Programación II

Curso 21-22

Práctica 4

1 Cuestiones previas

En esta práctica modificaremos las clases de la práctica anterior y añadiremos alguna nueva para introducir el concepto de herencia de la Programación Orientada a Objetos en la aplicación que estamos desarrollando (gestión de un parking). Guarda en P4 todos los ficheros fuente y **.txt** de esta práctica y los **.class** generados, dentro del paquete P4.

2 Actividad 1: Crear el árbol de herencia de la clase Car

En esta actividad vas a crear el árbol de herencia de las clases relacionadas con los coches. Para ello:

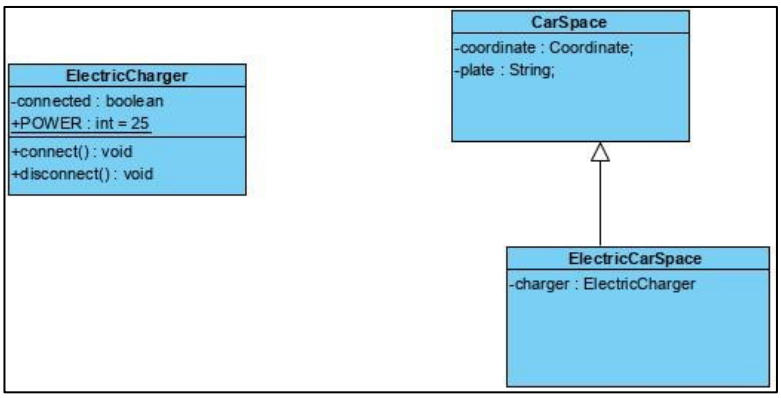
- **Superclase `Car`**. Crea una nueva clase `Car`. Incorpora en ella todos los atributos y métodos (estáticos y de instancia) comunes a las 3 clases `CombustionCar`, `ElectricCar`, e `HybridCar`.
  - Los atributos de instancia no serán accesibles desde el exterior de la clase. Define un constructor vacío y otro completo (con argumentos para los atributos de la clase).
  - Añade a la clase `Car` la constante:

    private static final long serialVersionUID = 7999008895737219541L;
  - Crea en la clase el método abstracto `getTotalPower`, que devuelve un entero, cuya funcionalidad será devolver la potencia total del coche (la potencia mecánica, la potencia eléctrica, o la suma de las potencias en el caso del coche híbrido). Como dentro de la clase `Car` no se sabe el tipo de coche, el método debe ser abstracto, ya que no puede saber cómo calcular su potencia.
  - Incluye el método `toString` en la clase `Car`. Para saber lo que debe devolver, lee primero el siguiente punto.
- **Subclases.** Copia las clases `CombustionCar`, `ElectricCar`, e `HybridCar` de la práctica P3b y modifica su código de forma que queden como subclases de la clase `Car`. Los atributos de instancia propios de cada clase también serán inaccesibles desde el exterior de la clase. Además:
  - Añade a las tres clases las siguientes constantes:
    - `CombustionCar` → *private static final long serialVersionUID = 2650292707495938005L;*
    - `ElectricCar` → *private static final long serialVersionUID = 2314952153963792566L;*
    - `HybridCar` → *private static final long serialVersionUID = 7558906366093469276L;*
  - Modifica el código de sus constructores de forma que inicialicen los atributos de la superclase `Car`.
  - Renombra su método `toText` a `toString`, de tal forma que los métodos `toString` de las hijas usen el del padre para lograr el mismo efecto que tenían antes.
  - Implementa en las tres subclases el método `getTotalPower`, con el código correspondiente para devolver la potencia total del coche según su tipo.

3 Actividad 2. Plazas con cargador

Copia las clases `Coordinate` y `CarSpace` de la práctica P3. En la clase `Coordinate`, renombra el método `esIgualA` para llamarlo `equals`, sobreescribiendo el correspondiente método de la clase `Object`.

Para modelar una plaza con cargador para eléctricos e híbridos (para cargar la batería durante el tiempo que el coche esté aparcado), crearemos las clases del siguiente diagrama UML:



- Clase `ElectricCharger`, para modelar el cargador, con atributos:
  - **connected**. Un valor `boolean` que indica si el cargador está conectado (`true`) o no (`false`).
  - **POWER**. Un valor entero constante, que indica la potencia del cargador. Todos los cargadores tendrán la misma potencia de 25 KW.

  Añade a esta clase un constructor vacío (en el que el atributo **connected** se inicialice a `false`), y también los métodos:

  - `connect()`. Este método debe poner el atributo **connected** a `true`.
  - `disconnect()`. Este método debe poner el atributo **connected** a `false`.

- Clase `ElectricCarSpace`**.** Subclase de la clase `CarSpace`, con el siguiente atributo extendido para incluir un cargador:
  - **charger**. Un objeto de clase `ElectricCharger`**.**

  Crea el constructor para esta subclase, que cree y asigne el objeto **charger** e inicialice los atributos de su superclase `CarSpace`**.**

4 Actividad 3. Trabajando con un único array general de coches

Crea la clase `CarDB`. En esa clase:

- Crea un único array denominado `cityCars` de objetos de clase *Car*, de tamaño 100 coches.
- Copia a esta clase el método `readCityCarsFile` de la clase P3a. Modifica su código para que lea las líneas del fichero `cityCars.txt`, cree objetos `Car`, y los guarde en el nuevo array `cityCars`. Al igual que en la practica P3a añade el coche al array siempre y cuando todos sus datos sean correctos.
- Añade un método `Car getCarFromPlate(String)` que reciba una matrícula y devuelva el objeto `Car` del array `cityCars` correspondiente a esa matrícula.
- Igual que en la práctica P3a, añade el método `computeTotalPower`, que calcule y devuelva la potencia total de todos los coches, llamando al método `getTotalPower` de cada coche que añadimos en la actividad 1.
- Crea el método `computeAverageBatteryLevel` para calcular y devolver el nivel de batería medio de todos los coches eléctricos e híbridos.

  ```
  public float computeAverageBatteryLevel () {...}
  ```

  Para ello debes recorrer el array de coches `cityCars`. Fíjate que sólo puedes pedir el nivel de batería a los coches que tengan batería. Para ello, usa el operador de Java `instanceof`. Este operador devuelve `true` si su operando por la izquierda es un objeto de la clase definida por su operando por la derecha. Es decir, la operación:

  ```
  objetoCar instanceof HybridCar
  ```

  devuelve `true` sólo si `objetoCar` es de tipo `HybridCar`.

5 Actividad 4. Modificando la clase Parking

Copia la clase `Parking` de la practica P3b, y modifica su código para tener en cuenta los cambios introducidos.

5.1 Creación del parking

Modifica el constructor de la clase `Parking` para que, al inicializar el array de plazas, si la plaza corresponde a un coche eléctrico o híbrido se añada al array `carSpaces` un objeto de clase `ElectricCarSpace`.

5.2 Especificación del fichero de entradas y salidas

Eliminamos de la especificación de la practica P3b el campo de `typeES`. Así, cada línea del fichero tendrá ahora el siguiente formato (además de comentarios):

```
<typeES>;<plate>
```

- **typeES***:* un carácter que indica si es una entrada (I) o una salida (O).
- **plate**: matrícula del coche que entra/sale.

Un ejemplo, con una entrada y una salida, sería:

```
I;2408FKL
O;2408FKL
```

5.3 Entradas al parking

Para implementar la entrada de un coche al parking, cambia el método `carEntry` por el siguiente:

```
public void carEntry (Car car) {...}
```

A partir del objeto `Car` recibido, mediante el operador `instanceof`, averigua de qué tipo es para saber dónde aparcarlo. Si es un eléctrico o híbrido debes aparcarlo en las plazas reservadas para ellos, y conectar el cargador.

5.4 Salidas del parking

Actualiza el código del método `carDeparture`, para que, en el caso de que la plaza corresponda a la de un coche eléctrico o híbrido, se desconecte el cargador.

6 Actividad 5. Creando la clase principal

Crea la clase P4, que contendrá el método `main` y será la clase principal de la práctica. En el método `main`:

- Recibe y lee tres argumentos, los nombres de tres ficheros: `file1` (un fichero existente con la estructura y contenido de un parking), `file2` (un fichero existente de entradas y salidas), y `file3` (el nombre del fichero en el que guardar el resultado de la actualización del parking).
- Crea un objeto `cdb` de la clase `CarDB` e invoca su método `readCityCarsFile` para leer el fichero `cityCars.txt` con los coches y que estos se guarden en el array `cityCars` del objeto creado `cdb`.
- Invoca el método `computeTotalPower` del objeto `cdb` para averiguar la potencia total de los coches, y muestra el resultado por pantalla según formato *"Total power = 9999".*
- Invoca el método `computeAverageBatteryLevel` del objeto `cdb` para averiguar el nivel de batería medio de todos los coches eléctricos e híbridos, y muestra el resultado por pantalla según el formato "*Median Battery Charge Level = resultado*".
- Crea un objeto de la clase `Parking` (asígnalo a una variable global) a partir del fichero `file1`.
- Añade a la clase P4 el método `processIO (String fileName)`, que lee el fichero de entradas/salidas `fileName` y actualiza el `parking` invocando a `carEntry` y `carDeparture` para cada movimiento.

  En el caso de las entradas, tras leer una matrícula del fichero, tendrás que llamar al método `getCarFromPlate` del objeto `cdb` para obtener el objeto `Car` con el que invocar a `carEntry`.
- Actualiza el parking a partir del fichero de entradas/salidas `file2` (llamando a `processIO(file2)`), y lo guarda, invocando el método `saveParking` con `file3`.