
	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación

Universidad Politécnica Salesiana

Vicerrectorado Docente

Código del Formato:	GUIA-PRL-001
Versión:	VF1.0
Elaborado por:	Directores de Área del Conocimiento Integrantes Consejo Académico
Fecha de elaboración:	2016/04/01
Revisado por:	Consejo Académico
Fecha de revisión:	2016/04/06
Aprobado por:	Lauro Fernando Pesántez Avilés Vicerrector Docente
Fecha de aprobación:	2016/14/06
Nivel de confidencialidad:	Interno

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Descripción General

Propósito


El propósito del presente documento es definir un estándar para elaborar documentación de guías de práctica de laboratorio, talleres o centros de simulación de las Carreras de la Universidad Politécnica Salesiana, con la finalidad de lograr una homogenización en la presentación de la información por parte del personal académico y técnico docente.

Alcance

El presente estándar será aplicado a toda la documentación referente a informes de prácticas de laboratorio, talleres o centros de simulación de las Carreras de la Universidad Politécnica Salesiana.

Formatos


- Formato de Guía de Práctica de Laboratorio / Talleres / Centros de Simulación – para Docentes
- Formato de Informe de Práctica de Laboratorio / Talleres / Centros de Simulación – para Estudiantes

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

		FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES	
CARRERA: COMPUTACIÓN		ASIGNATURA: Programación Aplicada	
NRO. PRÁCTICA:	1	TÍTULO PRÁCTICA: Reflexión en Java	
OBJETIVO: Identificar los cambios importantes de Java Diseñar e Implementar las nuevas tecnicas de programación Entender cada una de las características nuevas en Java			
INSTRUCCIONES (Detallar las instrucciones que se dará al estudiante):		1. Revisar los conceptos fundamentales de Java	
		2. Establecer las características de Java en reflexión	
		3. Implementar y diseñar los nuevos componentes de reflexión	
		4. Realizar el informe respectivo según los datos solicitados.	
ACTIVIDADES POR DESARROLLAR (Anotar las actividades que deberá seguir el estudiante para el cumplimiento de la práctica)			
1. Revisar la teoría y conceptos de Java 8, 9 ,10, 11, 12, 13, 14, 15			
2. Diseñar e implementar las características de Java para generar la impresión de cualquier lista, de los modelos que tengan el campo id generar automaticamente.			
3. Probar y modificar el metodo validar para que nos permita utilizar excepciones, ademas de modificar el buscar para controlar el nullpointerexception.			
4. Realizar práctica codificando los codigos de las nuevas caracteristicas de Java y su uso dentro de una agenda telefónica.			
RESULTADO(S) OBTENIDO(S): Realizar procesos de investigación sobre los cambios importantes de Java Entender las aplicaciones de codificación de las nuevas características en base a la programación genérica Entender las funcionalidades adicionales de Java.			
CONCLUSIONES: Aprenden a trabajar en grupo dentro de plazos de tiempo establecidos, manejando el lenguaje de programación de Java.			
RECOMENDACIONES: Realizar el trabajo dentro del tiempo establecido.			

Docente / Técnico Docente: _____

Firma: _____

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		



FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES

CARRERA: COMPUTACIÓN

ASIGNATURA: Programación Aplicada

NRO. PRÁCTICA:

TÍTULO PRÁCTICA: Reflexión Java

OBJETIVO ALCANZADO:

Se puso en práctica conocimientos de programación genérica utilizando reflexiones en Java, Streams y MVC

ACTIVIDADES DESARROLLADAS

1. En esta práctica se reutilizo la aplicación de una Agenda Telefónica con login del anterior ciclo en cual se utilizo MVC y DAO como también archivos binarios para la base de datos del programa, en este caso se dará a conocer los conocimientos previos de programación genérica y Reflexión en Java por la cual el controlador se lo definirá como una clase abstracta la cual los demás controladores puedan heredar sus métodos, igualmente se utilizo la arquitectura MVC para la organización del proyecto, los datos se lo guardaran en un ArrayList temporal. Dentro del Modelo tenemos las clases Usuario y Telefono

2.

Clase Telefono:

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

```
/**
 *
 * @author
 */
```

```
public class Telefono {
```

```
    private int codigo;
    private String numero;
    private String tipo;
    private String operadora;
```

```
    private Usuario usuario;
```

```
    public Telefono() {
    }
```

```
    public Telefono(int codigo, String numero, String tipo, String operadora) {
        this.setCodigo(codigo);
        this.setNumero(numero);
        this.setTipo(tipo);
        this.setOperadora(operadora);
    }
```

```
    public int getCodigo() {
        return codigo;
    }
```

```
}

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

public String getNumero() {
    return numero;
}

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

public String getTipo() {
    return tipo;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}

public String getOperadora() {
    return operadora;
}


public void setOperadora(String operadora) {
    this.operadora = operadora;
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}

@Override
public int hashCode() {
    int hash = 5;
    hash = 37 * 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()) {
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

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

@Override
public String toString() {
    return "DATOS TELEFONO ** " + "codigo: " + codigo + " numero: " + numero + " tipo: " +
    tipo + " operadora: " + operadora + " ***";
}
}

```

Clase Usuario:

```

import java.util.Objects;

/**
 *
 * @author Juanc
 */
public class Usuario {

    /*
     */
    private String Cedula;
    private String nombre;
    private String apellido;
    private String correo;
    private String contrasena;

    public Usuario() {

    }

    public Usuario(String Cedula, String nombre, String apellido, String correo, String
    contrasena) {

        this.setCedula(Cedula);
        this.setNombre(nombre);
        this.setApellido(apellido);
        this.setCorreo(correo);
        this.setContrasena(contrasena);

    }

    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 String getCorreo() {
    return correo;
}


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

public String getContrasena() {
    return contrasena;
}

public void setContrasena(String contrasena) {
    this.contrasena = contrasena;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 97 * 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 Usuario other = (Usuario) obj;
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

    if (!Objects.equals(this.Cedula, other.Cedula)) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "Usuario{" + "Cedula=" + Cedula + ", nombre=" + nombre + ", apellido=" + apellido
+ ", correo=" + correo + ", contrasena=" + contrasena + '}';
}
}

```

Una vez definidas las clases con sus respectivos atributos, getters y setters, equals and Hashcode, se procede a crear los controladores, teniendo en cuenta de que la clase padre es abstracta. Para controlar el NullPointerException del método buscar() se lo realiza mediante un Optional el cual captura al objeto mediante el stream, después se lo validara como de tipo objeto mediante el metodo get() de la clase Optional

Clase AbstractControlador

```

package ec.edu.upes.controlador;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

/**
 *
 * @author user
 */
public abstract class AbstractControlador<E> {

    private List<E> lista;


    public AbstractControlador() {
        lista = new ArrayList();
    }

    public boolean crear(E objeto) {

        if (validar(objeto) == true) {
            return lista.add(objeto);
        }
        return false;
    }

    public Optional<E> buscar(E comparar) {
        return lista.stream().filter(objeto -> objeto.equals(comparar)).findFirst();
    }
}

```


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

public int posicion(E objetoC) {
    for (int i = 0; i < lista.size() ; i++) {
        E objetoL = lista.get(i);
        if (objetoL.equals(objetoC)) {
            return i;
        }
    }
    return -1;
}

public boolean eliminar(E objeto) {
    Optional<E> buscar = buscar(objeto);
    E objetoE = buscar.get();
    if (objetoE != null) {
        System.out.println("Verdadero");
        return lista.remove(objetoE);
    }
    System.out.println("Falso");
    return false;
}

public boolean actualizar(E objetoA) {
    int pos = posicion(objetoA);
    if (pos >= 0) {
        lista.set(pos, objetoA);
        System.out.println("TRUE");
        return true;
    }
    System.out.println("FALSE");
    return false;
}


public abstract boolean validar(E objeto);

public abstract int generarId();

public List<E> getLista() {
    return lista;
}

public void setLista(List<E> lista) {
    this.lista = lista;
}
}

```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Uno de las actividades a desarrollar era que las clases que contengan como atributo un Id se genere automáticamente en este caso solo la clase Telefono dispone de un Id puesto que la clase Usuario dispone como clave única su respectiva cedula, en la cual al generar un usuario se valida si la cedula es real. Por otro lado, solo se va a utilizar el método generar Id () en el Teléfono y se la define como un método abstracto.

Para el login se crea un método de validación si el usuario existe y se logea puede administrar sus teléfonos, para esto se creo una variable privada dentro del controlador Usuario para que respalde los datos del Usuario logeado, esto se logra instanciando los controladores desde la ventana principal y pasándolos como parámetros a las demás ventanas. Como se explico anteriormente para la validación del usuario se verifica que exista mediante la verificación de su cedula ciudadana ecuatoriana, el método de la validación de la cedula se lo reciclo de un programa del primer ciclo.

Clase Controlador Usuario

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

import ec.edu.ups.modelo.Usuario;

/**
 *
 * @author user
 */
public class ControladorUsuario extends AbstractControlador<Usuario> {

    private Usuario usuario;

    public ControladorUsuario() {

    }

    @Override
    public boolean validar(Usuario usuario) {
        int suma = 0;
        String x = usuario.getCedula();
        if (x.length() == 9) {
            return false;
        } else {
            int a[] = new int[x.length() / 2];
            int b[] = new int[(x.length() / 2)];
            int c = 0;
            int d = 1;
            for (int i = 0; i < x.length() / 2; i++) {
                a[i] = Integer.parseInt(String.valueOf(x.charAt(c)));
                c = c + 2;
                if (i < (x.length() / 2) - 1) {
                    b[i] = Integer.parseInt(String.valueOf(x.charAt(d)));
                    d = d + 2;
                }
            }

            for (int i = 0; i < a.length; i++) {
                a[i] = a[i] * 2;
                if (a[i] > 9) {
                    a[i] = a[i] - 9;
                }
            }
        }
    }
}
```

```
    }
    suma = suma + a[i] + b[i];
}
int aux = suma / 10;
int dec = (aux + 1) * 10;
if ((dec - suma) == Integer.parseInt(String.valueOf(x.charAt(x.length() - 1)))) {
    return true;
} else if (suma % 10 == 0 && x.charAt(x.length() - 1) == '0') {
    return true;
} else {
    return false;
}
}
}


@Override
public int generarId() {
    return 0;
}

public boolean iniciarSesion(String correo, String pass) {
    for (Usuario usu : super.getLista()) {
        Usuario u = (Usuario) usu;
        if (u.getCorreo().equals(correo) && u.getContrasena().equals(pass)) {
            this.usuario = u;
            return true;
        }
    }
    return false;
}

public Usuario getUsuario() {
    return usuario;
}

public void setUsuario(Usuario usuario) {
    this.usuario = usuario;
}
}
```

En cuanto a diseño de las ventanas de la interfaz gráfica es el siguiente

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		


Ventana Principal

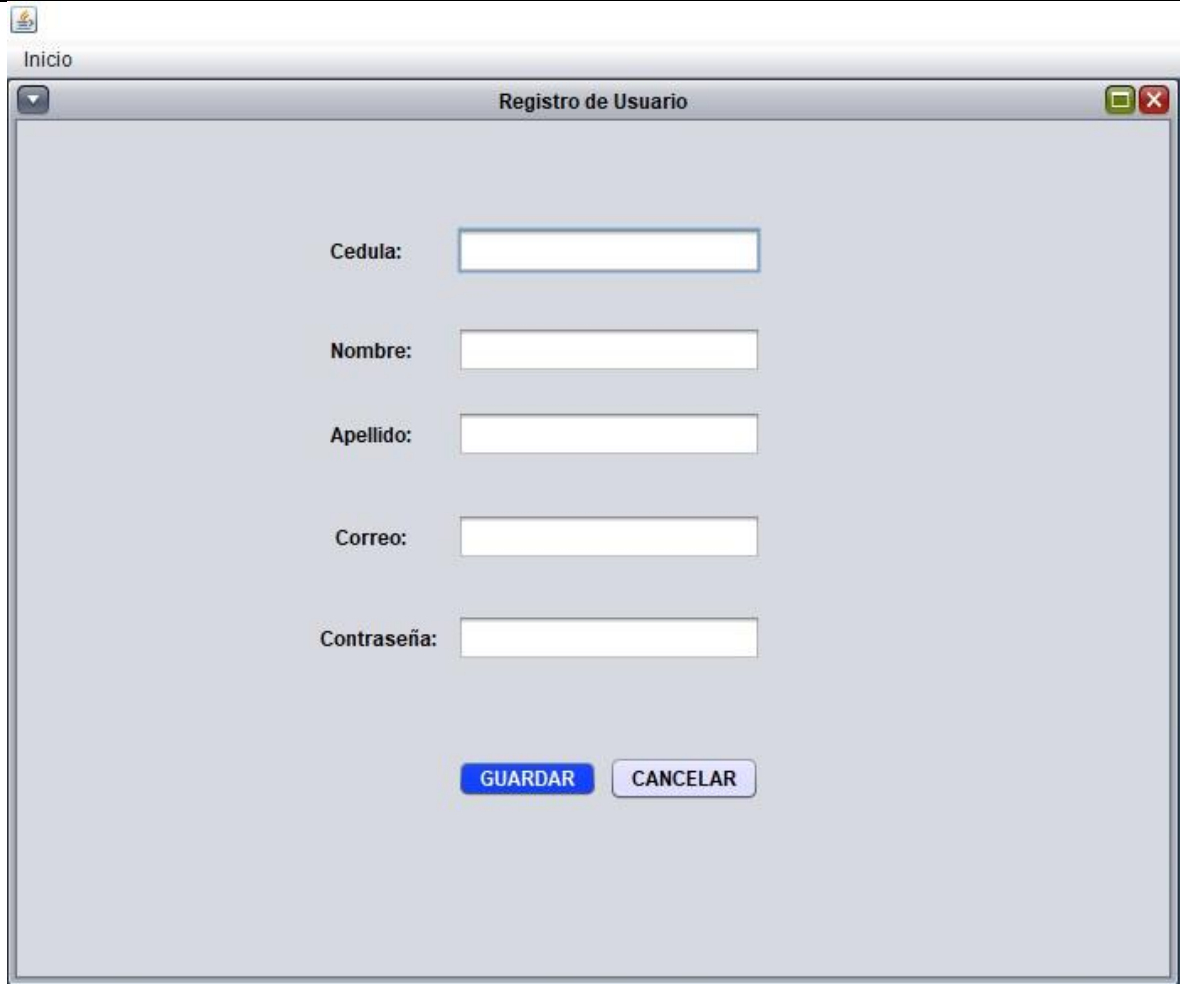


**Ventana
Login**




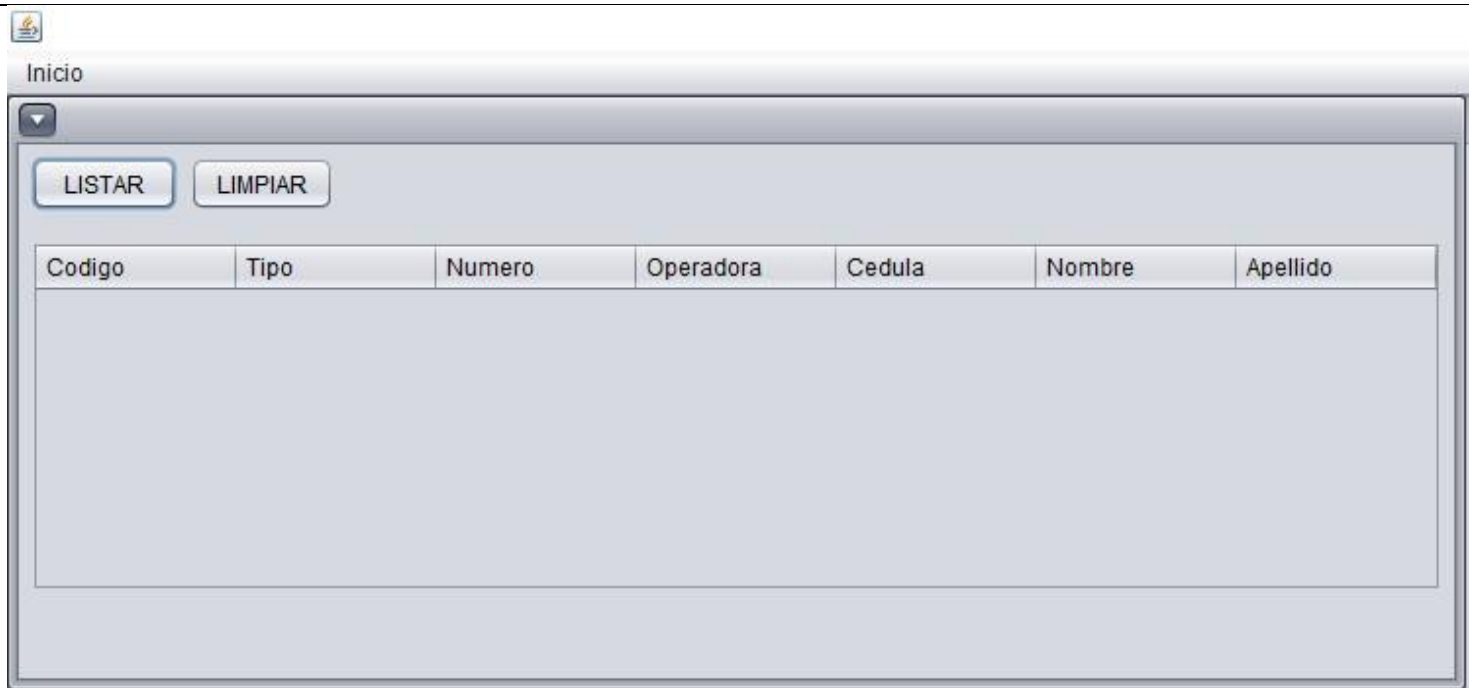
Ventana de registro de un usuario

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		



Ventana para listar todos los teléfonos de todos los usuarios

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

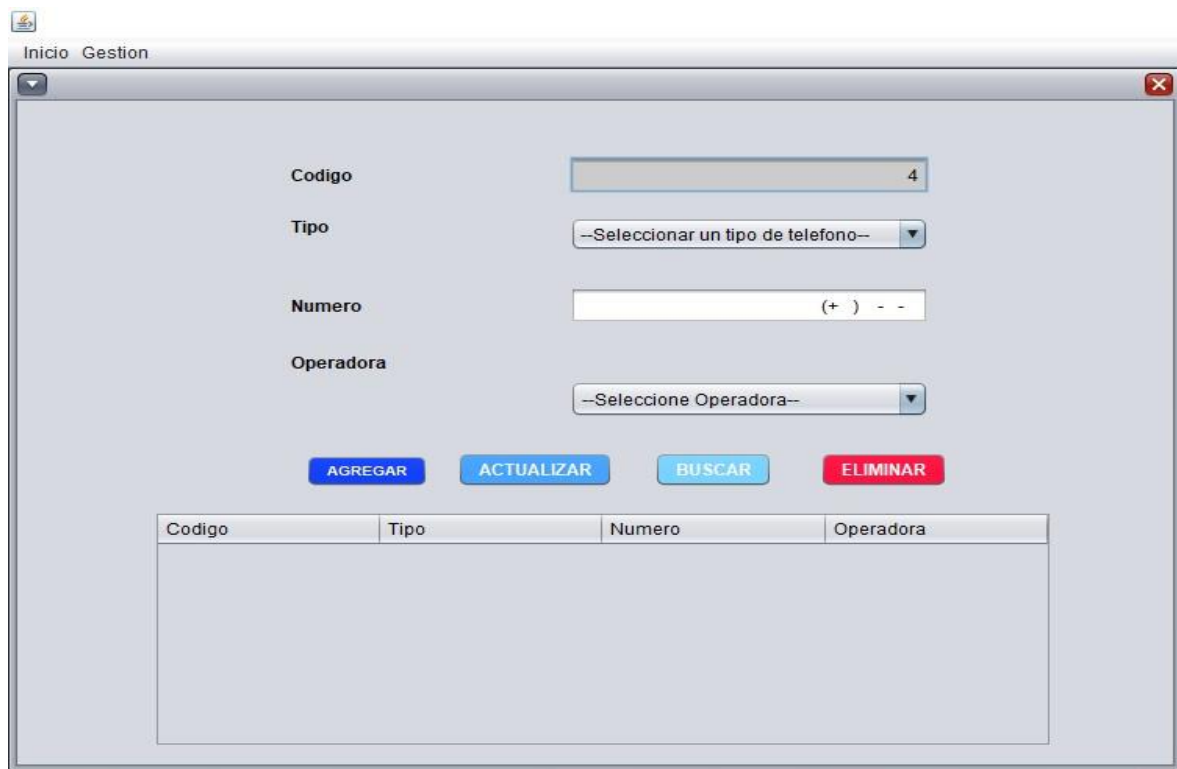


Inicio

LISTAR LIMPIAR

Codigo	Tipo	Numero	Operadora	Cedula	Nombre	Apellido
--------	------	--------	-----------	--------	--------	----------

Ventana que gestiona los teléfonos



Inicio Gestion

Codigo 4


Tipo --Seleccionar un tipo de telefono--

Numero (+) - -

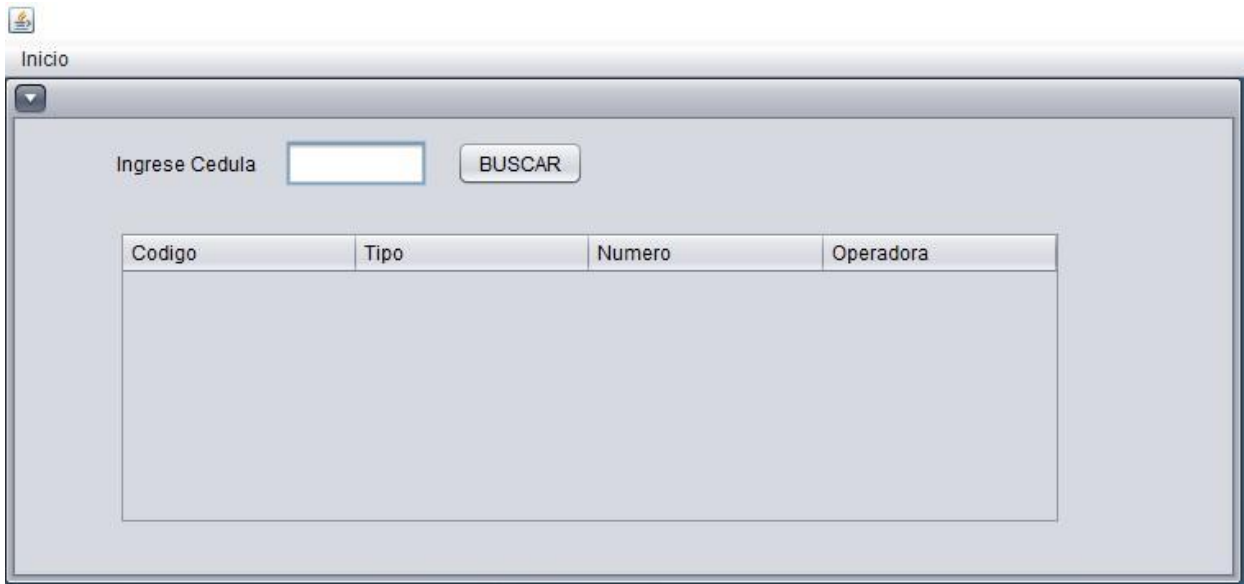
Operadora --Seleccione Operadora--

AGREGAR ACTUALIZAR BUSCAR ELIMINAR

Codigo	Tipo	Numero	Operadora
--------	------	--------	-----------

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Ventana que lista los teléfonos por usuario con la cedula como parámetro de búsqueda



Con respecto al código fuente de las ventanas, para una mejor presentación por favor revisar del GitHub mediante el siguiente link:


<https://github.com/juancvxpro/Reflexion-en-Java/tree/master/src/ec/edu/ups/vista>

RESULTADO(S) OBTENIDO(S):

Se logro la impresión de una lista en una tabla dentro de una interfaz utilizando conceptos sobre MVC, métodos CRUD, programación genérica y reflexión en Java, aparte de resolver el NullPointerException que se generaba debido a que no se guardaban parámetros dentro del Objeto, por lo tanto, se uso la interface Optional para rescatar el objeto desde el stream, luego des del método get () de la clase Optional se validó el objeto.

CONCLUSIONES:

- La aplicación de patrones de diseño y arquitecturas dentro de la aplicación creada son fundamentales y facilitan el trabajo a la vez que simplifican el código para que no venga a ser muy pesado y tenga una mayor aceptabilidad dentro de los ámbitos en los que se deseen proyectar su respectivo uso. Tanto como la programación genérica que ahorra recurso y código.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

RECOMENDACIONES: Para una revisión mas exacta revisar el código de las ventanas desde el link anticipado.

Nombre de estudiante: Juan José Cordova Calle

Firma de estudiante:

