C/ Caoba 1 - Tel. 91 506 46 10 - Fax 91 467 01 67 - 28005 Madrid - www.iesjuandelacierva.es Código de centro 28020910

PROGRAMACIÓN
Examen Convocatoria Ordinaria
25 de mayo de 2023

Nombre y Apellidos: _____

Grupo: _____

1. Herencia y Polimorfismo	/ 2
2. Clase EmpresaAlquilerVehiculos	/ 2
3. Lanzamiento y captura de excepciones	/ 2
4. Ordenamiento de colecciones	/ 2
5. Guardado y lectura en ficheros CSV	/ 2
6. Ampliaciones extra	/ 0,75
TOTAL	**/10**

INSTRUCCIONES

Entrega

AVISO: El incumplimiento de alguna de estas instrucciones puede suponer la pérdida de hasta 1 punto

1. El examen se entregará subiendo **un FICHERO ZIP** (no rar)**, que contendrá únicamente el directorio *src,*** en la tarea correspondiente al examen práctico de la Convocatoria Ordinaria en el aula virtual.
2. El nombre del fichero zip será **<Nombre><Apellidos>.zip**. Se deben omitir las tildes. Por ejemplo, María Pérez Ruiz subiría el fichero MariaPerezRuiz.zip que contendría únicamente el directorio src con su solución del ejercicio del examen.

Generales

AVISO: El incumplimiento de alguna de estas instrucciones supondrá suspender el examen

1. El **móvil** deberá estar **en silencio y en modo avión encima de la mesa con la pantalla hacia abajo**.
2. Está prohibido llevar ningún **tipo de reloj digital, smartwatch o pulsera similar**.
3. Aquellas personas que tengan el pelo largo deberán recogérselo para tener **las orejas a la vista**. Huelga decir que no se podrá llevar ningún tipo de auricular.
4. Se recomienda encarecidamente traer **tapones para los oídos** por si hubiese ruido en el patio, para que os podáis aislar y concentrar en los exámenes.
5. Se debe **grabar la pantalla del ordenador** que se utilice durante el examen, ya sea el del aula o un portátil, durante la totalidad del examen. Para ello se recomienda utilizar el software OBS Studio. **Cada estudiante se debe responsabilizar** de custodiar la **grabación** en vídeo de su pantalla durante la realización de **todos los exámenes, teóricos y prácticos**. La grabación es responsabilidad del estudiante y no proporcionar el vídeo de la grabación supondrá suspender el examen.
6. **No se permite el acceso a internet**. Esto incluye, entre otras muchas cosas, que no se permite hacer **búsquedas en Google** ni acceder a **foros** como **StackOverflow** o similares. En caso de duda, se debe preguntar al profesor antes de abrir cualquier aplicación o web que salga del entorno del cuestionario del aula virtual. Las únicas excepciones son:
    1. consultar el API de java (javadoc)
    2. consultar la guía de estilo para Java de Google
    3. probar expresiones regulares en la página regex101
7. **No se permite** consultar apuntes, resúmenes de teoría, ejemplos ni ejercicios.
8. Debe mostrarse, al inicio de la grabación del examen, el **listado de extensiones instaladas en el IDE**.
9. Se recuerda la **prohibición** de usar ninguna **extensión en el IDE** que provea funcionalidad basada en **IA**, como *IntelliCode* o *GitHub Copilot*.

Código

AVISO: El incumplimiento de alguna de estas instrucciones puede suponer la pérdida de hasta 1'5 puntos

1. Se debe seguir la guía de estilo para Java de Google, principalmente a la hora de nombrar las clases, las variables y los paquetes.
2. El código debe estar **bien indentado**, se recomienda utilizar la opción del IDE de *formatear documento*.
3. Las variables y clases deben tener **nombres claros** que identifiquen lo que hacen
4. No debe haber *números mágicos* ni *cadenas de caracteres mágicas*: se debe utilizar variables para almacenar esos números o cadenas. Si pueden declararse como *final* y/o como *static*, se debe hacer.

5. No se debe *abusar de la autogeneración* de código por parte del IDE: aquellos métodos que no se usen, no deben implementarse.
6. Se debe **respetar la estructura de directorios/packages**. Cualquier clase auxiliar que se necesite crear, debe incluirse en el package correspondiente.
7. El código debe compilar sin generar errores ni warnings.

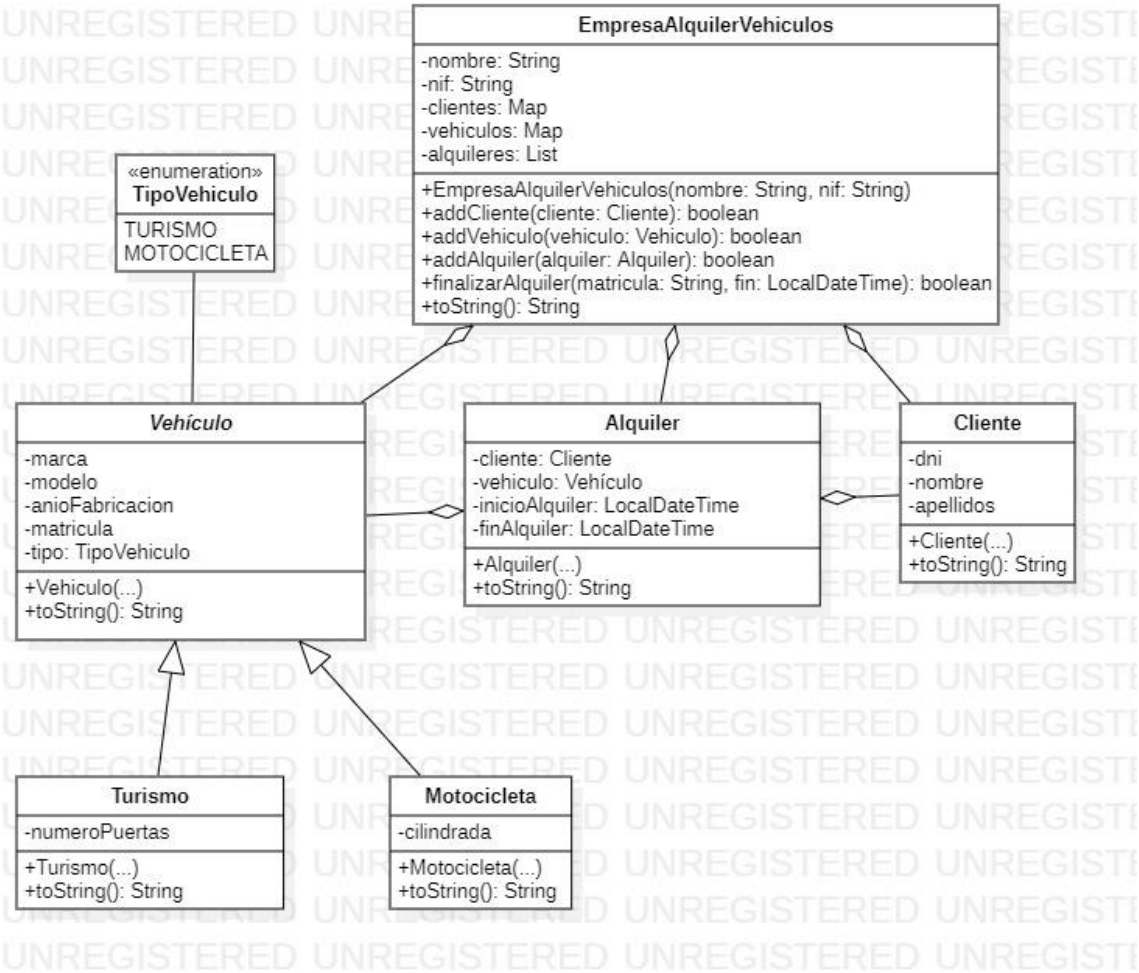
Empresa de alquiler de vehículos

Se propone implementar un sistema de gestión para una empresa de alquiler de vehículos. El sistema permitirá registrar clientes, registrar vehículos, crear alquileres y finalizarlos.

Como siempre, no hay una única solución correcta a ningún ejercicio, pero seguid lo explicado en clase y las guías del enunciado del examen. No utilicéis cosas que no hayamos visto en clase.

Diagrama UML

El siguiente diagrama UML pretende ser una ayuda para facilitar la comprensión de las diferentes clases y sus relaciones. Es solamente orientativo, hay que implementar muchos más métodos de los que aparecen en el diagrama.



Enunciado general

- El main proporcionado contiene un bucle do/while con un switch dentro para ejecutar la opción seleccionada del menú. Se recomienda encarecidamente comentar el código de cada uno de los bloques y la creación de la EmpresaAlquilerVehiculos que se hace antes del bucle do/while. Según vayáis avanzando en los diferentes apartados del examen, podréis ir descomentando bloques para comprobar que vuestras implementaciones funcionan. Si no lo hacéis así, el número de errores de compilación va a ser enorme y no vais a poder ir probando lo que desarrolléis.
- Las clases, por defecto, tendrán todos sus atributos ***private***. Esto puede hacer necesaria la creación de getters y setters. Sólo deben crearse los getters/setters que se vayan a utilizar, NO GENERAR TODOS CON EL IDE. Hay excepciones en las que un atributo no debe ser *private*, como variables *static final* que queramos crear una vez y que se puedan reutilizar (el ejemplo más claro son los comparators). En caso de duda, preguntad al profesor.
- Todas las clases, excepto los enum, deben sobrescribir el método *toString()*.
- No se enumeran todos los métodos que hacen falta, ni en el diagrama UML, ni en los enunciados de cada apartado, ni en el código proporcionado con guías de implementación. Tendréis que ir viendo qué otros métodos podéis necesitar y dónde implementarlos.
- Cada opción del menú está pensada para que sirva para validar el apartado correspondiente del examen. Las excepciones son la opción 0 que imprime el toString() de la empresa y la opción 9 que sale del programa.
- Tal y como se indica en el menú, las opciones 4 y 5 requieren que se haya ejecutado previamente la opción 2 para tener elemento en las colecciones.

El menú del main es el siguiente:

```
$ java MainExamenOrdinario
### Examen Ordinario Programacion
# Menu de opciones:
0) Imprimir empresa, con el contenido de sus colecciones
1) Crear Turismos, Clientes y Alquileres
2) Crear EmpresaAlquilerVehiculos y añadirle turismos, clientes y alquileres
3) Lanzamiento y captura de excepciones
4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA TENER
ELEMENTOS EN LAS COLECCIONES A ORDENAR)
5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA
TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
6) Ampliaciones Extras
9) SALIR
Introduce la opcion deseada:
```

Habrá que crear las siguientes clases:

1. *Vehiculo*: Clase **abstracta** que representará cualquier tipo de vehículo que se pueda alquilar.
   Sus atributos son la marca, el modelo, el año de fabricación, la matrícula y el tipo. El atributo tipo será de tipo enum TipoVehiculo y sus posibles valores son turismo y motocicleta.

Deben proporcionarse getters y setters públicos sólo cuando sean necesarios, pero harán falta más de los mostrados en el diagrama UML.

2. *Turismo*: clase concreta (no abstracta) y que hereda de *Vehiculo*. Tiene como atributo el número de puertas.
3. *Motocicleta*: clase concreta (no abstracta) y que hereda de *Vehiculo*. Tiene como atributo su cilindrada sin decimales.
4. *Cliente*: clase concreta (no abstracta) que representa cada uno de los clientes de la empresa. Sus atributos son dni, nombre y apellidos.
5. *Alquiler*: clase que representa el alquiler de un vehículo por parte de un cliente. Sus atributos son una referencia de clase Vehículo, una referencia de tipo Cliente, la fecha y hora del inicio del alquiler y la fecha y hora del fin del alquiler. Para representar fecha y hora se usará la clase *LocalDateTime*.
6. *EmpresaAlquilerVehiculos*: Clase principal que representa la empresa. Tendrá los atributos nombre, nif y las siguientes colecciones para almacenar los vehículos, los clientes y los alquileres.
    a. *vehículos*: Se utilizará un Map como colección para gestionar los vehículos. La clave/key del Map será la matrícula del vehículo, de tipo String. No podrá haber más de 1 vehículo con la misma matrícula.
    b. *clientes*: Se utilizará un Map como colección para gestionar los clientes. La clave/key del Map será el dni del cliente, de tipo String. No podrá haber más de 1 cliente con el mismo dni.
    c. *alquileres*: Se utilizará una List como colección para almacenar los alquileres. A la hora de buscar en la lista de alquileres, no tenemos otra opción que iterar sobre la lista hasta encontrar la condición que se busque.
    Si buscamos el alquiler en curso de un vehículo determinado, tendremos que recorrer todos los objetos de la lista hasta que encontremos un alquiler cuyo vehículo tenga la matrícula que buscamos y su fin de alquiler sea null.

Apartado 1. Herencia y polimorfismo (2 puntos)

Crear el enum TipoVehiculo y las clases Vehiculo, Turismo, Motocicleta, Cliente y Alquiler, con sus atributos y sus relaciones de herencia correspondientes. Deberéis crear también los constructores para cada clase.

La clase Alquiler es un caso especial ya que inicialmente solamente tiene las referencias del vehículo y del cliente y la fecha y hora del inicio del alquiler. Es decir, habrá un constructor de la clase Alquiler que solamente reciba el cliente, el vehículo y el inicio del alquiler.

El fin del alquiler es null inicialmente. El método *finalizarAlquiler(...)*, que se pide en el siguiente apartado, es el que asignará valor a la referencia de tipo *LocalDateTime* para el fin del alquiler.

Se deben implementar los métodos *toString()* de todas las clases. En la clase Alquiler, si la fecha de fin del alquiler es null, se imprimirá un asterisco (*) en vez de null.

Ejemplo de ejecución

```
$ java MainExamenOrdinario
### Examen Ordinario Programacion
# Menu de opciones:
0) Imprimir empresa, con el contenido de sus colecciones
1) Crear Turismos, Clientes y Alquileres
2) Crear EmpresaAlquilerVehiculos y añadirle turismos, clientes y alquileres
3) Lanzamiento y captura de excepciones
4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA TENER
ELEMENTOS EN LAS COLECCIONES A ORDENAR)
5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA
TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
6) Ampliaciones Extras
9) SALIR
Introduce la opcion deseada: 1

===> OPCION ELEGIDA: 1) Crear Turismos, Clientes y Alquileres
OK: Turismo creado: TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas
OK: Motocicleta creada: MOTOCICLETA: Suzuki/Burgman, 2005, Suzuki, 125 cc
OK: Motocicleta creada: MOTOCICLETA: Honda/CBR, 2015, Honda, 500 cc
OK: Turismo creado: TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
OK: Cliente creado: 01234567A - Turing, Alan
OK: Cliente creado: 89374813A - Turing, Abraham
OK: Cliente creado: 73048540F - Torvalds, Linus
OK: Alquiler creado: 73048540F - Torvalds, Linus | TURISMO: Kia/Niro, 2022,
8953LCC, 5 puertas | 2023-05-25T03:12:02.934330300 -> *

================================================
```

Apartado 2. Clase EmpresaAlquilerVehículos (2 puntos)

Crear la clase EmpresaAlquilerVehiculos, con los atributos descritos anteriormente. Se recomienda guiarse con el diagrama UML. Además, habrá que implementar los siguientes métodos:

1. *public boolean addCliente(Cliente cliente)*: Añadirá un cliente a la empresa, siempre que no haya otro cliente ya registrado con el mismo dni.
2. *public Cliente getCliente(String dni):* Buscará en el mapa de clientes y devolverá la referencia al cliente cuyo dni coincida con el proporcionado como argumento. Si no existe ningún cliente con ese dni, devolverá null.
3. *public boolean addVehiculo(Vehiculo vehiculo)*: Añadirá un vehículo a la empresa, siempre que no haya otro vehículo con la misma matrícula.
4. *public Vehiculo getVehiculo(String matricula)*: Buscará en el mapa de vehiculos y devolverá la referencia al vehículo cuya matrícula coincida con la proporcionada como argumento. Si no existe ningún vehículo con esa matrícula, devolverá null.
5. *public boolean addAlquiler(Alquiler alquiler)*: Añadirá un alquiler a la lista de alquileres de la empresa.
6. *public boolean finalizarAlquiler(String matricula, String fin)*: Se buscará una instancia de alquiler en la lista de alquileres cuya matrícula coincida con la recibida como argumento y cuyo fin de alquiler sea null y se actualizará el fin del alquiler con la fecha y hora recibida como argumento *fin*. Si no se encuentra un alquiler con esas condiciones o se produce cualquier otro error, se devolverá false.
7. *public String toString()*: Devolverá un String con el nombre y nif de la empresa, y el listado de clientes, vehículos y alquileres, utilizando los siguientes métodos toStringXXX() descritos a continuación.
8. *public String toStringClientes()*: Devolverá un String con el listado de clientes, da igual en qué orden.
9. *public String toStringVehiculos()*: Devolverá un String con el listado de vehículos, da igual en qué orden.
10. *public String toStringAlquileres()*: Devolverá un String con el listado de todos los alquileres, da igual en qué orden. Si el fin es null, se imprimirá un asterisco (*) en vez de null.

Ejemplo de ejecución

```
$ java MainExamenOrdinario
### Examen Ordinario Programacion
# Menu de opciones:
0) Imprimir empresa, con el contenido de sus colecciones
1) Crear Turismos, Clientes y Alquileres
2) Crear EmpresaAlquilerVehiculos y añadirle turismos, clientes y alquileres
3) Lanzamiento y captura de excepciones
4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA TENER
ELEMENTOS EN LAS COLECCIONES A ORDENAR)
5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA
TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
6) Ampliaciones Extras
9) SALIR
Introduce la opcion deseada: 2

===> OPCION ELEGIDA: 2) Crear EmpresaAlquilerVehiculos y añadirle turismos,
clientes y alquileres
OK: Cliente agregado: 01234567A - Turing, Alan
ERROR: No se pudo agregar el cliente: 01234567A - Turing, Alan
OK: Cliente agregado: 89374813A - Turing, Abraham
OK: Cliente agregado: 73048540F - Torvalds, Linus
OK: Turismo agregado: TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas
ERROR: No se pudo agregar el turismo: TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3
puertas
OK: Turismo agregado: TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
OK: Motocicleta agregada: MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
OK: Motocicleta agregada: MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
# Iniciando el alquiler del vehiculo con matrícula 9234BCJ con fecha 23 de
Mayo de 2023 a las 10h:
OK: Alquiler agregado: 01234567A - Turing, Alan | TURISMO: Ford/Fiesta, 2002,
9234BCJ, 3 puertas | 2023-05-23T10:00 -> *
# Iniciando el alquiler del vehiculo con matrícula 7920JFK con fecha 10 de
Mayo de 2023 a las 13h:
OK: Alquiler agregado: 73048540F - Torvalds, Linus | MOTOCICLETA: Honda/CBR,
2015, 7920JFK, 500 cc | 2023-05-10T13:00 -> *

## empresa.toStringAlquileres():
01234567A - Turing, Alan | TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas |
2023-05-23T10:00 -> *
73048540F - Torvalds, Linus | MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc |
2023-05-10T13:00 -> *

# Finalizando el alquiler del vehiculo con matrícula 7920JFK con fecha 20 de
Mayo de 2023 a las 17h:
OK: Alquiler vehiculo con matrícula 7920JFK finalizado.
# Si volvemos a intentar finalizar el alquiler del mismo vehiculo con
matrícula 7920JFK, el método falla y devuelve false:
ERROR: No se ha podido finalizar el alquiler de vehículo con matrícula
7920JFK.

## empresa.toString():
=== Empresa: Vehiculos La Cierva, NIF: 57849332A
== Clientes:
01234567A - Turing, Alan
73048540F - Torvalds, Linus
89374813A - Turing, Abraham

== Vehiculos:
MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas

== Alquileres:
01234567A - Turing, Alan | TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas |
2023-05-23T10:00 -> *
73048540F - Torvalds, Linus | MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc |
2023-05-10T13:00 -> 2023-05-20T17:00


================================================
```

Apartado 3. Lanzamiento y captura de excepciones (2 puntos)

Todos los constructores comprobarán sus argumentos y, en caso de que alguno sea inválido, se lanzará una excepción de tipo *ParametroInvalidoException* con un mensaje que describirá la razón por la que se lanza la excepción.

Las comprobaciones que se deben realizar son:

1. Ningún atributo de tipo String de ninguna clase puede ser vacío.
2. El año de fabricación, en la clase Vehiculo, no puede ser menor que 1867.
3. El número de puertas de un turismo tiene que ser mayor o igual que 1.
4. La cilindrada de una motocicleta tiene que ser mayor o igual que 50.

Se deben capturar todas las excepciones que los ejemplos del *main* proporcionado puedan lanzar.

A pesar de que nosotros solamente lanzaremos estas excepciones, debemos capturar todas las excepciones que se puedan lanzar.

Esto hay que tenerlo especialmente en cuenta cuando se lean, de los ficheros CSV, valores que haya que convertir de cadena a otro formato, por ejemplo el número de puertas o la cilindrada. Esto se verá en el apartado 5.

Ejemplo de ejecución

```
$ java MainExamenOrdinario
### Examen Ordinario Programacion
# Menu de opciones:
0) Imprimir empresa, con el contenido de sus colecciones
1) Crear Turismos, Clientes y Alquileres
2) Crear EmpresaAlquilerVehiculos y añadirle turismos, clientes y alquileres
3) Lanzamiento y captura de excepciones
4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA TENER
ELEMENTOS EN LAS COLECCIONES A ORDENAR)
5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA
TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
6) Ampliaciones Extras
9) SALIR
Introduce la opcion deseada: 3

===> OPCION ELEGIDA: 3) Lanzamiento y captura de excepciones
ERROR: ParametroInvalidoException capturada: Cliente: argumento dni vacío
ERROR: ParametroInvalidoException capturada: Cliente: argumento nombre vacío
ERROR: ParametroInvalidoException capturada: Cliente: argumento apellidos
vacío
ERROR: ParametroInvalidoException capturada: Vehiculo: argumento marca vacío
ERROR: ParametroInvalidoException capturada: Vehiculo: argumento modelo vacío
ERROR: ParametroInvalidoException capturada: Vehiculo: argumento año de
fabricación inválido, debe ser mayor a 1867
ERROR: ParametroInvalidoException capturada: Turismo: argumento numero de
puertas inválido, debe ser mayor o igual que 1
ERROR: ParametroInvalidoException capturada: Motocicleta: argumento cilindrada
inválido, debe ser mayor o igual que 50

================================================
```

Apartado 4. Ordenamiento de colecciones (2 puntos)

Se implementarán los siguientes métodos en la clase EmpresaAlquilerVehiculos para devolver cadenas de caracteres con los listados de sus elementos en el orden indicado en cada método. El orden de las colecciones originales no se debe modificar, sino que debemos crear nuevas colecciones y ordenarlas en cada método.

1. *String toStringClientesOrdenNatural()*: Devuelve un string con el listado de clientes ordenados por su orden natural.
   El orden natural de la clase Cliente se define por sus apellidos y, en caso de iguales apellidos, por el nombre.
   Ejemplo: Alan Turing iría antes que John Turing, porque a igual apellido, el nombre Alan va antes que John. En cambio, entre Alan Turing y Linus Torvalds, primero iría Linus Torvalds, porque primero se comparan los apellidos y Torvalds "es menor" que Turing.
   Recordatorio: tanto nombre como apellido deberían ser atributos String que ya tienen disponible el método *compareTo()*.
2. *String toStringClientesOrdenDni()*: Devuelve un string con listado de clientes ordenados por su dni.
3. *String toStringVehiculosOrdenNatural()*: Devuelve un string con el listado de vehículos ordenados por su orden natural.
   El orden natural de un vehículo se basa en su año de fabricación, de más reciente a más antiguo, de mayor a menor. Es decir, si hay un vehículo del 2023 y otro del 2015, ese es su orden natural.
4. *String toStringAlquileresOrdenNatural():* Devuelve un string con el listado de vehículos ordenados por su orden natural.
   El orden natural de un alquiler se basa en su fecha y hora de inicio: primero o menores los más antiguos, después o mayores los más recientes.

Ejemplo de ejecución (se había ejecutado anteriormente la opción 2):

```
===> OPCION ELEGIDA: 4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA
OPCION 2 ANTES PARA TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
## empresa.toStringClientesOrdenNatural() [apellido y nombre]:
73048540F - Torvalds, Linus
89374813A - Turing, Abraham
01234567A - Turing, Alan

## empresa.toStringClientesOrdenDni() [dni]:
01234567A - Turing, Alan
73048540F - Torvalds, Linus
89374813A - Turing, Abraham

## empresa.toStringVehiculosOrdenNatural() [anio de fabricacion descendente]:
TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas

## empresa.toStringAlquileresOrdenNatural() [inicio alquiler]:
73048540F - Torvalds, Linus | MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc |
2023-05-10T13:00 -> 2023-05-20T17:00
01234567A - Turing, Alan | TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas |
2023-05-23T10:00 -> *

==============================================
```

Apartado 5. Guardado y lectura en ficheros CSV (2 puntos)

Vamos a usar ficheros CSV para guardar y cargar los clientes, turismos y motocicletas. No hace falta implementar el guardado y carga de los alquileres, se deja como ampliación extra (ver apartado 6).

1. Los ficheros CSV no tendrán cabecera (header).
2. El separador de cada campo dentro de cada línea será el carácter "#".
3. En caso de que alguna línea de un fichero CSV tenga algún error que impida construir un objeto y/o cargarlo en la colección correspondiente, se mostrará un mensaje de error, se descartará la línea y se deberá continuar leyendo el resto de las líneas del fichero si las hubiese.
4. Al guardar los elementos en ficheros CSV no se guardará ningún valor anterior que pudiese haber en el fichero. Se sobrescribirá directamente, no se añadirán líneas al final.

Se recomienda revisar los ejemplos de ficheros CSV proporcionados para ver el formato. Hay ficheros *.in.csv que se usan en el main para cargar nuevos clientes, turismos y motocicletas, y ficheros *.out.csv que son el resultado de guardar los clientes, turismos y motocicletas añadidas en el apartado 2 (opción 2 del menú).

Se recomienda implementar el método *String toCsvLine(String csvSeparator)* en cada clase cuyos objetos se vayan a guardar en ficheros CSV.

Ejemplo de ejecución de la opción 5 (se había ejecutado anteriormente la opción 2) y después la opción 4, para mostrar cómo se siguen ordenando correctamente las colecciones, incluyendo los nuevos elementos cargados desde los ficheros CSV:

```
Introduce la opcion deseada: 5

===> OPCION ELEGIDA: 5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR
LA OPCION 2 ANTES PARA TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
## ESCRIBIMOS FICHEROS *.out.csv
# empresa.toStringClientes():
01234567A - Turing, Alan
73048540F - Torvalds, Linus
89374813A - Turing, Abraham

empresa.guardarClientesCsv(clientes.out.csv):
OK: Clientes guardados en fichero 'clientes.out.csv'
# empresa.toStringVehiculos():
MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas

empresa.guardarTurismosCsv(turismos.out.csv):
OK: Turismos guardados en fichero 'turismos.out.csv'
empresa.guardarMotocicletasCsv(motocicletas.out.csv):
OK: Motocicletas guardados en fichero 'motocicletas.out.csv'

## CARGAMOS FICHEROS *.in.csv
empresa.cargarClientesCsv(clientes.in.csv):
[cargarClientesCsv] Error en clientes.in.csv:1: se esperaban 3 columnas y se
han leído 2
[cargarClientesCsv] Error en clientes.in.csv:2: se esperaban 3 columnas y se
han leído 4
OK: Clientes cargados desde el fichero 'clientes.in.csv'
# empresa.toStringClientes():
01234567A - Turing, Alan
45027523C - Lamarr, Hedy
73048540F - Torvalds, Linus
78713217O - Conway, Lynn
89374813A - Turing, Abraham

empresa.cargarTurismosCsv(turismos.in.csv):
[cargarTurismosCsv] Error en turismos.in.csv:1: se esperaban 5 columnas y se
han leído 4
[cargarTurismosCsv] Error en turismos.in.csv:2 parseando anio de fabricación:
For input string: "dos mil veintitres"
[cargarTurismosCsv] Error en turismos.in.csv:3 parseando numero de puertas:
For input string: "Cuatro"
OK: Turismos cargados en fichero 'turismos.in.csv'
empresa.cargarMotocicletasCsv(motocicletas.in.csv):
[cargarMotocicletasCsv] Error en motocicletas.in.csv:1: se esperaban 5
columnas y se han leído 4
[cargarMotocicletasCsv] Error en motocicletas.in.csv:2 parseando anio de
fabricación: For input string: "dos mil veintitres"
[cargarMotocicletasCsv] Error en motocicletas.in.csv:3 parseando cilindrada:
For input string: "Quinientos"
OK: Motocicletas cargados en fichero 'motocicletas.in.csv'
# empresa.toStringVehiculos():
MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
MOTOCICLETA: Ducati/Monster, 2009, 4843GCB, 500 cc
MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
TURISMO: Renault/Twingo, 2003, 8953BVT, 5 puertas
TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas


================================================
# Menu de opciones:
0) Imprimir empresa, con el contenido de sus colecciones
1) Crear Turismos, Clientes y Alquileres
2) Crear EmpresaAlquilerVehiculos y añadirle turismos, clientes y alquileres
3) Lanzamiento y captura de excepciones
4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA TENER
ELEMENTOS EN LAS COLECCIONES A ORDENAR)
5) Guardado y lectura en ficheros CSV (SE DEBE EJECUTAR LA OPCION 2 ANTES PARA
TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
```

```
6) Ampliaciones Extras
9) SALIR
Introduce la opcion deseada: 4

===> OPCION ELEGIDA: 4) Ordenamiento de colecciones (SE DEBE EJECUTAR LA
OPCION 2 ANTES PARA TENER ELEMENTOS EN LAS COLECCIONES A ORDENAR)
## empresa.toStringClientesOrdenNatural() [apellido y nombre]:
78713217O - Conway, Lynn
45027523C - Lamarr, Hedy
73048540F - Torvalds, Linus
89374813A - Turing, Abraham
01234567A - Turing, Alan

## empresa.toStringClientesOrdenDni() [dni]:
01234567A - Turing, Alan
45027523C - Lamarr, Hedy
73048540F - Torvalds, Linus
78713217O - Conway, Lynn
89374813A - Turing, Abraham

## empresa.toStringVehiculosOrdenNatural() [anio de fabricacion descendente]:
TURISMO: Kia/Niro, 2022, 8953LCC, 5 puertas
MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc
MOTOCICLETA: Ducati/Monster, 2009, 4843GCB, 500 cc
MOTOCICLETA: Suzuki/Burgman, 2005, 3956DCB, 125 cc
TURISMO: Renault/Twingo, 2003, 8953BVT, 5 puertas
TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas

## empresa.toStringAlquileresOrdenNatural() [inicio alquiler]:
73048540F - Torvalds, Linus | MOTOCICLETA: Honda/CBR, 2015, 7920JFK, 500 cc |
2023-05-10T13:00 -> 2023-05-20T17:00
01234567A - Turing, Alan | TURISMO: Ford/Fiesta, 2002, 9234BCJ, 3 puertas |
2023-05-23T10:00 -> *


================================================
```

Apartado 6. Ampliaciones extra

Las siguientes ampliaciones se podrán hacer independientemente del resto del examen, y podrán sumar puntos no contemplados en los anteriores apartados. Obviamente, la nota máxima del examen es un 10, por mucho que se hagan todos los apartados y las ampliaciones extra.

No se proporciona código en el *main* que valide estas funcionalidades ni ejemplos de ejecución. Se deberán añadir ejemplos que muestren que estas ampliaciones funcionan.

1. **Validación de la matrícula mediante una regex (0,25 puntos)**
   Se deberá validar que la matrícula cumpla con el formato de contener 4 dígitos seguidos de 3 letras mayúsculas y consonantes (no puede haber vocales). En caso de no cumplir el formato, se lanzará una excepción.
   Un ejemplo de matrícula válida sería "1234BCD".
   Ejemplos de matrículas inválidas serían "1234bcd", "345BCD" o "1234ABC".
2. **Método *String toStringVehiculosOrdenMatricula()* (0,25 puntos)**
   Crear el método *String toStringVehiculosOrdenMatricula()* en la clase EmpresaAlquilerVehiculos que liste los vehículos en orden de matrícula, de más antigua a más moderna.
   Se recuerda que las matrículas comenzaron en 0000BBB, fueron avanzando los números hasta 9999BBB y la siguiente matrícula fue 0000BBC. Es decir, para ordenar las matrículas primero se deben comparar las letras y, en caso de que las letras sean iguales, se debe comparar la parte numérica.
   Se recomienda revisar el método String.substring (int beginIndex, int endIndex).
   Se penalizará especialmente el uso de números mágicos en este apartado.
3. **Cargar y guardar alquileres en formato CSV (0,25 puntos)**
   Dado que un alquiler está formado por un cliente, un vehículo, el inicio y el fin del alquiler, el cliente se representará en el fichero CSV con el dni y el vehículo con la matrícula. Con estos identificadores se tendrá que extraer la referencia de cliente y del vehículo a partir de los mapas de la clase EmpresaAlquilerVehiculos. Si al leer un fichero de alquileres no se encuentra un cliente por su dni o un vehículo por su matrícula, esa línea del fichero CSV se deberá descartar mostrando un mensaje de error y se deberá continuar leyendo el resto de las líneas del fichero. Por todo esto, primero se deben cargar los ficheros de clientes y vehículos y, por último, el fichero de alquileres. En el caso de haber alguna fecha y hora de fin a null, se representará con un asterisco.