Proxy

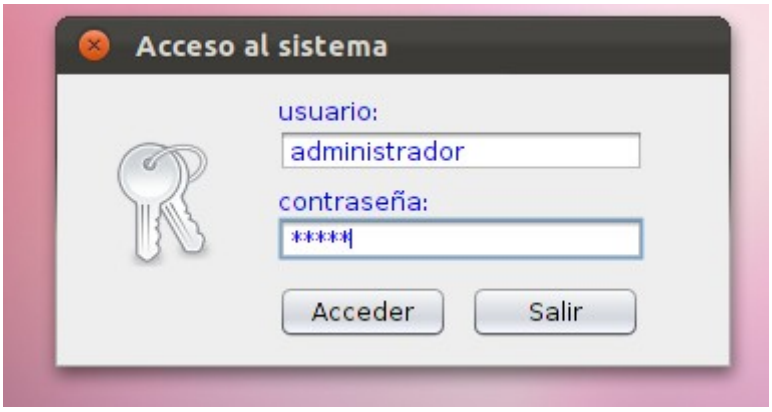
Una aplicación de nóminas como ejemplo del patrón Proxy

Proxy de Protección. Un proxy para controlar el acceso a un servicio protegido

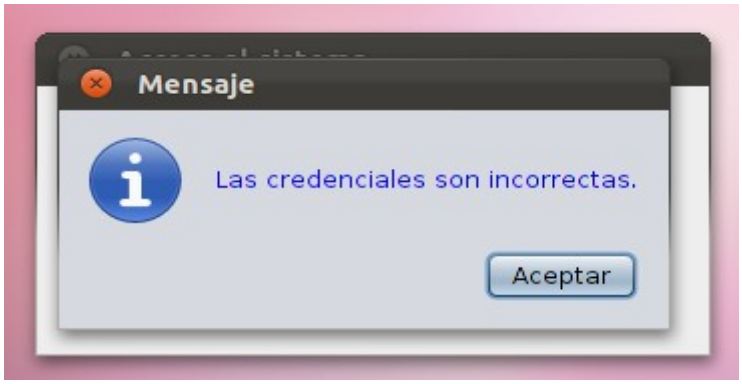
Supongamos una aplicación de gestión empresarial que dispone de un módulo de nóminas. Por seguridad, este módulo sólo debe ser accedido por los empleados pertenecientes al departamento de recursos humanos. Una manera de cumplir con este requisito es mediante un proxy de protección que se encargue de que los usuarios se acrediten correctamente para acceder al subsistema de nóminas.

A continuación se muestran algunas imágenes de la aplicación de ejemplo que vamos a analizar.

Al iniciar el programa el proxy de protección detecta que no estamos acreditamos, por lo que nos obliga a validarnos. Para simplificar la aplicación, las credenciales están almacenas en el propio código: admin/12345

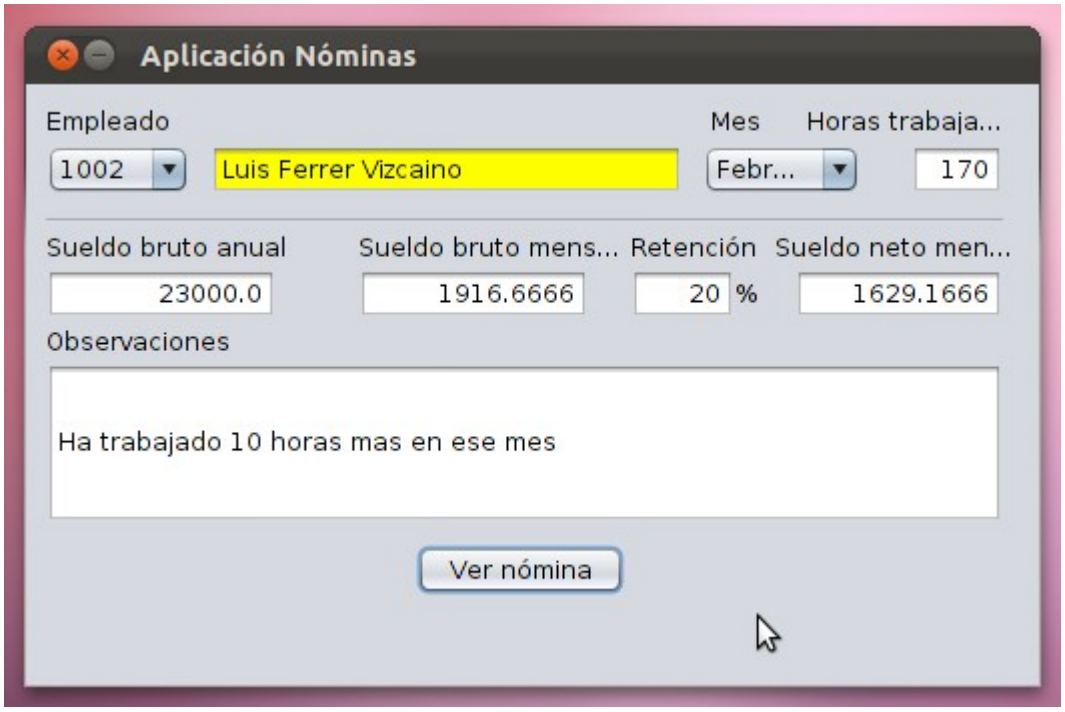


Si la validación no es correcta aparecerá un mensaje de aviso:



Tendremos tres oportunidades para acreditarnos antes de que el programa finalice.

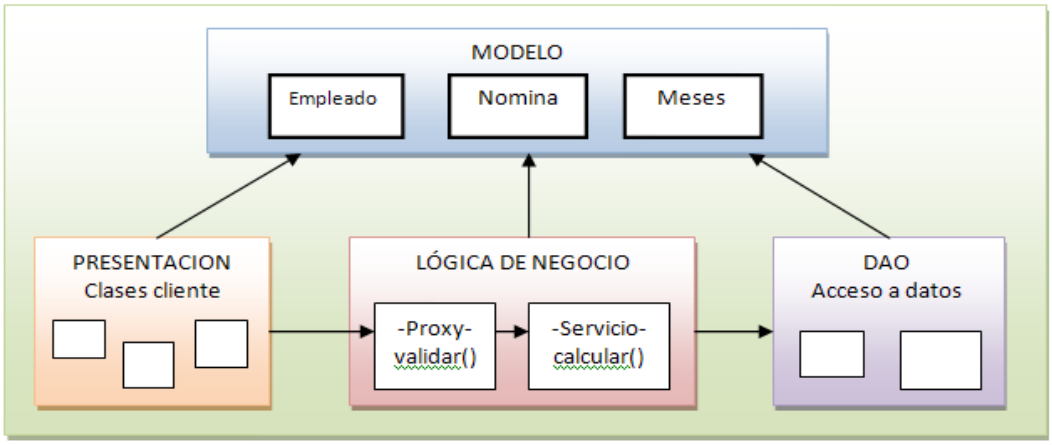
Una vez validados, accederemos al módulo de nóminas, consistente en una sencilla pantalla en la que seleccionamos un código de empleado, un mes e introducimos el número de horas trabajadas ese mes. Al pulsar el botón 'Ver nómina' obtendremos el sueldo neto mensual que le corresponde al empleado:



Por supuesto, al seleccionar otro empleado/mes y pulsar el botón 'Ver nómina', se obtienen nuevos datos y se muestra en la consola un listado con las nóminas existentes:

```
NOMINAS EXISTENTES ACTUALMENTE
------------------------------------------------
Empleado [codigo=1000, nombre=Xavier Garcia Fernandez, sueldoBruto anual=25000.0]
Nomina [mes=Enero, horas=180, bruto=2083.3333, neto=1875.0]
Observaciones=Ha trabajado 20 horas mas en ese mes
-----------------------------------------------------------------
Empleado [codigo=1000, nombre=Xavier Garcia Fernandez, sueldoBruto anual=25000.0]
Nomina [mes=Febrero, horas=180, bruto=2083.3333, neto=1875.0]
Observaciones=Ha trabajado 20 horas mas en ese mes
-----------------------------------------------------------------
------------------------------------------------
NOMINAS EXISTENTES ACTUALMENTE
------------------------------------------------
Empleado [codigo=1001, nombre=Raquel Mateo Gonzalez, sueldoBruto anual=27000.0]
Nomina [mes=Enero, horas=155, bruto=2250.0, neto=1743.75]
Observaciones=Ha trabajado 5 horas menos en ese mes.
-----------------------------------------------------------------
Empleado [codigo=1000, nombre=Xavier Garcia Fernandez, sueldoBruto anual=25000.0]
Nomina [mes=Enero, horas=180, bruto=2083.3333, neto=1875.0]
Observaciones=Ha trabajado 20 horas mas en ese mes
-----------------------------------------------------------------
Empleado [codigo=1000, nombre=Xavier Garcia Fernandez, sueldoBruto anual=25000.0]
Nomina [mes=Febrero, horas=180, bruto=2083.3333, neto=1875.0]
Observaciones=Ha trabajado 20 horas mas en ese mes
-----------------------------------------------------------------
```

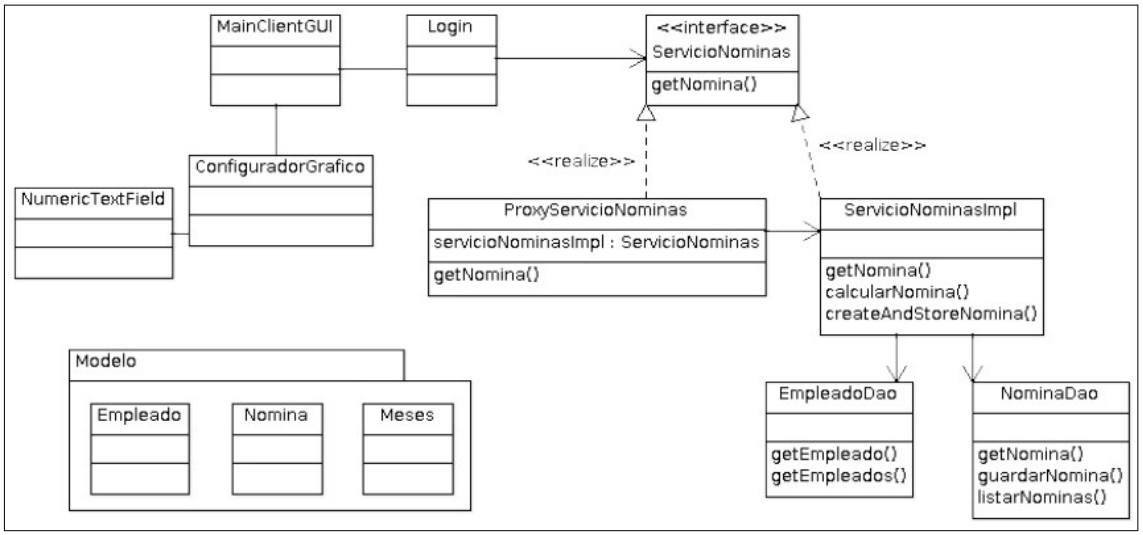
La figura siguiente muestra la arquitectura en capas que sigue la aplicación:



Decisiones de diseño:

- Las clases Empleado, Nomina y Meses forman el modelo de la aplicación.
- La capa de presentación la componen un conjunto de clases Swing.
- La capa de negocio está formada por clases POJO (Plain Old Java Objects). En esta capa es donde se implementa el patrón Proxy.
- La capa DAO, por simplicidad, utiliza como mecanismo de persistencia un HashMap, por lo que los nuevos datos introducidos se pierden entre diferentes sesiones de trabajo. No obstante, la aplicación tiene datos de ejemplo para que todo funcione correctamente.

El siguiente diagrama de clases muestra un diseño más detallado de la aplicación:



Empleado, Nomina y Meses son clases de entidad que son accedidas por el resto de módulos, aunque con diferente propósito. Por ejemplo, la capa de negocio accederá a ellas con la finalidad de calcular una nómina, mientras que la capa de presentación lo hará para obtener el importe de la misma y mostrarlo por pantalla. En cambio, el módulo de acceso a datos, lo hará para recuperar o almacenar del sistema de persistencia alguna de estas entidades.

Veamos el código.

Modelo de datos (modelo subyacente)

Comenzamos por las clases que conforman el modelo.

Meses.java (es una enum)

```
package estructurales.proxy.protection_proxy.modelo;


public enum Meses {
	Enero, Febrero, Marzo, Abril,
	Mayo, Junio, Julio, Agosto,
	Septiembre, Octubre, Noviembre, Diciembre;
}
```

A continuación la clase que modela empleados. Notad que se implementan los métodos hashCode() y equals(). Esto es altamente aconsejable cuando una clase se va a utilizar con contenedores de objetos (colecciones), así como para poder comparar objetos mediante la igualdad (que no la identidad).

Empleado.java

```
package estructurales.proxy.protection_proxy.modelo;


public class Empleado implements Comparable<Empleado> {
	private int codigo;
```

```
	private String nombre;
	private float sueldoBruto;

	public Empleado(int codigo, String nombre, float sueldoBruto) {
		this.codigo = codigo;
		this.nombre = nombre;
		this.sueldoBruto = sueldoBruto;
	}

	public int getCodigo() { return codigo;  }
	public String getNombre() { return nombre; }
	public float getSueldoBruto() {    return sueldoBruto;    }

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + codigo;
		result = prime * result + ((nombre == null) ? 0 :
nombre.hashCode());
		result = prime * result +
Float.floatToIntBits(sueldoBruto);
		return result;
	}

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

```
                return false;
        if (getClass() != obj.getClass())
                return false;
        Empleado other = (Empleado) obj;
        if (codigo != other.codigo)
                return false;
        if (nombre == null) {
                if (other.nombre != null)
                        return false;
        } else if (!nombre.equals(other.nombre))
                return false;
        if (Float.floatToIntBits(sueldoBruto) != Float
                        .floatToIntBits(other.sueldoBruto))
                return false;
        return true;
    }

    @Override
    public String toString() {
        return "Empleado [codigo=" + codigo + ", nombre=" + nombre
                        + ", sueldoBruto anual=" + sueldoBruto + "]";
    }

    @Override
    public int compareTo(Empleado e) {
        return this.codigo < e.codigo ? -1: 1;
    }

}
```

La última clase del modelo es la que modela nominas:

Nomina.java

```
package estructurales.proxy.protection_proxy.modelo;


import estructurales.proxy.protection_proxy.dao.EmpleadoDao;


public class Nomina {
	private int codEmp, horasTrabajadas;
	private Meses mes;
	private float bruto, neto;
	private String observaciones;

	public Nomina(int codEmp, int horasTrabajadas, Meses mes,
			float bruto, float neto, String observaciones)
	{
		this.codEmp = codEmp;
		this.horasTrabajadas = horasTrabajadas;
		this.mes = mes;
		this.bruto = bruto;
		this.neto = neto;
		this.observaciones = observaciones;
	}

	@Override
	public String toString() {
		Empleado empleado = EmpleadoDao.getEmpleado(codEmp);
		String parteEmpleado = empleado.toString();
```

```
            String parteNomina = "Nomina [mes=" + mes + ", horas=" +
            horasTrabajadas   + ", bruto=" + bruto + ", neto=" + neto +
"]";

            String parteObs = observaciones != "" ?
                        "Observaciones=" + observaciones : "";

            return parteEmpleado + "\n" + parteNomina + "\n" +
parteObs + "\n" +
                  "-----------------------------------------------
-------------";
      }

      public int getHorasTrabajadas() { return horasTrabajadas; }
      public int getCodEmp() { return codEmp;  }
      public Meses getMes() { return mes;}
      public float getBruto() { return bruto;  }
      public float getNeto() { return neto; }
      public String getObservaciones() { return observaciones; }

}
```

Capa de persistencia

Veamos ahora las clases de acceso a datos. Estas clases definen un HashMap que hace las veces de base de datos, así como algunas operaciones para gestionar los datos del HashMap. Esto se hace por simplicidad, para no complicar el ejemplo más de lo necesario con código que acceda realmente a una base de datos.

Estas clases son estáticas, ya que no tiene sentido crear diferentes instancias de ellas, de la misma forma que no tiene sentido tener una base de datos para cada usuario de nuestra aplicación. Para evitar crear instancias los constructores se definen privados.

Comenzamos viendo la clase que se encarga de gestionar la persistencia de los empleados. Notad que se define un inicializados estático que crea algunos empleados y los añade a un HashMap. Con esto nos aseguramos que siempre que se ejecuta el programa tenemos unos cuantos empleados con los que hacer pruebas.

También hay que observar que no se define ningún método para añadir nuevos empleados. Esto se debe a que la aplicación cliente, la GUI, por simplicidad no contempla esta operación.

EmpleadoDao.java

```
package estructurales.proxy.protection_proxy.dao;


import java.util.HashMap;
import java.util.Map;


import estructurales.proxy.protection_proxy.modelo.Empleado;


public class EmpleadoDao {
	private static Map<Integer, Empleado> empleados;
	static {
		empleados = new HashMap<Integer, Empleado>();
		empleados.put(1000, new Empleado(1000, "Xavier Garcia
Fernandez", 25000f));
		empleados.put(1001, new Empleado(1001, "Raquel Mateo
Gonzalez", 27000f));
		empleados.put(1002, new Empleado(1002, "Luis Ferrer
Vizcaino", 23000f));
		empleados.put(1003, new Empleado(1003, "Vanesa Garriga
Teruel", 25000f));
	}
```

```
	/*
	 * Constructor privado para evitar que se creen instancias de
esta clase
	 */
	private EmpleadoDao() {


	}


	public static Empleado getEmpleado(int codEmp) {
		return empleados.get(codEmp);
	}


	public static Map<Integer, Empleado> getEmpleados() {
		return empleados;
	}


}
```

Ahora veamos la clase que se encarga de gestionar la persistencia de las nóminas. Esta clase es un poco más compleja que la anterior, aunque sigue la misma tónica. A diferencia de la clase EmpleadoDao, esta clase permite guardar nóminas, así como mostrar por consola un listado con todas las nóminas generadas.

Notad que en el inicializador estático se crea un mapa donde:

- La clave es el código de empleado de cada uno de los empleados definidos en EmpleadoDao.
- El valor consiste en un HashMap de doce entradas en el que la clave es uno de los doce meses y el valor es una nómina inicializada a null.

NominaDao.java

```
package estructurales.proxy.protection_proxy.dao;


import java.util.HashMap;
import java.util.Set;
import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.modelo.Nomina;


public class NominaDao {


	private static HashMap<Integer, HashMap<Meses, Nomina>> nominas;


	static {
		nominas = new HashMap<Integer, HashMap<Meses, Nomina>>();


		Set<Integer> codigosEmp =
EmpleadoDao.getEmpleados().keySet();
		for (int codEmp : codigosEmp) {
			nominas.put(codEmp, inicializarNominasEmpleado());
		}
	}
	/*
	 * Constructor privado para evitar que se creen instancias de
esta clase
	 */
	private NominaDao() {


	}
```

```
	public static Nomina getNomina(int codEmp, Meses mes) {
		return nominas.get(codEmp).get(mes);
	}

	public static void guardarNomina(int codEmp, Meses mes, Nomina
nomina) {
		nominas.get(codEmp).put(mes, nomina);
		listarNominas();
	}

	private static void listarNominas() {

System.out.println("-----------------------------------------------");
		System.out.println("NOMINAS EXISTENTES ACTUALMENTE");

System.out.println("-----------------------------------------------");
		Set<Integer> codEmps = nominas.keySet();

		for (Integer codEmp : codEmps) {
			for (Meses mes : Meses.values()) {
				Nomina nomina = nominas.get(codEmp).get(mes);
				if (nomina != null)
					System.out.println(nomina);
			}
		}
	}

	private static HashMap<Meses, Nomina>
inicializarNominasEmpleado() {
```

```
            HashMap<Meses, Nomina> nominasEmpleado =
                  new   HashMap<Meses, Nomina>();

            for (Meses mes : Meses.values()) {
                  nominasEmpleado.put(mes, null);
            }

            return nominasEmpleado;
      }

}
```

Capa de Negocio o de Servicios (Lógica de negocio)

Veamos ahora las clases e interfaces que conforman el core de la aplicación.

Comenzamos con la interfaz que define las operaciones que proporciona el servicio de nominas. Esta interfaz tiene el rol 'Subject' dentro del patrón proxy:

ServicioNominas.java

```
package estructurales.proxy.protection_proxy.servicios;

import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.modelo.Nomina;

/** Subject */
public interface ServicioNominas {

      public static int porcentRet = 20;
```

```
      public static int horasMensualesOficiales = 40*4;

    public Nomina getNomina(int codEmp, Meses mes,
                                    int horasTrabajadas);
}
```

A continuación, se muestra el código de la clase que implementa realmente el servicio de nóminas (implementa la interfaz anterior). Contiene toda la lógica de negocio necesaria para calcular la nómina de un empleado. Esta clase desempeña el rol 'RealSubject' en el patrón proxy, por lo que el código cliente no tiene acceso a ella, es más, incluso puede desconocer su existencia.

ServicioNominasImpl.java

```
package estructurales.proxy.protection_proxy.servicios;

import estructurales.proxy.protection_proxy.dao.EmpleadoDao;
import estructurales.proxy.protection_proxy.dao.NominaDao;
import estructurales.proxy.protection_proxy.modelo.Empleado;
import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.modelo.Nomina;

/** RealSubject */
class ServicioNominasImpl implements ServicioNominas {

      /*
       * La construcción de este objeto puede ser computacionalmente
       * costosa. Cuando es así, el hecho de que el proxy sostenga
       * una referencia hacia él evita este coste.
       */
    public ServicioNominasImpl() {
```

```
        try {
            System.out.println("Construyendo un objeto
ServicioNominasImpl...");
            // Para ver un beneficio añadido del patrón proxy podemos
            // simular que la creación de este objeto es costosa
            //Thread.sleep(5000);
            Thread.sleep(50);
            System.out.println("...hecho");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    /*
     * Si la nómina ya existe y las horas trabajadas son las mismas,
     * entonces se recupera la nómina tal cual. En caso contrario,
     * se calcula, se almacena y se retorna.
     */
    @Override
    public Nomina getNomina(int codEmp, Meses mes, int horasTrabajas)
{
            Nomina nomina = NominaDao.getNomina(codEmp, mes);
            if (existeNomina(nomina) &&
                  mismasHorasTrabajadas(nomina, horasTrabajas))
            {
                  return nomina;
            }
            return createAndStoreNomina(codEmp, mes, horasTrabajas);
      }
```

```
    private boolean existeNomina(Nomina nomina) {
      return nomina != null;
    }

    private boolean mismasHorasTrabajadas(Nomina nomina, int newHoras)
{
      int oldHoras = nomina.getHorasTrabajadas();
            return oldHoras == newHoras ? true : false;
      }

    /*
     * Se obtiene la nueva nomina calculada y se almacena
     */
      private Nomina createAndStoreNomina(int codEmp, Meses mes, int
horasTrabajadas) {
            Nomina nomina = calcularNomina(codEmp, mes,
horasTrabajadas); // calcularla
            NominaDao.guardarNomina(codEmp, mes, nomina); // guardarla
            return nomina;
      }

      /*
       * El cálculo de un nomina es algo realmente complejo,
       * por lo que simplificamos tanto como nos apetezca.
       */
      private Nomina calcularNomina(int codEmp, Meses mes,
                  int horasTrabajadas)
      {
            Empleado empleado = EmpleadoDao.getEmpleado(codEmp);
```

```
            float bruto = empleado.getSueldoBruto();
            float bruto_mes_teorico = bruto / 12; // 12 meses

            /*
             * Calculamos el sueldo en bruto para el mes.
             * El empleado tiene un sueldo en bruto mensual si ha
trabajado
             * exactamente el numero de horas fijado por
horasMensualesOficiales.
             * Tenemos que mirar cuántas horas ha trabajado realmente
para
             * estimar su sueldo bruto mensual.
             */
            float bruto_mes_calculado =
                  (bruto_mes_teorico * horasTrabajadas) /
horasMensualesOficiales;

            // Descontamos la retencion IRPF
            float neto_mes = (1 - porcentRet/100f) *
bruto_mes_calculado;

            String observaciones = "Ha trabajado ";

            if (horasTrabajadas > horasMensualesOficiales) {
                  observaciones += (horasTrabajadas-
horasMensualesOficiales) + " horas mas ese mes";
            } else if (horasTrabajadas < horasMensualesOficiales) {
                  observaciones += (horasMensualesOficiales-
horasTrabajadas) + " horas menos ese mes.";
            } else {
                  observaciones += " las horas normales: " +
horasMensualesOficiales + ".";
```

```
            }

            Nomina nomina = new Nomina(codEmp, horasTrabajadas, mes,
                        bruto_mes_calculado, neto_mes, observaciones);
            return nomina;
      }

}
```

Veamos por fin la clase proxy. Esta clase:

- Implementa la interfaz ServicioNominas, igual que la clase ServicioNominasImpl.
- Define un atributo de tipo ServicioNominas en el que guardará una instancia de la clase ServicioNominasImpl (utilizará este atributo para efectuar la técnica de la Delegación).

La finalidad de esta clase es impedir que el código cliente acceda libremente (sin control) a los servicios de nóminas. Para ello obliga a pasar las credenciales del usuario en su constructor. Si éstas son correctas, el atributo booleano 'validado' toma el valor verdadero. El valor de este atributo se comprueba en el método getNomina(), el cual sólo accede al servicio real de nóminas si 'validado' vale verdadero. En caso contrario se lanza una excepción y finalizará el programa.

Por simplicidad, el proxy almacena en un HashMap las credenciales de los usuarios con permiso para acceder al módulo de nóminas.

Notad además que el método getNomina() sólo crea el objeto ServicioNominasImpl si este no existe, es decir, la primera vez. El resto de ejecuciones de getNomina() utilizan el objeto cacheado, con lo que suele mejorarse el rendimiento cuando el objeto tiene una construcción computacionalmente laboriosa.

ProxyServicioNominas.java

```
package estructurales.proxy.protection_proxy.servicios;
```

```java
import java.util.HashMap;
import java.util.Map;

import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.modelo.Nomina;

public class ProxyServicioNominas implements ServicioNominas {

    private ServicioNominas servicio;
      private boolean validado;
    private static Map<String, String> credenciales;

    /*
     * Creamos algunas credenciales de usurios del departamento
     * de nominas y las guarfamos en un Map.
     * Se tendrían que obtener de una BD o de un LDAP.
     */
    static {
      credenciales = new HashMap<String, String>();
            credenciales.put("admin", "12345");
            credenciales.put("test", "test");
      }

    /*
     * Constructor
     */
    public ProxyServicioNominas(String presuntoLogin,
            String presuntoPassword) throws ValidacionException
```

```
    {
      System.out.println("Construyendo un objeto
ProxyServicioNominas...");
      System.out.println("Validando usuario...");

      comprobarCredenciales(presuntoLogin, presuntoPassword);

    }

    /*
     * Constrastar las credenciales suministradas con las
     * almacenadas en el HashMap.
     */
    private void comprobarCredenciales(String presuntoLogin,
                String presuntoPassword) throws ValidacionException
    {
      String passwordRecuperado =
                credenciales.get(presuntoLogin);

      if (passwordRecuperado != null) {
            if (passwordRecuperado.equals(presuntoPassword)) {
                  validado = true;
                  System.out.println("...credenciales correctas.");
            } else {
                  System.out.println("...credenciales incorrectas.");
                  throw new ValidacionException("El password
especificado no es correcto.");
            }
      } else {
            System.out.println("...credenciales incorrectas.");
```

```
            throw new ValidacionException("El login especificado no
existe.");
      }
      }

      @Override
    public Nomina getNomina(int codEmp, Meses mes, int
horasTrabajadas) {
      if (validado) {
            if (servicio == null) {
                  servicio = new ServicioNominasImpl();
            }
            return servicio.getNomina(codEmp, mes, horasTrabajadas);
      } else {
            throw new RuntimeException("Acceso denegado. El usuario no
se ha validado.");
      }
    }
}
```

Para finalizar la capa de negocio nos falta por ver la clase que implementa la excepción de validación que podría lanzar el proxy:

ValidacionException.java

```
package estructurales.proxy.protection_proxy.servicios;


public class ValidacionException extends Exception {


      private static final long serialVersionUID = 1L;
      private static final String defaultMsg = "Error de validacion";
```

```
	public ValidacionException() {
		super(defaultMsg);
	}
	public ValidacionException(String msg) {
		super(msg);
	}
	public ValidacionException(Throwable excep) {
		super(excep);
	}
	public ValidacionException(String msg, Throwable excep) {
		super(msg, excep);
	}
}
```

Capa de presentación

Veamos ahora las clases de la capa de presentación. La aplicación utiliza dos ventanas. La ventana inicial que solicita las credenciales al usuario y la ventana con la que interactúa el usuario para calcular nóminas. La primera ventana se compone de una única clase, Login, mientras que la segunda se compone de tres clases: MainClientGUI, ConfiguradorGrafico y NumericTextField.

Aunque al ejecutar la aplicación lo primero que se aprecia es la ventana de login, realmente la aplicación se inicia desde la clase MainClientGUI. Por esto, dejemos la clase Login para el final.

NumericTextField es una clase que extiende a JTextField de Swing con la finalidad de sólo permitir escribir dígitos en una caja de texto. Esto nos va muy bien para que el usuario introduzca las horas trabajadas de los empleados, ya que se trata de números enteros.

MainClientGUI y ConfiguradorGrafico son en realidad una única clase, aunque para evitar crear una clase con más de 400 líneas de código, lo cual dificultaría su legibilidad, se ha optado por separar el código en dos, delegando la responsabilidad de dibujar y configurar los controles gráficos en la clase ConfiguradorGrafico. MainClientGUI tiene el resto de código necesario (de hecho, un código más interesante) para hacer operativa la interfaz de usuario.

Veamos ahora el código de estas tres clases y posteriormente comentamos y vemos el código para la clase Login.

Comenzamos con la clase por la que se inicia la aplicación. Notad que en su constructor, antes de hacerse visible, se ocupa de invocar a la clase Login. Las credenciales devueltas por Login las utiliza como argumentos en la llamada al constructor de ProxyServicioNominas.

MainClientGUI.java

```
package estructurales.proxy.protection_proxy.presentacion;


import java.awt.*;
import java.awt.Dialog.ModalityType;
import java.awt.event.*;

import java.util.*;


import javax.swing.JFrame;


import estructurales.proxy.protection_proxy.dao.EmpleadoDao;
import estructurales.proxy.protection_proxy.modelo.Empleado;
import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.modelo.Nomina;
import estructurales.proxy.protection_proxy.servicios.ServicioNominas;
```

```
public class MainClientGUI {

	private ConfiguradorGrafico cg; // JFrame que soporta toda la
GUI

	private int codEmpSeleccionado;
	private Meses mesSeleccionado;
	private ServicioNominas servicioNominas;

	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				MainClientGUI frame = new MainClientGUI();
				frame.cg.setVisible(true);
			}
		});
	}

	// Constructor
	public MainClientGUI() {
		Login login = validarUsuario();
		adquirirServicioNominas(login);
		if (servicioAdquirido()) {
			inicializarGUI();
		} else {
			throw new RuntimeException("No se ha podido adquirir
el servicio protegido.");
		}
	}
```

```
	/*
	 * Muestra una pantalla de login para que el usuario
	 * se autentifique.
	 */
	private Login validarUsuario() {
		JFrame ventanaPadre = null;
		Login login = new Login(ventanaPadre , "Acceso al
sistema",
				ModalityType.APPLICATION_MODAL);
		login.setVisible(true);
		return login;
	}

	/*
	 * Si la utenticacion ha sido correcta se obtiene el
	 * servicio de nominas que proporciona el proxy.
	 */
	private void adquirirServicioNominas(Login login) {
		servicioNominas = login.getServicio();
	}

	private boolean servicioAdquirido() {
		return servicioNominas != null;
	}

	private void inicializarGUI() {
		dibujarGUI();
		limpiarYmostrarDatosEmpleado();
	}
```

```
	/*
	 * Con el propósito de no cargar a la clase de código rutinario,
	 * delegamos en otra clase la confeccion de la interfaz grafica
	 */
	private void dibujarGUI() {
		cg = new ConfiguradorGrafico(this);
		cg.crearGUI();
	}

	private void limpiarYmostrarDatosEmpleado() {
		limpiarFormulario();
		mostrarDatosEmpleado();
	}

	private void limpiarFormulario() {
		cg.txtNomEmpleado.setText("");

cg.txtHorasTrabajadas.setText(getStringHorasMensualesOficiales());
		cg.txtSBAnual.setText("");
		cg.txtSBMensual.setText("");
		cg.txtSNMensual.setText("");
		cg.txtObservaciones.setText("");
	}

	/*
	 * En funcion del empleado seleccionado actualmente en el
	 * combo, mostrar el nombre y el sueldo bruto del empleado.
	 */
```

```
	private void mostrarDatosEmpleado() {
		int codEmp = getIntegerCodEmpFromCombo();
		Empleado empleado = EmpleadoDao.getEmpleado(codEmp);
		cg.txtNomEmpleado.setText(empleado.getNombre());

	cg.txtSBAnual.setText(getStringSueldoBrutoEmpleado(empleado));
		cg.txtHorasTrabajadas.requestFocusInWindow();
	}

	private int getIntegerCodEmpFromCombo() {
		String elementoSeleccionado =
			cg.cmbEmpleado.getSelectedItem().toString();
		return Integer.valueOf(elementoSeleccionado);
	}

	private String getStringSueldoBrutoEmpleado(Empleado empleado) {
		return String.valueOf(empleado.getSueldoBruto());
	}

	String getStringHorasMensualesOficiales() {
		return
String.valueOf(ServicioNominas.horasMensualesOficiales);
	}

	private int getIntegerHorasTrabajadas() {
		return
Integer.valueOf(cg.txtHorasTrabajadas.getText().toString());
	}

	private String getStringBrutoMensualNomina(Nomina nomina) {
```

```
            return String.valueOf(nomina.getBruto());
      }

      private String getStringNetoMensualNomina(Nomina nomina) {
            return String.valueOf(nomina.getNeto());
      }

      /*
       * Cuando el usuario selecciona un nuevo empleado:
       * 1) Lo establecemos como el empleado actual para
       *    el calculo de la nomina.
       * 2) Mostramos sus datos en el resto de campos
       *    del formulario.
       */
      void cmbEmpleadoHaCambiado(ItemEvent e) {
            codEmpSeleccionado =
Integer.valueOf(e.getItem().toString());
            mostrarDatosEmpleado();
      }

      /*
       * Cuando el usuario selecciona un nuevo mes lo
       * establecemos com el mes actual para el calculo
       * de la nomina.
       */
      void cmbMesesHaCambiado(ItemEvent e) {
            mesSeleccionado =
                  Meses.valueOf(e.getItem().toString());
      }
```

```
	/*
	 * Rellena el combo de empleados.
	 * 1) Obtiene los empleados de EmpleadoDao
	 * 2) Los ordena.
	 * 3) Devuelve un String[] que contiene el codigo
	 *    de cada empleado.
	 */
	String[] getEmpleados() {
		final String[] EMPTY_EMPLEADOS = new String[0];

		java.util.List<Empleado> empleados = new
ArrayList<Empleado>(
				EmpleadoDao.getEmpleados().values());

		Collections.sort(empleados);

		if (!empleados.isEmpty()) {
			Collection<String> codEmpleados = new
ArrayList<String>(
					empleados.size());
			for (Empleado empleado : empleados) {

	codEmpleados.add(String.valueOf(empleado.getCodigo()));
			}
			return codEmpleados.toArray(EMPTY_EMPLEADOS);
		}

		return EMPTY_EMPLEADOS;
	}
```

```
	void btnVerNominaPulsado(ActionEvent e) {
		if (codEmpSeleccionado == 0) {
			codEmpSeleccionado =
Integer.valueOf(cg.cmbEmpleadoModel
					.getSelectedItem().toString());
		}
		if (mesSeleccionado == null) {
			mesSeleccionado =
Meses.valueOf(cg.cmbMes.getSelectedItem()
					.toString());
		}
		getNomina();
	}


	/*
	 * Obtiene la nomina para el cliente/mes seleccionado
	 */
	private void getNomina() {
		if (servicioAdquirido()) {
			Nomina nomina =
servicioNominas.getNomina(codEmpSeleccionado,
					mesSeleccionado,
getIntegerHorasTrabajadas());


	cg.txtSBMensual.setText(getStringBrutoMensualNomina(nomina));


	cg.txtSNMensual.setText(getStringNetoMensualNomina(nomina));


	cg.txtObservaciones.setText(nomina.getObservaciones());
```

```
            } else {
                  throw new RuntimeException("No se ha podido adquirir
el servicio protegido.");
            }
      }
}
```

A continuación, el código para la clase que dibuja los widgets:

ConfiguradorGrafico.java

```
package estructurales.proxy.protection_proxy.presentacion;


import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.*;
import javax.swing.UIManager.LookAndFeelInfo;
import javax.swing.border.EmptyBorder;

import estructurales.proxy.protection_proxy.modelo.Meses;
import estructurales.proxy.protection_proxy.servicios.ServicioNominas;

public class ConfiguradorGrafico extends JFrame {

      private static final long serialVersionUID = 1L;
```

```
JPanel panelBase;

DefaultComboBoxModel cmbEmpleadoModel;
JComboBox cmbEmpleado, cmbMes;

JLabel lblEmpleado, lblMes, lblHorasTrabajadas;
JLabel lblSBAnual, lblRetencion, lblSBMensual, lblSNMensual;
JLabel lblObservaciones, lblSimboloPorcenRet;

JButton btnVerNomina;
JSeparator separator;

// ojo, no son editables
JTextField txtSBAnual, txtRetencion, txtSBMensual;
JTextField txtNomEmpleado, txtSNMensual, txtObservaciones;

// Unico campo editable, es una especializacion de JTextField
NumericTextField txtHorasTrabajadas;

private MainClientGUI gui;

public ConfiguradorGrafico(MainClientGUI gui) {
      this.gui = gui;
}

public ConfiguradorGrafico getConfiguradorGrafico() {
      return this;
}
```

```
	public void crearGUI() {
		dibujarGeneralidades();
		dibujarPanelBase();
		dibujarSeparador();
		dibujarEtiquetas();
		dibujarCajasTexto();
		dibujarCombos();
		dibujarBotones();
		crearManejadores();
	}

	private void dibujarGeneralidades() {
		setLookAndFeel();
		setResizable(false);
		setTitle("Aplicaci\u00F3n N\u00F3minas");
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 506, 300);
		setLocationRelativeTo(null);
	}

	static void setLookAndFeel() {
		try {
			for (LookAndFeelInfo info :
UIManager.getInstalledLookAndFeels()) {
				if ("Nimbus".equals(info.getName())) {

	UIManager.setLookAndFeel(info.getClassName());
					break;
				}
			}
```

```
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	private void dibujarPanelBase() {
		panelBase = new JPanel();
		panelBase.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(panelBase);
		panelBase.setLayout(null);
	}

	private void dibujarSeparador() {
		separator = new JSeparator();
		separator.setBounds(10, 66, 478, 2);
		panelBase.add(separator);
	}

	private void dibujarBotones() {
		btnVerNomina = new JButton("Ver n\u00F3mina");
		btnVerNomina.setBounds(195, 230, 104, 25);
		panelBase.add(btnVerNomina);
	}

	private void dibujarCombos() {

		// Obtener del dataSource todos los codigo de empleado
		cmbEmpleadoModel =
			new DefaultComboBoxModel(gui.getEmpleados());
```

```
        cmbEmpleado = new JComboBox();
        cmbEmpleado.setModel(cmbEmpleadoModel);
        cmbEmpleado.setBounds(10, 30, 72, 25);
        panelBase.add(cmbEmpleado);

        cmbMes = new JComboBox(Meses.values());
        cmbMes.setBounds(338, 30, 79, 25);
        panelBase.add(cmbMes);
    }

    private void dibujarEtiquetas() {
        lblEmpleado = new JLabel("Empleado");
        lblEmpleado.setBounds(10, 11, 90, 14);
        panelBase.add(lblEmpleado);

        lblMes = new JLabel("Mes");
        lblMes.setBounds(342, 11, 46, 14);
        panelBase.add(lblMes);

        lblSBMensual = new JLabel("Sueldo bruto mensual");
        lblSBMensual.setBounds(166, 74, 133, 14);
        panelBase.add(lblSBMensual);

        lblHorasTrabajadas = new JLabel("Horas trabajadas");

    lblHorasTrabajadas.setHorizontalAlignment(SwingConstants.RIGHT);
        lblHorasTrabajadas.setBounds(384, 11, 104, 14);
        panelBase.add(lblHorasTrabajadas);
```

```
        lblRetencion = new JLabel("Retenci\u00F3n");
        lblRetencion.setBounds(302, 74, 72, 14);
        panelBase.add(lblRetencion);

        lblSNMensual = new JLabel("Sueldo neto mensual");
        lblSNMensual.setBounds(374, 74, 124, 14);
        panelBase.add(lblSNMensual);

        lblSimboloPorcenRet = new JLabel("%");
        lblSimboloPorcenRet.setBounds(355, 97, 24, 14);
        panelBase.add(lblSimboloPorcenRet);

        lblSBAnual = new JLabel("Sueldo bruto anual");
        lblSBAnual.setBounds(10, 74, 133, 14);
        panelBase.add(lblSBAnual);

        lblObservaciones = new JLabel("Observaciones");
        lblObservaciones.setBounds(10, 121, 117, 14);
        panelBase.add(lblObservaciones);
    }

    private void dibujarCajasTexto() {
        txtNomEmpleado = new JTextField("----");
        txtNomEmpleado.setBounds(92, 30, 236, 25);
        txtNomEmpleado.setEditable(false);
        txtNomEmpleado.setBackground(Color.yellow);
        panelBase.add(txtNomEmpleado);
```

```
            txtHorasTrabajadas = new NumericTextField();

txtHorasTrabajadas.setText(gui.getStringHorasMensualesOficiales());

      txtHorasTrabajadas.setHorizontalAlignment(SwingConstants.RIGHT);
            txtHorasTrabajadas.setBounds(442, 30, 46, 25);
            panelBase.add(txtHorasTrabajadas);

            txtSBMensual = new JTextField();
            txtSBMensual.setHorizontalAlignment(SwingConstants.RIGHT);
            txtSBMensual.setBounds(166, 92, 115, 25);
            txtSBMensual.setEditable(false);
            panelBase.add(txtSBMensual);
            txtSBMensual.setColumns(10);

            txtRetencion = new JTextField();
            txtRetencion.setHorizontalAlignment(SwingConstants.RIGHT);
            txtRetencion.setColumns(10);
            txtRetencion.setBounds(302, 92, 52, 25);
            txtRetencion.setEditable(false);

txtRetencion.setText(String.valueOf(ServicioNominas.porcentRet));
            panelBase.add(txtRetencion);

            txtSNMensual = new JTextField();
            txtSNMensual.setHorizontalAlignment(SwingConstants.RIGHT);
            txtSNMensual.setColumns(10);
            txtSNMensual.setBounds(384, 92, 104, 25);
            txtSNMensual.setEditable(false);
```

```
            panelBase.add(txtSNMensual);

            txtObservaciones = new JTextField();
            txtObservaciones.setBounds(10, 139, 478, 80);
            panelBase.add(txtObservaciones);
            txtObservaciones.setEditable(false);
            txtObservaciones.setColumns(10);

            txtSBAnual = new JTextField();
            txtSBAnual.setHorizontalAlignment(SwingConstants.RIGHT);
            txtSBAnual.setEditable(false);
            txtSBAnual.setColumns(10);
            txtSBAnual.setBounds(10, 92, 115, 25);
            panelBase.add(txtSBAnual);
      }

      public void crearManejadores() {
            // Controlar evento pulsacion boton ver nominas
            btnVerNomina.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
                        gui.btnVerNominaPulsado(e);
                  }
            });

            // Controlar evento seleccion codigo empleado en combo
empleados
            cmbEmpleado.addItemListener(new ItemListener() {
                  public void itemStateChanged(ItemEvent e) {
                        gui.cmbEmpleadoHaCambiado(e);
```

```
                }
            });

            // Controlar evento seleccion mes en combo meses
            cmbMes.addItemListener(new ItemListener() {
                public void itemStateChanged(ItemEvent e) {
                    gui.cmbMesesHaCambiado(e);
                }
            });
      }
}
```

Ahora vemos el código de la subclase de JTextField:

NumericTextField.java

```
package estructurales.proxy.protection_proxy.presentacion;

import java.util.regex.Pattern;

import javax.swing.JTextField;
import javax.swing.text.AttributeSet;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.PlainDocument;

public class NumericTextField extends JTextField{

      private static final long serialVersionUID = 1L;
```

```
      @Override
    protected Document createDefaultModel()
    {
        return new NumericDocument();
    }

    private static class NumericDocument extends PlainDocument
    {
            private static final long serialVersionUID = 1L;
            // Expresion regular con la que debe coincidir lo que
teclee el usuario (sólo digitos)
        private final static Pattern DIGITS = Pattern.compile("\\d*");
        // No queremos el campo tenga mas de 3 digitos
        private final static int MAX_LEN_PERMITIDA = 3;

        @Override
        public void insertString(int offs, String str, AttributeSet a)
throws BadLocationException
        {
            // Solo admitir el texto si es un numero y no se excede el
tamaño permitido
            if (str != null) {
                if (caracterEsValido(str) && longitudEsValida(str)) {
                    super.insertString(offs, str, a);
                }
            }
        }

        private boolean caracterEsValido(String str) {
            return DIGITS.matcher(str).matches();
```

```
        }

        private boolean longitudEsValida(String str) {
            int lenMaxima = getLength() + str.length();
            return lenMaxima <= MAX_LEN_PERMITIDA ? true : false;
        }
    }
}
```

Ahora tan sólo nos resta ver el código de la clase Login:

Login.java

```
package estructurales.proxy.protection_proxy.presentacion;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


import
estructurales.proxy.protection_proxy.servicios.ProxyServicioNominas;
import estructurales.proxy.protection_proxy.servicios.ServicioNominas;
import
estructurales.proxy.protection_proxy.servicios.ValidacionException;


/*
 * Clase que muestra una cuadro de dialogo para que el
 * usuario introduzca sus credenciales. En caso de no
 * validarse correctamente no se permite el acceso a la
 * aplicacion y finaliza.
 */
```

```
public class Login extends JDialog {

	private static final long serialVersionUID = 1L;
	private static final int LIMITE_ERRORES_VALIDACION = 3;

	protected JFrame propietario;

	private JLabel lblUsuario, lblContrasena, lblIconoLlaves;
	protected JTextField txtUsuario;
	protected JPasswordField txtContrasena;
	protected JButton btnAcceder, btnSalir;
	protected int contador_errores;

	private ServicioNominas servicio;

	/*
	 * Constructor.
	 * @propietario: la ventana que contiene al cuadro de dialogo
	 * @modalidad: modal/no modal (normalmente modal)
	 */
	public Login(JFrame propietario, String tit, Dialog.ModalityType
modalidad) {
		// Invocamos a super, indicando adicionalmente
		// el titulo para la ventana
		super(propietario, tit, modalidad);
		this.propietario = propietario;
		this.setResizable(false);
		this.setSize(337, 175);
```

```
        this.setLocationRelativeTo(null);

        create(); // Creacion de los controles
    }

    /*
     * Creacion de la GUI
     */
    private void create() {

        ConfiguradorGrafico.setLookAndFeel();

        // Listener de la ventana. Si el usuario
        // la cierra, finaliza la aplicacion
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                salir();
            }
        });

        // No usamos ningun layout manager
        setLayout(null);


        lblUsuario = new JLabel();
        lblUsuario.setHorizontalAlignment(SwingConstants.LEFT);
        lblUsuario.setForeground(new Color(0, 0, 255));
        lblUsuario.setText("usuario:");
        lblUsuario.setBounds(110, 9, 106, 18);
```

```
            add(lblUsuario);


            lblContrasena = new JLabel();
            lblContrasena.setHorizontalAlignment(SwingConstants.LEFT);
            lblContrasena.setForeground(new Color(0, 0, 255));
            lblContrasena.setText("contrase\u00F1a:");
            lblContrasena.setBounds(110, 54, 97, 18);
            add(lblContrasena);


            // Cargamos la imagen del icono manualmente
            lblIconoLlaves = new
JLabel(crearImageIcon(this.getClass(), "llaves.png"));
            lblIconoLlaves.setBounds(6, 24, 86, 81);
            add(lblIconoLlaves);


            txtUsuario = new JTextField();
            txtUsuario.setForeground(new Color(0, 0, 255));
            txtUsuario.setSelectedTextColor(new Color(0, 0, 255));
            txtUsuario.setToolTipText("Introducir un nombre de usuario
valido");
            txtUsuario.setBounds(110, 27, 183, 22);
            add(txtUsuario);


            txtContrasena = new JPasswordField();
            txtContrasena.setForeground(new Color(0, 0, 255));
            txtContrasena.setToolTipText("Introducir la
contrase\u00F1a");
            txtContrasena.setBounds(110, 71, 183, 22);
            txtContrasena.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) {
```

```
                btnAccederPulsado(e);
            }
        });
        add(txtContrasena);

        btnAcceder = new JButton();
        btnAcceder.setBounds(110, 105, 85, 27);
        btnAcceder.setText("Acceder");
        btnAcceder.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                btnAccederPulsado(e);
            }

        });
        add(btnAcceder);

        btnSalir = new JButton();
        btnSalir.setBounds(206, 105, 85, 27);
        btnSalir.setText("Salir");
        btnSalir.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                salir();
            }
        });
        add(btnSalir);
    }


    /*
```

```
	 * Manejador para la pulsacion del boton 'Acceder'.
	 */
	protected void btnAccederPulsado(ActionEvent e) {

		String username = txtUsuario.getText();
		String password = new String(txtContrasena.getPassword());

		/*
		 * Se comprueba que ni el nombre de usuario ni la
		 * contrasena esten en blanco. En caso contrario se
		 * muestra una ventana informando de este hecho.
		 */
		if (username.equals("") || password.equals("")) {
			String msg="Debe proporcionar un nombre de usuario y
una contrase\u00F1a.";
			btnAcceder.setEnabled(false);
			JLabel errorFields = new JLabel(
					"<HTML><FONT COLOR =
Blue>"+msg+"</FONT></HTML>");
			JOptionPane.showMessageDialog(null, errorFields);

			btnAcceder.setEnabled(true);
			txtUsuario.requestFocusInWindow();

		} else { // Ambos campos tienen valor
			try {
				// Intentamos acceder al proxy
				servicio = new ProxyServicioNominas(username,
password);
```

```
                        if (propietario != null) {
                              String tit = propietario.getTitle() + " -
usuario: " + username;
                              propietario.setTitle(tit);
                        }
                        dispose();


                  } catch (ValidacionException e1) {
                        String msg = "Las credenciales son
incorrectas.";
                        btnAcceder.setEnabled(false);
                        contador_errores++;


                        if (contador_errores ==
LIMITE_ERRORES_VALIDACION) {
                              String msg2 = msg +
                                    "<br/>Ha superado el numero de
intentos maximo. <br/>La aplicacion no puede continuar.";
                              JLabel errorFields = new JLabel(
                              "<HTML><FONT COLOR = Red>" + msg2 +
"</FONT></HTML>");
                              JOptionPane.showMessageDialog(null,
errorFields);
                              salir();
                        } else {
                              JLabel errorFields = new JLabel(
                                    "<HTML><FONT COLOR = Blue>" + msg +
"</FONT></HTML>");
                              JOptionPane.showMessageDialog(null,
errorFields);
                        }
                        btnAcceder.setEnabled(true);
```

```
                        txtContrasena.setText("");
                        txtUsuario.setSelectionStart(0);
      txtUsuario.setSelectionEnd(txtUsuario.getText().length());
                        txtUsuario.requestFocusInWindow();
                  }  catch (Exception e2) {
                        e2.printStackTrace();
                        salir();
                  }
            }
      }

      protected void salir() {
            System.exit(0);
      }

      /*
       * @param clase: this.getClass()
       * @param path: El paquete relativo a la clase
       *
       * Retorna un objeto ImageIcon a partir de una ruta,
       * o null si el path no es correcto.
       */
      private static ImageIcon crearImageIcon(Class<?> clase, String
path) {
            java.net.URL imgURL = clase.getResource(path);
            if (imgURL != null) {
                  return new ImageIcon(imgURL);
            } else {
```

```
                throw new RuntimeException("Error al crear la
imagen");
            }
      }

      public ServicioNominas getServicio() {
            return servicio;
      }
}
```