Smart Parking

Manejo de recursos escasos

Marcelo Bustos Ramírez
Sebastián Robles González
Cristian Vidal Sepúlveda

Temuco, agosto 2018

Tabla de contenido

Introducción

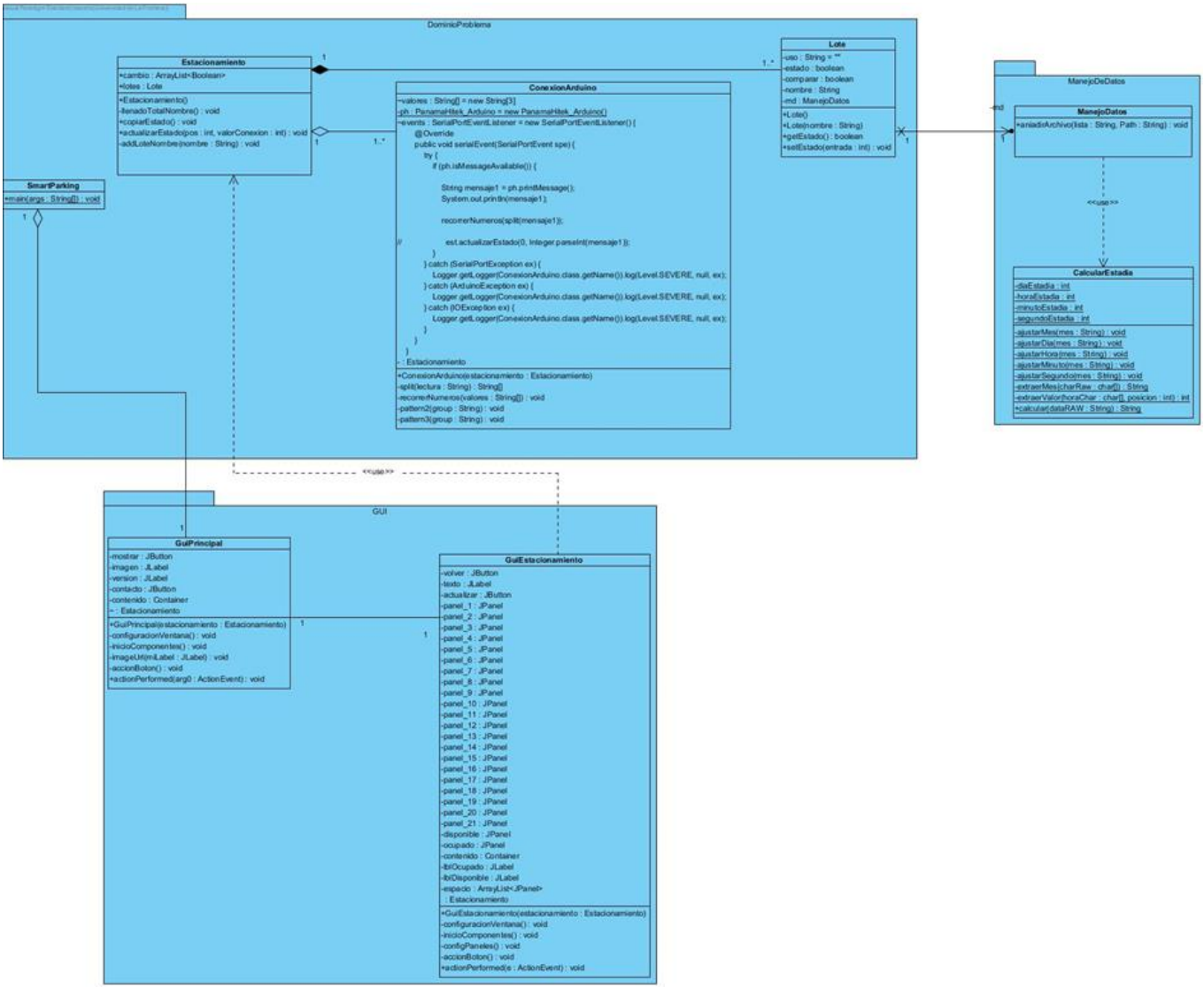
En el mundo existen más de 1,1 billones de vehículos, en Chile más de 5 millones, en la Araucanía más de 220 mil (datos entregados por el instituto nacional de estadística en el año 2016), con el rápido avance de las ciudades y del crecimiento demográfico los espacios para construir estacionamientos se han reducido radicalmente y la demanda de estos crece exponencialmente cada año. Por esta razón se vuelve cada vez más necesario un sistema que sea capaz de optimizar y facilitar su manejo.

Con esto en mente nuestro equipo de trabajo busca desarrollar un sistema que permita detectar y manejar el estado de los espacios de un estacionamiento y con ello facilitar su gestión, navegabilidad además de reducir el tiempo invertido en estacionar. Todo esto con el fin de optimizar el uso de este recurso cada vez más escaso en la sociedad actual.

Para este software nos basamos principalmente en tecnologías de Java y Arduino, para generar un enlace entre software y hardware que nos permitiera obtener datos desde el medio y realizar su tratamiento, la conexión entre Java y Arduino fue gestionada por una biblioteca llamada PanamaHiTek. Adjunto a este informe se entrega una versión de software que ya puede ser manejada por el usuario final.

Diseño UML y Documentación

Diseño UML de clases y paquetes



Documentación Proyecto

Clase	Atributos	Métodos
Smartparking(main)	Ninguno	main():void
ConexionArduino	+valores:String[3] -est:Estacionamiento -ph:PanamaHitek_Arduino +events:SerialEventListener	-split(lectura:String):String[] -recorrerNumeros(valores: String[]): void -pattern2(group: String): void -pattern3(group: String): void
Estacionamiento	+lotes: ArrayList<Lote> +cambio: ArrayList<Boolean>	-llenadoTotalNombre() : void +copiarEstado(): void +actualizarEstado(posicion: int valorConexion: int) : void -addLoteNombre(): void +botonActualizar(camb: ArrayList<Boolean>, lables: ArrayList<JPanel>): void
Lote	-uso: String -estado: boolean -comparar: boolean -md: ManejoDatos -nombre: String	+getEstado(): boolean +setEstado(entrada: int): void
GuiPrincipal	-mostrar:JButton -imagen: JLabel -version: JLabel -contenido: Container +est:Estacionamiento	-configuracionVentana(): void -inicioComponentes(): void -imageURL(miLabel: JLabel): void -accionBoton(): void +actionPerformed(e : ActionEvent) : void

Clase	Atributos	Métodos
GuiEstacionamiento	-volver: JButton -text: JLabel -actualizar: JButton -panel_1: JPanel -panel_2: JPanel -panel_3: JPanel -panel_4: JPanel -panel_5: JPanel -panel_6: JPanel -panel_7: JPanel -panel_8: JPanel -panel_9: JPanel -panel_10: JPanel -panel_11: JPanel -panel_12: JPanel -panel_13: JPanel -panel_14: JPanel -panel_15: JPanel -panel_16: JPanel -panel_17: JPanel -panel_18: JPanel -panel_19: JPanel -panel_20: JPanel -panel_21: JPanel -disponible: JPanel -ocupado: JPanel -contenido: Container -est: Estacionamiento -lblOcupado: JLabel -lblDisponible: JLabel -espacio: ArrayList<JPanel>	-configuracionVentana(): void -inicioComponentes(): void -configPaneles(): void -accionBoton(): void +actionPerformed(e : ActionEvent) : void

Clases	Atributos	Métodos
CalcularEstadia	-diaEstadia -horaEstadia -minutoEstadia -segundoEstadia	-ajustarMes(mes: String): void -ajustarDia(mes: String): void -ajustarHora(mes: String): void -ajustarMinuto(mes: String): void -ajustarSegundo(mes: String): void -extraerMes(charRaw: char[]): String -extraerValor(horaChar : char[], posicion : int) : int +calcular(dataRAW : String) : String
ManejoDatos	Ninguno	+aniadirArchivo(lista : String, Path : String) : void

La clase “SmartParking” es el main del programa, y como tal, su única función es inicializar algunas variables y servir como punto de inicio para el programa.

La clase “ConexionArduino” se encarga de establecer la conexión entre el programa principal (escrito en Java) y el programa controlador del dispositivo Arduino.

La clase “Estacionamiento” sirve como un “almacén”, guardando la información relevante más reciente. Su método “addLoteNombre” agrega un nuevo “Lote” para ser registrado por la clase “Estacionamiento”.

El método “llenadoTotalNombre” itera el método “addLoteNombre” hasta alcanzar la capacidad máxima de la clase “Estacionamiento”.

El método “actualizarEstado”, tal como indica su nombre, actualiza el estado actual de todos los “Lotes” del “Estacionamiento”. Este método es ejecutado a intervalos de tiempo regulares.

El método “botonActuallizar” actualiza la información del “Estacionamiento” de manera manual.

El método “copiarEstado” obtiene el estado actual de los “Lotes” del “Estacionamiento” para ser usado por otros métodos.

La clase “Lote” representa una parte de un “Estacionamiento” y su función es indicar su estado actual. Dado lo anterior, sus únicos métodos son un “getter” y un “setter” para su propio estado (disponible u ocupado).

La clase “GuiPrincipal” es la encargada de crear y mostrar la ventana principal para interacción entre el usuario final y el programa.

El método” configuracionVentana” configura la ventana, “inicioComponentes” inicia las componentes de la ventana, “imageURL” descarga la imagen central que utiliza la ventana.

El método “accionBoton” despliega al usuario información de contacto del desarrollador para poder contactar a este.

El método” actionPerformed” cierra la ventana actual y crea una nueva ventana a partir de la clase “GuiEstacionamiento”.

La clase “GuiEstacionamiento” es la encargada de crear y mostrar una ventana donde se despliega la información más reciente del “Estacionamiento”.

Al igual que la clase anterior, “GuiEstacionamiento posee sus propios métodos “cofiguracionVentana” e “inicioCOmponentes”, los cuales cumplen un rol similar que en la clase anterior.

El método “configPaneles” configura los “JPanel’s” que utiliza “GuiEstacionamiento”.

El método “accionBoton” solicita una actualización del estado actual del “Estacionamiento” y modifica el despliegue en la ventana para que refleje el estado actual.

El método “actionPerformed” cierra la ventana actual y retorna al usuario a la “ventana principal”.

La clase “CalcularEstadia” es un conjunto de atributos y métodos necesarios para estimar la cantidad de tiempo que un “Lote” se encontró “ocupado”.

El método “extraerMes” obtiene el mes en el cual el “Lote” volvió a estar “disponible”.

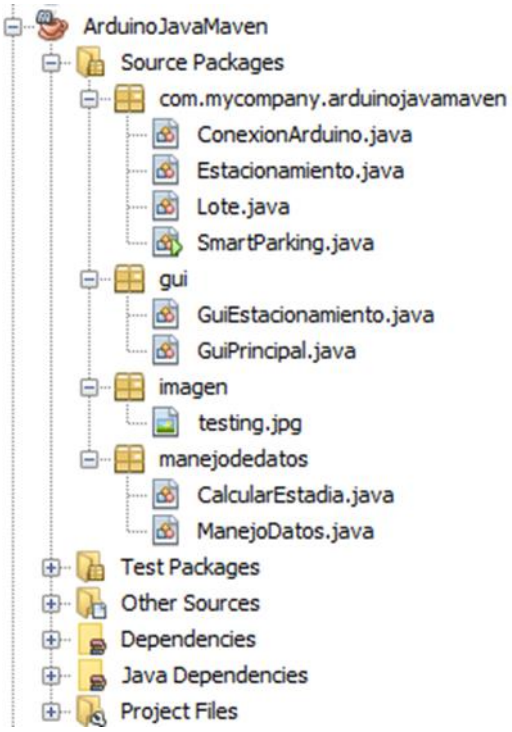
El método “extraerValor” extrae de un “String” el valor numérico del dato solicitado (hora, dia, minuto o segundo).

El método “calcular” se encarga de estimar el “tiempo de estadia” de un vehiculo en un “Lote” determinado. Para ello solicita la información obtenida por “extraerMes” y “extraerValor” y luego asegura valores coherentes con los métodos ajustarMes”, “ajustarDia”, “ajustarHora”, “ajustarMinuto” y “ajustarSegundo”.

Los métodos “ajustarMes”, “ajustarDia”, “ajustarHora”, “ajustarMinuto” y “ajustarSegundo” aplican una corrección a los valores estimados por “calcular”, ya que dicho método opera sin las consideraciones necesarias a la hora de trabajar con múltiples unidades de tiempo.

La clase “ManejoDatos” se encarga de almacenar información relevante en un archivo de texto. Para ello utiliza su único método “aniadirArchivo”.

Estructura del proyecto en NetBeans IDE



Gui’s diseño y navegación

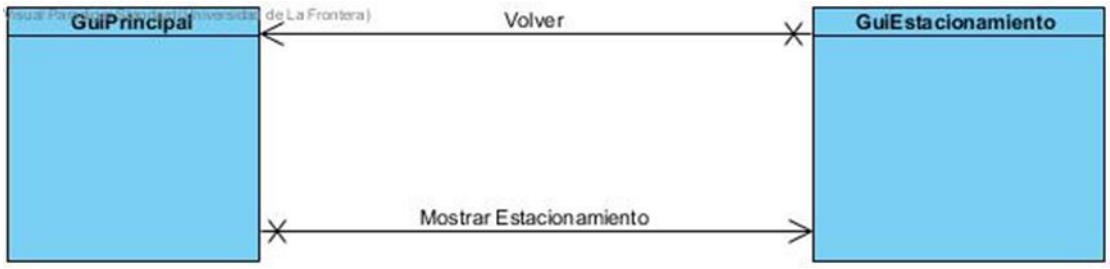
Existen dos clases en el paquete gui, GuiPrincipal y GuiEstacionamiento.

GuiPrincipal es la presentación del software, en la cual se presenta la información de contacto, versión del software y un botón el cual comunica la ventana con GuiEstacionamiento.

GuiEstacionamiento, la ventana más importante, es en la cual se muestra el mapa del estacionamiento, consta de JLabel´s para para representar los espacios. Tiene 2 botones, volver el cual nos envía de regreso a la GuiPrincipal, y el botón Actualizar el cual genera una consulta a la tabla que guarda el estado de los espacios, cambiando JLabel correspondiente de color según el estado del espacio, azul si está ocupado y plomo si está disponible.

Navegación entre las Gui’s del problema

GuiPrincipal lleva a GuiEstacionamiento al presionar el botón mostrar estacionamiento y desde estacionamiento se vuelve a GuiPrincipal al presionar volver.



Diseño y Gestión de los datos manejados

El software genera un archivo de nombre “registro” del tipo txt el cual en cada línea almacena un String con la siguiente estructura, “nombre del espacio+fecha de ingreso+fecha de estadia+duracion de la estadía”.

```
A12 Fecha ingreso Thu Aug 09 12:36:03 CLT 2018 Fecha salida Thu Aug 09 12:36:23 CLT 2018 Su estadia fue de 0 dia(s) con 0 : 0 : 20
A12 Fecha ingreso Thu Aug 09 12:37:25 CLT 2018 Fecha salida Thu Aug 09 12:37:26 CLT 2018 Su estadia fue de 0 dia(s) con 0 : 0 : 1
```

El método que realiza esta acción se llama aniadirArchivo en el paquete manejodedatos, el cual recibe el String desde la clase Lote, con la información mostrada anteriormente y lo almacena el registro.txt.

Gestión de Errores

Paquete GUI

Clase relacionada: GuiPrincipal.java

Posible fuente de error: URL de la imagen no disponible o URL errónea

La portabilidad del proyecto creó la necesidad de utilizar una URL y un BufferedImage para mostrar un archivo de imagen dentro de un JLabel llamado imagen.

Como se gestionó la solución: Añadiendo bloques try{ }catch

```
private void setImagen() {
    try {
        rutaImagen = new URL("URL DE LA IMAGEN");
        miImagen = ImageIO.read(rutaImagen);
        imagen.setIcon(new ImageIcon(miImagen));
    } catch (IOException e) {
        imagen.setText("Imagen no disponible");
    }finally {
        this.add(imagen);
    }
}
```

- La URL rutaImagen es una dirección que viene ya definida dentro del programa.
- En caso de que rutaImagen no esté disponible el JLabel en vez de enseñar una imagen mostrará un texto diciendo “Imagen no disponible”.

Paquete Manejo de Datos

Clase Relacionada: ManejoDatos.java

Posibles fuentes de error: Errores al escribir en un archivo de texto:

- El archivo que se especifica en la ruta no existe.
- No se puede modificar el archivo de texto.

Como se gestionó la solución: Añadiendo bloques try{ }catch

```
 public  void aniadirArchivo(String lista, String Path) {
    FileWriter flwriter = null;
    try {//además de la ruta del archivo recibe un parámetro de tipo boolean, que le indican que se va añadir más registros
        flwriter = new FileWriter(Path, true);
        BufferedWriter bfwriter = new BufferedWriter(flwriter);

            bfwriter.write(lista);
            bfwriter.newLine();
        bfwriter.close();
        System.out.println("Archivo modificado satisfactoriamente..");

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (flwriter != null) {
            try {
                flwriter.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}


public void archivo(String uso) {
    System.out.println("archivo guaradadp");
    try {
        //Crear un objeto File se encarga de crear o abrir acceso a un archivo que se especifica en su constructor
        File archivo = new File("registro.txt");

        //Crear objeto FileWriter que sera el que nos ayude a escribir sobre archivo
        FileWriter escribir = new FileWriter(archivo, true);

        //Escribimos en el archivo con el metodo write
        escribir.write(uso);

        //Cerramos la conexion
        escribir.close();
    } //Si existe un problema al escribir cae aqui
    catch (IOException e) {

        System.out.println("Error al escribir");
    }
}
```

Pruebas Unitarias

@Test de Manejo de Datos

```
@Test (expected = java.lang.RuntimeException.class)
public void ManejoDatosTestFallo(){

    ManejoDatos.aniadirArchivo(5);

}

@Test ()
public void ManejoDatosTestExcito() throws IOException{

    ManejoDatos.aniadirArchivo("hola");

}
```

@Test de descarga de imagen

```
@Test(expected = IOException.class)
public void agregarImagenTest() throws IOException {
    Estacionamiento est = new Estacionamiento();
    GuiPrincipal gp = new GuiPrincipal(est);

}

@Test
public void agregarImagenTestCorrecta() throws IOException{
        Estacionamiento est = new Estacionamiento();
    GuiPrincipal gp = new GuiPrincipal(est);
}
}
```

@Test Conexión con Arduino

```
//con la conexion faltando
    @Test (expected = ArduinoException.class)
    public void ConexionArduinoTest() throws ArduinoException, ArduinoException{
    Estacionamiento est = new Estacionamiento();
    ConexionArduino conexion = new ConexionArduino(est);

    }
//con conexion a arduinos
   public void ConexionArduinoTestCorrecta() throws ArduinoException, ArduinoException{
    Estacionamiento est = new Estacionamiento();
    ConexionArduino conexion = new ConexionArduino(est);

    }
}
```

Conclusiones

Si bien el proyecto tiene mucho por donde crecer y mejorar, como agregar nuevas funciones para una mejor interpretación de los datos y generar una asociación más fuerte entre lote y los lectores, nos encontramos satisfechos con el nivel alcanzado por la solución implementada, queda pendiente según lo mencionado anteriormente mejoras en el manejo de los datos y optimización de los datos a guardar para incluir solo los aspectos más relevantes.

Link GitHub:

https://github.com/cvidalse/SmartParking

Link JavaDoc:

https://drive.google.com/drive/u/3/folders/1g1d4gMC8xK_4ltSzwOZJIAAuniZchS8v