Universidad Nacional de Quilmes

Departamento de Ciencia y Tecnología

Informe

Sistema de estacionamiento medido

Alumnos:

Ferro Ignacio

Neira Guitera Tomas

Rodriguez Joaquin

Contacto:

Ferro Ignacio: nachoferro112233@gmail.com
Neira Guitera Tomas: tomasneira181@gmail.com
Rodriguez Joaquin: jdr123r@gmail.com

Decisiones de desarrollo

Registrar todas las compras

Para llevar a cabo este punto lo que desarrollamos fue la clase Ticket, con las subclases TicketDeRecargaCelular y TicketDeHoras, que nos permite almacenar en el SistemaDeEstacionamientoMedido cada una de las compras que se realizan en un PuntoDeVenta.

Registrar los estacionamientos

Para registrar los estacionamientos dentro del sistema desarrollamos la clase Estacionamiento, con las subclases EstacionamientoPorCompraPuntual y EstacionamientoMedianteApp que nos permite almacenar en el SistemaDeEstacionamientoMedido cada uno de los estacionamientos que se generan.

Inicializacion de una AplicacionUsuario

En el mensaje de creación de la aplicación se establece el estado SinEstacionamientoVigente, y las notificaciones activadas por defecto.
Además debe recibir como argumento un modo de uso y un celular, ese número no va a cambiar en la instancia.

Detalles de implementación:

En el mensaje iniciarEstacionamientoSEM(patente)
La patente que se recibe como argumento es seteada en el atributo patente, para que si se cambia al modo de uso automático ya se encuentra la patente registrada.

En el mensaje finalizarEstacionamientoSEM()
Se finaliza el estacionamiento con el número que se encuentra registrado en la aplicación.

El funcionamiento fue pensado de esta manera para que un usuario pueda iniciar estacionamientos con distintos vehículos.

Funcionamiento de la AplicacionUsuario con los estados:

Métodos driving() y walking():

Los métodos driving() y walking() los implementa la AplicacionUsuario pero son delegados al estado de la forma: estado.driving(this) y estado.walking(this).

La clase EstadoDeEstacionamiento es una clase abstracta que define los métodos sin comportamiento:

- driving(AplicacionUsuario).
- walking(AplicacionUsuario).
- notificarPosibleInicioEstacionamiento(AplicacionUsuario).
- notificarPosibleFinEstacionamiento(AplicacionUsuario).

El estado EstacionamientoVigente ignora el mensaje walking(AplicacionUsuario). Pero cuando recibe el mensaje driving(AplicacionUsuario a) debe enviarle a la aplicación el mensaje: notificarPosibleInicioEstacionamiento(String).

El estado SinEstacionamientoVigente ignora el mensaje driving(AplicacionUsuario). Pero cuando recibe el mensaje walking(AplicacionUsuario) debe enviarle a la aplicación el mensaje: notificarPosibleFinEstacionamiento().

Modo manual:

iniciarEstacionamiento le delega el mensaje al estado:
estado.iniciarEstacionamiento(Aplicacionusuario, String).

El estado EstacionamientoVigente ignora el mensaje.

El estado SinEstacionamientoVigente cuando recibe el mensaje debe enviarle a la instancia que recibe en el parámetro el mensaje: iniciarEstacionamientoSEM(String).

Dentro del método iniciarEstacionnamietnoSEM se realiza un: this.setEstado(new EstacionamientoVigente()) para que rote el estado de no vigente a vigente.

finalizarEstacionamiento le delega el mensaje al estado:
estado.finalizarEstacionamiento(AplicacionUsuario).

El estado SinEstacionamientoVigente ignora el mensaje.

El estado EstacionamientoVigente cuando recibe el mensaje debe enviarle a la instancia que recibe en el parámetro el mensaje:
finalizarEstacionamientoSEM().

Dentro del método finalizarEstacionnamietnoSEM se realiza un: this.setEstado(new SinEstacionamientoVigente()) para que rote el estado de vigente a no vigente.

Funcionamiento de la AplicacionUsuario con las estrategias:

Modo manual:

La aplicación cuando recibe los métodos:

- notificarPosibleInicioEstacionamiento()
- notificarPosibleFinEstacionamiento()
- activarODesactivarNotificaciones()

Le delega a la estrategia de la forma:

- estrategia.notificiarPosibleInicioEstacionamiento(this)

En los métodos siguientes se debe verificar que las notificaciones se encuentren activas, para poder accionarlas, lo solucionamos de la siguiente manera:

* notificarPosibleInicioEstacionamiento(AplicaionUsuario)
* notificarPosibleFinEstacionamiento(AplicaionUsuario)
Dentro de cada método hay un if con: a.notificacionesActivas() para saber si hacer los print o no.

El método notificarPosibleInicioEstacionamiento(AplicacionUsuario) lo único que realiza es un print de tipo notificación donde se le indica al usuario que debe iniciar el estacionamiento en la app.

El método notificarPosibleFinEstacionamiento(AplicacionUsuario) lo único que realiza es un print de tipo notificación donde se le indica al usuario que debe finalizar el estacionamiento en la app.

El metodo activarODesactivarNotificaciones(AplicacionUsuario a) debe enviarle el mensaje a la app: a.ADNotificaciones()

Modo automático:

El método notificarPosibleInicioEstacionamiento(AplicacionUsuario) le envía a la app el mensaje: a.iniciarEstacionamientoSEM(**null**), la patente que le pasamos por parámetro es null, esto es porque el iniciar el estacionamiento de forma automática se debe realizar con una patente ya registrada con antelación, entonces si la aplicación usuario cuenta con una patente registrada entonces se inicia el estacionamiento y si no le indica al usuario que debe registrar una patente.

Además el método realiza un print indicando que se inició el estacionamiento de forma automática.

El método notificarPosibleFinEstacionamiento(AplicacionUsuario) debe enviarle a la app el mensaje:
a.finalizarEstacionamientoSEM()

Además el método debe hacer un print indicando que se finalizó el estacionamiento de forma automática.

El metodo activarODesactivarNotificaciones(AplicacionUsuario a) no debe realizar nada, ya que cuando la app se encuentra en modo automático no se deben poder desactivar las notificaciones

Patrones de diseño utilizados

Para lograr el desarrollo del trabajo fue necesario implementar tres patrones de diseño, estos fueron:

Un State para definir el estado de un usuario, si se encuentra con un estacionamiento vigente o sin el estacionamiento vigente.
Los roles de cada una de las clases en el desarrollo del estado son:

- AplicacionUsuario es el “context”.
- EstadoDeEstacionamiento es el “state”.
- SinEstacionamientoVigente es uno de los “concrete state”.
- EstacionamientoVigente es el otro de los “concrete state”.

Un Strategy para elegir el modo de uso de la aplicación usuario, las estrategias son ModoDeUsoManual y ModoDeUsoAutomatico.
Los roles de cada una de las clases en el desarrollo del strategy son:

- AplicacionUsuario es el “context”.
- ModoDeUso es el “strategy”.
- Manual es uno de los “concrete strategy”.
- Automatico es el otro “concrete strategy”.

Un Observer para notificar a entidades sobre inicios de estacionamientos, fines de estacionamientos y carga de crédito de los diferente usuarios del sistema.
La clase SistemaDeEstacionamientoMedido es la “observable” y los observadores son las entidades que implementen la interfaz Notificable.
El observer que realizamos es el conocido como “Complex With Listeners”.