**Proyecto Tema 2 de múltiples procesos por hilos diferentes de un PARKING con el registro de entrada/salida de vehículos.**

Índice

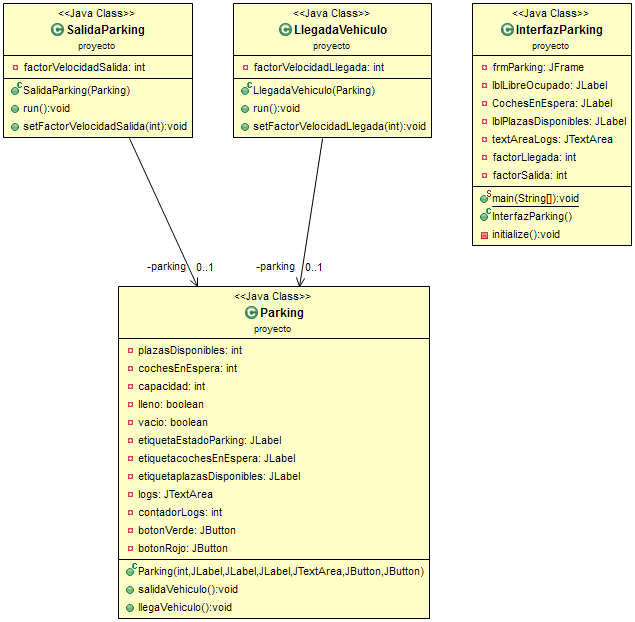
[**UML del proyecto PARKING** 3](#_Toc438141175)

[**Interface grafica** 4](#_Toc438141176)

[**Código** 5](#_Toc438141177)

[**GitHubs** 10](#_Toc438141178)

# **UML del proyecto PARKING**



La aplicación consta de cuatro clases:

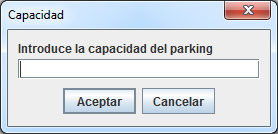
-Parking: Clase que actúa de buffer y controla la entrada y salida de vehículos del parking

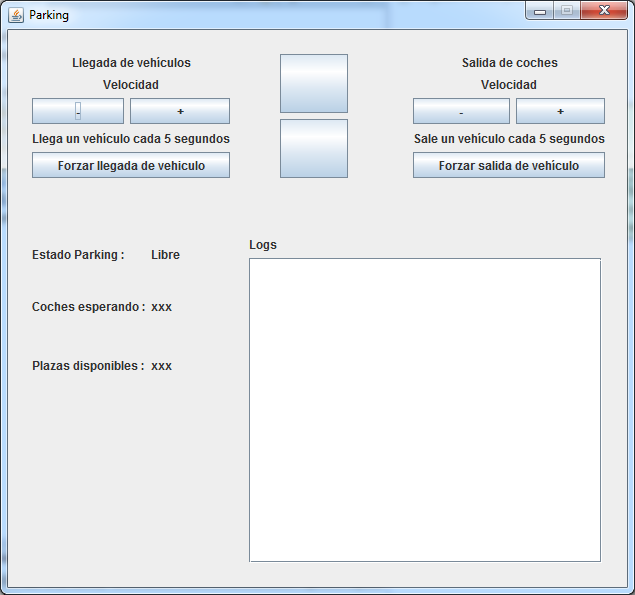
-LlegadaVehiculo y SalidaParking : Controlan dos hilos independientes que

funcionan en bucle llamando a los métodos de la clase parking.

-InterfazParking: Contiene el main, que inicializa la interfaz y las demás clases.

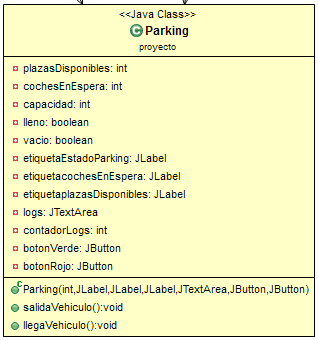
# **Interface grafica**





# **Código**

Parking.java



Dispone de todos los parámetros para gestionar el parking como su capacidad máxima, número de plazas disponibles, coches en espera para entrar.

El parking se inicializará indicando su máxima capacidad, las plazas disponibles al inicializarlo serán la mitad de su capacidad, y el resto de atributos que recibe por parámetros son los elementos de la interfaz que tiene que manipular.

**public** Parking(**int** capacidad, JLabel etiquetaEstadoParking, JLabel etiquetacochesEnEspera, JLabel etiquetaplazasDisponibles, JTextArea logs,

JButton botonVerde, JButton botonRojo) {

//partimos de que el parking esta vacio y que tiene todas sus plazas libres

**this**.capacidad = capacidad;

**this**.plazasDisponibles = capacidad/2;

**this**.lleno = **false**;

**this**.vacio = **true**;

**this**.etiquetacochesEnEspera = etiquetacochesEnEspera;

**this**.etiquetaEstadoParking = etiquetaEstadoParking;

**this**.etiquetaplazasDisponibles = etiquetaplazasDisponibles;

**this**.logs = logs;

**this**.botonVerde=botonVerde;

**this**.botonRojo=botonRojo;

}Los métodos principales de la clase parking son:

-salidaVehiculo

-llegaVehiculo

**public** **synchronized** **void** salidaVehiculo(){

//en el caso de que el estado del parking sea vacio (tiene todas sus plazas disponibles)

//el hilo esperará

**while**(**this**.vacio){

**try** {

wait();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

//cuando haya entrado un vehiculo y por tanto, haya ocupado una plaza, el hilo que utilice este metodo

//continuará generando plazas

**this**.plazasDisponibles++;

etiquetaEstadoParking.setText("Libre");

botonVerde.setBackground(Color.***GREEN***);

botonRojo.setBackground(Color.***GRAY***);

**this**.lleno = **false**;

//si el numero de plazas disponibles es igual a su capacidad significa que esta vacio y por tanto no puede generar mas plazas

**this**.vacio = **this**.plazasDisponibles == **this**.capacidad;

String aux = logs.getText();

logs.setText(contadorLogs+"- Ha salido un vehiculo\n"+aux);

contadorLogs++;

**if**(cochesEnEspera>0){

**int** vehiculosEntraron = plazasDisponibles;

cochesEnEspera = cochesEnEspera - plazasDisponibles;

plazasDisponibles = 0;

**this**.lleno = **true**;

botonVerde.setBackground(Color.***GRAY***);

botonRojo.setBackground(Color.***RED***);

**this**.vacio = **false**;

etiquetaplazasDisponibles.setText(plazasDisponibles+"");

etiquetaEstadoParking.setText("Lleno");

etiquetacochesEnEspera.setText(cochesEnEspera+"");

aux = logs.getText();

logs.setText(contadorLogs+" - Vehiculos que entraron : " +vehiculosEntraron+"\n"+aux);

contadorLogs++;

}

etiquetaplazasDisponibles.setText(""+**this**.plazasDisponibles);

}

Este método se ocupa de generar plazas disponibles en el parking y está sincronizado por lo que no pueden acceder dos simultáneamente a él. En primer lugar comprueba de que el parking se encuentra completamente vacío, y en el caso de que lo esté, el hilo que haya lanzado el método esperará. Generará una plaza y volverá a comprobará si hay vehículos esperando para entrar al parking, en el caso de que los haya, dejará entrar tantos como plazas queden libres. También modificará los valores de los atributos con los valores de los atributos del parking con los valores resultantes tras estas operaciones, plazasDisponibles, lleno ...

Y modificará los elementos de la interfaz mostrando estos valores.

Si al final del proceso quedan plazas, el semáforo se mostrará verde, en caso contrario, rojo.

**public** **synchronized** **void** llegaVehiculo(){

String aux = logs.getText();

logs.setText(contadorLogs+" - Ha llegado un vehiculo \n"+aux);

contadorLogs++;

**int** cochesEntraron = 0;

**this**.cochesEnEspera++;

//en el caso de que el numero de plazas disponibles sea mayor que los coches que haya en espera

//entraran todos los que haya en espera y esta cantidad se restará del numero de plazas disponibles

**if**(plazasDisponibles > cochesEnEspera){

plazasDisponibles = plazasDisponibles - cochesEnEspera;

cochesEntraron = cochesEnEspera;

cochesEnEspera = 0;

**this**.vacio = **false**;

}**else**{

//si el numero de coches en espera es igual o superior al numero de plazas disponibles

//se ocuparan todas las plazas y estas se reduciran a 0, volviendo al estado de lleno

//y los coches que quedaran en espera serán la cantidad anterior menos las plazas disponibles en el parking

cochesEntraron = plazasDisponibles;

cochesEnEspera = cochesEnEspera - plazasDisponibles;

plazasDisponibles = 0;

**this**.lleno = **true**;

botonVerde.setBackground(Color.***GRAY***);

botonRojo.setBackground(Color.***RED***);

etiquetaEstadoParking.setText("Lleno");

**this**.vacio = **false**;

}

//como ambas situaciones modifican el estado de vacio a false, despertarán si hay algun hilo intentando producir nuevas plazas para

//que sigan haciendolo

**if** (cochesEntraron>0){

aux = logs.getText();

logs.setText(contadorLogs+" - Vehiculos que entraron : "+ cochesEntraron+"\n"+aux);

contadorLogs++;

}

notifyAll();

etiquetacochesEnEspera.setText(""+**this**.cochesEnEspera);

etiquetaplazasDisponibles.setText(""+**this**.plazasDisponibles);

}

Este método se ocupa de gestionar los vehículos que llegan al parking y está sincronizado por lo que no pueden acceder dos simultáneamente a él. Utiliza como buffer una variable interna (cochesEnEspera) que incrementa en 1 cada vez que se le llama. Dejará entrar a tantos vehículos como quepan en ese momento en el parking, modificando los correspondientes atributos y modificará los elementos de la interfaz con estos valores. Si al final del proceso quedan plazas, el semáforo se mostrará verde, en caso contrario, rojo, volverá a poner en funcionamiento al hilo que lanzo al método salidaParking por que en cualquier caso el parking no estará vacío.

LlegadaVehiculo.java y SalidaParking.java

**public** **class** LlegadaVehiculo **extends** Thread {

**private** Parking parking;

**private** **int** factorVelocidadLlegada;

**public** LlegadaVehiculo(Parking parking) {

**this**.parking = parking;

**this**.factorVelocidadLlegada = 5;

}

@Override

**public** **void** run() {

**while** (**true**){

**try** {

**if**(factorVelocidadLlegada==10){

Thread.*sleep*(1000);

}

**else**{

parking.llegaVehiculo();

Thread.*sleep*(1000\*factorVelocidadLlegada);

}

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **void** setFactorVelocidadLlegada(**int** factorVelocidadLlegada) {

**this**.factorVelocidadLlegada = factorVelocidadLlegada;

}

}

**public** **class** SalidaParking **extends** Thread{

**private** Parking parking;

**private** **int** factorVelocidadSalida;

**public** SalidaParking(Parking p) {

**this**.parking = p;

**this**.factorVelocidadSalida = 5;

}

@Override

**public** **void** run() {

**while** (**true**){

**try** {

**if**(factorVelocidadSalida==10){

Thread.*sleep*(1000);

}**else**{

parking.salidaVehiculo();

Thread.*sleep*(1000\*factorVelocidadSalida);

}

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**public** **void** setFactorVelocidadSalida(**int** factorVelocidadSalida) {

**this**.factorVelocidadSalida = factorVelocidadSalida;

}

}

Estas clases extienden de thread y se ocupan de gestionar la frecuencia con las que hacen las llamadas a sus respectivos métodos de la clase parking en función del atributo (factorVelocidad) que se modificará dinámicamente desde la interfaz a través del set de este atributo en cada una de estas clases. Los valores de este atributo estarán comprendidos entre 1 y 10 indicando el número de segundos que dormirá el hilo, en el caso de que el valor sea 10, el hilo no hará nada, solo dormirse sin hacer llamadas al método.

InterfazParking.java

Por defecto solicitará un valor para indicar la capacidad del parking, el valor tiene que ser un numero entero entre 1 y 999, si no se introduce esto, el programá finalizará

**if**(capacidadParkingString == **null** || !capacidadParkingString.matches("^\\d{1,3}")|| capacidadParkingString.equals("0")){

JOptionPane.*showMessageDialog*(frmParking, "Debe introducir una capacidad \n entre 1 y 999");

System.*exit*(0);

}

Inicializará la clase Parking pasando como parámetros los elementos que se encuentran en esta clase y la capacidad solicitada. Inicializara las clases LlegadaVehiculo y SalidaParking y controlará el atributo que controla el tiempo que duermen con los

SalidaParking

JButton btnNewButton = **new** JButton("-");

btnNewButton.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if**(factorSalida<10){

factorSalida++;

salidaparking.setFactorVelocidadSalida(factorSalida);

lblSaleUnVehículo.setText("Sale un vehículo cada "+factorSalida+" segundos");

**if**(factorSalida==10){

lblSaleUnVehículo.setText("Han dejado de salir vehiculos");

}

}

}

});

LlegadaParking

JButton btnDisminuirVelocidadLlegada = **new** JButton("-");

btnDisminuirVelocidadLlegada.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**if**(factorLlegada<10){

factorLlegada++;

llegadavehiculo.setFactorVelocidadLlegada(factorLlegada);

lblLlegaUnVehculo.setText("Llega un veh\u00EDculo cada "+factorLlegada+" segundos");

**if**(factorLlegada==10){

lblLlegaUnVehculo.setText("Han dejado de llegar vehiculos");

}

}

}

});

Los botones Forzar Llegada o Forzar Salida generan una clase interna que se ejecuta haciendo una sola llamada a los metodos

JButton btnForzarLlegada = **new** JButton("Forzar llegada de vehiculo");

btnForzarLlegada.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

p.llegaVehiculo();

}

}).start();

}

});

# **GitHubs**

Proyecto: <https://github.com/berna87/ProyectoTema2PSS>

Bernabé Fernández Ogayar: <https://github.com/berna87>

Luis Usero Reyes: <https://github.com/Luis54>

Jorge Lapeña Antón: <https://github.com/jorgela92>