	Computación	Docente: Diego Quisi Peralta
	Programación Aplicada	Período Lectivo: Marzo 2020 – Julio 2020

		<b>FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</b>	
<b>CARRERA:</b> COMPUTACIÓN/INGENIERÍA DE SISTEMAS		<b>ASIGNATURA:</b> PROGRAMACIÓN APLICADA	
<b>NRO. PROYECTO:</b>	1.1	<b>TÍTULO PROYECTO:</b> Proyecto Integrador Interciclo Desarrollo e implementación de un sistema de gestión de datos del parqueadero de la empresa EMOV-EC	
<b>OBJETIVO:</b> Reforzar los conocimientos adquiridos en clase sobre la programación aplicada (POO, Interfaz gráfica, etc) en un contexto real.			
<b>INSTRUCCIONES:</b>		1. Revisar el contenido teórico y práctico del tema	
		2. Profundizar los conocimientos revisando los libros, guías, los enlaces contenidos en los objetos de aprendizaje Java y la documentación disponible en fuentes académicas en línea.	
		3. Deberá desarrollar un sistema informático para la gestión de parqueaderos.	
		4. Deberá generar un informe empleando una herramienta Web 2.0 o Prezi (Tutorial o manual técnico).	
		5. Tomar en consideración que la evaluación del trabajo a realizarse de forma individual y dependerá de los siguientes parámetros: Nivel de precisión y explicación de la propuesta planteada del sistema informático. <b>50%</b> Tutorial o manual técnico del sistema <b>25%</b> (Página Web o Prezi) Exposición, funcionamiento y validación del sistema <b>25%</b> . <b>Puntos extras:</b> Funcionalidad o librería no vista en clase serán valorados como puntos adicionales al intercurso. 6. <b>Fecha de entrega:</b> El sistema debe ser subido al AVAC y presentado el día miércoles <b>13 de Diciembre del 2020</b> .	
<b>ACTIVIDADES POR DESARROLLAR</b>			

1. Investigue, diseñe y desarrolle e implemente un sistema informático que permita gestionar los espacios de parqueo de la empresa EMOV-EC del cancho del parque de la Madre que tiene una capacidad de más de 50 estacionamientos.

### **DEFINICIÓN DEL PROBLEMA:**

El ofrecer una atención cordial y eficiente a sus clientes es un objetivo de vital importancia para una empresa de este tipo. No es tarea fácil gestionar espacios de un estacionamiento ya que se debe de tomar en cuenta diferentes factores como son: el número de espacios con que se cuenta, los espacios que ya se encuentran ocupados en la actualidad, así como las que ya han sido reservadas para una fecha determinada, también hay que tomar en cuenta los servicios que se han contratado para cada vehículo y el precio que esta presenta. Finalmente es importante generar una factura la misma que puede o ser emitido el comprobante impreso.

### **OBJETIVOS DEL SISTEMA:**

El sistema debe de ser desarrollado en el modo monousuario para que pueda:

- 1.3.1. Establecer usuarios en el sistema con dos niveles de operación: Administrador (Todas las operaciones) y Usuario simple (Solo gestiona el parqueadero), con ello mantiene y distingue las posibilidades de operación del usuario correspondiente.
- 2.3.2. Hacer el ingreso y egreso de vehículos desde un solo punto o puesto emitiendo su comprobante de entrada (para el cliente) y luego el de cobro.
- 3.3.3. Hacer ingreso y egreso de vehículos que se estacionan por un determinado tiempo (horario fijo) o que utilizan el espacio mediante un contrato de arrendamiento preestablecido emitiendo el correspondiente recibo de entrada (registro) y salida (cobro) si se quiere.
- 4.3.4. Emitir diversos reportes de ingresos y espacios disponibles.

### **ESPECIFICACIÓN DE REQUERIMIENTOS:**

#### **2.1 Requerimientos No Funcionales**

- 2.1.1. Aprendizaje: El sistema debe permitir el aprendizaje fluido del usuario.
- 2.1.2. Facilidad de uso: El sistema debe poseer una interfaz visual para facilidad del usuario final.

#### **2.2 Requerimientos Funcionales:**

- 2.2.1. Hacer contrato de espacio: El sistema debe gestionar la información correspondiente a las reservas del estacionamiento.
- 2.2.2. Realizar las entradas y salidas de los vehículos: ya sea de vehículos con control de tiempo o de vehículos con control de espacio.
- 2.2.3. Consultar Importe total: El sistema calculará la cuenta total del cliente por los servicios prestados, se deberá tener una tabla para gestionar el valor por hora del parqueadero permitiendo asignar descuentos a clientes.
- 2.2.4. Consultar el Precio de cada espacio: El sistema deberá registrar y mostrar el precio de los espacios disponibles.
- 2.2.5. Ver listado de espacios disponibles: El sistema deberá mostrar la lista de espacios disponibles con que cuenta el estacionamiento de modo **grafico**.
- 2.2.6 Permitirá consultar si alguno de los espacios contratados cuenta con algún tipo de servicio de arrendamiento o multa (Después de una semana de no pago se debe calcular multiplicando el valor de la deuda por 10%).

### **2. Tutorial técnico del uso (Manual técnico):**

- Generar una página web o presentación que contenga lo siguiente:
  - Planteamiento y descripción del problema.
  - Proceso de solución.

- Diagramas de Clases.
- Arquitectura del sistema.
- Descripción de la solución y pasos seguidos.
- Tutorial del uso del sistema (básico).
- Requerimientos de HW y SF (Java).
- Conclusiones y recomendaciones.
- Resultados.

**RESULTADO(S) OBTENIDO(S):**

- Interpreta de forma correcta los algoritmos de programación y su aplicabilidad.
- Identifica correctamente qué herramientas de programación se pueden aplicar.

**CONCLUSIONES:**

Los estudiantes identifican las principales estructuras para la creación de sistemas informáticos. Los estudiantes implementan soluciones gráficas en sistemas.

**RECOMENDACIONES:**

Revisar la información proporcionada por el docente previo a la práctica. Haber asistido a las sesiones de clase.

**Consultar con el docente las dudas que puedan surgir al momento de realizar la práctica.**

**BIBLIOGRAFIA:**

[1]: <https://www.ups.edu.ec/evento?calendarBookingId=98892>

**Docente / Técnico Docente:** Ing. Diego Quisi Peralta Msc.

**Firma:** \_\_\_\_\_

**CARRERA:** COMPUTACION / INGENIERIA EN  
SIATEMAS

**ASIGNATURA:** PROGRAMACION APLICADA

**NRO. PROYECTO:** 1

**TÍTULO PROYECTO:** Proyecto Integrador Interciclo

Desarrollo e implementación de un sistema de gestión de datos del parqueadero de la empresa EMOV-EC

**OBJETIVO:**

Reforzar los conocimientos adquiridos en clase sobre la programación aplicada (POO, Interfaz gráfica, etc) en un contexto real.

**ACTIVIDADES DESARROLLADAS**

1. **Investigue, diseñe y desarrolle e implemente un sistema informático que permita gestionar los espacios de parqueo de la empresa EMOV-EC del cancho del parque de la Madre que tiene una capacidad de más de 50 estacionamientos.**

**Diagrama de clase:**

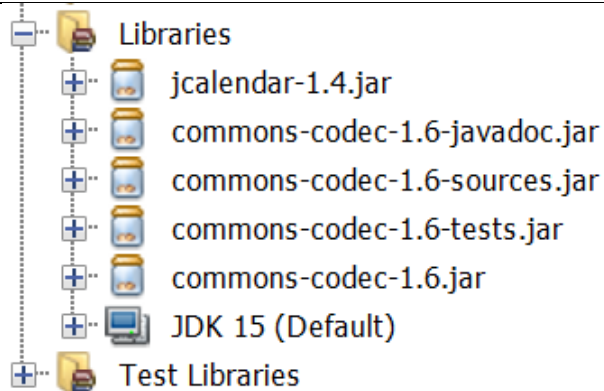
[https://lucid.app/lucidchart/34233cb1-1caf-4ec5-b08d-e301032536cc/view?page=0\\_0#?folder\\_id=home&browser=icon](https://lucid.app/lucidchart/34233cb1-1caf-4ec5-b08d-e301032536cc/view?page=0_0#?folder_id=home&browser=icon)

**Página Web:**

<https://sites.google.com/view/gestin-parqueadero-emov/inicio>

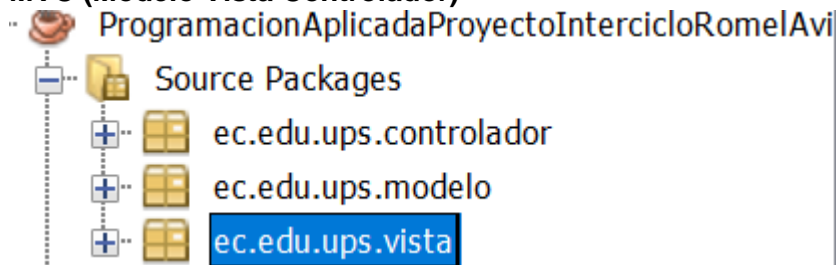
**Requerimientos para el uso correcto del sistema.**

- El sistema funciona con el lenguaje de programación java por lo que debemos tener java en nuestros computadores
- El sistema está codificado en el IDE de NetBeans versión 12.1 con un JDK15 por lo cual es recomendable tener instalado una versión de java de la 10 en adelante para que funcione de una manera correcta.
- El sistema cuenta con dos librerías extras las cuales tienen que ser puestas en el mismo las cuales son el jCalendar y Md5, Jcalendar usado para los que sin fechas y las librerías del MD5 usadas para encriptar la contraseña de los usuarios



## Patrones de Diseño

### MVC (Modelo Vista Controlador)



Uno de los patrones mas usados en una programación orientada a objetos para poder crear de una mejor manera aplicaciones.

### Singleton

Con este patrón aseguramos la unicidad de una clase la cual nos es muy útil para hacer que un docente inicie sesión y así solo a él se le desplieguen ciertas ventanas las cuales solo va a manejar el usuario y a la vez controla que ningún otro usuario o administrador puede iniciar sesión en ese momento como vamos a poder observar en las siguientes imágenes que son parte del código del Usuario clase que se encuentra en el paquete modelo. Entonces este getInstance no ayuda a verificar si un usuario ya inicio sesión o no con el constructor privado creando atributos estáticos en la clase.

```

32     private Usuario(String correo) {
33         this.correo = correo;
34     }
35
36
37     public static Usuario getInstance(){
38         if(Usuario.instance==null){
39             Constructor<Usuario> constructor;
40             try{
41                 constructor=Usuario.class.getDeclaredConstructor();
42                 constructor.setAccessible(true);
43
44                 Usuario usuario=constructor.newInstance();
45
46                 Usuario.instance=usuario;
47             }catch(Exception e){
48                 e.printStackTrace();
49             }
50             return Usuario.instance;
51         }
52         return Usuario.instance;
53     }

```

## Reflexión

La reflexión nos ayuda a crear los usuarios de una manera en la que vamos setiando parte por parte ya que esta nos ayuda a acceder a constructor de la clase debido a que es privado por el uso del patrón de diseño singleton.

```

try{
    Constructor<Usuario> constructor = Usuario.class.getDeclaredConstructor();
    constructor.setAccessible(true);
    Usuario u = constructor.newInstance();
    u.setCorreo(correo);
    if (cedula.isEmpty() || nombre.isEmpty() || Apellido.isEmpty() || correo.isEmpty() || contra.isEmp
        JOptionPane.showMessageDialog(this, "Llene todos los campos solicitados");
    } else {
        if (controladorUsuario.read(u) != null) {
            u.setCodigo(codigo);
            u.setCedula(cedula);
            u.setNombre(nombre);
            u.setApellido(Apellido);
            u.setFechaNacimiento(fecha);
            u.setCorreo(correo);
            u.setContraseña(contra);
            resultado=controladorUsuario.update(u);
        }else{
            u.setCodigo(codigo);
            u.setCedula(cedula);
            u.setNombre(nombre);
            u.setApellido(Apellido);
            u.setFechaNacimiento(fecha);
            u.setCorreo(correo);
        }
    }
}

```

```

        u.setApellido(Apellido);
        u.setFechaNacimiento(fecha);
        u.setCorreo(correo);
        u.setContraseña(contra);
        resultado=controladorUsuario.create(u);
    }
    limpiar();
    JOptionPane.showMessageDialog(this, "Operación : " + resultado);
    cargarDatos();
    dispose();
}

} catch (InstantiationException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
} catch (IllegalAccessException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
} catch (IllegalArgumentException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
} catch (InvocationTargetException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
} catch (NoSuchMethodException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
} catch (SecurityException ex) {
    Logger.getLogger(VentanaCrearUsuario.class.getName()).log(Level.SEVERE, null, ex);
}

```

## Uso de las nuevas actualizaciones de las versiones java 8 en adelante.

### Streams

creados en la parte del controlador para reducir código en el método de buscar es así como podemos ver en la siguiente imagen uno de los usos ya en el código de la aplicación también se usa dentro del método de iniciar sesión con la ayuda del patrón singleton y final mente usado también en un foreach para recorrer los objetos de una mejor manera.

```

public T read(T objeto) {
    if (listaGenerica.contains(objeto)) {
        return (T) listaGenerica.stream().filter(obj -> obj.equals(objeto)).findFirst().get();
    } else {
        return null;
    }
}

}

public Usuario iniciarSesion(String correo, String contrase) {

    var listaDocente= (List<Usuario>) getListGenerica();

    Usuario usuario= listaDocente.stream().filter(usu-> usu.getCorreo().equals(correo)&& usu.getContraseña().

    if(usuario!=null){
        Usuario.instance=usuario;
        return Usuario.getInstance();
    }
    return null;
}
}

```

```

public void validarEstado() {
    controladorSitio.getListGenerica().stream().map(sitio -> {
        if(sitio.getNumero().equalsIgnoreCase(btn1.getText())){
            if(sitio.getEstado().equals("Libre")){
                btn1.setEnabled(true);
                txt1.setBackground(Color.WHITE);
                txt1.setText("L");
            }else{
                if(sitio.getEstado().equals("Ocupado")){
                    btn1.setEnabled(false);
                    txt1.setBackground(Color.RED);
                    txt1.setText("O");
                }else{
                    if(sitio.getEstado().equals("Contrato")){
                        btn1.setEnabled(false);
                        txt1.setBackground(Color.BLUE);
                        txt1.setText("C");
                    }
                }
            }
        }
    })
    return sitio;
}).map(sitio -> {

```

## Sentencia var

Uso de la sentencia var que es una nueva característica que tiene java desde su 10 versión para no especificar el tipo de variable a utilizar como vamos a ver en los siguientes ejemplos utilizados en el código del sistema.

```

public Usuario iniciarSesion(String correo, String contrase) {

    var listaDocente= (List<Usuario>) getListGenerica();

    Usuario usuario= listaDocente.stream().filter(usu-> usu.getCorreo().equals(correo)&& usu.getContraseña().

    if(usuario!=null){
        Usuario instance=usuario;
        return Usuario.getInstance();
    }
    return null;
}

```

```

public void crearSitios() {
    var listaSitios=(List<Sitio>) getListGenerica();
    int numero=0;

    while(numero<61) {
        numero++;
        Sitio sitio= new Sitio(String.valueOf(numero), "Libre", null);
        listaSitios.add(sitio);
    }
    guardarDatos(listaSitios);
}

```

## Paquete Modelo



En todas las clases del controlador se implementa del paquete io. La clase Serializable para poder escribir los datos o en si guardarlos en un archivo de tipo objeto de una manera correcta.

### **Clase Persona**

```
package ec.edu.ups.modelo;
```

```
import java.io.Serializable;
```

```
import java.util.Calendar;
```

```
import java.util.Objects;
```

```
/**
```

```
*
```

```
* @author NANCY
```

```
*/
```

```
public class Persona implements Serializable {
```

```
    private int codigo;
```

```
    private String cedula;
```

```
    private String nombre;
```

```
    private String apellido;
```

```
    private Calendar fechaNacimiento;
```

```
    public Persona() {
```

```
    }
```

```
    public Persona(int codigo, String cedula, String nombre, String apellido) {
```

```
        this.codigo = codigo;
```

```
        this.cedula = cedula;
```

```
        this.nombre = nombre;
```

```
        this.apellido = apellido;
```

```
    }
```

```
    public Persona(String cedula) {
```

```
        this.cedula = cedula;
```

```
    }
```

```
    public Persona(int codigo, String cedula, String nombre, String apellido, Calendar  
    fechaNacimiento) {
```

```
        this.codigo = codigo;
```

```
        this.cedula = cedula;
```

```
        this.nombre = nombre;
```

```
        this.apellido = apellido;
```

```
        this.fechaNacimiento = fechaNacimiento;
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getCedula() {
        return cedula;
    }

    public void setCedula(String cedula) {
        this.cedula = cedula;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public Calendar getFechaNacimiento() {
        return fechaNacimiento;
    }

    public void setFechaNacimiento(Calendar fechaNacimiento) {
        this.fechaNacimiento = fechaNacimiento;
    }

    @Override
    public int hashCode() {
        int hash = 7;
```

```

        hash = 79 * hash + Objects.hashCode(this.cedula);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Persona other = (Persona) obj;
        if (!Objects.equals(this.cedula, other.cedula)) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Persona{" + "codigo=" + codigo + ", cedula=" + cedula + ", nombre=" +
        nombre + ", apellido=" + apellido + ", fechaNacimiento=" + fechaNacimiento + '}';
    }
}

```

- La clase persona es la clase padre de las clases Administrador, Cliente y Usuario debido a que estas clases comparten los mismo atributos entonces en esos casos lo mejor es hacerle una clase padre como en este caso la clase persona que tiene como atributos datos específicos que una persona puede tener como cedula nombre y apellido y cosas así entonces esta clase es muy importante debido a que nos ayuda a optimizar el código una manera muy fácil.

### **Clase Administrador**

```

package ec.edu.ups.modelo;

```

```

import java.io.Serializable;
import java.util.Calendar;
import java.util.Objects;

```

```

/**

```

```

*
* @author NANCY
*/
public class Administrador extends Persona implements Serializable {
    private String correo;
    private String contraseña;

    public Administrador() {
    }

    public Administrador(String correo, String contraseña, int codigo, String cedula, String
nombre, String apellido) {
        super(codigo, cedula, nombre, apellido);
        this.correo = correo;
        this.contraseña = contraseña;
    }

    public String getCorreo() {
        return correo;
    }

    public void setCorreo(String correo) {
        this.correo = correo;
    }

    public String getContraseña() {
        return contraseña;
    }

    public void setContraseña(String contraseña) {
        this.contraseña = contraseña;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 23 * hash + Objects.hashCode(this.correo);
        hash = 23 * hash + Objects.hashCode(this.contraseña);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {

```

```

        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Administrador other = (Administrador) obj;
    if (!Objects.equals(this.correo, other.correo)) {
        return false;
    }
    if (!Objects.equals(this.contraseña, other.contraseña)) {
        return false;
    }
    return true;
}
}

```

- La clase del administrador hereda de la clase persona sus atributos y esta clase como tal tiene como atributos un correo y una contraseña para poder iniciar sesión es por ello por lo que en su equals y hash code ponemos estos dos atributos.

### Clase Cliente

```

package ec.edu.ups.modelo;

import java.io.Serializable;
import java.util.Calendar;

/**
 *
 * @author NANCY
 */
public class Cliente extends Persona implements Serializable {

    public Cliente(int codigo, String cedula, String nombre, String apellido, Calendar fechaNacimiento) {
        super(codigo, cedula, nombre, apellido, fechaNacimiento);
    }

    public Cliente(String cedula) {
        super(cedula);
    }
}

```

- En la clase cliente como podemos ver hereda los atributos de la clase persona entonces el objetivo de esta clase es distinguir un cliente y así después poder crear un controlador para acceder a los distintos métodos Crud y no utilizar la clase persona para ello debió a que es una clase con mas hijos entonces podemos acceder de una mejor manera creando este tipo de clase.

## Clase Usuario

```
package ec.edu.ups.modelo;

import java.io.Serializable;
import java.lang.reflect.Constructor;
import java.util.Calendar;
import java.util.Objects;

/**
 *
 * @author NANCY
 */
public class Usuario extends Persona implements Serializable{
    private String correo;
    private String contraseña;

    public static Usuario instance;

    private Usuario() {
    }

    private Usuario(String correo, String contraseña, int codigo, String cedula, String
nombre, String apellido, Calendar fechaNacimiento) {
        super(codigo, cedula, nombre, apellido, fechaNacimiento);
        this.correo = correo;
        this.contraseña = contraseña;
    }

    private Usuario(String correo) {
        this.correo = correo;
    }

    public static Usuario getInstance(){
        if(Usuario.instance==null){
            Constructor<Usuario> constructor;
            try{
                constructor=Usuario.class.getDeclaredConstructor();
                constructor.setAccessible(true);
```

```

        Usuario usuario=constructor.newInstance();

        Usuario.instance=usuario;
    }catch(Exception e){
        e.printStackTrace();
    }
    return Usuario.instance;
}
return Usuario.instance;
}

public String getCorreo() {
    return correo;
}

public void setCorreo(String correo) {
    this.correo = correo;
}

public String getContraseña() {
    return contraseña;
}

public void setContraseña(String contraseña) {
    this.contraseña = contraseña;
}

@Override
public int hashCode() {
    int hash = 5;
    hash = 23 * hash + Objects.hashCode(this.correo);
    hash = 23 * hash + Objects.hashCode(this.contraseña);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }

```

```

    }
    final Usuario other = (Usuario) obj;
    if (!Objects.equals(this.correo, other.correo)) {
        return false;
    }
    if (!Objects.equals(this.contraseña, other.contraseña)) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return super.toString()+"Usuario{" + "correo=" + correo + ", contraseña=" +
    contraseña + '}';
}
}

```

Esta es la ultima clase que hereda los atributos de la clase persona y al igual que la clase del administrador contiene solo como atributos un correo y una contraseña los cuales nos van a permitir iniciar sesión después la diferencia con la clase administrador radica en como el administrador se va encontrar creado por defecto en el programa no es necesario implementar el método get instances como lo tiene esta clase este método nos ayuda para el uso del patrón de diseño singleton y a su vez el uso de la reflexión debido a que los constructores serán privados en esta clase.

### **Clase Vehículo**

```
package ec.edu.ups.modelo;
```

```
import java.io.Serializable;
import java.util.Objects;
```

```
/**
```

```
*
```

```
* @author NANCY
```

```
*/
```

```
public class Vehiculo implements Serializable{
```

```
    private int codigo;
```

```
    private String color;
```

```
    private String marca;
```

```
    private String modelo;
```

```
    private String placa;
```

```
    private Cliente cliente;
```

```
    public Vehiculo() {
```



```
}

public Vehiculo(String placa) {
    this.placa = placa;
}

public Vehiculo(int codigo, String color, String marca, String modelo, String placa,
Cliente cliente) {
    this.codigo = codigo;
    this.color = color;
    this.marca = marca;
    this.modelo = modelo;
    this.placa = placa;
    this.cliente = cliente;
}

public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public String getColor() {
    return color;
}

public void setColor(String color) {
    this.color = color;
}

public String getMarca() {
    return marca;
}

public void setMarca(String marca) {
    this.marca = marca;
}

public String getModelo() {
    return modelo;
}

public void setModelo(String modelo) {
```

```

        this.modelo = modelo;
    }

    public String getPlaca() {
        return placa;
    }

    public void setPlaca(String placa) {
        this.placa = placa;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 71 * hash + Objects.hashCode(this.placa);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Vehiculo other = (Vehiculo) obj;
        if (!Objects.equals(this.placa, other.placa)) {
            return false;
        }
        return true;
    }

```

```

@Override
public String toString() {
    return "Vehiculo{" + "codigo=" + codigo + ", color=" + color + ", marca=" + marca +
", modelo=" + modelo + ", placa=" + placa + ", cliente=" + cliente + "}";
}

}

```

- La clase vehiculo contiene los atributos con los cuales nosotros vamos a crear los vehículos los cuales son placa, maraca, modelo, color y un código o numero para llevar un registro de cuantos vehículos están registrados y a su vez esta clase contiene como atributo un cliente para poder identificar quie será el dueño del auto que ocupe un sitio de parqueo.

### **Clase Sitio**

```

package ec.edu.upse.modelo;

import java.io.Serializable;
import java.util.Objects;

/**
 *
 * @author NANCY
 */
public class Sitio implements Serializable{
    private String numero;
    private String estado;
    private Vehiculo vehiculo;

    public Sitio() {
    }

    public Sitio(String numero) {
        this.numero = numero;
    }

    public Sitio(String numero, String estado, Vehiculo vehiculo) {
        this.numero = numero;
        this.estado = estado;
        this.vehiculo = vehiculo;
    }

    public String getNumero() {
        return numero;
    }
}

```

```

public void setNumero(String numero) {
    this.numero = numero;
}

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public Vehiculo getVehiculo() {
    return vehiculo;
}

public void setVehiculo(Vehiculo vehiculo) {
    this.vehiculo = vehiculo;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 71 * hash + Objects.hashCode(this.numero);
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Sitio other = (Sitio) obj;
    if (!Objects.equals(this.numero, other.numero)) {
        return false;
    }
    return true;
}

```

```

@Override
public String toString() {
    return "Sitio{" + "numero=" + numero + ", estado=" + estado + ", vehiculo=" +
vehiculo + '}';
}

}

```

- La Clase Sitio contiene como atributos primero un numero de sitio que siempre en el cual se podrá aparcar un estado para ver si se encuentra libre, ocupado o bajo un contrato y finalmente un vehículo para saber que vehículo se encuentra estacionado en ese lugar y como un vehículo contiene un cliente se sabe que cliente ocupa el lugar.

### **Clase Factura**

```

package ec.edu.upse.modelo;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.Calendar;

/**
 *
 * @author NANCY
 */
public class Factura implements Serializable {
    private int numero;
    private Calendar fechaYHoraDeIngreso;
    private Calendar fechaYHoraDeSalida;
    private String estado;
    private double total;
    private Cliente cliente;
    private Sitio sitio;

    public Factura() {
    }

    public Factura(int numero) {
        this.numero = numero;
    }

    public Factura(int numero, Calendar fechaYHoraDeIngreso, String estado, Cliente
cliente, Sitio sitio) {
        this.numero = numero;

```

```
        this.fechaYHoraDeIngreso = fechaYHoraDeIngreso;
        this.estado = estado;
        this.cliente = cliente;
        this.sitio = sitio;
    }

    public Factura(int numero, Calendar fechaYHoraDeIngreso, Calendar
fechaYHoraDeSalida, String estado, double total, Cliente cliente, Sitio sitio) {
        this.numero = numero;
        this.fechaYHoraDeIngreso = fechaYHoraDeIngreso;
        this.fechaYHoraDeSalida = fechaYHoraDeSalida;
        this.estado = estado;
        this.total = total;
        this.cliente = cliente;
        this.sitio = sitio;
    }


    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public Calendar getFechaYHoraDeIngreso() {
        return fechaYHoraDeIngreso;
    }

    public void setFechaYHoraDeIngreso(Calendar fechaYHoraDeIngreso) {
        this.fechaYHoraDeIngreso = fechaYHoraDeIngreso;
    }

    public Calendar getFechaYHoraDeSalida() {
        return fechaYHoraDeSalida;
    }

    public void setFechaYHoraDeSalida(Calendar fechaYHoraDeSalida) {
        this.fechaYHoraDeSalida = fechaYHoraDeSalida;
    }
}
```

```
public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public double getTotal() {
    return total;
}

public void setTotal(double total) {
    this.total = total;
}

public Cliente getCliente() {
    return cliente;
}

public void setCliente(Cliente cliente) {
    this.cliente = cliente;
}

public Sitio getSitio() {
    return sitio;
}

public void setSitio(Sitio sitio) {
    this.sitio = sitio;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 37 * hash + this.numero;
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
}
```

```

    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Factura other = (Factura) obj;
    if (this.numero != other.numero) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "Factura{" + "numero=" + numero + ", fechaYHoraDeIngreso=" +
fechaYHoraDeIngreso + ", fechaYHoraDeSalida=" + fechaYHoraDeSalida + ",
estado=" + estado + ", total=" + total + ", cliente=" + cliente + ", sitio=" + sitio + '}';
}

}

```

- Esta clase contiene como atributos algunos componentes que se necesitan para poder crear una factura o un tiket de ingreso por así llamarlo entonces los atributos de esta clase son primero un numero para saber cuantos tikets o facturas se están generando des pues una hora de entrada para registrar el ingreso del vehículo al parqueadero y a su vez una hora de salida para registrar el egreso del mismo tendremos lo que es un valor o precio a pagar, un cliente para poder sacarlo a nombre del mismo y finalmente un sitio para tener el registro del sitio usado.

### **Clase Contrato**

```

package ec.edu.ups.modelo;

import java.io.Serializable;
import java.util.Calendar;

/**
 *
 * @author NANCY
 */
public class Contrato implements Serializable {
    private int codigo;
    private Cliente cliente;
}

```



```
private Sitio Sitio;
private Calendar fechaInicio;
private Calendar fechaFinal;
private String estadoPago;
private double valorAPagar;

public Contrato() {
}

public Contrato(int codigo) {
    this.codigo = codigo;
}

public Contrato(int codigo, Cliente cliente, Sitio Sitio, Calendar fechaInicio,
Calendar fechaFinal, String estadoPago, double valorAPagar) {
    this.codigo = codigo;
    this.cliente = cliente;
    this.Sitio = Sitio;
    this.fechaInicio = fechaInicio;
    this.fechaFinal = fechaFinal;
    this.estadoPago = estadoPago;
    this.valorAPagar = valorAPagar;
}

public int getCodigo() {
    return codigo;
}

public void setCodigo(int codigo) {
    this.codigo = codigo;
}

public Cliente getCliente() {
    return cliente;
}

public void setCliente(Cliente cliente) {
    this.cliente = cliente;
}

public Sitio getSitio() {
    return Sitio;
}

public void setSitio(Sitio Sitio) {
    this.Sitio = Sitio;
}
```

```
}

public Calendar getFechaInicio() {
    return fechaInicio;
}

public void setFechaInicio(Calendar fechaInicio) {
    this.fechaInicio = fechaInicio;
}

public Calendar getFechaFinal() {
    return fechaFinal;
}

public void setFechaFinal(Calendar fechaFinal) {
    this.fechaFinal = fechaFinal;
}

public String getEstadoPago() {
    return estadoPago;
}

public void setEstadoPago(String estadoPago) {
    this.estadoPago = estadoPago;
}

public double getValorAPagar() {
    return valorAPagar;
}

public void setValorAPagar(double valorAPagar) {
    this.valorAPagar = valorAPagar;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 89 * hash + this.codigo;
    return hash;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
}
```

```

        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Contrato other = (Contrato) obj;
        if (this.codigo != other.codigo) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "Contrato{" + "codigo=" + codigo + ", cliente=" + cliente + ", Sitio=" + Sitio
        + ", fechaInicio=" + fechaInicio + ", fechaFinal=" + fechaFinal + ", estadoPago=" +
        estadoPago + ", valorAPagar=" + valorAPagar + '}';
    }
}

```

Esta clase es muy parecida a la clase de la factura debido a que contienen los mismos atributos o similares la razón de esta clase es que las facturas serán solo por un tiempo de hora en cambio los contratos será por un tiempo más prolongado sin exceder más de un mes

## **Paquete controlador**

Las clases que contiene el paquete de controlador son bastante cortas debido a que vamos a tener un solo controlador ya que el sistema usa una programación genérica que nos ayuda minimizar el código teniendo todos los métodos del crud en un solo controlador y en los demás controladores se tendrá métodos específicos que solo usaran ellos en caso de tenerlos.

## **Clase Controlador**

```

package ec.edu.ups.controlador;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

```

```

/**
 *
 * @author NANCY
 * @param <T>
 */
public abstract class Controlador <T>{
    private List<T> listaGenerica;
    private String ruta;

    public Controlador(String ruta) {
        listaGenerica = new ArrayList<>();
        this.ruta = ruta;

        this.cargarDatos();
    }

    //public abstract boolean validar(T obj);

    public void cargarDatos() {
        try {
            FileInputStream archivo = new FileInputStream(ruta);
            ObjectInputStream objetoLectura = new ObjectInputStream(archivo);

            listaGenerica = (List<T>) objetoLectura.readObject();

        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public boolean guardarDatos(List<T> listaGuardar) {
        try {
            FileOutputStream archivo = new FileOutputStream(ruta);
            ObjectOutputStream objetoEscritura = new ObjectOutputStream(archivo);

            objetoEscritura.writeObject(listaGuardar);
            return true;
        } catch (IOException e) {
            e.printStackTrace();
        }

        return false;
    }
}

```

```

public boolean create(T objeto) {
    listaGenerica.add(objeto);
    return guardarDatos(listaGenerica);

}

public T read(T objeto) {

    if (listaGenerica.contains(objeto)) {
        return (T) listaGenerica.stream().filter(obj
obj.equals(objeto)).findFirst().get();      ->
    } else {
        return null;
    }

}

public boolean update(T objetoActualizado) {
    for (T objeto : listaGenerica) {
        if (objeto.equals(objetoActualizado)) {
            listaGenerica.set(listaGenerica.indexOf(objeto), objetoActualizado);
            return guardarDatos(listaGenerica);
        }
    }
    return false;
}

public boolean delete(T objeto) {
    listaGenerica.remove(objeto);
    return guardarDatos(listaGenerica);
}

public List<T> getListaGenerica() {
    return listaGenerica;
}

public int cargarCodigo(){
    if (getListaGenerica().size() > 0) {
        return getListaGenerica().size() + 1;
    } else {
        return 1;
    }
}
}

```

- La clase del controlador es una clase abstracta la cual usa una programación genérica y como nos podemos dar cuenta de eso es por que al principio entre una <>esta definida una variable que después será llamada al momento de instanciar en los otros controladores cámbiala por el objeto necesario para que accedan a los métodos en esta clase, esta clase contiene los métodos del crud además de esos se encuentran los métodos para guardar y leer la información de los archivos objetos y por ultimo un método de cargar código que nos sirve para generar el código de una manera mas automatizada.

### **Controlador Administrador**

```
package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Administrador;

/**
 *
 * @author NANCY
 */
public class ControladorAdministrador {
    Administrador administrador = new Administrador("a", "123", 1, "0567123489",
"Juan", "Mendez");

    public Administrador iniciarSesion(String correo, String contrase){
        if (administrador.getCorreo().equals(correo) &&
administrador.getContraseña().equals(contrase)) {
            return administrador;
        }
        return null;
    }
}
```

El controlador administrador es el único controlado que no se extiende del controlador principal debido a que solo podremos tener un administrador el cual ya esta generado por defecto entonces lo creamos ya aquí como tal entonces el único método que necesita y tiene va a hacer el de iniciar sesión con el cual vamos a comprobar si es el administrador creado el cual quiere acceder al sistema.

### **Controlador Usuario**

```
package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Usuario;
import java.security.MessageDigest;
import java.util.Arrays;
import java.util.List;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
```

```

import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Base64;

/**
 *
 * @author NANCY
 */
public class ControladorUsuario extends Controlador<Usuario>{

    public ControladorUsuario(String ruta) {
        super(ruta);
    }

    public Usuario iniciarSesion(String correo, String contrase,String keyUnlock) {

        var listaDocente= (List<Usuario>) getListaGenerica();

        Usuario usuario= listaDocente.stream().filter(usu->
usu.getCorreo().equals(correo)&&deecnode(keyUnlock,
usu.getContraseña()).equals(contrase)).findFirst().get();
        if(usuario!=null){
            Usuario.instance=usuario;
            return Usuario.getInstance();
        }
        return null;
    }

    public String ecryptarContraseña(String key, String contraseña){
        String encriptacion= " ";
        try{
            MessageDigest msmd= MessageDigest.getInstance("MD5");
            byte[] llaveContrase=msmd.digest(key.getBytes("utf-8"));
            byte[] bytesKey= Arrays.copyOf(llaveContrase, 24);
            SecretKey secretKey= new SecretKeySpec(bytesKey, "DESede");

            Cipher cifrado= Cipher.getInstance("DESede");
            cifrado.init(Cipher.ENCRYPT_MODE, secretKey);

            byte[] bytesContrase= contraseña.getBytes("utf-8");
            byte[] buffer= cifrado.doFinal(bytesContrase);
            byte[] base64Byte= Base64.encodeBase64(buffer);

            encriptacion= new String(base64Byte);
            return encriptacion;
        } catch (Exception ex) {

```

```

        System.out.println("algo esta mal");
    }
    return null;
}

public String deecnode(String secretKey, String contraseñaencode) {
    String desenscriptacion = "";
    try {
        byte[] message = Base64.decodeBase64(contraseñaencode.getBytes("utf-8"));
        MessageDigest md5 = MessageDigest.getInstance("MD5");
        byte[] digestOfPassword = md5.digest(secretKey.getBytes("utf-8"));
        byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);
        SecretKey key = new SecretKeySpec(keyBytes, "DESede");
        Cipher decipher = Cipher.getInstance("DESede");
        decipher.init(Cipher.DECRYPT_MODE, key);
        byte[] plainText = decipher.doFinal(message);
        desenscriptacion = new String(plainText, "UTF-8");
        return desenscriptacion;

    } catch (Exception ex) {
        System.out.println("algo esta mal");
    }
    return null;
}

```

El controlador del usuario tiene algunos métodos definidos de el mismo como lo son el iniciar sesión y los métodos para encriptar y desencriptar la contraseña con las librerías del encriptado MD5

### **Controlador Cliente**

```

public class ControladorCliente extends Controlador<Cliente> {

    public ControladorCliente(String ruta) {
        super(ruta);
    }

}

```

### **Controlador Vehículo**

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Vehiculo;

/**

```



```

*
* @author NANCY
*/
public class ControladorVehiculo extends Controlador<Vehiculo> {

    public ControladorVehiculo(String ruta) {
        super(ruta);
    }

}

```

### **Controlador Sitio**

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Sitio;

/**
 *
 * @author NANCY
 */
public class ControladorSitio extends Controlador<Sitio>{

    public ControladorSitio(String ruta) {
        super(ruta);
    }

}

```

### **Controlador Factura**

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Factura;

/**
 *
 * @author NANCY
 */
public class ControladorFactura extends Controlador<Factura> {

    public ControladorFactura(String ruta) {
        super(ruta);
    }

}

```

## Controlador Contrato

```
package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Contrato;

/**
 *
 * @author NANCY
 */
public class ControladorContrato extends Controlador<Contrato> {

    public ControladorContrato(String ruta) {
        super(ruta);
    }

}
```

Como podemos ver en el código de estos controladores son muy similares debido a que lo único que cambia es el objeto con el cual se va a trabajar y esto es gracias a que todos extienden la clase del controlador principal y nos ahorramos escribir en cada uno de ellos los métodos del curd entre otros más optimizando así gran parte del código.

## Paquete Vista

**Debido a lo extensos que son los códigos de las ventanas se pondrá el link del github en donde se podrá visualizar el código respectivo de cada ventana**

### Ventana Principal

La ventana principal es una ventana de tipo MDI en la cual podemos acceder o abrir ventanas del JInternalFrame para la ejecución del programa esta es la única ventana de ejecución por lo tanto en esta venta se instancia por única vez cada una de las ventanas y los controladores que serán pasados a los controladores de las ventanas que necesiten para que así en las JInternalFrame no tendríamos que volver a instanciar ningún constructor ni tipo de ventana

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaPrincipal.java>

### Ventana Crear Usuario

Ventana que solo se puede activar si entra el administrador ya que es el único que puede crear diferentes usuarios que trabajaran en el sistema llevando un registro en una tabla del numero de usuarios creados siendo posible una actualización de los mismos o su eliminación respectiva, pero para hacer estas actividades el sistema lo que hace es setear uno por uno cada atributo utilizando reflexión.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaCrearUusuario.java>

### **Ventana Crear Cliente**

Ventana a la cual pueden acceder tanto como el usuario como el administrador esta es la ventana donde se puede crear un cliente llenando todos los parámetros requeridos para poder pasar al constructor para ser guardados y leídos de los archivos objetos.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaCrearCliente.java>

### **Ventana Crear Vehículo**

Ventana la cual también accede tanto el administrador como los usuarios en esta ventana se pueden crear vehículos siempre y cuando busquemos un cliente y llenemos todos los campos requeridos para pasar la información y enviarla al constructor para que sea guardada en archivos objetos y así llenar la tabla de datos con los vehículos ya registrados.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaGestionVehiculo.java>

### **Ventana Iniciar Sesión**

Ventana en donde primero elegimos quien va a iniciar sesión si el administrador o un usuario y de acuerdo con eso pasamos a un if para verificar lo elegido y si es el correcto hacer visibles los menús items de la ventana principal a los cuales tienen acceso respectivamente.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaIniciarSesion.java>

### **Ventana Listar Facturas y Contratos.**

A esta ventana solo tendrá acceso el administrador que es el único que tendrá una lista de las facturas o los contratos realizados con solo presionar un botón se accede al método respectivo indicando en un txt el tipo de lista a la que se accede para que no haya confusiones y así cargamos los datos respectivos a la tabla.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaListarFacturasYContratos.java>

### **Ventana Sitio**

Esta es la ventana en donde esta la forma del parqueadero interactivo posee una serie de botones con los cuales vamos a saber cuál espacio está disponible o no mediante un método denominado validar el cual con un stream map vamos recorriendo la lista de sitios creados y verificando su estado respectivo para así tener una señalética del estado de los sitios del parqueadero y a su vez esta ventana hace visible otra ventana donde sale la información del ticket de entrada del vehículo solo si están llenos todos los campos para poder hacer una entrada por hora esto es muy importante debido a que esta ventana no permite hacer la entrada por contrato pero si muestras los espacios que son contratados. Al presionar los botones cargar el número del sitio entonces es por eso por lo que tenemos que presionar el botón del sitio que se quiere ocupar.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaSitios.java>

### **Ventana Crear Sitio**

Ventana a la cual solo accede el administrador y permite hacer visible 10 sitios más según se valla presionando el botón de abrir sitio y a su vez también permite eliminar sitios haciéndoles que no se puedan ver y por la tanto inhabilitando el poder ser seleccionados esto funciona a través de un contador el cual a medida que presionamos cualquiera de los botones suman o restan al contador para abrir o eliminar el sitio.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaCrearSitios.java>

### **Ventana Tiket de Ingreso**

Esta ventana solo se hace visible si en la ventana sitios se llenan los espacios respectivos y muestran los datos del sitio que se ocupara ya que esta ventana recibe como parámetro el lugar seleccionado y así permite buscar el sitio y extraer la información para poder entregar el tiket al ingresar el vehículo.

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaTiketIngreso.java>

### **Ventana cobro**

Ventana en donde primero tenemos que buscar el tiket a pagar para poder generar la factura entonces al presionar el botón accedemos al método del controlador read para buscar el tiket respectivo entonces si queremos generar un descuento aplicamos el botón aplicar descuento el cual calcula el tanto por ciento y vuelve a escribir sobre el txtTotal para mostrar cuanto tiene que pagar entonces para terminar se presiona el botón pagar el cual actualiza el estado de tiket y lo escribe como pagado y a su vez actualiza el sitio para poder ponerlo en libre y se puede seleccionar nuevamente ese sitio.

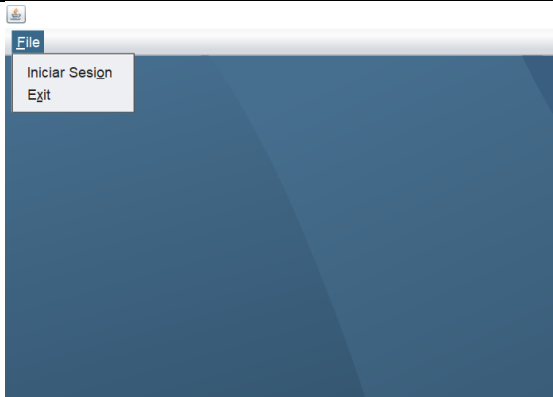
<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaCobro.java>

### **Ventana Contrato**

En esta ventana podemos generar un contrato seleccionado la fecha de salida y según eso el contrato ver al presionar calcular si esta por día por semana o mes debido a que el contrato tiene la opción de no pagado no permite generar contratos mayores aun mes seleccionamos el sitio donde queremos el contrato y en caso de estar libre se llama al controlador del sitio y se manda a crear o a actualizar el sitio como tal para que en la ventana ya salga que está ocupado para poder pagar o terminar el contrato presionamos el botón de buscar primero el contrato y al darle terminar se verifica si se termina en el tiempo establecido o en ese caso pasa a apagar una multa del 10%

<https://github.com/RomelAvila2001/Proyecto-Integrador-Interciclo-Programacion-Aplicada-Parqueadero-EMOV/blob/master/src/ec/edu/ups/vista/VentanaContrato.java>

## **Funcionalidad**



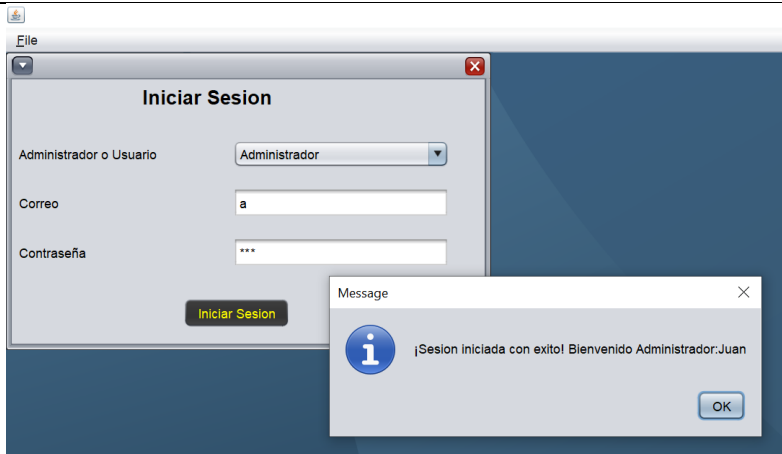
**Al ejecutar el programa y presionar el menú File se nos despliegan las dos opciones de terminar el programa o iniciar sesión**



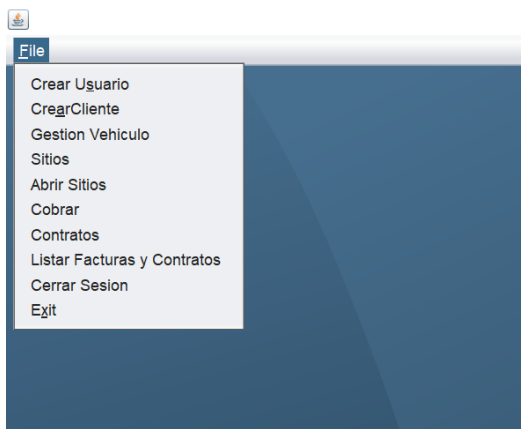
**Al presionar iniciar sesión abrimos la siguiente ventana que nos permitirá iniciar sesión**



**Para poder iniciar sesión tenemos que seleccionar primero quien va a ingresar al sistema si un administrador o un usuario y después se llenaran los datos del correo y la contraseña para ver si se encuentra en los datos guardados.**



**Al ingresar como administrador podemos ver y ser encontrado podemos ver que sale el siguiente mensaje y solo tenemos que dar a ok para continuar**



**Entonces observamos que el menú de selecciona cambia al entrar como administrador y tenemos una serie de opciones las cuales podremos realizar para hacer funcionar el sistema.**

**Registrar Usuario**

Codigo: 2

Cedula:

Nombre:

Apellido:

Fecha de Nacimiento:

Correo:

Contraseña:

**Guardar o Actualizar** **Eliminar**

Codigo	Cedula	Nombre	Apellido	Fecha de Nacimi...	Correo	Contraseña
1	0105944110	Romel	Avila	java.util.Gregoria...	r	123

Si presionamos en crear usuario se nos despliega la siguiente ventana en onde el código ya nos sale cargado por defecto

**Registrar Usuario**

Codigo: 2

Cedula: 0102257920

Nombre: Rene

Apellido: Avila

Fecha de Nacimiento:

Correo:

Contraseña:

Calendar: diciembre 2020

Codigo	Ced	Nombre	Apellido	Fecha de Nacimi...	Correo	Contraseña
1	010	Rene	Avila	13	r	123

Al llenar los datos y llenar la fecha de nacimiento damos clic en la imagen del calendario y se nos despliega esta opción para poder seleccionar la fecha respectiva

**Registrar Usuario**

Codigo: 3

Cedula:

Nombre:

Apellido:

Fecha de Nacimiento:

Correo:

Contraseña:

**Guardar o Actualizar** **Eliminar**

Codigo	Cedula	Nombre	Apellido	Fecha de Nacimiento	Correo	Contraseña
1	0105944110	Romel	Avila	java.util.Gregoria...	r	123
2	0102257920	Rene	Avila	java.util.Gregoria...	t	123

Al presionar guardar y actualizar se cerrará la ventana y si volvemos abrir claramente podemos ver que se carga ya en la tabla los datos del nuevo usuario registrado.

**Crear Cliente**

Codigo: 2

Cedula:

Nombre:

Apellido:

Fecha de Nacimiento:

**Gurdar/Actualizar** **Eliminar**

Codigo	Cedula	Nombre	Apellido	Fecha de Nacimiento
1	0105194195	Santiago	Avila	java.util.Gregorian...

Si presionamos en crear cliente podemos se nos abrirá esta ventana en donde tendremos que llenar cada dato para poder crear un cliente nuevo. O en caso de que se dese actualizar algún dato de algún cliente se selecciona desde la tabla el cliente deseado y se cargaran los datos para ser modificados y se vera de la siguiente manera



**Crear Cliente**

Codigo: 1

Cedula: 0105194195

Nombre: Santiago

Apellido: Avila

Fecha de Nacimiento: 12 jun. 1998

**Guardar/Actualizar** **Eliminar**

Codigo	Cedula	Nombre	Apellido	Fecha de Nacime...
1	0105194195	Santiago	Avila	java.util.Gregorian...

**Gestion Vehiculo**

Ingrese numero de cedula para buscar al cliente

Codigo: 2

Color:

Marca:

Modelo:

Placa:

Cedula:

Nombre:

Apellido:

Fecha de Nacimiento:

**Buscar**

**Guardar/Actualizar** **Eliminar**

Codigo	Color	Marca	Modelo	Placa	Cliente
2	blanco	Toyota	1000	aaa123	Persona(codigo=1, cedu...

Si presionamos en gestión vehículo se nos abre esta ventana con la cual llenaremos cada uno de los datos para poder crear un nuevo vehículo, pero en la parte del cliente tenemos que escribir la cedula del mismo para buscarlo y asignarle a un vehículo en caso de no ser encontrado el cliente nos saldrá el siguiente mensaje

**Gestion Vehiculo**

Ingrese numero de cedula para buscar al cliente

Codigo: 2

Color:

Marca:

Modelo:

Placa:

Codigo:

Cedula: 1

Nombre:

Buscar

Gurdar/Actualizar

Select an Option

? Cliente no encontrado desea crear uno

Yes No Cancel

Codigo	Color	Marca	Modelo	Placa	Cliente
2	blanco	Toyota	1000	aaa123	Persona{codigo=1, cedu...

En caso de presionar que si se abrirá la ventana para poder crear un nuevo cliente de esta manera

**Crear Cliente**

Codigo: 2

Cedula:

Nombre:

Apellido:

Fecha de Nacimiento: 12 jun. 1998

Gurdar/Actualizar Eliminar

Codigo:

Cedula: 1

Nombre:

Apellido:

Fecha de Nacimiento:

Buscar

Codigo	Cedula	Nombre	Apellido	Fecha de Nacimiento
1	0105194195	Santiago	Avila	java.util.Gregorian...

Cliente

Persona{codigo=1, cedu...

File

Numero del Sitio:  Estado: Libre Placa del Vehículo:   ☐ Ocupado ☒ Bajo Contrato  Libre

Nombre del Cliente:  Cedula:

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Al presionar en sitios se nos abre la siguiente ventana en donde podemos ver ya el modelo del parqueadero y hacer el registro de entrada de algún vehículo por hora ya que en esta ventana no se puede hacer el registro por contrato ya que eso se hace desde otra ventana.

File

Numero del Sitio: 15 Estado: Libre Placa del Vehículo:   ☐ Ocupado ☒ Bajo Contrato  Libre

Nombre del Cliente:  Cedula:

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Entonces primero para hacer el registro de la entrada tendremos que seleccionar el sitio donde se va a ocupar y así se cargará el número del sitio después seleccionamos el estado y ponemos en ocupado y finalmente escribimos la placa del vehículo que

hace su ingreso en ese momento para buscar en los datos guardados en caso de no ser encontrado el vehículo nos saldara el siguiente mensaje

File

Numero del Sitio: 15 Estado: Ocupado Placa del Vehiculo: aaa12 **Buscar** O Ocupado C Bajo Contrato L Libre

Nombre del Cliente: Cedula: **Generar Tiket**

Grid of 50 slots (1-50). Slots 1 and 2 have wheelchair icons. Slots 4, 11, 15, 21, 28, 31, 38, 41, 48, and 50 have car icons. Slots 1, 4, 11, 15, 21, 28, 31, 38, 41, 48, and 50 are labeled 'L'.

**Select an Option**

? Vehiculo no encontrado desea crear uno

**Yes** **No** **Cancel**

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Si presionamos que si podremos acceder a la ventana de gestión vehículos en donde podremos crear un nuevo vehículo y aparecerá de la siguiente manera

**Gestion Vehiculo**

Ingrese numero de cedula para buscar al cliente

Codigo: 2 Color: Marca: Modelo: Placa:

Cedula: 1 **Buscar**

Nombre: Apellido: Fecha de Nacimiento:

**Guardar/Actualizar** **Eliminar**

Codigo	Color	Marca	Modelo	Placa	Cliente
2	blanco	Toyota	1000	aaa123	Persona(codigo=1, cedu...

File

Numero del Sitio: 15 Estado: Ocupado Placa del Vehiculo: aaa123 Buscar Ocupado Bajo Contrato L Libre

Nombre del Cliente: Santiago Cedula: 0105194195 Generar Tiket

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Al tener ya todos los datos llenos podemos presionar el botón de generar tiket y nos aparecerá la siguiente ventana con el tiket de ingreso el cual se le entregará al cliente entrante para que después pueda hacer el respectivo pago

File

Tiket numero: 5  
Fecha y hora de ingreso: Sun Dec 13 16:17:46 ECT 2020  
Cliente: Santiago Avila  
Cedula: 0105194195  
Placa vehiculo: aaa123  
Marca: Toyota  
Modelo: 1000  
Numero sitio parqueado: 15

OK

Numero del Sitio: 15 Estado: Ocupado Placa del Vehiculo: aaa123 Buscar Ocupado Bajo Contrato L Libre

Nombre del Cliente: Santiago Cedula: 0105194195 Generar Tiket

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50



















































- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Presionamos en ok y se cerrara esa ventana del tiket

File

Numero del Sitio  Estado Libre Placa del Vehículo  **Buscar** O Ocupado C Bajo Contrato L Libre

Nombre del Cliente  Cedula  **Generar Tiket**

 1	 2	 3	 4	 5	 6	 7	 8	 9	 10
L			L						
 11	 12	 13	 14	 15	 16	 17	 18	 19	 20
				<span style="color: red;">O</span>					
 21	 22	 23	 24	 25	 26	 27	 28	 29	 30
 31	 32	 33	 34	 35	 36	 37	 38	 39	 40
			L				L		
 41	 42	 43	 44	 45	 46	 47	 48	 49	 50

- (30 min o menos) Valor a Pagar: \$0.60 ctn    - (31 min a 1 Hora) Valor a Pagar: \$1.00    - (Pasada la Hora) Valor por Minuto: \$0.02 ctn















































Podemos ver que al cerrar esa ventana tiket y nuevamente activarse la ventana de los sitios se encuentra bloqueado y muestra que está ocupado el sitio seleccionado recientemente y no podrá ser ocupado hasta que quede libre nuevamente

File

Placa del Vehículo  **Buscar** O Ocupado C Bajo Contrato L Libre

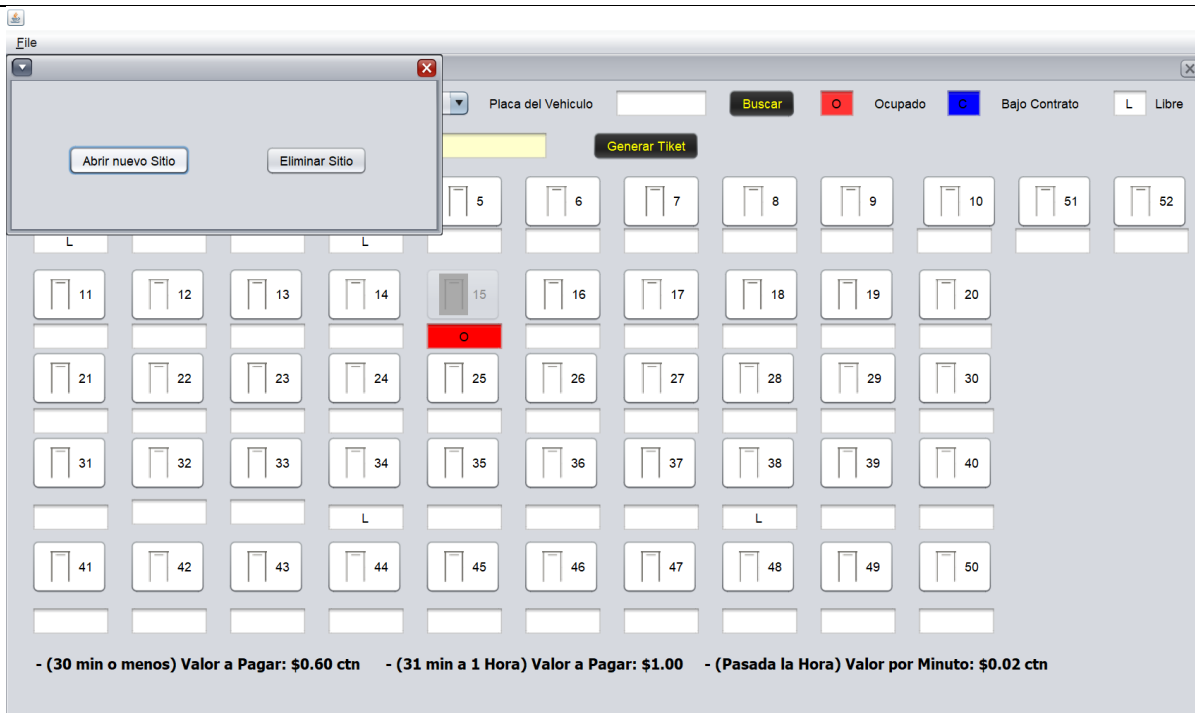
**Generar Tiket**

Abrir nuevo Sitio
Eliminar Sitio

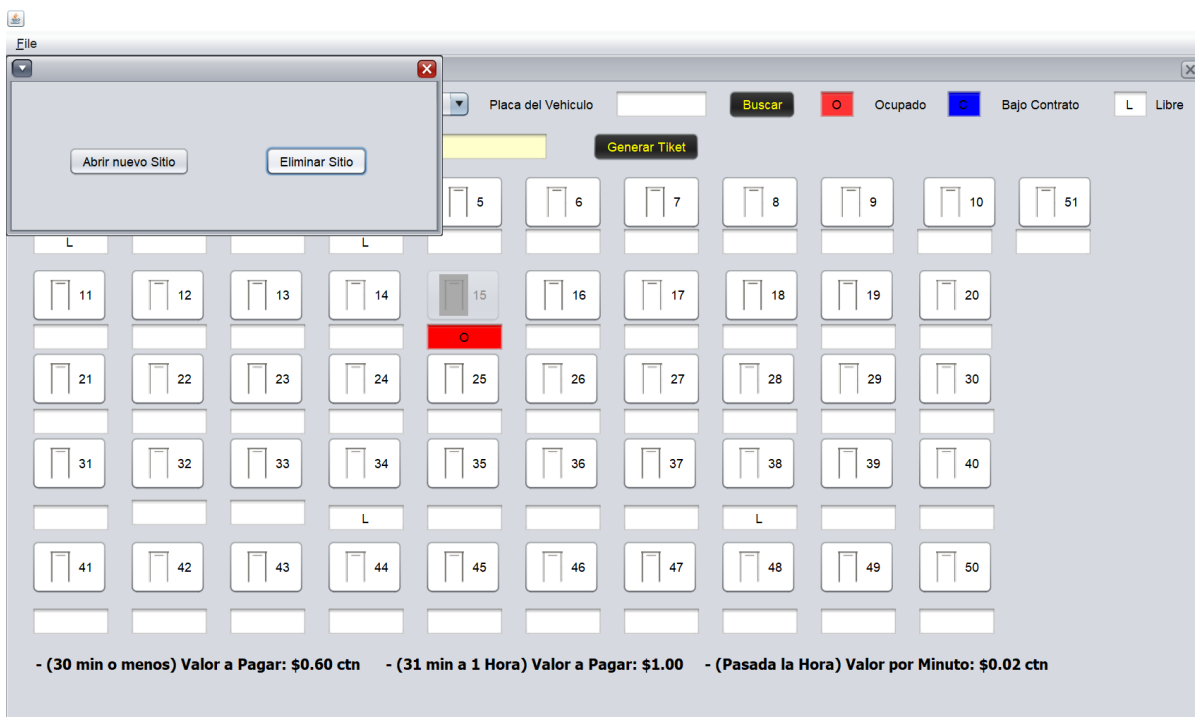
				 5	 6	 7	 8	 9	 10
L			L						
 11	 12	 13	 14	 15	 16	 17	 18	 19	 20
				<span style="color: red;">O</span>					
 21	 22	 23	 24	 25	 26	 27	 28	 29	 30
 31	 32	 33	 34	 35	 36	 37	 38	 39	 40
			L				L		
 41	 42	 43	 44	 45	 46	 47	 48	 49	 50

- (30 min o menos) Valor a Pagar: \$0.60 ctn    - (31 min a 1 Hora) Valor a Pagar: \$1.00    - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

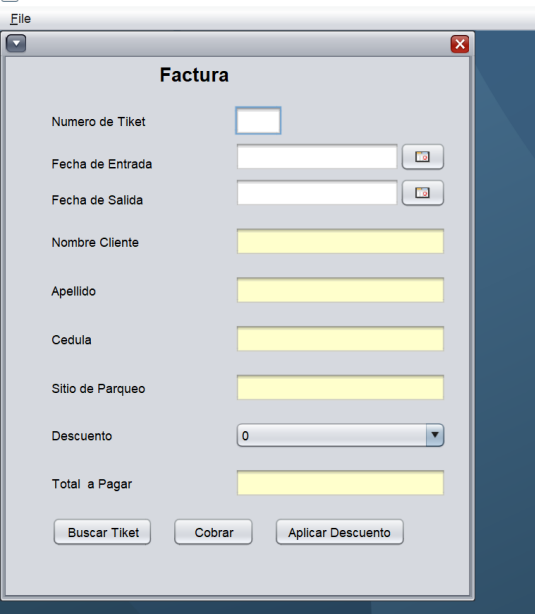
Al presionar la ventana abrir sitios no sale la siguiente ventana con dos únicos botones abrir nuevo sitio y eliminar sitio



Cada que el botón abrir nuevo sitio es presionado aparecerá un nuevo sitio en la venta sitio en este caso el botón fue pulsado 2 veces. El número máximo de veces que puede ser pulsado va a hacer 10 porque no puede abrir más de 10 sitios debido a que el espacio estará lleno.

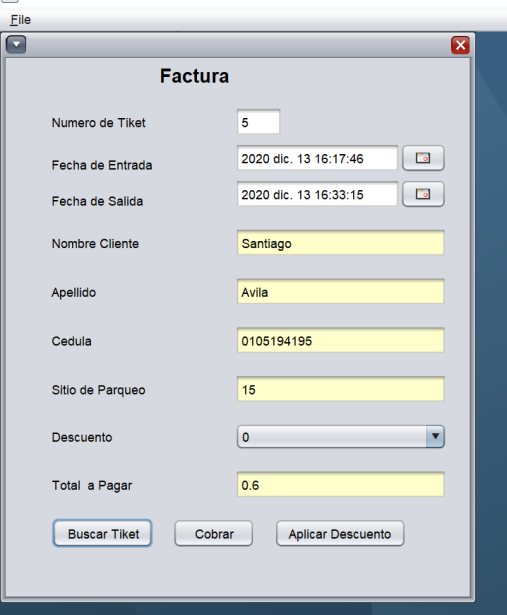


**Al presionar el botón eliminar sitio podemos ver que se borra uno de los sitios abiertos recientemente entonces cada que presione este botón eliminara un sitio empezando desde el ultimo. Solo se puede eliminar hasta que el numero de sitios sea 50 otra vez**



A screenshot of a software window titled 'Factura'. It contains several input fields for creating a ticket: 'Numero de Tiket' (empty), 'Fecha de Entrada' (empty), 'Fecha de Salida' (empty), 'Nombre Cliente' (empty), 'Apellido' (empty), 'Cedula' (empty), 'Sitio de Parqueo' (empty), 'Descuento' (set to 0), and 'Total a Pagar' (empty). At the bottom are three buttons: 'Buscar Tiket', 'Cobrar', and 'Aplicar Descuento'.

**Al presionar en cobrar se abre la siguiente ventana para realizar el cobro del tiket de los vehículos ya ingresados**



A screenshot of the same 'Factura' window, but now with data entered. The fields are: 'Numero de Tiket' (5), 'Fecha de Entrada' (2020 dic. 13 16:17:46), 'Fecha de Salida' (2020 dic. 13 16:33:15), 'Nombre Cliente' (Santiago), 'Apellido' (Avila), 'Cedula' (0105194195), 'Sitio de Parqueo' (15), 'Descuento' (0), and 'Total a Pagar' (0.6). The buttons 'Buscar Tiket', 'Cobrar', and 'Aplicar Descuento' remain at the bottom.

**Al escribir el numero del tiket a buscar y presionar en el botón buscar tiket se llenara los datos de ese tiket para hacer el respectivo cobro**



The screenshot shows a window titled 'Factura' with a menu bar containing 'File'. The form contains the following fields and controls:

- Numero de Tiket: 5
- Fecha de Entrada: 2020 dic. 13 16:17:46
- Fecha de Salida: 2020 dic. 13 16:33:15
- Nombre Cliente: Santiago
- Apellido: Avila
- Cedula: 0105194195
- Sitio de Parqueo: 15
- Descuento: A dropdown menu is open, showing options 0, 5, 10, 15, 20, and 25. The '0' option is currently selected.
- Total a Pagar: (empty)

At the bottom, there are two buttons: 'Buscar Tiket' and 'Cobrar'.

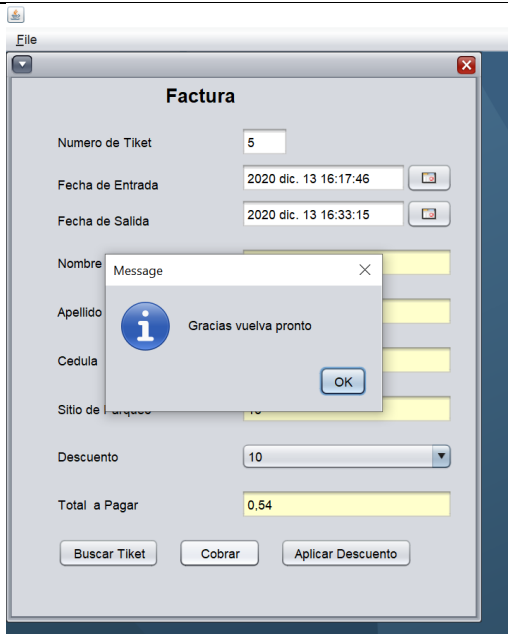
**Si se presiona en el combo box de descuento se le puede generar un descuento al cliente en ese momento seleccionado el cuanto quiere descontar.**

The screenshot shows the same 'Factura' form, but with the following changes:

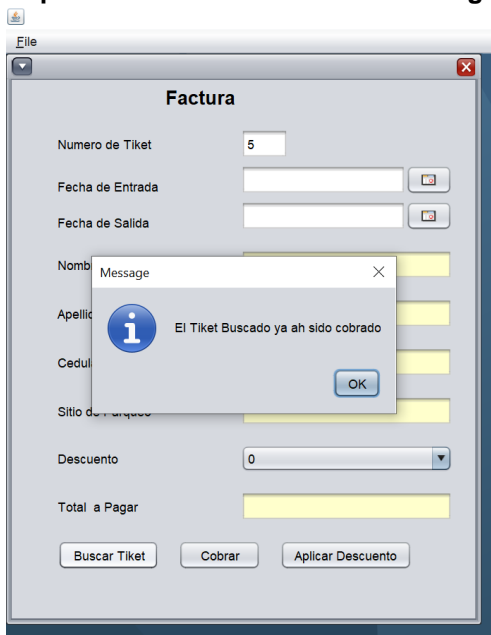
- Descuento: The dropdown menu is now closed, and '10' is selected.
- Total a Pagar: 0.54

The buttons at the bottom are now 'Buscar Tiket', 'Cobrar', and a new button 'Aplicar Descuento'.

**Si seleccionamos un descuento de los de ahí y presionamos en el botón aplicar descuento vemos que el total a pagar reduce**



**Al presionar cobrar se nos abre la siguiente ventana**



**Si buscamos un tiket ya cobrado nos saldrá el siguiente mensaje que podemos ver en la imagen**

File

Numero del Sitio  Estado **Libre** Placa del Vehiculo  **Buscar** **Ocupado** **Bajo Contrato** **L** Libre

Nombre del Cliente  Cedula  **Generar Tiket**

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
				L					
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

**Al realizar el pago y regresar a la ventana sitio podemos ver que ya se encuentra libre el sitio que fue pagado**

File

Numero de Contrato  2 Borre el numero solo en caso de busqueda del contrato

Fecha de Entrada 20 dic. 13 16:43:06

Fecha de Salida

Placa del vehiculo  **Busca Vehiculo**

Nombre Cliente  Contrato por dia: \$ 15

Apellido  Contrato por semana: \$ 50

Cedula  Contrato por mes: \$ 90

Sitio de Parqueo  Contratos no superior a un mes

Estado **pagado**

Total a Pagar  **Calcular valor**

**Crear Contrato** **Buscar Contrato** **Terminar Contrato**

**Al presionar en contrato se abre la siguiente ventana en donde tenemos que llenar cada uno de los datos del contrato que queremos hacer.**

File

Numero de Contrato: 2 Borra el numero solo en caso de busqueda del contrato

Fecha de Entrada: 20 dic. 13 16:43:06

Fecha de Salida: 020 dic. 17 16:44:36

Placa del vehiculo: aaa123 Busca Vehiculo

Nombre Cliente: Santiago Contrato por dia:\$ 15

Apellido: Avila Contrato por semana:\$ 50

Cedula: 0105194195 Contrato por mes:\$ 90

Sitio de Parqueo: 50 Contratos no superior a un mes

Estado: no pagado

Total a Pagar: 60 Calcular valor

Crear Contrato Buscar Contrato Terminar Contrato

Para poder calcular el valor a pagar tenemos que presionar en el botón de calcular valor y calcula de acuerdo a la fecha seleccionada para la salida entonces del día actual que se genera el contrato tiene un lapso de 7 días para que se le cobre contrato por día pasado los 7 días desde el octavo día hasta el día 24 se le cobrara por numero de semanas y finalmente desde el día 25 hasta el día 30 se le cobra contrato por mes. Y es de esa manera como calcula el valor a pagar.

File

Numero del Sitio: Libre Estado: Libre Placa del Vehiculo: Buscar Ocupado Bajo Contrato Libre

Nombre del Cliente: Generar Tiket

Cedula: Generar Tiket

1	2	3	4	5	6	7	8	9	10
L			L						
11	12	13	14	15	16	17	18	19	20
				L					
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
			L				L		
41	42	43	44	45	46	47	48	49	50
									C

- (30 min o menos) Valor a Pagar: \$0.60 ctn - (31 min a 1 Hora) Valor a Pagar: \$1.00 - (Pasada la Hora) Valor por Minuto: \$0.02 ctn

Si abrimos nuevamente la ventana sitio podemos ver como ya el sitio antes seleccionado en el contrato se encuentra ya bloqueado e indica que se encuentra bajo contrato.

File

Numero de Contrato: 2 Borre el numero solo en caso de busqueda del contrato

Fecha de Entrada: 20 dic. 13 16:43:06

Fecha de Salida:

Placa del vehiculo: aaa123 Busca Vehiculo

Nombre Cliente: Santiago Contrato por dia: \$ 15

Apellido: Avila Contrato por semana: \$ 50

Cedula: 0105194195 Contrato por mes: \$ 90

Sitio de Parqueo: 50 Contratos no superior a un mes

Estado: no pagado

Total a Pagar: 60.0 Calcular valor

Crear Contrato Buscar Contrato Terminar Contrato

Para poder terminar el contrato se escribe el numero del contrato para poder buscar y así activar la opción de terminar contrato presionamos el botón y el contrato habrá terminado con una multa del 10% en caso de no terminar en la fecha acordada.

File

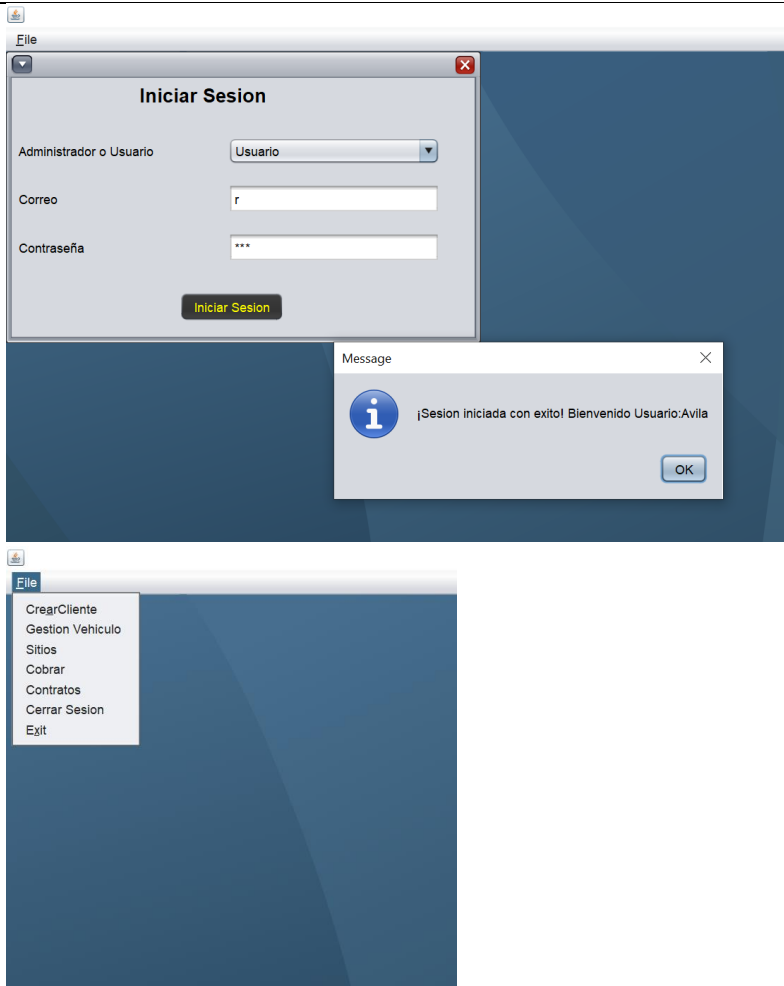
Listar Facturas y Contratos

Lista de Facturas Cargada

Listar Facturas Listar Contratos

Numero	Fecha Y Hora Ingreso	Fecha Y Hora Salida	Cliente	Sitio	Estado	Total
1	Sun Dec 13 13:07:4...	Sun Dec 13 13:09:0...	Persona(codigo=1, ...	Sitio(numero=4, est...	Pagado	0.6
2	Sun Dec 13 13:14:1...	Sun Dec 13 13:14:3...	Persona(codigo=1, ...	Sitio(numero=1, est...	Pagado	0.57
3	Sun Dec 13 14:32:2...	Sun Dec 13 14:32:4...	Persona(codigo=1, ...	Sitio(numero=1, est...	Pagado	0.51
4	Sun Dec 13 15:25:5...	Sun Dec 13 15:27:4...	Persona(codigo=1, ...	Sitio(numero=38, es...	Pagado	0.51
5	Sun Dec 13 16:17:4...	Sun Dec 13 16:33:1...	Persona(codigo=1, ...	Sitio(numero=15, es...	Pagado	0.54

Si presionamos en listar facturas y contratos si nos abre esta ventana en donde al presionar los botones se podrá listar tanto las facturas como los contratos ya cancelados.



**Si Iniciamos sesión como usuario podemos ver como el menú cambia y se le quitan algunas opciones que solo tiene el administrador.**

#### **RESULTADO(S) OBTENIDO(S):**

Como resultado podemos generar un aplicación más completa como es esta de los parqueaderos que tiene más interacción con los usuarios con el modo grafico utilizado para registrar el ingreso de un vehículo utilizando los nuevos convencionalismos de java como los Streams y a su vez nos alienta a investigar nuevas librerías las cuales podremos usar para hacer más seguro y efectivo el código como es el caso de las librerías de encriptamiento del MD5, también facilitamos la ejecución del código utilizando el patrón de diseño singleton que nos ayuda a ver las únicas instancias de un usuario.

#### **CONCLUSIONES:**

En conclusión, este proyecto ayuda a utilizar todo lo aprendido en las unidad revisadas hasta el momento desde las nuevas aplicaciones de las nuevas versiones de java hasta reflexión y los distintos patrones de diseño que se pueden ir aplicando en el programa. También fomenta a investigar mas librerías que son fáciles de usar y ayudan a facilitar algunos comandos como la librería calendar para fechas y el Md5 para encriptaciones.

#### **RECOMENDACIONES:**

No hay recomendaciones

**Estudiantes:** Romel Ávila

**Firma:**