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

CARRERA: Computación		**ASIGNATURA**: Programación aplicada
Examen Final	2	**TÍTULO PRÁCTICA**: Examen Final Juego Ruleta

OBJETIVO ALCANZADO:

- Consolidar los conocimientos adquiridos en la asignatura, aplicando nuevas características de java y persistencia de datos.

ACTIVIDADES DESARROLLADAS

Para el desarrollo del examen se siguió el siguiente diagrama a continuación:

La arquitectura con la que se diseñó el programa fue la misma de todas las buenas prácticas de la asignatura modelo, vista y controlador.

Entonces dentro del paquete **modelo** tenemos únicamente una clase de tipo Entity donde cuenta con todos los atributos de los jugadores de la ruleta.

EntityClass Jugador Ruleta

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

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class JugadorRuleta implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;

     @Column(name ="nombre")
    private String nombre;

    @Column(name ="numero")
    private int numero; //el número de la ruleta o 0/1

    @Column(name ="saldo")
```

```
     private float saldo;

    @Column(name ="cantidadApuesta")
    private int cantidadApuesta;

    @Column(name ="isPar")
     private int isPar; //Se verifica si se escogió par caso contrario, impar

    @Column(name ="nApuestas")
     private int nApuestas; //numero de apuestas

    @Column(name ="nGanadas")
     private int nWins; //numero de apuestas ganadas

    @Column(name ="nPerdidas")
     private int nLost; //numero de apuestas perdidas

    @Column(name ="dAupuesta")
     private boolean isDuplicar; //Se verifica si se escogió 'martingala' y se
duplican sus próximas apuestas


    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getNumero() {
        return numero;
    }

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

    public float getSaldo() {
        return saldo;
    }

    public void setSaldo(float saldo) {
        this.saldo = saldo;
    }

    public int getCantidadApuesta() {
        return cantidadApuesta;
    }

    public void setCantidadApuesta(int cantidadApuesta) {
        this.cantidadApuesta = cantidadApuesta;
    }
```

```
    public int getIsPar() {
        return isPar;
    }

    public void setIsPar(int isPar) {
        this.isPar = isPar;
    }

    public int getnApuestas() {
        return nApuestas;
    }

    public void setnApuestas(int nApuestas) {
        this.nApuestas = nApuestas;
    }

    public int getnWins() {
        return nWins;
    }

    public void setnWins(int nWins) {
        this.nWins = nWins;
    }

    public int getnLost() {
        return nLost;
    }

    public void setnLost(int nLost) {
        this.nLost = nLost;
    }

    public boolean isIsDuplicar() {
        return isDuplicar;
    }

    public void setIsDuplicar(boolean isDuplicar) {
        this.isDuplicar = isDuplicar;
    }


    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not
set
        if (!(object instanceof JugadorRuleta)) {
            return false;
        }
        JugadorRuleta other = (JugadorRuleta) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
```

```
        return true;
    }

    @Override
    public String toString() {
        return "ec.edu.ups.modelo.JugadorRuleta[ id=" + id + " ]";
    }

}
```

Controlador

Utilizando los conceptos de programación genérica creamos un controlador abstracto el cual tendrán los métodos crud para la persistencia de datos en PostgreSQL

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



import ec.edu.ups.utils.JPAUtils;
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManager;

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

    private List<E> lista;
    private Class<E> clase;
    private EntityManager em;

    /**
     *
     */
    public AbstractControlador() {
        lista= new ArrayList<>();
        Type t = getClass().getGenericSuperclass();
        ParameterizedType pt =(ParameterizedType) t;
        clase= (Class) pt.getActualTypeArguments()[0];
        em=JPAUtils.getEntityManager();
    }

    public AbstractControlador(EntityManager em) {
        lista= new ArrayList<>();
        Type t= getClass().getGenericSuperclass();
        ParameterizedType pt = (ParameterizedType) t;
        this.clase = (Class) pt.getActualTypeArguments()[0];
        this.em=em;
    }
```

```
   public boolean crear (E objeto){
        try {
            if(this.validar(objeto)){
                em.getTransaction().begin();
                em.persist(objeto);
                em.getTransaction().commit();
                lista.add(objeto);
                return true;
            }} catch (Exception ex) {
            Logger.getLogger(AbstractControlador.class.getName()).log(Level.SEVERE,
null, ex);
        }


       return false;
}

   public boolean eliminar (E objeto){
        em.getTransaction().begin();
        em.remove(em.merge(objeto));
        em.getTransaction().commit();
        lista.remove(objeto);
        return true;

   }
     public boolean actualizar (E objeto){
        try {
            if(this.validar(objeto)){
                em.getTransaction().begin();
                objeto=em.merge(objeto);
                em.getTransaction().commit();
                this.lista=buscarTodo();
                return true;
            }
        } catch (Exception ex) {
            Logger.getLogger(AbstractControlador.class.getName()).log(Level.SEVERE,
null, ex);
        }


       return false;
}

     public E buscar  (Object id){
        return (E) em.find(clase, id);

     }

     public List<E> buscarTodo (){

        return em.createQuery("Select t from "+ clase.getSimpleName()+ "
t").getResultList();
     }

     public abstract boolean  validar(E objeto) throws Exception;

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

    public void setLista(List<E> lista) {
```

```
        this.lista = lista;
    }

    public Class<E> getClase() {
        return clase;
    }

    public void setClase(Class<E> clase) {
        this.clase = clase;
    }

    public EntityManager getEm() {
        return em;
    }

    public void setEm(EntityManager em) {
        this.em = em;
    }


}
```

Como clase genérica tenemos la clase de tipo JugadorRuleta

En este caso no existe atributos el cual validar así que retorna true.

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

import ec.edu.ups.modelo.JugadorRuleta;

/**
 *
 * @author user
 */
public class controladorJugador extends AbstractControlador<JugadorRuleta> {

    @Override
    public boolean validar(JugadorRuleta objeto) throws Exception {
        return true;
    }

}
```

Ahora para la creación de los hilos se realizo de la siguiente manera:

```
package ec.edu.ups.Hilos;


import ec.edu.ups.controlador.controladorJugador;
import ec.edu.ups.modelo.JugadorRuleta;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JTextArea;
import java.util.concurrent.ThreadLocalRandom;
import javax.swing.JLabel;
/**
 *
 * @author user
```

```
 */
public class Ruleta implements Runnable{
    private ArrayList<JugadorRuleta>jugadores;
    private String juego;
    private  JTextArea descripcion;
    private  int segundos;
    private  controladorJugador control;
    private  boolean iterar=true;

    private JLabel saldoBanca;

    private JLabel numeroB;

    public Ruleta(List<JugadorRuleta> jugadores, int segundos,JTextArea
descripcion,JLabel saldoBanca,JLabel numeroB,controladorJugador control,String
juego) {
        this.jugadores=(ArrayList<JugadorRuleta>) jugadores;
        this.segundos=segundos;
        this.descripcion= descripcion;
        this.control=control;
        this.juego=juego;
        this.saldoBanca=saldoBanca;
        this.numeroB=numeroB;
    }

     int numeroBanca=0;

    @Override
    public void run() {

        while (iterar){

            numeroBanca=RandomNumber();
            restarSaldoJugador();
            tiempoEsperar (segundos);
            if(numeroBanca==0){

            }
            switch (juego) {
                case "concreto":
                    jugadores.stream().map(jugador -> {
                        if(numeroBanca==jugador.getNumero()){
                            JugadorRuleta j = jugador;
                            j.setSaldo(jugador.getSaldo()+(360));
                            j.setnWins(jugador.getnWins()+1);
                            descripcion.setText(jugador.getNombre()+ "Gana 360
euros");

saldoBanca.setText(""+(Float.parseFloat(saldoBanca.getText()+"")-360));
                            control.actualizar(j);
                            tiempoEsperar (segundos);
                        }
                    return jugador;
                }).filter(jugador -> (numeroBanca!=jugador.getNumero())).map(jugador
-> {
                    JugadorRuleta j = jugador;
                    j.setnLost(jugador.getnLost()+1);
                    descripcion.setText(jugador.getNombre()+ "Pierde");
                    control.actualizar(j);
                    return jugador;
                }).forEachOrdered((JugadorRuleta _item) -> {
```

```
                    tiempoEsperar (segundos);

                });
                    iterar=false;
                    break;

                case "parImpar":
                    jugadores.forEach(jugador -> {
                        if(parImpar(numeroBanca)==jugador.getIsPar()){
                            JugadorRuleta j = jugador;
                            j.setSaldo(jugador.getSaldo()+20);
                            j.setnWins(jugador.getnWins()+1);
                            descripcion.setText(jugador.getNombre()+ "Gana 20
euros");

saldoBanca.setText(""+(Float.parseFloat(saldoBanca.getText()+"")-20));
                             control.actualizar(j);
                            tiempoEsperar (segundos);
                        }else{
                            JugadorRuleta j = jugador;
                            j.setnLost(jugador.getnLost()+1);
                            descripcion.setText(jugador.getNombre()+ "Pierde");
                            tiempoEsperar (segundos);
                            ;
                        }
                });
                 iterar=false;
                    break;

                case "martingala":
                    jugadores.stream().map(jugador -> {
                        if(numeroBanca==jugador.getNumero()){
                            JugadorRuleta j = jugador;
                            j.setSaldo(jugador.getSaldo()+(360));
                            j.setnWins(jugador.getnWins()+1);
                            descripcion.setText(jugador.getNombre()+ "Gana 360
euros");

saldoBanca.setText(""+(Float.parseFloat(saldoBanca.getText()+"")-360));
                            control.actualizar(j);
                        }
                    return jugador;
                }).filter(jugador ->
(numeroBanca!=jugador.getNumero())).forEachOrdered(jugador -> {
                    JugadorRuleta j = jugador;
                    j.setnLost(jugador.getnLost()+1);
                    j.setIsDuplicar(true);
                    descripcion.setText(jugador.getNombre()+ "Pierde y se duplica
apuesta");
                    control.actualizar(j);
                    tiempoEsperar (segundos);

                });
                    iterar=false;
break;

                default:
                    break;
            }
```

```
        }
    }
    public void reanudar(){
    iterar=true;
    }
    public boolean isIterar() {
        return iterar;
    }

    public void setIterar(boolean iterar) {
        this.iterar = iterar;
    }

    private void tiempoEsperar(int segundos){

        segundos= segundos * 1000;
             try {
                        descripcion.setText("Esperando..... "+segundos);
                        Thread.sleep(segundos);

                } catch (InterruptedException e) {

                }
        }



    private void restarSaldoJugador(){
     for (JugadorRuleta jugador : jugadores) {
         if (jugador.isIsDuplicar()){
              JugadorRuleta j = jugador;
                    j.setSaldo(jugador.getSaldo()-(jugador.getCantidadApuesta())*2);
                    j.setCantidadApuesta((jugador.getCantidadApuesta())*2);
                    j.setnApuestas(jugador.getnApuestas()+1);
                    tiempoEsperar(segundos);
                    //control.actualizar(j);
         }else {


                    descripcion.setText(jugador.getNombre()+" apuesta 10 euros al
numero "+jugador.getNumero());
                    JugadorRuleta j = jugador;
                    j.setSaldo(jugador.getSaldo()-10);
                    j.setCantidadApuesta(jugador.getCantidadApuesta()+10);
                    j.setnApuestas(jugador.getnApuestas()+1);
                    tiempoEsperar(segundos);
                   // control.actualizar(j);
         }

            }
    }


    private int RandomNumber (){
        int numero = ThreadLocalRandom.current().nextInt(0, 36 + 1);
        numeroB.setText(""+numero);
        return numero;

    }

    private int parImpar(int numero){
```

```
        if(numero%2==0){
        return 2;
        }else {
        return 1;
        }
    }

    public void stop(){
    iterar=false;
    }
    }
```

Primero creamos la clase Ruleta y implementamos la clase Runnable, luego definimos los atributos correspondientes, se crea una lista temporal la cual va a estar almacenados los jugadores que participarán en el juego, en este caso los jugadores que ya ingresaron en el juego podrán jugarlo de nuevo.
La Lista temporal (ArrayList<>): se genera temporalmente cuando se agrega de jugador en jugador al juego mediante su Id.

En lanzador de Hilos es el siguiente y se lo declara en la ventana JuegoRuleta

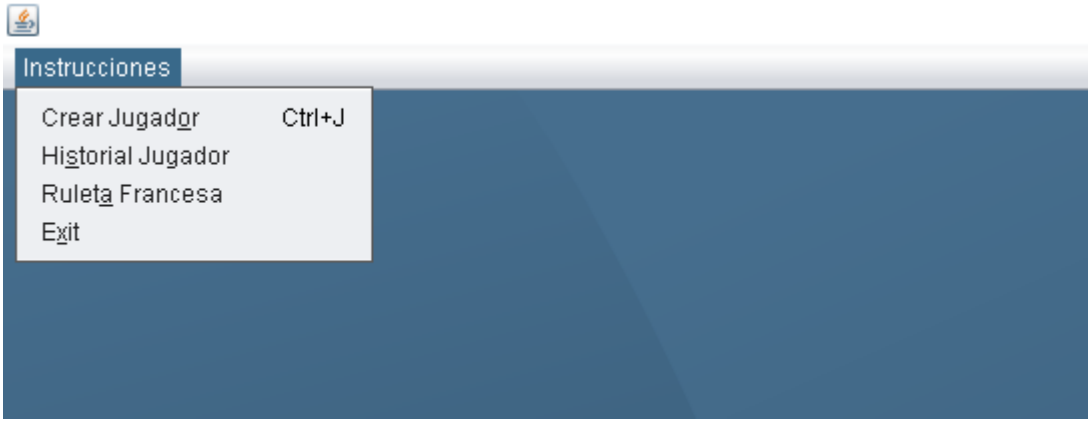
```
private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {

    if(jugadores!=null){
      Ruleta[] vHilos = new Ruleta[numList()];
      for(int i=0;i<numList();i++){

          vHilos[i]=new Ruleta(jugadores,segundos,txtArea,lblValorS,lblValorBanca,controladorJugador,cbxItemJuego.getSelectedItem().toString().trim());
          Thread hilo = new Thread (vHilos[i]);
          hilo.start();
         /* if(isStop){
          hilo.stop();
          }
          if(isPause.equals("verdad")){
          vHilos[i].stop();
          }else if(isPause.equals("resume")){
          vHilos[i].reanudar();
          }*/

      }
   }
}
```

Simulación:

Verificamos la conexión a la base de datos:

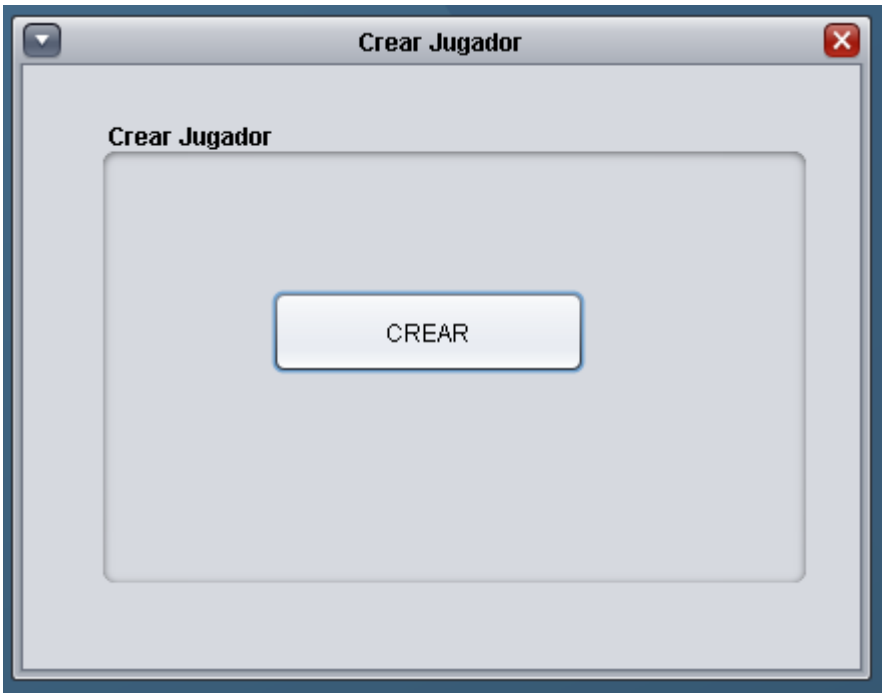
```
[EL Info]: transaction: 2021-02-07 17:43:29.694--ServerSession(24140544)--property eclipselink.jdbc.user is deprecated, property javax.persistence.jdbc.user should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.698--ServerSession(24140544)--property eclipselink.jdbc.driver is deprecated, property javax.persistence.jdbc.driver should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.698--ServerSession(24140544)--property eclipselink.jdbc.url is deprecated, property javax.persistence.jdbc.url should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.698--ServerSession(24140544)--property eclipselink.jdbc.password is deprecated, property javax.persistence.jdbc.password should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.813--ServerSession(24140544)--property eclipselink.jdbc.user is deprecated, property javax.persistence.jdbc.user should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.813--ServerSession(24140544)--property eclipselink.jdbc.driver is deprecated, property javax.persistence.jdbc.driver should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.813--ServerSession(24140544)--property eclipselink.jdbc.url is deprecated, property javax.persistence.jdbc.url should be used instead.
[EL Info]: transaction: 2021-02-07 17:43:29.813--ServerSession(24140544)--property eclipselink.jdbc.password is deprecated, property javax.persistence.jdbc.password should be used instead.
[EL Info]: 2021-02-07 17:43:29.861--ServerSession(24140544)--EclipseLink, version: Eclipse Persistence Services - 2.7.7.v20200504-69f2c2b80d
```

Ventana Principal

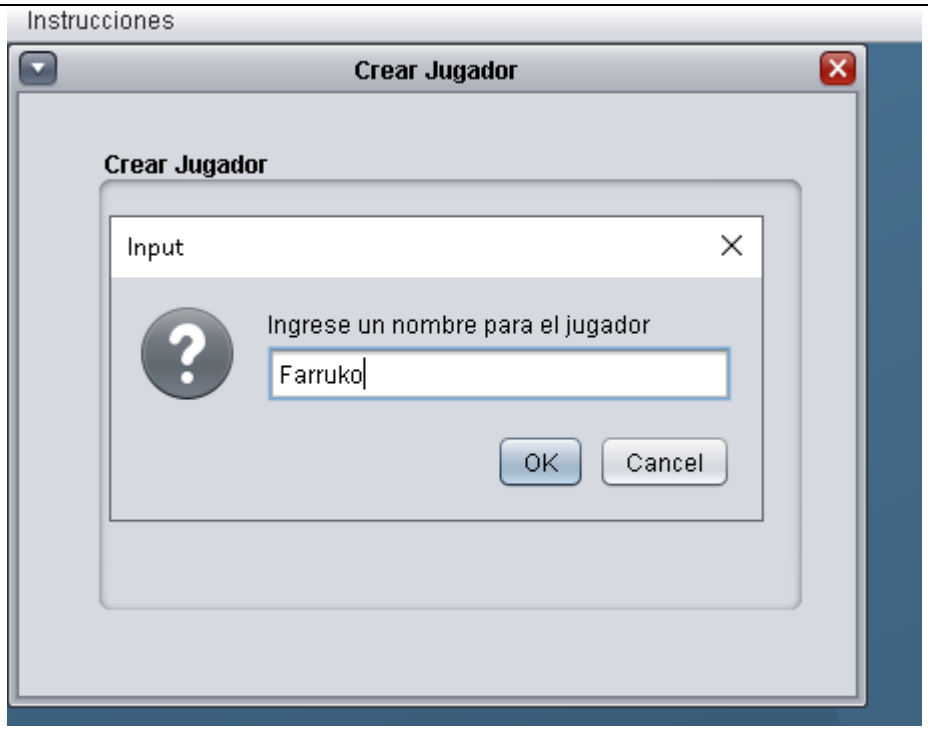


En la ventana principal tenemos las opciones del juego la cual nos da para crear un Jugador, el historial de los jugadores, el juego y para salir.

Vamos a crear un jugador:



En este caso solo cree un botón para utilizar ventanas JOptionPane para el ingreso de datos.



Ahora vamos a poner un juego en la ruleta en este caso la martingala:
Igualmente utilizamos solo botones para la entrada de datos con JOptionPane.


Juego Ruleta
RULETA FRANCESA
Agregar Jugador
Input
Ingrese numero a apostar 1-36
25
OK
Cancel
--Seleccionar Juego--


Juego Ruleta
RULETA FRANCESA
Agregar Jugador
Message
Jugador: Farruko asignado exitosamente
OK
--Seleccionar Juego--


Juego Ruleta
RULETA FRANCESA
Agregar Jugador
Asignar Tiempo
SALDO: 50000
Numero de la banca: -
--Seleccionar Juego--
Input
Ingrese tiempo de intervalo entre procesos
2
OK
Cancel

RESULTADO(S) OBTENIDO(S):

Se aplico todos los conocimientos obtenidos en el ciclo en este programa, entra programación genérica, Java 8, Reflexión, Hilos, MVC, JPA en Java.

CONCLUSIONES:

En conclusión, JPA fue de mucha utilidad para la persistencia de datos de manera muy sutil, ya que en las clases entity se pueden generar automáticamente los primary key y no hay necesidad de crear un método aparte en los controladores.

RECOMENDACIONES:

No existe alguna recomendación.

Nombre de estudiante: JUAN JOSE CORDOVA CALLE

Firma del estudiante: