
	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: Examen Practico Java	
OBJETIVO: Identificar los cambios importantes de Java Diseñar e Implementar expresiones regulares Entender la cada uno 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 programación genérica	
		3. Implementar y diseñar los nuevos componentes de programación genérica	
		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			
2. Diseñar e implementar las características de Java para generar una expresion regular.			
3. Probar su funcionamiento y rendimiento dentro de los equipos de cómputo de programación genérica.			
4. Realizar práctica codificando los codigos de las nuevas características de Java y su uso dentro de un sistema escolar.			
Enunciado Se desea generar un sistema que me permita extraer infomación del internet a traves de expresiones regulares, esta informacion permitira vincular actividades desarrolladas del los niños con aplicaciones mobiles que permitan apoyar en el desarrollo de las actividades planteadas (https://play.google.com/store?hl=es&gl=US). Adicionalmente, se debe realizar un sistema de gestion de alumnos y actividades planificadas por curso, dentro de este sistema se debe realizar un procesos de administracion de usuarios los mismo que son los docentes de cada curso escolar, en este sentido solo debemos tener un administrador (Rector) el encargado de crear docentes y el curso que se le asigna. Ejemplo Rector: Docentes: 1. Diego Quisi 2. Vladimir Robles 3. Etc. Cursos: 1 de basica 2 de basica 3 basica			

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

Asignacion de Curso – Docente

1 Basica -> Diego Quisi

2 Basica -> Vladimir Robles

Dentro de cada curso el docente gestionara los estudiantes y las actividades planificadas para el curso, estas actividades tendra una opcion de buscar aplicaciones moviles dentro de la tiendas de play store, obtenidas desde el internet, dentro de esta información lo importante es mostrar el link y una descripción para ello deberán utilizar expresiones regulares.

Ejemplo Docentes:

Alumnos

1. Juan Perez

2. Maria Peralta

3. .

Actividades:

1. Suma de numeros -> Obtener aplicaciones moviles (Link y Titulo)
2. Resta de numeros -> Obtener aplicaciones moviles
3. Oraciones compuestas -> Obtener aplicaciones moviles
4. Etc.

Toda esta infomación sera almacenada dentro de archivos y deberan tener aplicado al menos una patron de diseño y las nuevas características de programación de Java 8 o superior.

Al finalizar, generar el informe de la practica en formato PDF y subir todo el proyecto incluido el informe al repositorio personal.

La fecha de entrega: 23:55 del 01 de diciembre del 2020.

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 y expresiones regulares.

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		

```

URL      urlObject      =      new      URL("https://play.google.com/store/search?q=" +
textoBusqueda.replaceAll("\\s", "\\+"));
URLConnection urlConnection = urlObject.openConnection();
urlConnection.setRequestProperty("User-Agent", "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.11 (KHTML, like Gecko) Chrome/23.0.1271.95 Safari/537.11");

BufferedReader      bufferedReader      =      new      BufferedReader(new
InputStreamReader(urlConnection.getInputStream(), "UTF-8"));
String inputLine;
while ((inputLine = bufferedReader.readLine()) != null) {
    stringBuilder.append(inputLine);
}
}catch(IOException e){

}

resultado= controladorActividades.obtenerUrlGoogle(stringBuilder.toString());
Lista(resultado);

```

3. Probar su funcionamiento y rendimiento dentro de los equipos de cómputo de programación genérica.

Programación Genérica

Con la ayuda de la programación genérica es que nosotros solo tenemos un controlador el cual es abstracto para así nosotros poder acceder a todos los métodos en comunes que tienen los otros controladores solo declarando la variable o el objeto con el cual se va a trabajar estos métodos especialmente son los métodos del CRUD, además debido a que estamos trabajando y almacenando la información dentro de los archivos objetos tenemos que adaptar los controladores de las clases que heredan de la clase abstracta debido a que así nosotros pasaremos una ruta adecuada para poder manejar la información dentro de esos archivos

A continuación, se encuentran los códigos de todos los controladores del programa en donde se podrá constatar el uso de una programación genérica adecuada

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;

```

/**

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

```

*
* @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;
        }
    }
}
```

Controlador Alumno

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

import ec.edu.ups.modelo.Alumno;
```


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

```

/**
 *
 * @author NANCY
 */
public class ControladorAlumno extends Controlador<Alumno>{

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

    @Override
    public boolean validar(Alumno obj) {
        return true;
    }

}

```

Controlador Curso

```

package ec.edu.ups.controlador;

import ec.edu.ups.modelo.Curso;

/**
 *
 * @author NANCY
 */
public class ControladorCurso extends Controlador<Curso>{

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

    @Override
    public boolean validar(Curso obj) {
        return true;
    }

}

```


Controlador Rector

El controlador Rector no accede a la clase genérica debido a que este ya está creado previamente como podemos observar entonces no tiene la necesidad de crearse en un archivo obj ni acceder a los métodos del Crud es por eso que no se le pasa al controlador general

```

package ec.edu.ups.controlador;

```

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

```
import ec.edu.ups.modelo.Rector;

/**
 *
 * @author NANCY
 */
public class ControladorRector {

    Rector rector = new Rector("rector","123","0123456789", "Juan", "Perez");


    public Rector iniciarSesion(String correo, String contrase){
        if (rector.getCorreo().equals(correo) && rector.getContrase().equals(contrase)) {
            return rector;
        }
        return null;
    }
}
```

Patrón de Diseño 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 el s ele desplieguen ciertas ventanas las cuales solo va a manejar el docente y a la vez controla que ningún otro docente o rector puede iniciar sesión en ese momento como vamos a poder observar en las siguientes imágenes que son parte del código del Docente clase que se encuentra en el paquete modelo. Entonces este getInstance no ayuda a verificar si un docente ya inicio sesión o no con el constructor privado creando atributos estáticos en la clase.

```
31 private Docente(int codigo, String correo, String contrase, Curso curso, String cedula, String nombre, St
32     this.codigo = codigo;
33     this.correo = correo;
34     this.contrase = contrase;
35     this.curso = curso;
36     this.cedula = cedula;
37     this.nombre = nombre;
38     this.apellido = apellido;
39 }
40 public static Docente getInstance(){
41     if(Docente.instance==null){
42         Constructor<Docente> constructor;
43         try{
44             constructor=Docente.class.getDeclaredConstructor();
45             constructor.setAccessible(true);
46
47             Docente docente=constructor.newInstance();
48
49             Docente.instance=docente;
50         }catch(Exception e){
51             e.printStackTrace();
52         }
53         return Docente.instance;
54     }
55     return Docente.instance;
```

Reflexión

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


La reflexión nos ayuda a crear los docentes de una manera en la que vamos setiando parte por parte ya que esta nos ayuda a acceder a constructor

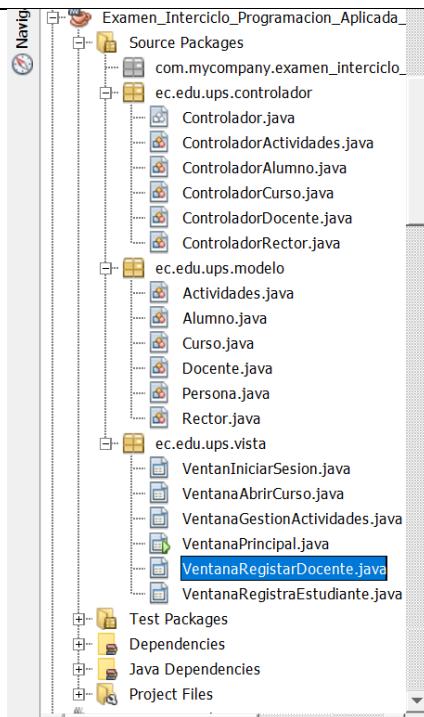
```

315         try{
316             Constructor<Docente> constructor= Docente.class.getDeclaredConstructor();
317             constructor.setAccessible(true);
318             Docente docente= constructor.newInstance();
319             int codigoc = Integer.parseInt(txtCodigoCurso.getText());
320             Curso c = new Curso(codigoc);
321             Curso curso = controladorCurso.read(c);
322             docente.setCodigo(codigo);
323             docente.setCorreo(correo);
324             docente.setContrase(contra);
325             docente.setCurso(curso);
326             docente.setCedula(cedula);
327             docente.setNombre(nombre);
328             docente.setApellido(Apellido);
329             boolean resultado = controladorDocente.create(docente);
330             limpiar();
331             JOptionPane.showMessageDialog(this, "Operación : " + resultado);
332             dispose();
333
334         } catch (NoSuchMethodException ex) {
335             Logger.getLogger(VentanaRegistrarDocente.class.getName()).log(Level.SEVERE, null, ex);
336         } catch (SecurityException ex) {
337             Logger.getLogger(VentanaRegistrarDocente.class.getName()).log(Level.SEVERE, null, ex);
338         } catch (InstantiationException ex) {
339             Logger.getLogger(VentanaRegistrarDocente.class.getName()).log(Level.SEVERE, null, ex);

```

Imagen de las clases creadas en el proyecto utilizando patrón MVC

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



Código de las clases en el paquete del modelo

```
public class Alumno implements Serializable {
```

```
    private int codigo;
    private String comportamiento;
    private Curso curso;
    private String cedula;
    private String nombreApellido;
```

```
    public Alumno() {
    }

```

```
    public Alumno(int codigo, String comportamiento, Curso curso, String cedula, String nombreApellido) {
        this.codigo = codigo;
        this.comportamiento = comportamiento;
        this.curso = curso;
        this.cedula = cedula;
        this.nombreApellido = nombreApellido;
    }

```

```
    public Alumno(String nombreApellido) {
        this.nombreApellido = nombreApellido;
    }

```

```
public String getComportamiento() {  
    return comportamiento;  
}  
  
public void setComportamiento(String comportamiento) {  
    this.comportamiento = comportamiento;  
}  
  
public Curso getCurso() {  
    return curso;  
}  
  
public void setCurso(Curso curso) {  
    this.curso = curso;  
}  
  
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 getNombreApellido() {  
    return nombreApellido;  
}  
  
public void setNombreApellido(String nombreApellido) {  
    this.nombreApellido = nombreApellido;  
}  
  
@Override  
public int hashCode() {  
    int hash = 3;  
    hash = 53 * hash + Objects.hashCode(this.nombreApellido);  
    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 Alumno other = (Alumno) obj;
    if (!Objects.equals(this.nombreApellido, other.nombreApellido)) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "Alumno{" + "codigo=" + codigo + ", comportamiento=" + comportamiento + ", curso=" + curso +
", cedula=" + cedula + ", nombreApellido=" + nombreApellido + '}';
}

}

package ec.edu.ups.modelo;

import java.io.Serializable;

/**
 *
 * @author NANCY
 */
public class Curso implements Serializable {
    private int codigo;
    private String nombre;

    public Curso() {
```

```
}

public Curso(int codigo, String nombre) {
    this.codigo = codigo;
    this.nombre = nombre;
}

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

public String getNombre() {
    return nombre;
}

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

public int getCodigo() {
    return codigo;
}

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

@Override
public int hashCode() {
    int hash = 5;
    hash = 97 * 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 Curso other = (Curso) obj;  
if (this.codigo != other.codigo) {  
    return false;  
}  
return true;  
}
```

```
@Override  
public String toString() {  
    return "Curso{" + "codigo=" + codigo + ", nombre=" + nombre + '}';  
}
```

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

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

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

```
public class Docente implements Serializable {
```

```
    private int codigo;  
    private String correo;  
    private String contrase;
```

```
    private Curso curso;  
    public static Docente instance;
```

```
    private String cedula;  
    private String nombre;  
    private String apellido;
```

```
    private Docente() {  
    }
```

```
    private Docente(int codigo, String correo, String contrase, Curso curso, String cedula, String nombre,  
String apellido) {  
        this.codigo = codigo;  
        this.correo = correo;  
        this.contrase = contrase;  
        this.curso = curso;  
        this.cedula = cedula;
```



```
this.nombre = nombre;
this.apellido = apellido;
}
public static Docente getInstance(){
    if(Docente.instance==null){
        Constructor<Docente> constructor;
        try{
            constructor=Docente.class.getDeclaredConstructor();
            constructor.setAccessible(true);

            Docente docente=constructor.newInstance();

            Docente.instance=docente;
        }catch(Exception e){
            e.printStackTrace();
        }
        return Docente.instance;
    }
    return Docente.instance;
}

public int getCodigo() {
    return codigo;
}

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

public String getCorreo() {
    return correo;
}

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

public String getContrase() {
    return contrase;
}

public void setContrase(String contrase) {
    this.contrase = contrase;
}

public Curso getCurso() {
    return curso;
```

```
}

public void setCurso(Curso curso) {
    this.curso = curso;
}

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;
}

@Override
public int hashCode() {
    int hash = 7;
    hash = 97 * hash + Objects.hashCode(this.correo);
    hash = 97 * hash + Objects.hashCode(this.contrase);
    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 Docente other = (Docente) obj;
if (!Objects.equals(this.correo, other.correo)) {
    return false;
}
if (!Objects.equals(this.contrase, other.contrase)) {
    return false;
}
return true;
}

@Override
public String toString() {
    return "Docente{" + "codigo=" + codigo + ", correo=" + correo + ", contrase=" + contrase + ", curso=" +
    curso + ", cedula=" + cedula + ", nombre=" + nombre + ", apellido=" + apellido + '}';
}

public class Persona {
    private String cedula;
    private String nombre;
    private String apellido;

    public Persona() {
    }

    public Persona(String cedula, String nombre, String apellido) {
        this.cedula = cedula;
        this.nombre = nombre;
        this.apellido = apellido;
    }

    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;
}

@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 Persona other = (Persona) obj;
    if (!Objects.equals(this.cedula, other.cedula)) {
        return false;
    }
    return true;
}

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

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

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

```

import java.io.Serializable;

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

    public Rector() {
    }

    public Rector(String correo, String contrase, String cedula, String nombre, String apellido) {
        super(cedula, nombre, apellido);
        this.correo = correo;
        this.contrase = contrase;
    }

    public String getCorreo() {
        return correo;
    }

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


    public String getContrase() {
        return contrase;
    }

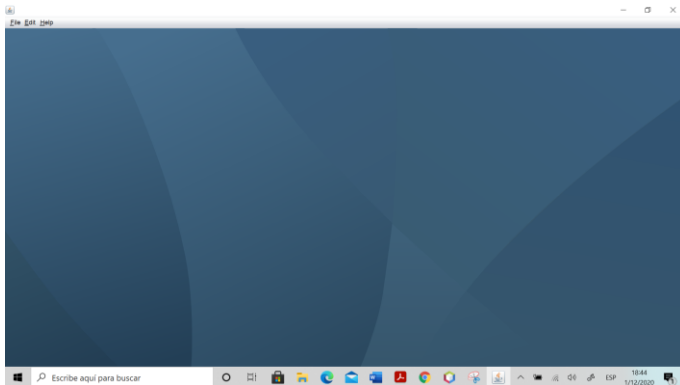
    public void setContrase(String contrase) {
        this.contrase = contrase;
    }

    @Override
    public String toString() {
        return "Rector{" + "correo=" + correo + ", contrase=" + contrase + '}';
    }
}

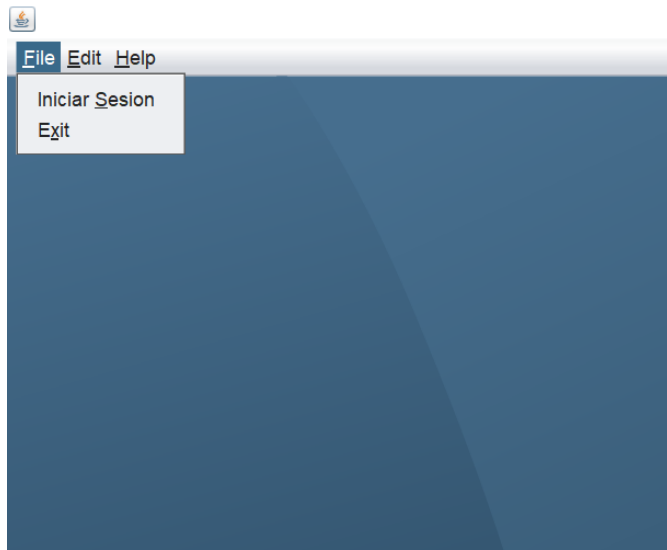
```

- Realizar práctica codificando los códigos de las nuevas características de Java y su uso dentro de un sistema escolar.

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




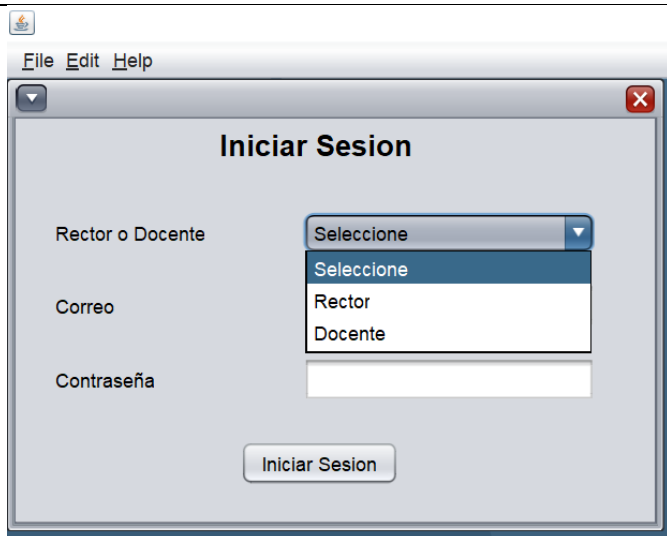
Al Ejecutar el programa primero aparece la siguiente ventana



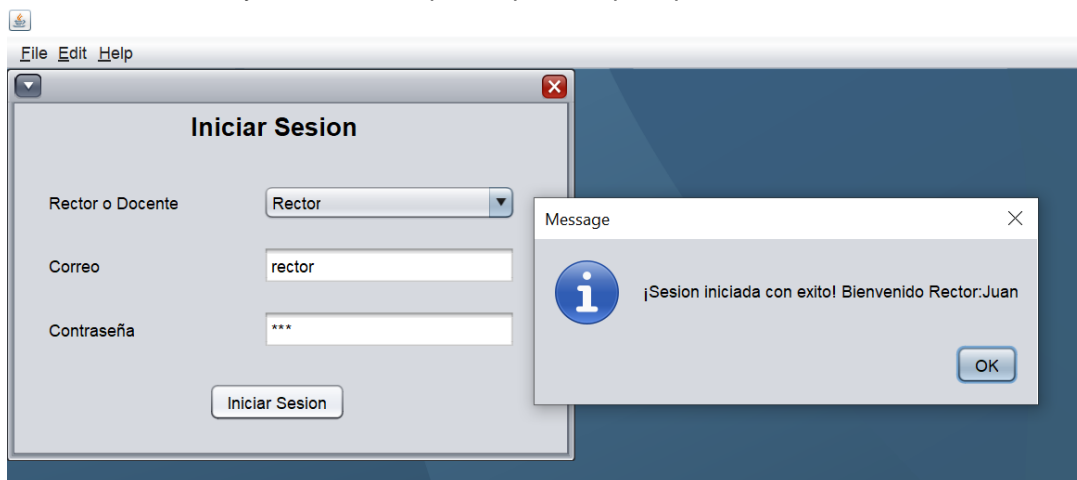
Si abrimos el menú file clara mente vemos que solo podemos iniciar sesión




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

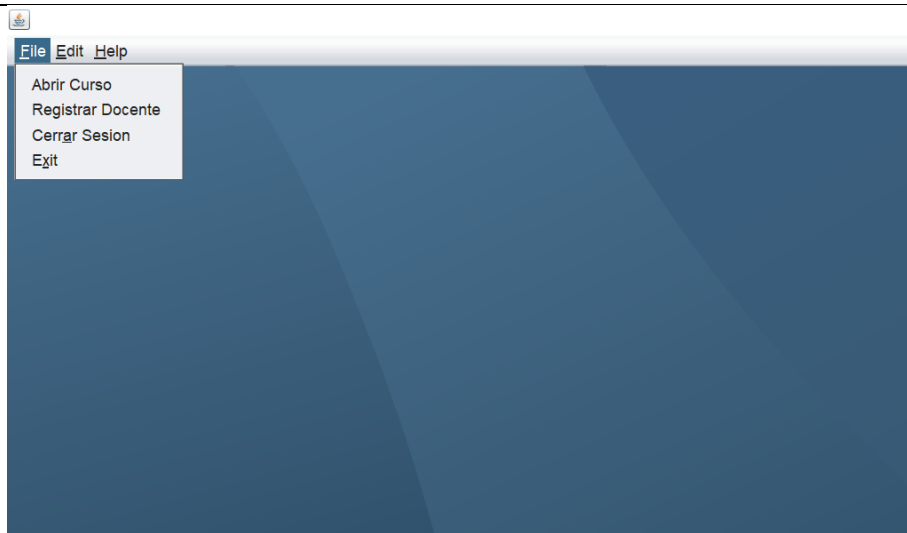


Al presionar iniciar sesión se abre la siguiente ventana en donde tenemos que escoger si va ingresar el rector o un docente y llenar los campos requeridos para poder iniciar.

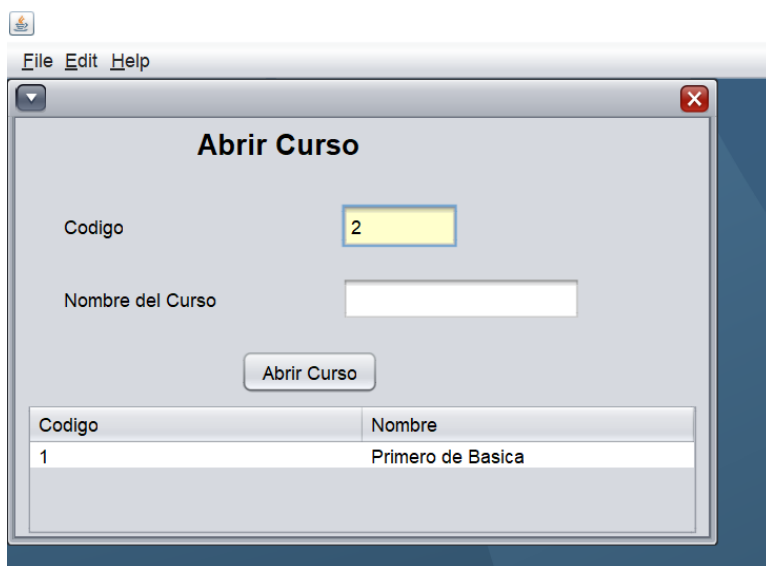


Si inicia Sesión el rector nos saldrá la siguiente pantalla


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

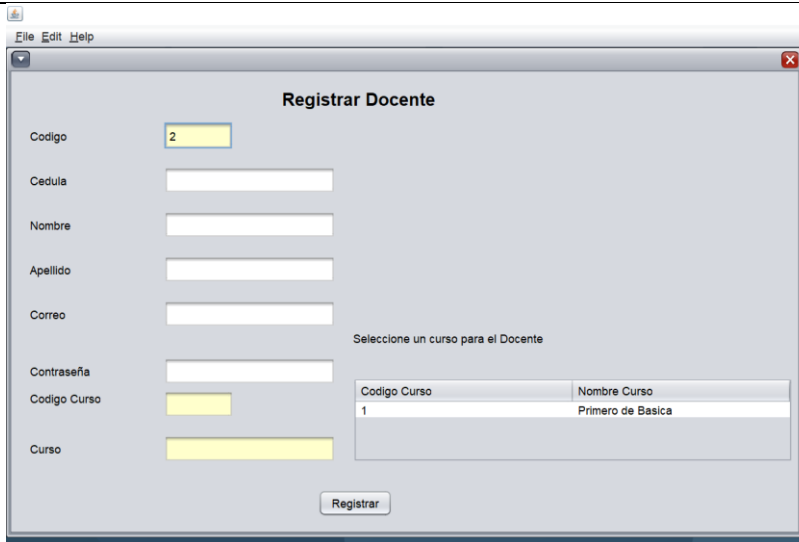


Se abre el siguiente menú bar cuando inicia el rector

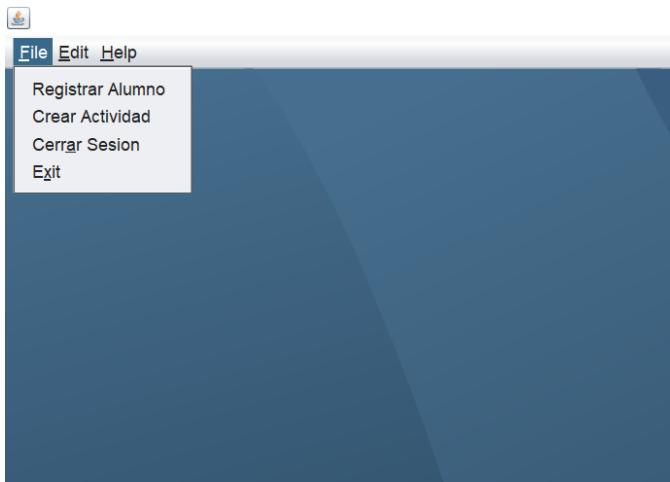


Si seleccionamos abrir curso se nos despliega la siguiente ventana en donde pasamos un nombre al curso que deseamos abrir y a la vez si seleccionamos en la tabla podemos actualízalo es decir cambiarlo de nombre


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

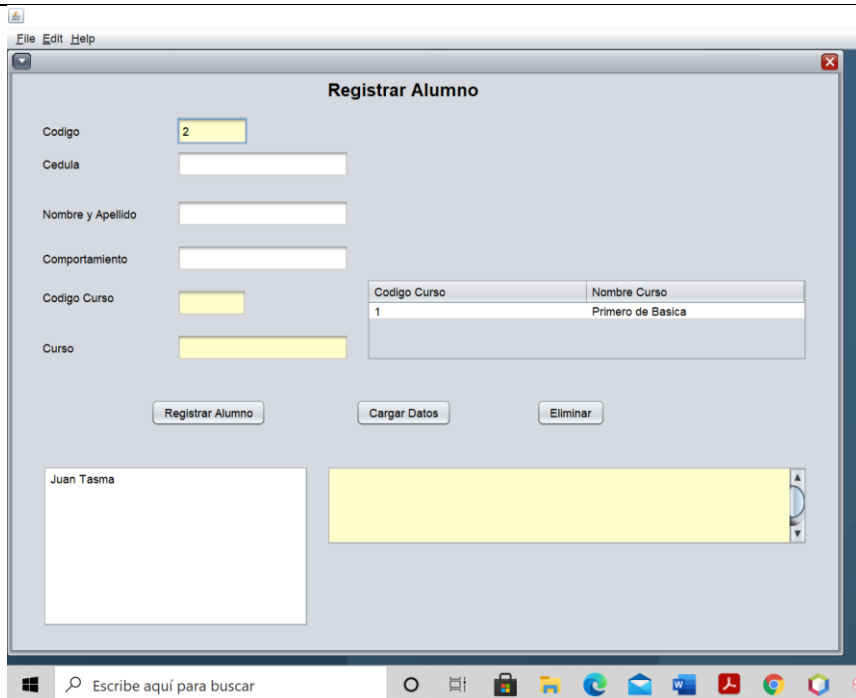


Si seleccionamos registrar Docente se abre la siguiente ventana en donde tenemos que llenar cada campo para poder registrar a un docente y a su vez se selecciona de la tabla el curso al cual se le va a asignar.

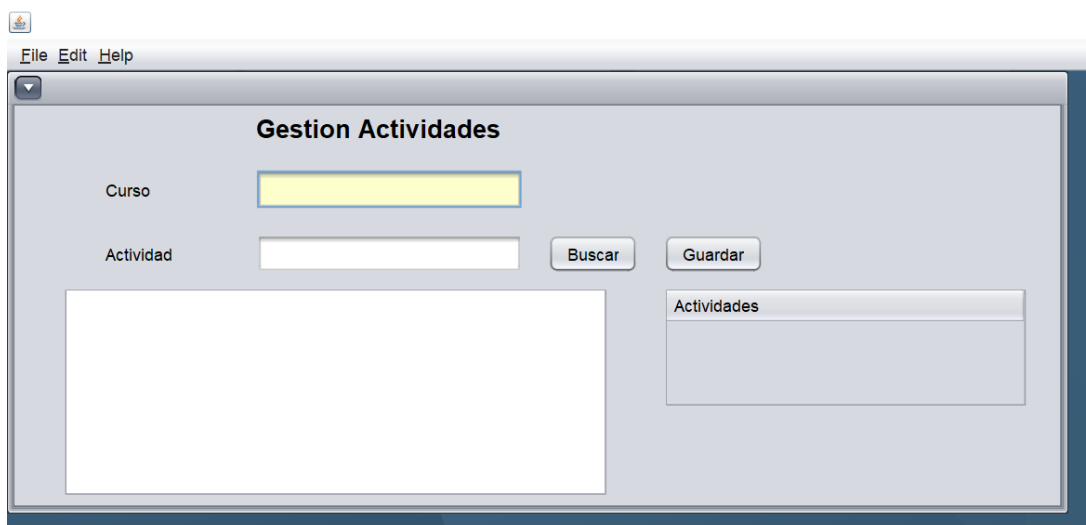


Si iniciamos sesión como docente se nos despliega el siguiente menú bar con otras opciones diferentes que al rector


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



Si seleccionamos registrar alumno se abre esta ventana en donde llenamos todos los datos y se va a crear un alumno con el curso seleccionado esto se hace en caso de actualizar los datos del alumno sea cambiado de curso se lo podrá hacer porque en el jlist donde se ven los nombres saldrá solo la lista de alumnos del curso del profesor que inicio sesión, para poder actualizar los datos se debe seleccionar un nombre del jlist presionar en cargar para poder obtener los datos nuevamente y en el panel txt se vera todos los datos antes de ser modificados.



Finalmente tenemos la ventana de gestión de actividades en donde nos obtiene las aplicaciones pasadas desde la play store

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

RESULTADO(S) OBTENIDO(S):

Como resultados es que podemos generar una aplicación en donde se pueda buscar ciertas cosas en internet para obtenerlas en la computadora gracias a las expresiones regulares.

CONCLUSIONES:

En conclusión, esta parte práctica del examen fue de mucha ayuda para poder aplicar todos los conceptos vistos en estas dos unidades de una manera correcta para así poder llegar a generar un aplicación funcional con un código menos extenso gracias a las nuevas técnicas de programación vistas en estas dos primeras unidades.

RECOMENDACIONES:

No existen recomendaciones

Estudiantes: Romel Ávila

Firma:

