

# (SIMULACIÓN PARKING) PROGRAMACIÓN ORIENTADA A OBJETOS BÁSICA

Angel Maroto Chivite  
19/01/2023

# Índice

## Contenido

Índice .....	1
1. Java & Kotlin .....	2
2. Ejecución de la simulación .....	2
2.1. Instanciar objetos .....	2
2.2. Recorremos la matriz del parking .....	2
2.3. Introducir vehículos en el parking.....	2
2.3. Desplegar Menú .....	4
2.3.1 Continuar.....	4
2.3.2 Listado Parking .....	4
2.3.3 Recaudación Total .....	5
2.3.4 Búsqueda vehículo .....	5
2.3.5 Finalizar simulación .....	5
2.4. Verificar finalización de simulación.....	6
2.4.1 Vehículos aparcados.....	6
2.4.2 Parking completo .....	7
3. Pruebas unitarias.....	8
4. Diseño implementado .....	9
4.1 Enums Class .....	9
4.1.1 Color .....	9
4.1.2 TipoVehiculo.....	9
4.2 Factories Class .....	9
4.2.1 ConductorFactory.....	9
4.2.2 VehiculoFactory.....	9
4.2.3 ParkingFactory.....	9
5. Models Class.....	9
5.1 Conductor.....	9
5.2 Vehículo.....	9
5.3 Parking.....	9
5.4 Vigilante.....	9

## 1. Java & Kotlin

Primero realicé la práctica en Java, desde mi punto de vista la calidad de código está mejor en la versión de Java.

Para la versión de Kotlin me he apoyado mediante el conversor del IntelliJ, después he realizado cambios puntuales para eliminar los errores de la conversión.

(La versión de Kotlin está sujeta a cambios de mejora)

## 2. Ejecución de la simulación

### 2.1. Instanciar objetos

Instanciamos Parking gracias a ParkingFactory que nos ha generado aleatoriamente un Parking

Creamos un almacén de conductores para tener un control sobre la simulación de cuantos conductores con sus vehículos podrán probar entrar en el Parking, introducimos en el almacén gracias a ConductorFactory y VehiculoFactory.

Le vigilante no hace falta instanciarlo, ya que como querré utilizar siempre el mismo, con llamar a la clase es suficiente.

### 2.2. Recorremos la matriz del parking

Recorremos la matriz del parking a la vez que nos desplazamos por el almacén de conductores y sus correspondientes vehículos.

### 2.3. Introducir vehículos en el parking

Si la posición de la matriz presenta un Vehículo.tipoVACIO y favorece a las siguientes condiciones, introducimos un vehículo en el Parking y el Vigilante informará donde podrá aparcar el vehículo y se imprimirá en consola el parking:

#### 2.3.1 Introducir vehículo camión

Si el vehículo que estamos intentando introducir es un camión ocupará 4 posiciones en el parking, ya que es más grande.

Si no entra nuestro camión en la fila que nos encontremos, lo introducirá en la siguiente

#### 2.3.2 Introducir vehículo coche

Si el vehículo que estamos intentando introducir es un coche ocupará 2 posiciones en el parking, ya que es más grande.

Si no entra nuestro coche en la fila que nos encontremos, lo introducirá en la siguiente

#### 2.3.3 Introducir vehículo moto, bici, patinete

Si el vehículo que estamos intentando introducir es una moto, bici o patinete ocupará una posición en el parking, ya que presenta un tamaño estándar.

```

CONDUCTOR→ Hola soy: Elena
DNI: 216281624K, vehículos aparcados: 1
VEHÍCULOS EN PROPIEDAD
VEHÍCULO 1 Vehiculo{id='01', matricula='YQBHK-0764', anno='1994', tipo=BICI, estado=APARCADO}
VEHÍCULO 2 Vehiculo{id='02', matricula='YDCMK-8872', anno='2012', tipo=MOTO, estado=NO_APARCADO}

VIGILANTE→ Hola soy Angel puedes aparcar tu B en el parking fila:1 columna:1 por un precio de 3.75€

===== LEYENDA =====
Camión (N), Coche (C), Moto (M), Bici (B), Patinete (P)
Espacio vacío: (--)
ID Vehículo: (TipoVehiculo nº)
===== PARKING =====

    B01 --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---
    --- --- --- --- --- --- ---

=====

```

Disponemos de un contador para pasar al siguiente vehículo, en caso de sobrepasar los vehículos que tenga en propiedad un Conductor pasamos al siguiente Conductor.

### 2.3. Desplegar Menú

Damos la opción al usuario si quiere desplegar un menú extra sobre la información del parking

```
¿Quieres desplegar un menú interactivo? (S/N)
```

```
(USUARIO) → Selecciona la opción deseada:
```

```
1: Continuar simulación
2: Listado Parking (ordenados: antigüedad ascendete)
3: Recaudación total (caja actual del Parking)
4: Encontrar tu vehículo (matricula)
0: Finalizar simulación
```

#### 2.3.1 Continuar

Recorreremos nuevamente el parking para introducir vehículos

#### 2.3.2 Listado Parking

Nos demuestra un listado de todos los vehículos que disponemos en el parking, un contador de cuantos se encuentra aparcados y el parking actual:

```
Disponemos de un total actual de: 1 vehículos
Vehiculo{id='01', matricula='YQBHK-0764', anno='1994', tipo=BICI, estado=APARCADO}
===== LEYENDA =====
Camión (N), Coche (C), Moto (M), Bici (B), Patinete (P)
Espacio vacío: (--)
ID Vehículo: (TipoVehiculo n°)
===== PARKING =====
  B01 --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
=====
Volver al menú (S)
```

### 2.3.3 Recaudación Total

Calcula el total que hemos recaudado en la simulación, como hemos introducido un vehículo, solo disponemos de la recaudación de 3,75€

```
La recaudación total actual es de: 3.75€
```

```
Volver al menú (S)
```

### 2.3.4 Búsqueda vehículo

Mediante la matrícula se realiza la búsqueda y nos devuelve la posición donde se encuentre:

```
Puedes realizar la búsqueda de tu vehículo mediante → matrícula:
```

```
YQBHK-0764
```

```
Tu vehículo se encuentra en la fila 1 columna 1
```

```
¿Quieres realizar de nuevo la búsqueda? (S/N)
```

### 2.3.5 Finalizar simulación

En caso de que el usuario quiera terminar la simulación en cualquier momento.

## 2.4. Verificar finalización de simulación

### 2.4.1 Vehículos aparcados

En el caso de que hayamos introducido todos los vehículos de todos los conductores, finalizará nuestra simulación.

Creo 3 conductores, para que se almacenen todos los vehículos:

```
// 3 conductores
Conductor[] almacenConductores = new Conductor[3];
for (int i = 0; i < almacenConductores.length; i++) {
    almacenConductores[i] = ConductorFactory.create();
}
```

```
===== LEYENDA =====
Camión (N), Coche (C), Moto (M), Bici (B), Patinete (P)
Espacio vacío: (--)
ID Vehículo: (TipoVehículo n°)
===== PARKING =====

  B01 N02 N02 N02 N02 C03 C03 M05 M07
  N04 N04 N04 N04 M08 --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---
  --- --- --- --- --- --- --- ---

=====
```

Angel ha hecho un buen trabajo de vigilante y ha terminado su jornada

Process finished with exit code 0

### 2.4.2 Parking completo

En el caso de que el parking esté completo, finalizará nuestra simulación, con el aviso del Vigilante.

Creo 25 conductores, para que se rellene el parking:

```
// 25 conductores
Conductor[] almacenConductores = new Conductor[25];
for (int i = 0; i < almacenConductores.length; i++) {
    almacenConductores[i] = ConductorFactory.create();
}
```

```
===== LEYENDA =====
Camión (N), Coche (C), Moto (M), Bici (B), Patinete (P)
Espacio vacío: (--)
ID Vehículo: (TipoVehiculo n°)
===== PARKING =====

M01 N02 N02 N02 N02 M03 M04 B06 B07
C05 C05 N08 N08 N08 N08 C09 C09 M11
N10 N10 N10 N10 M12 C13 C13 P19 B20
N14 N14 N14 N14 N21 N21 N21 N21 B22
P23 N24 N24 N24 N24 B25 C27 C27 M28
N26 N26 N26 N26 N29 N29 N29 N29 B30
B31 C32 C32 M33 M34 B36 B37 B38 B45
N35 N35 N35 N35 M46 M47 C48 C48 M49
C50 C50 C51 C51 C52 C52 C53 C53 P54

=====

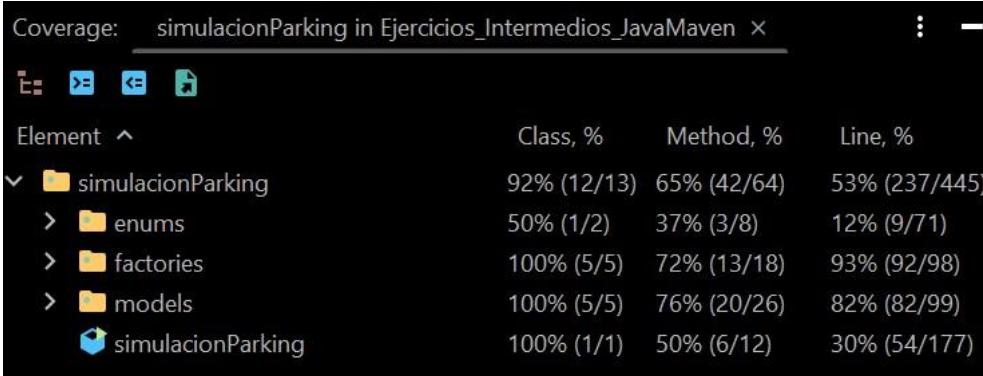
VIGILANTE→ Hola soy Angel, no puedes aparcar, tenemos el Parking completo!

Process finished with exit code 0
```



### 3. Pruebas unitarias

Los Enum Class, Record y Data Class, y el código de impresión en pantalla no se encuentran probados, el resto del código está probado:



The screenshot shows the coverage report for the 'simulacionParking' module. The table lists the following data:

Element	Class, %	Method, %	Line, %
simulacionParking	92% (12/13)	65% (42/64)	53% (237/445)
enums	50% (1/2)	37% (3/8)	12% (9/71)
factories	100% (5/5)	72% (13/18)	93% (92/98)
models	100% (5/5)	76% (20/26)	82% (82/99)
simulacionParking	100% (1/1)	50% (6/12)	30% (54/177)

Para la versión en Java he utilizado el gestor de dependencias Maven mediante la dependencia de Test Junit5.

Para la versión de Kotlin he utilizado el gestor de dependencias Gradle mediante la dependencia de Test Junit5.

## 4. Diseño implementado

### 4.1 Enums Class

#### 4.1.1 Color

Colores para mejorar la estética de la impresión en pantalla

#### 4.1.2 TipoVehiculo

Tipo de vehículos que podemos encontrar en nuestra simulación

(Este apartado tengo previsto reformular el diseño, mediante un abstract class Vehículo, y heredar de él sus campos a las distintas clases (Camion, Coche, Moto, Bici, Patinete)

### 4.2 Factories Class

Creación de objetos aleatorios mediante factories.

#### 4.2.1 ConductorFactory

Presenta un campo de clase entero ID, para aumentar el ID que pasaremos al constructor una clase Conductor.

#### 4.2.2 VehiculoFactory

Presenta un campo de clase entero ID, para aumentar el ID que pasaremos al constructor una clase Vehiculo.

#### 4.2.3 ParkingFactory

## 5. Models Class

Objetos que instanciaré a lo largo de la ejecución del programa

### 5.1 Conductor

He tenido que asignar un constructor secundario, para poder instanciar correctamente el objeto.

### 5.2 Vehículo

He tenido que asignar un constructor secundario, para poder instanciar correctamente el objeto.

Y por defecto su estado será NO\_APARCADO.

### 5.3 Parking

Record/Data Class, donde solo me interesa que almacene los campos y me imprima en pantalla el Parking.

### 5.4 Vigilante

Vigilante va a ser único por tanto tendré su nombre fijo.