PRÁCTICA A001.- Parking (Array de Plazas de aparcamiento)

En esta práctica se va a gestionar la ocupación de un parking.

Cada parking tiene un número de plazas donde se van a poder aparcar vehículos. Los datos del parking son las plazas de aparcamiento y el coste por minuto.

Se dispone de un **interface Parking** que define las operaciones que se pueden realizar en el parking. Ademas en el proyecto hay una clase **ParkingArrayImpl** que implementa el interface Parking y por tanto debe implementar todas las operaciones del interface.

IMPORTANTE: Las plazas del parking están numeradas a partir de 1. (Cuidado porque en el array la plaza 1 se almacena en la posición 0 del array).

- **Entorno de prácticas para la asignatura: Eclipse con JUnit 4.**

Elegir un espacio de trabajo (un `workspace`) para las prácticas.

Todos los ficheros de texto estarán codificados en `UTF-8`: Cuando se haya iniciado el IDE, comprobar que en las preferencias se tiene asignada `UTF-8` como codificación por defecto para el espacio de trabajo.

PASOS PARA LA REALIZACIÓN DE LA PRÁCTICA:

1. Descargar el proyecto de la página de la asignatura en agora.unileon.es:

   - edi_a001_2023.zip

2. Importar dicho proyecto en Eclipse :File-> Import… Existing proyects into Workspace… Select archive file…

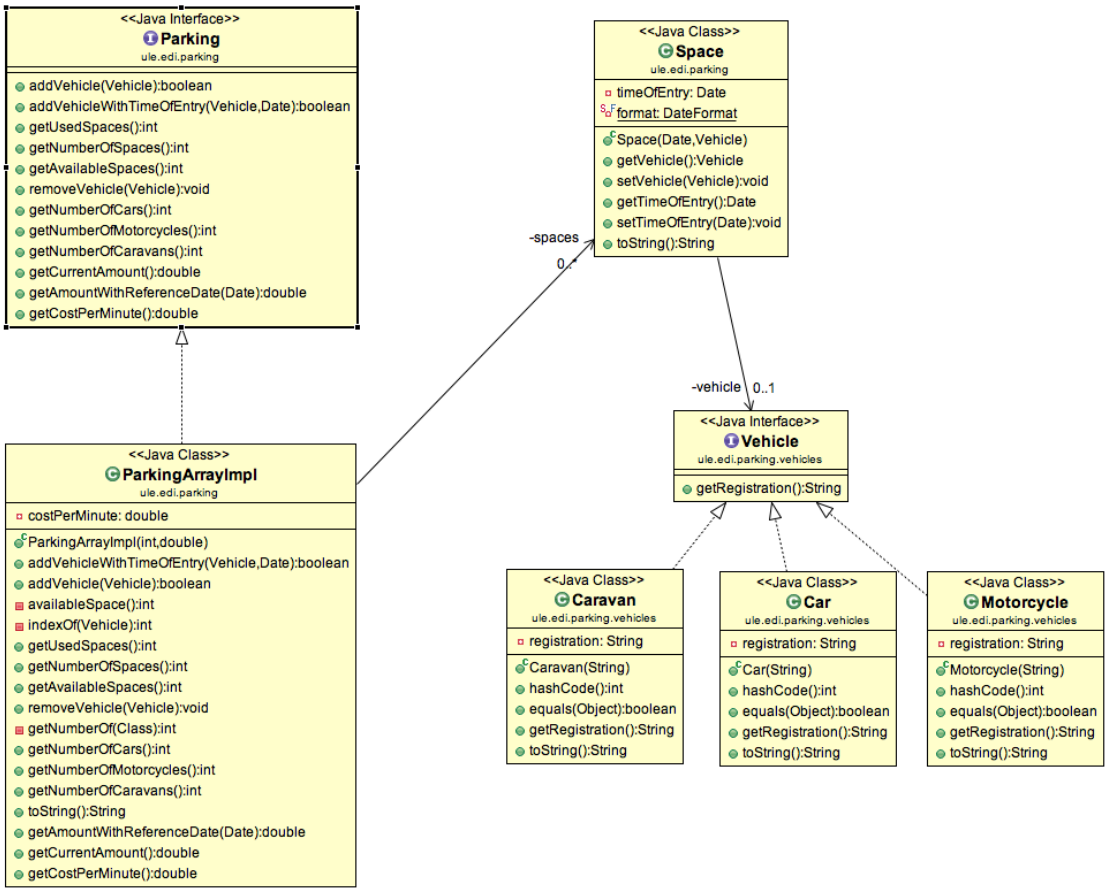
   (indicar el archivo ZIP descargado)

Los proyectos normalmente contendrán la estructura a respetar (y probablemente muchos errores de compilación ya que todo el trabajo está por hacer).

3. El interface **Parking.java NO PUEDE MODIFICARSE**.

   Para esta práctica tenemos el paquete ule.edi.parking en el que se definen las estructuras de datos utilizadas (**Interface Parking**, **clase Space, clase ParkingArrayImpl y clase ParkingArrayImplTests). Además hay un paquete ule.edi.parking.vehicles en el que sedefinen los vehiculos que se pueden aparcar en** el parking.

DIAGRAMA DE CLASES:




```
public interface Parking {
        /**
         * Marca la llegada de un vehículo al aparcamiento.
         *
         * Guarda el vehículo y su fecha de entrada (la fecha actual). Si ya existe
         * en el aparcamiento una posición ocupada por otro vehículo igual al
         * dado, no se hace nada.
         *
         * @param r vehículo que llega al aparcamiento.
         * @return true si se guardó con éxito el vehículo en una nueva posición;
   * false en caso contrario
         */
        public boolean addVehicleWithTimeOfEntry(Vehicle r, Date toe);


        /**
         * Marca la llegada un una fecha dada de un vehículo al aparcamiento.
         *
         * Guarda el vehículo y su fecha de entrada (la fecha actual).
         * Se puede llamar al método anterior pasandole la fecha actual (new Date()) ).
         * Si ya existe en el aparcamiento una posición ocupada por otro vehículo igual
         * al dado, no se hace nada.
         *
         * @param r vehículo que llega al aparcamiento.
         * @param toe fecha de llegada del vehículo.
         * @return true si se guardó con éxito el vehículo en una nueva posición;
   *       false en caso contrario
         */
    public boolean addVehicle(Vehicle  r);
```

```
        /**
         * Indica el número de plazas de aparcamiento ocupadas.
         *
         * @return número de plazas ocupadas.
         */
        public int getUsedSpaces();

        /**
         * Indica el número de plazas totales en este aparcamiento.
         *
         * @return número total de plazas.
         */
        public int getNumberOfSpaces();

        /**
         * Indica el número de plazas de aparcamiento disponibles.
         *
         * @return número de plazas disponibles.
         */
        public int getAvailableSpaces();

        /**
         * Elimina el vehículo dado del aparcamiento.
         *
         * Elimina el vehículo del aparcamiento que sea igual al dado
         * por parámetro.
         *
         * @param r
         */
        public void removeVehicle(Vehicle r);


        /**
         * Devuelve el número de coches en el aparcamiento.
         *
         * (El número de vehículos de tipo Car)
         *
         *  Se puede usar el método getClass para obtener el tipo del objeto que está en
    * la posición i del array: spaces[i].getVehicle().getClass().equals(Car.class)
         *
         * @return número de coches en el aparcamiento.
         */
        public int getNumberOfCars();

        /**
         * Devuelve el número de motos en el aparcamiento.
         *
         * (El número de vehículos de tipo Motorcycle)
         *
         * @return número de motos en el aparcamiento.
         */
        public int getNumberOfMotorcycles();

        /**
         * Devuelve el número de caravanas en el aparcamiento.
         *
         * (El número de vehículos de tipo Caravan)
         *
         * @return número de caravanas en el aparcamiento.
         */
        public int getNumberOfCaravans();

        /**
         * Devuelve el valor actual del aparcamiento.
         *
         * Con la fecha actual como referencia, calcula cuánto tiempo ha estado
         * cada vehículo en el aparcamiento. Con el total y el coste por minuto,
```

```
	 * calcula el valor del aparcamiento.
	 *
	 * Los cálculos se realizarán apoyándose en {@link Date#getTime()}.
	 *
	 * @return valor actual del aparcamiento.
	 */
	public double getCurrentAmount();

	/**
	 * Devuelve el valor del aparcamiento para una fecha indicada.
	 *
	 * Con la fecha dada como referencia, calcula cuánto tiempo ha estado
	 * cada vehículo en el aparcamiento. Con el total y el coste por minuto,
	 * calcula el valor del aparcamiento.
	 *
	 * Si un vehículo entró en el aparcamiento después de la fecha dada, no
	 * se tiene en cuenta para el cálculo.
	 *
	 * Los cálculos se realizarán apoyándose en {@link Date#getTime()}.
	 *
	 * @param reference fecha de referencia para el cálculo.
	 * @return valor del aparcamiento con esa referencia.
	 */
	public double getAmountWithReferenceDate(Date reference);

	/**
	 * Indica el coste por minuto de este aparcamiento.
	 *
	 * @return coste por minuto de este aparcamiento.
	 */
	public double getCostPerMinute();
}
```

```
public class Space {

	private Vehicle vehicle;

	private Date timeOfEntry;

	public Space(Date toe, Vehicle v) {
		setTimeOfEntry(toe);
		setVehicle(v);
	}

	public Vehicle getVehicle() {
		return vehicle;
	}

	public void setVehicle(Vehicle vehicle) {
		this.vehicle = vehicle;
	}

	public Date getTimeOfEntry() {
		return timeOfEntry;
	}

	public void setTimeOfEntry(Date timeOfEntry) {
		this.timeOfEntry = timeOfEntry;
	}

	//		formato con la zona horaria, e.g. "01/03/2017 10:50:56 +0100"
	private static final DateFormat format = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss Z");

	@Override
	public String toString() {
		// TODO Auto-generated method stub
		return "{ \"Vehículo\":" + vehicle.toString() + ",
```

```
                \"FechaDeEntrada\":\"" + format.format(timeOfEntry) + "\"}";
        }
}
```

```
public interface Vehicle {

        /**
         * Devuelve la matrícula de este vehículo.
         *
         * @return matrícula de este vehículo.
         */
        String getRegistration();
}
```

4. En el **proyecto ed_a001_2023, hay que completar**.

    a. El paquete **ule.edi.parking**, contiene dos ficheros java a completar (siguiendo los comentarios situados en la definición del interface que implementa):

        i. **ParkingArrayImpl**: clase que implementa el interface Parking. Se implementa el parking mediante un array de objetos de tipo Space, que representa una plaza de aparcamiento (con un atributo de tipo Vehicle, y otro atributo de tipo fecha que almacena el momento de entrada de ese vehículo al parking).

        ii. **ParkingArrayImplTests**: archivo de tests a completar con más tests de alumno. (A la vez que se van desarrollando el código de la práctica se deben crear las correspondientes clases de pruebas JUnit 4 (cuyo nombre debe acabar en Test) para ir comprobando su correcto funcionamiento.

    b. El paquete **ule.edi.parking.vehicles** contiene una clase para cada uno de los tipos de vehículo: Car, Caravan y Motorcycle. TODAS ELLAS IMPLEMENTAN EL INTERFACE Vehicle por lo que pueden ser asignadas como valor del atributo vehicle de la clase Space.

    En estas tres clases hay que redefinir el método equals.

    NOTA: Dos vehículos se considerarán iguales si son del mismo tipo de clase y su matrícula es la misma.

8. **Además se deberá entregar en agora.unileon.es la versión final de la práctica** (proyecto exportado como zip) que deberá coincidir con la última evaluación realizada en la plataforma de evaluación. La semana que viene se completará la especificación de esta práctica.

FECHA LIMITE de entrega de la práctica A001-2023: 10de Marzo de 2023 23:55