



**Prueba 3**

29/01/2021

Objetivo:

- Consolidar los conocimientos adquiridos en clase sobre JPA.

Enunciado:

Realizar un sistema implementando todos los conceptos vistos en clases para gestionar la hipoteca de las casas con las siguientes características:

- Las personas compran casas y se convierten en propietarios.
- Para pagarlas es habitual que el propietario formalice un préstamo hipotecario con una entidad bancaria.
- El banco toma la casa en forma de aval en caso de impago de las mensualidades.
- En el caso de que el capital fiado supera el valor de tasación de la casa y el sueldo del propietario no es suficiente, el banco suele exigir la presencia de un avalista (garante).
- Para formalizar la hipoteca se necesitan los datos personales del propietario, además de su cédula, dirección de la casa, su dirección, nombres, apellidos y fecha de nacimiento y del garante de ser necesario.
- El capital de la hipoteca se ajusta teniendo en cuenta el valor de tasación de la casa y los datos de dirección.
- Toda hipoteca se formaliza detallando el capital, el interés (8,99 - 16,99%) y la duración (fecha de inicio y fecha de fin).
- A partir de estos datos se calcula el importe de cada mensualidad para el total del tiempo que pide el préstamo.
- No es necesario guardar los datos del banco, pero si un sistema de autenticación.
- Generar los datos con el sistema de amortización alemán [1].

**Ejemplo** simulador de crédito para guía: <https://www.pichincha.com/portal/simuladores/simulador-de-creditos>

Se calificará de la siguiente forma:

- JPA: 40%
- Excepciones: 10%
- MVC: 20%
- Diagrama de clases: 10%
- Usabilidad - Vista: 15%
- Programación genérica, Java 8, reflexión: 15%


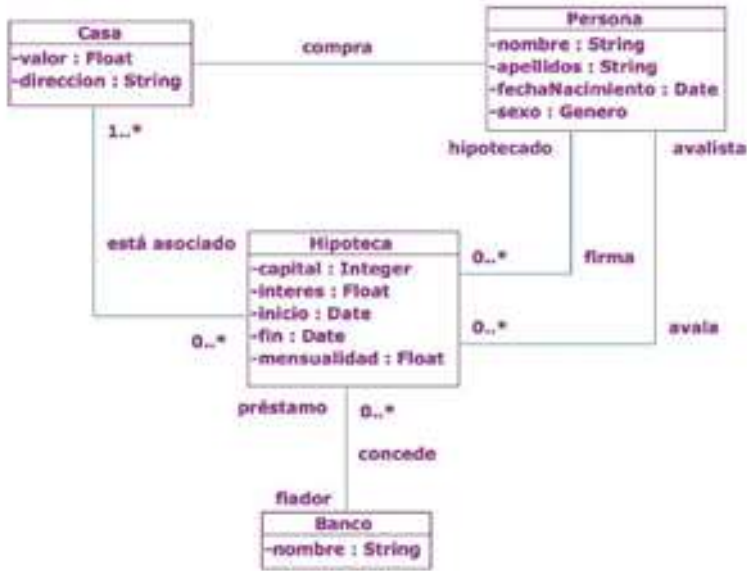
**Bibliografía**

- [1] <https://www.produbanco.com.ec/banca-minorista/cr%C3%A9ditos/hipotecario/simulador-de-cr%C3%A9dito-hipotecario/>

**Entrega:** Subir al Git el documento en formato PDF de los resultados y código hasta las **23:55** del



domingo 31 de enero del 2021.

 <b>FORMATO DE INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES</b>		
<b>CARRERA:</b> Computación		<b>ASIGNATURA:</b> Programación Aplicada
<b>NRO. PRUEBA</b>	3	<b>TÍTULO PRÁCTICA:</b> Prueba JPA
<b>OBJETIVO ALCANZADO:</b> <ul style="list-style-type: none"><li>Consolidar los conocimientos adquiridos en clase sobre JPA, MVC, Excepciones, Programación genérica</li></ul>		
<b>ACTIVIDADES DESARROLLADAS</b>		
<p>1. Para la prueba primero se planteo el diagrama de clases, y se siguió el mismo formato</p>  <p style="text-align: center;"><b>La arquitectura utilizada MVC</b></p> <p><b>Dentro del modelo tenemos las siguientes clases:</b></p> <p>La clase Persona: Esta cuenta como padre para la clase propietario, se encuentran los datos personales</p> <pre>package ec.edu.ups.modelo;  import java.io.Serializable; import java.util.Date; import javax.persistence.Column; import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id; import javax.persistence.Temporal; import javax.persistence.TemporalType;</pre>		



Prueba 3

29/01/2021

```
/**
 *
 * @author user
 */
@Entity
public class Persona implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "cedula")
    private String cedula;
    @Column(name = "nombre")
    private String nombre;
    @Column(name = "apellido")
    private String apellido;
    @Column(name = "fechaNacimiento")
    @Temporal(TemporalType.DATE)
    private Date fechaNacimiento;
    @Column(name = "direccion")
    private String direccion;

    public Long getId() {
        return id;
    }

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

    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 Date getFechaNacimiento() {
        return fechaNacimiento;
    }
}
```



Prueba 3

29/01/2021

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

public String getDireccion() {
    return direccion;
}

public void setDireccion(String direccion) {
    this.direccion = direccion;
}

@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 Persona)) {
        return false;
    }
    Persona other = (Persona) object;
    if ((this.id == null && other.id != null) || (this.id != null
&& !this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("Persona{id=").append(id);
    sb.append(", cedula=").append(cedula);
    sb.append(", nombre=").append(nombre);
    sb.append(", apellido=").append(apellido);
    sb.append(", fechaNacimiento=").append(fechaNacimiento);
    sb.append(", direccion=").append(direccion);
    sb.append('}');
    return sb.toString();
}

}
```

La clase Propietario: Esta clase hereda de la clase padre los atributos de persona, y aparte tiene de atributos dirección de la casa a hipotecar, y el garante asignado



Prueba 3

29/01/2021

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

import java.io.Serializable;
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.OneToMany;
import javax.persistence.OneToOne;

/**
 *
 * @author user
 */
@Entity
public class Propietario extends Persona implements Serializable {

    @Column(name="direccionCasa")
    @OneToOne
    private Casa casa;

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

    @OneToMany(mappedBy="propietario",targetEntity=Hipoteca.class)
    private Set<Hipoteca> hipotecas;

    public Casa getCasa() {
        return casa;
    }

    public void setCasa(Casa casa) {
        this.casa = casa;
    }

    public Set<Hipoteca> getHipotecas() {
        return hipotecas;
    }

    public void setHipotecas(Set<Hipoteca> hipotecas) {
        this.hipotecas = hipotecas;
    }

    public String getGarante() {
        return garante;
    }

    public void setGarante(String garante) {
        this.garante = garante;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Propietario{casa=").append(casa);
        sb.append(", garante=").append(garante);
        sb.append('}');
        return sb.toString();
    }
}
```



Prueba 3

29/01/2021

}

}

La clase Usuario: Esta clase contiene los atributos para realizar el sistema de autenticación para el sistema

```
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;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;

/**
 *
 * @author user
 */
@Entity
public class Usuario implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "username")
    private String username;
    @Column(name = "password")
    private String password;
    @OneToOne
    @JoinColumn (name="idPersona")
    private Persona persona;
    public Long getId() {
        return id;
    }

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

    public static long getSerialVersionUID() {
        return serialVersionUID;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
```



Prueba 3

29/01/2021

```
        return password;
    }

    public Persona getPersona() {
        return persona;
    }

    @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 Usuario)) {
            return false;
        }
        Usuario other = (Usuario) object;
        if ((this.id == null && other.id != null) || (this.id != null
&& !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Usuario{id=").append(id);
        sb.append(", username=").append(username);
        sb.append(", password=").append(password);
        sb.append(", persona=").append(persona);
        sb.append('}');
        return sb.toString();
    }
}
```

La clase hipoteca: En esta clase contiene los atributos para generar el importe de la mensualidad del préstamo

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ec.edu.ups.modelo;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
```



Prueba 3

29/01/2021

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 *
 * @author user
 */
@Entity
public class Hipoteca implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column (name="capital")
    private double capital;
    @Column (name="interes")
    private double interes;

    @Column(name = "fechaInicio")
    @Temporal(TemporalType.DATE)
    private Date fechaInicio;

    @Column(name = "fechaFin")
    @Temporal(TemporalType.DATE)
    private Date fechaFin;

    @ManyToOne
    @JoinColumn(name="fk_propietario")
    private Propietario propietario;
    public Long getId() {
        return id;
    }

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

    public double getCapital() {
        return capital;
    }

    public void setCapital(double capital) {
        this.capital = capital;
    }

    public double getInteres() {
        return interes;
    }

    public void setInteres(double interes) {
        this.interes = interes;
    }
}
```





Prueba 3

29/01/2021

```
public Date getFechaInicio() {
    return fechaInicio;
}

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

public Date getFechaFin() {
    return fechaFin;
}

public void setFechaFin(Date fechaFin) {
    this.fechaFin = fechaFin;
}

public Propietario getPropietario() {
    return propietario;
}

public void setPropietario(Propietario propietario) {
    this.propietario = propietario;
}

@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 Hipoteca)) {
        return false;
    }
    Hipoteca other = (Hipoteca) object;
    return !((this.id == null && other.id != null) || (this.id != null
&& !this.id.equals(other.id)));
}

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

La clase casa: Contiene los atributos del valor de la casa (Capital) y la dirección de la misma

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

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
```



Prueba 3

29/01/2021

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

/**
 *
 * @author user
 */
@Entity
public class Casa implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column (name="valor")
    private double valor;

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

    @OneToOne(mappedBy = "casa")
    private Propietario propietario;

    public Long getId() {
        return id;
    }

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

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

    public Propietario getPropietario() {
        return propietario;
    }

    public void setPropietario(Propietario propietario) {
        this.propietario = propietario;
    }

    public double getValor() {
        return valor;
    }

    public void setValor(double valor) {
        this.valor = valor;
    }

    public String getDireccion() {
```



### Prueba 3

29/01/2021

```
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    @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 Casa)) {
            return false;
        }
        Casa other = (Casa) 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.Casa[ id=" + id + " ]";
    }
}
```

**Dentro del controlador con la ayuda de programación genérica y reflexión en java tenemos:**

AbstractControlador: Este cuenta con los métodos crud el cual será implementados en los demás controladores a continuación:

```
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
 */
public abstract class AbstractControlador <E> {

    private List<E> lista;
```



Prueba 3

29/01/2021

```
private Class<E> clase;
private EntityManager em;

/**
 *
 */
public AbstractControlador() {

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



Prueba 3

29/01/2021

```
        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;
}
}
```

Controlador Persona: Este controlador se usa para implementar los métodos crud de la clase padre (AbstractControlador) la cual crea, elimina, actualiza, busca y lista objetos de tipo Persona.

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

import ec.edu.ups.Excepciones.ExcepcionCedula;
import ec.edu.ups.modelo.Persona;

/**
```



Prueba 3

29/01/2021

```
*
* @author user
*/
public class controladorPersona extends AbstractControlador<Persona> {

    @Override
    public boolean validar(Persona objeto) throws ExcepcionCedula {
        int suma = 0;
        String x = objeto.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 {
                throw new ExcepcionCedula();
            }
        }
    }
}

}
```

Controlador Propietario: Este controlador se usa para implementar los métodos crud de la clase padre (AbstractControlador) la cual crea, elimina, actualiza, busca y lista objetos de tipo Propietario.

```
public class controladorPropietario extends AbstractControlador<Propietario>{

    @Override
    public boolean validar(Propietario objeto) throws Exception {
```



### Prueba 3

29/01/2021

```
        return true;
    }
}
```

Controlador Usuario: Este controlador se usa para implementar los métodos crud de la clase padre (AbstractControlador) la cual crea, elimina, actualiza, busca y lista objetos de tipo Usuario

```
public class controladorUsuario extends AbstractControlador<Usuario> {

    private Usuario usuario;

    @Override
    public boolean validar(Usuario objeto) {

        return true;
    }

    public boolean iniciarSesion(String correo, String pass) {

        for (Usuario usu : super.getLista()) {
            Usuario u = (Usuario) usu;
            if (u.getUsername().equals(correo) && u.getPassword().equals(pass)) {
                this.usuario = u;
                return true;
            }
        }
        return false;
    }

    public Usuario getUsuario() {
        return usuario;
    }

}
```

Controlador Hipoteca: Este controlador se usa para implementar los métodos crud de la clase padre (AbstractControlador) la cual crea, elimina, actualiza, busca y lista objetos de tipo Hipoteca

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

import ec.edu.ups.modelo.Hipoteca;

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



### Prueba 3

29/01/2021

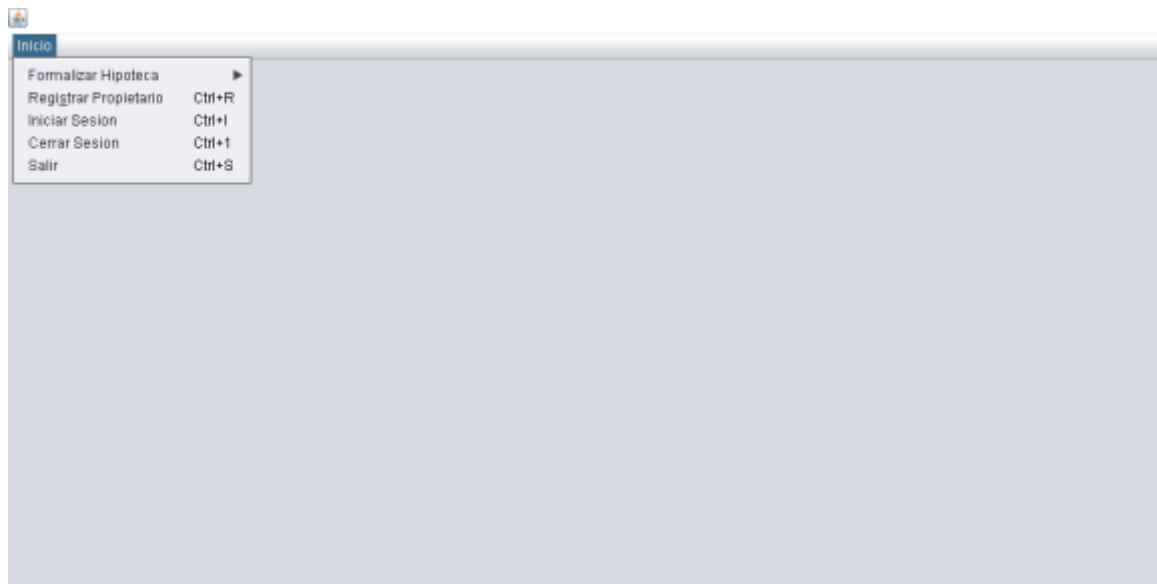
```
public class controladorHipoteca extends AbstractControlador<Hipoteca>{  
  
    @Override  
    public boolean validar(Hipoteca objeto) throws Exception {  
        return true;  
    }  
  
}
```

Controlador Casa: Este controlador se usa para implementar los métodos crud de la clase padre (AbstractControlador) la cual crea, elimina, actualiza, busca y lista objetos de tipo Casa

```
package ec.edu.ups.controlador;  
  
import ec.edu.ups.modelo.Casa;  
  
/**  
 *  
 * @author user  
 */  
public class controladorCasa extends AbstractControlador<Casa>{  
  
    @Override  
    public boolean validar(Casa objeto) throws Exception {  
        return true;  
    }  
  
}
```

**Dentro de la vista tenemos las siguientes ventanas:**

**Ventana Principal:**







Prueba 3

29/01/2021

Ventana para registrar un propietario:

Registro propietario

Propietario

Nombre:

Cedula:

Fecha Nacimiento: enero 2021  
lun. mar. mié. jue. vie. sáb. dom.  
1 2 3  
4 5 6 7 8 9 10  
11 12 13 14 15 16 17  
18 19 20 21 22 23 24  
25 26 27 28 29 30 31

Apellido:

Direccion:

Direccion Casa:

Nombre Garante:

GUARDAR ELIMINAR ACTUALIZAR

Ventana Iniciar Sesión

INICIAR SESION

Correo:

Contraseña:

Iniciar Sesion

Ventana para formalizar la hipoteca



Prueba 3

29/01/2021

Generar Hipoteca

Formalizar Hipoteca

Capital

Interés

Mensualidad

Fecha Inicio

enero

2021

lun.

mar.

mié.

jue.

vie.

sáb.

dom.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

Fecha Fin

enero

2021

lun.

mar.

mié.

jue.

vie.

sáb.

dom.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

GUARDAR

ELIMINAR

ACTUALIZAR