes`, las clases `PatinetesException`, `Patinete`, `Empresa`, `EmpresaSeleccion`, `SeleccionAutonomia`, `SeleccionPosicion` y la interfaz `Seleccion`. También **se debe crear**, en el paquete "por defecto" del proyecto, la clase denominada `PruebaPatinete` para la prueba de algunas de las clases anteriores. Todas estas clases e interfaces a desarrollar aparecen en el ***diagrama de clases*** que se suministra en el campus virtual.

Por otro lado, **se proporcionan** en el campus virtual las clases `Posicion` y `Empleado`, que deben guardarse en el paquete `prPatinetes`. También **se facilitan** `Main1` y `Main2` para realizar pruebas de las clases, que deben guardarse en el paquete "por defecto" del proyecto. Además**, se suministra** el fichero `patinetes.txt` con los datos necesarios para realizar alguna de las operaciones que se indican más adelante. Este fichero debe almacenarse directamente en la carpeta del proyecto `prPatinetes`. Por último, también **se suministran** dos ficheros: `salidaConsolaMain1.txt` (con los datos mostrados por pantalla al ejecutar la clase `Main1`) y `salidaConsolaMain2.txt` (con los datos mostrados por pantalla al ejecutar la clase `Main2`). A continuación, se describen cada una de las clases pedidas:

1) (**0.25 *ptos.***) La clase `PatinetesException` se utilizará para tratar las diferentes situaciones excepcionales. Se trata de una excepción *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 `Patinete` mantendrá información sobre un patinete eléctrico. En concreto, el nombre de su empresa (`String`), su código (`int`), su posición (`Posicion`) y su autonomía disponible en kilómetros (`double`).

Nota: La clase `Posicion` se suministra en el campus virtual y sirve para almacenar una determinada posición, dadas su latitud y su longitud, las cuales se pueden consultar y modificar. Además, esta clase también dispone de un método para calcular la distancia entre dos posiciones y otro para establecer la representación como cadena de caracteres de una posición (consultar el código suministrado).

La clase `Patinete` dispondrá de constructores y métodos de instancia públicos necesarios para:

a. Construir un objeto de la clase dados los valores para los cuatro datos descritos anteriormente. Si el código o su autonomía son negativos, se debe lanzar una excepción del tipo `PatinetesException`, informando en cada caso del error correspondiente.
b. Construir un objeto de la clase dados solamente el código, el nombre de la empresa y la posición. En este caso, la autonomía en kilómetros se inicializará a 0.
c. Obtener el nombre de la empresa (`String getNombreEmpresa()`), el código (`int getCodigo()`), la posicion (`Posicion getPosicion()`) y la autonomía (`double getAutonomia()`). Modificar la posición y la autonomía (`void setPosicion(Posicion)`, `void setAutonomia(double)`). En este último método hay que controlar el mismo tipo de error que se detalló en el apartado a.
d. Dos objetos de la clase `Patinete` son iguales si coinciden los nombres de sus empresas (ignorando mayúsculas y minúsculas) y sus códigos.

e. El *orden natural* de dos objetos de la clase `Patinete` se define de la siguiente forma: se aplica el orden natural de los nombres de sus empresas (ignorando mayúsculas y minúsculas); en caso de igualdad, se considera menor el patinete con un código menor.
f. La representación de un patinete eléctrico viene dada por una cadena de caracteres con el siguiente formato (un ejemplo para un patinete de la empresa cuyo nombre es Campero, con código 100, con una posición con latitud -4.4214862 y longitud 36.7179875 y una autonomía de 2 kilómetros):

```
(Empresa: Campero; Codigo: 100; Lat:  -4.4214862; Lon: 36.7179875; Autonomia: 2)
```

3) (***0.75 ptos.***) Crea una aplicación, `PruebaPatinete`, para probar las clases anteriores. Ten en cuenta que debes capturar y tratar (mostrar el correspondiente mensaje de error por pantalla) las posibles excepciones que se lancen (`PatinetesException`). En esta aplicación se crearán tres objetos de la clase `Patinete`. El primero de la empresa "Campero", con código 100, en la posición (latitud: -4.4204216, longitud: 36.7182771) y con una autonomía de 3 kilómetros. El segundo de la empresa "campero", con código 100, en la posición (latitud: -4.4495993, longitud: 36.7015323) y con una autonomía de 2 kilómetros. El tercero de la empresa "Biznaga", con código 101, en la posición (latitud: -4.4150382, longitud: 36.7306184) y con una autonomía de 5 kilómetros. Después se mostrarán los datos de los tres objetos por pantalla. A continuación, se comprobará si el objeto primero y segundo son iguales o no, indicándolo por pantalla. Después se creará un conjunto ordenado vacío de objetos de la clase `Patinete` (la ordenación seguirá el *orden natural* de los objetos), al que se añadirán los tres patinetes anteriores. Posteriormente, se mostrará por pantalla el contenido de este conjunto. Finalmente, se modificará la autonomía del tercer patinete dándole un valor de -7 kilómetros. La ejecución de la aplicación producirá la siguiente salida por pantalla (el mensaje de error puede aparecer en una posición diferente):

```
Patinete 1: (Empresa: Campero; Codigo: 100; Lat: -4.4204216; Lon: 36.7182771; Autonomia: 3.0)
Patinete 2: (Empresa: campero; Codigo: 100; Lat: -4.4495993; Lon: 36.7015323; Autonomia: 2.0)
Patinete 3: (Empresa: Biznaga; Codigo: 101; Lat: -4.4150382; Lon: 36.7306184; Autonomia: 5.0)
Los patinetes 1 y 2 son iguales
Conjunto: [(Empresa: Biznaga; Codigo: 101; Lat: -4.4150382; Lon: 36.7306184; Autonomia: 5.0),
(Empresa: Campero; Codigo: 100; Lat: -4.4204216; Lon: 36.7182771; Autonomia: 3.0)]
ERROR: Valor negativo al modificar la autonomía
```

4) (***6 ptos.***) La clase `Empresa` almacenará la información relativa a una determinada empresa. En concreto, el nombre de la empresa (`String`), un conjunto ordenado de patinetes pertenecientes a la misma (`SortedSet<Patinete>`), una lista de errores producidos en la lectura de datos desde fichero (`List<String>`) y una asociación o correspondencia que asocia cada empleado de la empresa con el conjunto ordenado de códigos de patinetes asignados a dicho empleado (`Map<Empleado, SortedSet<Integer>`). Los conjuntos estarán en todos los casos ordenados atendiendo al *orden natural* de los objetos **.**

Nota: La clase `Empleado` se suministra en el campus virtual y sirve para almacenar información sobre un empleado de alguna empresa de patinetes eléctricos. En concreto, almacenará el dni y la posición del empleado. Además, esta clase también dispone de métodos típicos para obtener y modificar datos, comprobar la igualdad entre dos empleados y representar como cadena de caracteres la información de un empleado (consultar el código suministrado).

La clase `Empresa` dispondrá de un constructor y varios métodos de instancia públicos necesarios para:

a. Construir un objeto de la clase dados el nombre de la empresa y el nombre de un fichero de datos. La correspondencia o asociación se crea vacía y se deja así. Por el contrario, el conjunto de patinetes y la lista de errores se crean vacíos inicialmente, pero se irán rellenando conforme se lean datos del fichero. El formato de los datos se puede observar en el fichero suministrado como ejemplo en el campus virtual (`patinetes.txt`). En cada línea del fichero aparece la información de un patinete:

el nombre de la empresa, el código, la latitud, la longitud y la autonomía). Todos estos datos están separados por el carácter ';'. Un ejemplo podría ser:

```
Campero;100;-4.4204216;36.7182771;3.0
```

Si el fichero no puede abrirse, se debe lanzar una excepción del tipo `PatinetesException`, informando de ello.
Si el nombre de empresa leído no coincide con el nombre de la empresa que se está creando (ignorando mayúsculas y minúsculas), esa línea no se tiene en cuenta. En otro caso, si se produce algún error de formato en una línea (falta algún dato, el dato no es del tipo correcto o el código o la autonomía son negativos), se creará una cadena de caracteres compuesta por "ERROR: ", el tipo de error, "EN LINEA: " y la línea correspondiente. Esta cadena se almacenará en la lista de errores. Si todo es correcto, se creará un objeto de la clase `Patinete` y se almacenará en el conjunto de patinetes de la empresa.

b. El método `void asignaPatinetesEmpleado(Empleado, SortedSet<Integer>)` en primer lugar crea un nuevo conjunto (`SortedSet<Integer>`) con los códigos de los patinetes recibidos como segundo parámetro que no estén ya en la correspondencia (asignados al mismo o distinto empleado) y se correspondan con algún patinete almacenado en el conjunto ordenado de patinetes (para esto último se puede utilizar la operación del apartado c). Después, si el empleado no está todavía en la correspondencia, asigna al empleado ese nuevo conjunto creado. En caso de que el empleado ya estuviese en la correspondencia, se añadirán los códigos de patinetes del nuevo conjunto creado a los que ese empleado ya tuviera previamente asignados.
c. El método `Patinete buscaPatinete(int)` devuelve el patinete almacenado en el conjunto de patinetes cuyo código se recibe como parámetro o `null` si no está.
d. La representación de los objetos de esta clase muestra toda la información contenida en la misma: nombre de la empresa, conjunto de patinetes, lista de errores y la asignación de códigos de patinetes a los empleados. El formato que se ha de utilizar es el que se indica en el siguiente esquema (se pueden incluir saltos de línea donde se considere oportuno para una mejor legibilidad). Se debe utilizar *StringBuilder* o *StringJoiner*.

```
NombreEmpresa
Patinetes: [patinete1,patinete2,…]
Errores: [error1, error2,…]
Empleados: {dni1:[codigo1,codigo2,…], dni2: [codigo1,codigo2,…],…}
```

e. El método `void modificaPatinete(int,Posicion,double)` modifica la posición y la autonomía del patinete almacenado en el conjunto de patinetes cuyo código coincida con el que se recibe como primer parámetro, con la posición y autonomía pasadas como segundo y tercer parámetro. Si el patinete no está almacenado, se lanzará una excepción del tipo `PatinetesException` con el mensaje adecuado informando de tal situación. Este método hará uso del método del apartado c.

5) 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). El orden de los empleados mostrados puede variar.

6) (***0.25 ptos.***) La interfaz `Seleccion` define un método que permite seleccionar un determinado patinete en función de diferentes criterios:

El método `boolean seleccionar(Patinete)` devuelve `true` si el patinete recibido debe ser seleccionado. En otro caso, devuelve `false`.

7) (***0.25 ptos.***) La clase `SeleccionAutonomia` implementa la interfaz `Seleccion`, y contiene información sobre un determinado umbral de autonomía (`double`). Permite seleccionar un

determinado patinete si su autonomía es menor que el umbral establecido en la creación del objeto de esta clase. La clase dispondrá de un constructor y un método de instancia público:

a. El constructor `SeleccionAutonomia(double)` almacena en una variable de instancia el umbral de autonomía recibido como parámetro, en base al cual se seleccionarán después los patinetes.
b. El método `boolean seleccionar(Patinete)` devolverá `true` si la autonomía del patinete que se pasa como parámetro es menor que el umbral de autonomía establecido.

8) (***0.25 ptos.***) La clase `SeleccionPosicion` implementa la interfaz `Seleccion`, y contiene información sobre una determinada posición de referencia (`Posicion`) y una determinada distancia máxima (`double`). Permite seleccionar un determinado patinete si la distancia desde él hasta la posición de referencia es menor que la distancia máxima establecida. La clase dispondrá de un constructor y un método de instancia público:

a. El constructor `SeleccionPosicion(Posicion,double)` almacena los valores de los parámetros (la posición de referencia y la distancia máxima) en dos variables de instancia.
b. El método `boolean seleccionar(Patinete)` devolverá `true` si la distancia desde el patinete recibido como parámetro hasta la posición de referencia es menor que la distancia máxima.

9) (***1.5 ptos.***) La clase `EmpresaSeleccion` *se comporta como la clase* `Empresa`, con la diferencia de que, a la hora de asignar códigos de patinetes a los empleados, sólo se considerarán aquellos patinetes que sean seleccionados por un objeto de tipo `Seleccion` almacenado en la variable de instancia `sel`. La clase dispondrá de un constructor y un método de instancia público:

a. El constructor `EmpresaSeleccion(String, String, Seleccion)` recibe como primer parámetro el nombre de la empresa. El segundo parámetro es el nombre del fichero con los datos de los patinetes. El tercer parámetro es el objeto de tipo `Selección` que contendrá el criterio de selección para que un patinete pueda ser asignado a un empleado.
b. El método `void asignaPatinetesEmpleado(Empleado, SortedSet<Integer>)` será una redefinición del método correspondiente de la clase `Empresa`. En este caso, sólo aquellos códigos de patinetes que sean seleccionables por el objeto de tipo `Seleccion` (`sel`) serán tenidos en cuenta para la asignación que se realiza en el método de la clase `Empresa`.

10) 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.