DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN	**PROGRAMACIÓN ORIENTADA A OBJETOS** (Prueba realizada el 17 de junio de 2014)	
APELLIDOS, Nombre	TITULACIÓN Y GRUPO	
	MÁQUINA	

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- El ejercicio se almacenará en el directorio **C:\POO**. En caso de que no exista deberá crearse, y si ya existiese, deberá borrarse todo su contenido antes de comenzar.
- Al inicio de cada fichero deberá indicarse **el nombre del alumno, titulación, grupo y código del equipo** que está utilizando**.**
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, el alumno no debe pararse en él indefinidamente. Puede abordar otros.
- **Está permitido:**
  - Consultar el API.
- **No está permitido**:
  - Utilizar otra documentación electrónica o impresa.
  - Intercambiar documentación con otros compañeros.
  - Utilizar soportes de almacenamiento.
- Una vez terminado el ejercicio subir un fichero comprimido (.rar) sólo con los **fuentes**

Este ejercicio trata de la construcción de un programa para la simulación del funcionamiento de un servicio de reserva de plazas de aparcamiento en un conjunto de parkings, para esto se deberá crear un proyecto **`prParking`** con las clases siguientes:

1) (***0.25 ptos.***) La clase **`ParkingException`** que se utilizará para tratar todas las situaciones excepcionales (argumentos incorrectos de los métodos, ficheros, que no se encuentran, errores de formato, …). Se tratará de una excepción **no comprobada**.

2) La clase **`Posicion`**, disponible en el campus virtual, que mantiene información sobre las coordenadas cartesianas (`double`) de una posición en un plano. Esta clase dispone de: un constructor a partir de sus dos coordenadas, el método `String toString()` que devuelve un `String` con las coordenadas entre paréntesis, los métodos de consulta de las coordenadas `double getX()` y `double getY()`, el método `double distancia(Posicion p)` para calcular la distancia a otra posición y los métodos `equals` y `hashCode`.

3) (***0.50 ptos.***) La clase **`Vehiculo`** que almacenará la información necesaria de un vehículo que solicita una plaza de aparcamiento, a saber: la matrícula (`String`), su posición actual (`Posicion`), la hora de entrada y la hora de salida (`int`) teniendo en cuenta que sólo se admiten peticiones de reserva por un número entero de horas, dentro de un mismo día, con entradas desde la hora 0 hasta la hora 23 y salidas desde la hora 1 hasta la hora 24. La hora de salida siempre tiene que ser posterior a la de entrada y en la hora de salida NO se ocupa plaza, p.e. si un vehículo solicita una reserva de 10 a 14, se entenderá que desea ocupar una plaza durante las horas 10, 11, 12 y 13 y esta será su *franja horaria*. Esta clase dispondrá de:
   a. *(0.25 ptos.)* Un constructor que recibe como parámetros, la matrícula, la posición y las horas de entrada y de salida (cuya validez deberá comprobar), y una representación textual (`String toString()`) en la que aparecerá toda la información de un vehículo.

b. *(0.10)* Unos métodos, `String getMatricula()`, `Posicion getPosicion()`, `int getEntrada()` y `int getSalida()`, para consultar cada uno de los atributos.
c. *(0.15)* Un método para decidir si dos vehículos son iguales atendiendo únicamente a las matrículas sin distinguir entre letras mayúsculas y minúsculas.

4) (***2.85 ptos.***) La clase **`Parking`** que mantiene un identificador (`String`) de un parking, su posición (`Posicion`), una constante LIBRE con valor "libre" y un array bidimensional de `String` con la ocupación, por horas, de cada una de las plazas del parking de la forma siguiente

	Hora 0	Hora 1	… … … …	Hora 23
Plaza 0	"libre"	"5643HFS"	… … … …	"libre"
Plaza 1	"MA3450AT"	"MA3450AT"	… … … …	"libre"
… … …	… … … …	… … … …	… … … …	… … … …
Plaza N	"libre"	"0891ASS"	… … … …	"libre"

La clase dispondrá de:
a. *(0.50 ptos.)* Un constructor que recibirá como argumentos: un identificador (`String`), una posición (`Posicion`) y un número de plazas (`int`), cuya validez deberá comprobar, y construirá el correspondiente parking con el número de plazas dado y pondrá en todas las horas del array de reservas la constante LIBRE.
b. *(0.10)* Métodos para consultar el identificador (`String getId()`) y la posición (`Posicion getPosicion()`).
c. *(0.50)* Un método `private boolean hayHuecoEnPlaza(Vehiculo v, int pz)`, para determinar si está libre la franja horaria solicitada por `v` en la plaza `pz`.
d. *(0.50)* Un método para buscar una plaza libre en el parking durante la franja horaria solicitada por un vehículo `v`, `public int buscarPlaza(Vehiculo v)`. El método devolverá el número de una plaza libre desde la hora de llegada del vehículo `v` hasta la hora de salida, o -1 si no encuentra ninguna plaza que esté libre durante ese periodo.
e. *(0.50)* Un método para reservar una determinada plaza `pz` para un vehículo `v` dado
```
public boolean reservarPlaza(Vehiculo v, int pz)
```
Este método comprobará la validez de sus argumentos, comprobará si la plaza `pz` está libre en el periodo que solicita `v` y, si todo es correcto, pondrá la matrícula de v en la correspondiente franja horaria de la plaza `pz` y devolverá `true`; en caso contrario no hará nada y devolverá `false`.
f. *(0.25)* Un método para comprobar la igualdad de dos parkings comparando los identificadores con independencia de mayúsculas o minúsculas y las posiciones de ambos.
g. *(0,50)* Un método (`String toString()`) para la representación textual de un parking presentando su identificador, su posición y el array de reservas de las distintas plazas

5) (***1.00 ptos.***) La clase satélite **`ComparadorParking`** cuyos objetos servirán para comparar dos parkings en función de sus distancias a un cierto vehículo. Para esta clase *se deberá definir un constructor* que recibirá como argumento un `Vehiculo v` y lo guardará en una variable de instancia. Un objeto de esta clase considerará un parking menor que otro si está más próximo al vehículo `v` que tiene como variable de

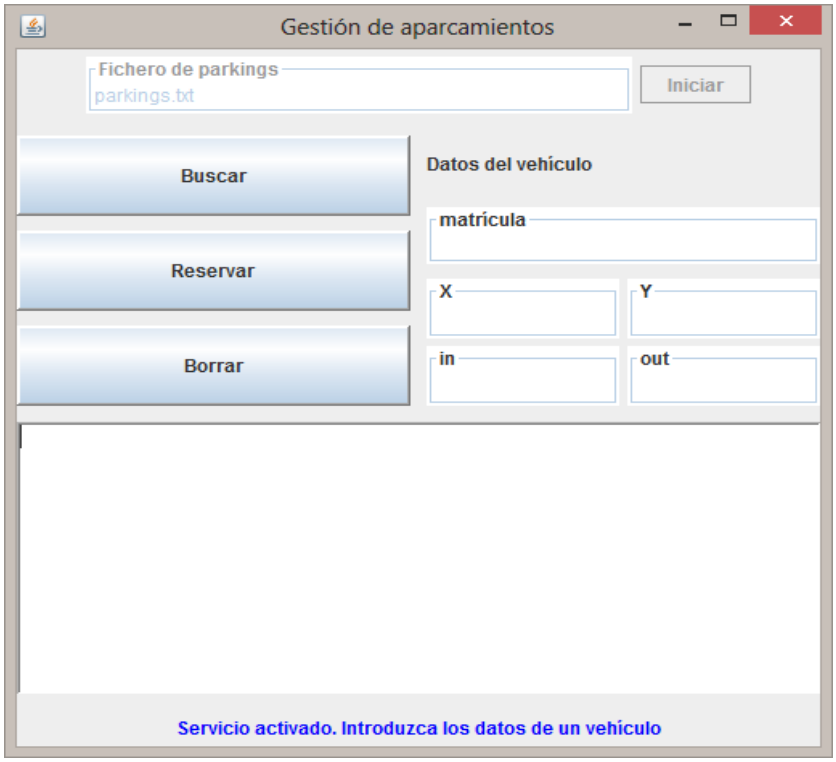
instancia. Nótese que cada objeto de esta clase se puede utilizar para ordenar los parkings en función de la distancia a su vehículo variable de instancia.

6) (***3.60 ptos.***) La clase **`ServicioDeReserva`** que mantiene un conjunto de parkings y se encarga de gestionar las solicitudes de reserva de los vehículos buscando plazas en los distintos parkings. Esta clase dispondrá de:
   a. *(0.35)* Un constructor sin argumento que inicializará el conjunto de parkings.
   b. *(1.00)* Un constructor que recibirá como argumento el nombre de un fichero de líneas de texto de la forma
      Parking_1 0.0 0.0 5
      Parking_2 4.0 0.0 3
      donde cada línea contiene el identificador de un parking, sus dos coordenadas y el número de plazas de aparcamiento. Este constructor deberá inicializar el conjunto de parkings, construir los parkings con los datos del fichero (con todas sus plazas libres en todas las horas) y agregarlos al conjunto de parkings.
   c. *(0.15)* Un método para agregar un parking al conjunto de parkings que gestiona el servicio de reservas
      `public boolean addParking(Parking pk)`
      este método devolverá `true` cuando el parking no se encuentre previamente en el conjunto de parkings y `false` en caso contrario.
   d. *(0.50)* Un método para reservar una plaza para un vehículo `v`, en un cierto parking `pk`,
      `public boolean reservarPlaza(Vehiculo v, Parking pk)`
      Este método comprobará si se puede reservar plaza para `v` en el parking `pk` y, en caso afirmativo, lo hará y devolverá `true`; en caso contrario no hará ninguna reserva y devolverá `false`.
   e. *(0.25)* Un método (`String toString()`) para la representación textual del conjunto de parkings del servicio de reservas.
   f. *(0.85)* Un método para localizar, para un vehículo dado `v`, el parking más próximo que dispone de una plaza libre en la franja horaria solicitada por `v`.
      `public Parking parkingLibreMasProximo(Vehiculo v)`
      Este método devolverá `null` cuando no se encuentre ningún parking que tenga libre la franja horaria solicitada por `v`.
   g. *(0.50)* Un método para reservar plaza en el parking más próximo para cada uno de los vehículos de una lista, siguiendo el orden en el que aparecen en la lista,
      `public Map<String,String>reservaDePlazas(List<Vehiculos> lv)`
      Este método realizará las reservas posibles y devolverá una aplicación asociando a la matrícula de cada vehículo el identificador del parking más próximo con plaza libre (se puede dar la circunstancia de que para algún vehículo no haya plaza disponible, en cuyo caso no aparecerá en la aplicación).

7) (***1.80 ptos.***) La clase **`ControladorDeReservas`** que controlará una vista **`PanelReservaDeParking`** (que se proporciona en el campus virtual junto con la interfaz **`VistaReservaDeParking`** donde se explica el comportamiento de los métodos de la vista) e interactuará con un modelo (compuesto por las clases anteriormente descritas).
   a. *(0.35)* Esta clase tendrá un constructor que recibirá la vista y habilitará la parte de inicialización: ventana de introducción del fichero de parkings, y el botón INICIO, y mostrará un mensaje en la parte baja de la misma indicando al usuario que introduzca el nombre del fichero y pulse el botón de inicio. El resto de la vista estará deshabilitado. La actuación de los objetos de esta clase será la siguiente:

b. *(0.35)* La pulsación del botón INICIAR dará lugar a la creación de un objeto `ServicioDeReserva` con los parkings construidos a partir de los datos leídos del fichero y habilitará el resto de la vista, deshabilitando la zona de inicio, además, presentará un mensaje en la parte baja invitando al usuario a introducir los datos de un vehículo. Si no está el nombre del fichero o el fichero no se encuentra, se mostrará un mensaje de error y se permanecerá en la situación inicial.
c. *(0.50)* La pulsación del botón BUSCAR provocará la lectura de los datos de un vehículo introducidos por el usuario (matrícula, posición construida a partir de las coordenadas, hora de entrada y hora de salida) y la búsqueda del parking más próximo con una plaza libre en la franja horaria leída. El identificador del parking se mostrará como información, junto con la distancia al vehículo. Si no hay plaza, también se anunciará como información. En estos casos se borrará la zona de salida de mensajes. Si faltan datos o los datos leídos no son correctos, se mostrará un mensaje de error.
d. *(0.50)* La pulsación del botón RESERVAR provocará también la lectura de los datos de un vehículo introducidos por el usuario, la búsqueda del parking más próximo con una plaza libre en la franja horaria leída y, si se encuentra, la reserva de la correspondiente plaza. En este caso, también se mostrará el identificador del parking y el número de la plaza reservada como información y se borrará la zona de salida de mensajes. Si faltan datos o los datos leídos no son correctos o no se encuentra plaza libre, se mostrará el correspondiente mensaje de error.
e. *(0.10)* La pulsación del botón BORRAR provocará que se borren todos los datos del vehículo y que se invite a usuario a introducir datos nuevos.

Las clases principales con ejemplos de uso (no exahustivos) de las clases anteriormente descritas (sin GUIs y con GUIs) están disponibles en el campus virtual.



(Aspecto que presenta la interfaz gráfica proporcionada)