Plan de Pruebas de Software

MiPandilla

Versión 1.0 Finalizado

Preparada por:

Alejandro Bermejo Vialas, Saad Ziyad Khalid

Universidad Carlos III de Madrid

6/12/2015

Contenido

1. Introducción

1.1. Propósito del documento

El objetivo principal del documento, es mostrar los resultados de las pruebas de software realizadas a la aplicación desarrollada para el proyecto. Analizaremos diferentes categorías y observaremos el comportamiento de las pruebas en el funcionamiento de la aplicación.

También, lo que se busca con este documento es garantizar que la aplicación y el código desarrollados para el proyecto, son lo suficientemente robustos y que todo funcionará como cabe esperar cuando el cliente haga uso de la aplicación.

Por eso, es muy importante realizar este plan de pruebas de manera rigurosa y a conciencia, para evitar comportamientos incorrectos de la aplicación y presentando al cliente una aplicación totalmente funcional y eficiente.

1.2. Glosario

- **JUnit:** es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.
- **Framework:** un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Prueba de software:** son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder.
- **Prueba unitaria:** es una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.
- **Pruebas de validación:** es una forma de comprobar y verificar, que el sistema de software producido, cumple con las especificaciones impuestas por el cliente.
- **Stakeholder:** quienes son afectados o pueden ser afectados por las actividades de una empresa.

1.3.Materiales de referencia

Para llevar a cabo este documento, nos apoyamos en los siguientes documentos como referencia:

- Documento de Diseño, MiPandilla.
- Documento ERS, MiPandilla.
- API de JUnit

1.4. Estructura del documento

El documento consta de dos partes principalmente, la primera hace una breve descripción del documento y esta formada por los siguientes apartados:

- **Propósito del documento:** descripción del ámbito del documento y cuáles son sus objetivos.
- **Glosario:** términos empleados en este documento.
- **Materiales referenciados**: lista con los documentos y otras referencias bibliográficas utilizadas en este documento.
- **Estructura del documento:** breve resumen de cómo se organizan los contenidos de este documento.

La segunda parte, es el plan de pruebas realizado, con las pruebas que se han llevado a cabo para comprobar que todo esta implementado correctamente. Esta parte, está formada por los siguientes apartados:

- **Plan de pruebas:** descripción los planes definidos para la realización de las pruebas y los resultados de su aplicación
- **Visión global del software a probar:** descripción general del software a probar y sus funcionalidad.
- **Restricciones:** descripción de la estrategia general para las pruebas.
- **Software a probar:** identificación de cada uno de los componentes de software que se van a probar

- **Pruebas unitarias:** descripción de las pruebas unitarias en los módulos seleccionados para las pruebas.
- **Pruebas de validación:** descripción de las pruebas de validación que se llevarán a cabo para cumplir con los requisitos funcionales especificados en el ERS.

2. Plan de pruebas

El plan que ha sido seguido consistió en hacer pruebas unitarias a 14 métodos perteneciendo a 3 clases utilizando para este motivo el Programa JUnit.

La primera clase que ha sido probada con pruebas unitarias es la clase Evento que es de tipo modelo. De la clase Evento 4 métodos fueron probados y estos métodos son:

El método getID() que tenia resultados correctos en las pruebas unitarias.

El método getNombre() que tenia resultados correctos en las pruebas unitarias.

El método getLugar() que tenia resultados correctos en las pruebas unitarias.

El método getFecha() que tenia resultados correctos en las pruebas unitarias.

La segunda clase que ha sido probada con pruebas unitarias es la clase Controller que es de tipo controlo. De la clase Controller 4 métodos fueron probados y estos métodos son:
El método limpiarConsola() que tenia resultados correctos en las pruebas unitarias.
El método inicio() que tenia resultados correctos en las pruebas unitarias.
El método menuPrincipal() que tenia resultados correctos en las pruebas unitarias.
eEl método buscarUsuarios() que tenia resultados correctos en las pruebas unitarias.

La tercera clase que ha sido probada con pruebas unitarias es la clase View que es de tipo vista. De la clase View 6 métodos fueron probados y estos métodos son:
El método limpiarPantalla() que tenia resultados correctos en las pruebas unitarias.
El método mostrarInicio() que tenia resultados correctos en las pruebas unitarias.
El método mostrarLogin() que tenia resultados correctos en las pruebas unitarias.
El método mostrarEventos() que tenia resultados correctos en las pruebas unitarias.
El método mostrarUsuarios() que tenia resultados correctos en las pruebas unitarias.
El método mostrarPerfil() que tenia resultados correctos en las pruebas unitarias.

Ademas a las pruebas unitarias, pruebas de validación fueron echas para comprobar que los requisitos funcionales han sido cumplidos correctamente.

1. Visión global del software a probar

El software que ha sido probado es parte del proyecto MiPandilla que consiste de varias clases que son de tipo control, vista o modelo. Cada clase proporciona cierta funcionalidad.

Dentro de las clases de tipo vista 6 métodos de la clase View han sido probados. El primer método probado es limpiarConsola() que al ser llamado debe limpiar la consola y dejar solo lo que debe ver el usuario. Esta funcionalidad del método ha sido probada y el resultado es correcto. El segundo método es mostrarInicio() que se encarga en mostrar el menu de inicio al usuario. Esta funcionalidad del método ha sido probada con éxito. El tercer método es mostrarLogin() que se encarga de mostrar el menu de login al usuario. Esta funcionalidad del método ha sido probada y el resultado es correcto. El cuarto método es mostrarEventos() que se encarga de mostrar eventos patrocinados y no patrocinados dado dos listas dinámicas una de eventos no patrocinados y otra de eventos patrocinados. Si los eventos no han sido representados con éxito el método devuelve false. Esta funcionalidad del método ha sido probada y el resultado es correcto. El quinto método es mostrarUsuarios() que se encarga de mostrar los usuarios al usuario y devuelve false al terminar de enseñar. Esta funcionalidad del método ha sido probada y el resultado es correcto. El sexto método es mostrarPerfil() que dado un usuario representa sus informaciones al usuario. Esta funcionalidad del método ha sido probada y el resultado es correcto.

Dentro de las clases de tipo control 4 métodos de la clase Controller han sido probados. El primer método probado es limpiarConsola() que debe llamar al método correspondiente en la clase View para enseñar al usuario solo lo que tiene que ver. Esta funcionalidad del método ha sido probada y el resultado es correcto. El segundo método probado es inicio() que al ser llamado debe llamar al método correspondiente de la clase View para mostrar el menu de inicio al usuario y conseguir la opción elegida por el usuario. Esta funcionalidad del método ha sido probada y el resultado es correcto. El tercer método probado es menuPrincipal() que al ser llamado enseña el menu principal al usuario y consigue la opción elegida por el. Esta funcionalidad del método ha sido probada y el resultado es correcto. El cuarto método probado es buscarUsuarios() que dado un usuario lo busca en la lista de usuarios y si no lo encuentra devuelve null. Esta funcionalidad del método ha sido probada y el resultado es correcto.

Dentro de las clases de tipo modelo 4 métodos del la clase Evento han sido probados. El premier método probado es getID() que dado un evento devuelve su identificador único. Esta funcionalidad del método ha sido probada y el resultado es correcto. El segundo método probado es getNombre() que dado un evento devuelve un string que indica el nombre del evento. Esta funcionalidad del método ha sido probada y el resultado es correcto. El tercer método probado es getLugar() que dado un evento devuelve un string que indica el lugar del evento. Esta funcionalidad del método ha sido probada y el resultado es correcto. El cuarto método probado es get fecha que dado un usuario devuelve un objeto de tipo GregorianCalendar que indique la fecha del evento. Esta funcionalidad del método ha sido probada y el resultado es correcto.

2. Restricciones

En las pruebas unitarias en general el método de probar es de crear variables con valores para lo cual se savia el resultado y pasar estos variables a los

métodos y comprobar utilizando los métodos assert de JUnit para comprobar si el resultado teórico coincide con lo que devuelven los métodos. En las pruebas de los requisitos funcionales e general el método seguido es de ejecutar el programa y interactuar con el para ver si el funcionamiento es correcto o no.

3. Software a probar

Componente	**Prueba**	**Tipo**	**numero**
getID()	Devolver el identificador correcto del evento	Unitaria	1
getNombre()	Devolver el nombre correcto del evento	Unitaria	2
getLugar()	Devolver el lugar correcto del eveto	Unitaria	3
getFecha()	Devolver la fecha correcta del evento	Unitaria	4
limpiarConsola()	Limpiar la consola para el usurario	Unitaria	5
inicio()	Enseñar el menu de inicio para el usuario	Unitaria	6
menuPrincipal()	Enseñar el menu principal al usuario	Unitaria	7
buscarUsuarios()	Devolver null si el usuario no existe	Unitaria	8
limpiarPantalla()	Limpiar la consola para el usurario	Unitaria	9
mostrarInicio()	Enseñar el menu de inicio para el usuario	Unitaria	10
mostrarLogin()	Enseñar el menu de login al usuario	Unitaria	11
mostrarEventos()	Devolver false al no enseñar todos los eventos	Unitaria	12
mostrarUsuarios()	Devolver false al terminar de mostrar los usuarios	Unitaria	13
mostrarPerfil()	Mostrar los datos correctos del usuario dado	Unitaria	14

4. Pruebas unitarias

Identificador	*PR01*
Nombre	*Prueba de conseguir el identificador de un evento*
Entrada	*Objeto de tipo usuario que contiene los datos correctos de un evento*
Prueba	*comprobar que el identificador del evento en el objeto evento es igual al identificador devuelto por el método*

Salida	*void*

Identificador	*PR02*
Nombre	*Prueba de conseguir el nombre de un evento*
Entrada	*Objeto de tipo usuario que contiene los datos correctos de un evento*
Prueba	*comprobar que el nombre del evento en el objeto evento es igual al nombre devuelto por el método*
Salida	*void*

Identificador	*PR03*
Nombre	*Prueba de conseguir el lugar de un evento*
Entrada	*Objeto de tipo usuario que contiene los datos correctos de un evento*
Prueba	*comprobar que el lugar del evento en el objeto evento es igual al lugar devuelto por el método*
Salida	*void*

Identificador	*PR04*
Nombre	*Prueba de conseguir la fecha de un evento*
Entrada	*Objeto de tipo usuario que contiene los datos correctos de un evento*
Prueba	*comprobar que la fecha del evento en el objeto evento es igual a la fecha devuelta por el método*
Salida	*void*

Identificador	*PR05*
Nombre	*Prueba de limpiar la consola para el usuario*
Entrada	*void*
Prueba	*Comprobar que la consola se limpia al llamar al método*
Salida	*void*

Identificador	*PR06*
Nombre	*Prueba de mostrar el menu de inicio por la clase controller*
Entrada	*void*
Prueba	*Comprobar que el menu de inicio se muestra al usuario*

Salida	*void*

Identificador	*PR07*
Nombre	*Prueba de enseñar el menu principal el usuario*
Entrada	*void*
Prueba	*Comprobar que el menu principal se muestra al usuario*
Salida	*void*

Identificador	*PR08*
Nombre	*Prueba de buscar usuario en la lista de usuarios*
Entrada	*Objeto de tipo usuarios que contiene todos los usuarios guardados por el sistema*
Prueba	*Comprobar que el método devuelve null al no encontrar el usuario*
Salida	*void*

Identificador	*PR09*
Nombre	*Prueba de limpiar la consola para el usuario por la clase View*
Entrada	*void*
Prueba	*Comprobar que la consola se limpia al llamar al método*
Salida	*void*

Identificador	*PR010*
Nombre	*Prueba de mostrar el menu de inicio por la clase View*
Entrada	*void*
Prueba	*Comprobar que el menu de inicio se muestra al usuario*
Salida	*void*

Identificador	*PR011*
Nombre	*Prueba de enseñar el menu de login al usuario*
Entrada	*void*
Prueba	*Comprobar que el menu de login se muestra correctamente al usuario*
Salida	*void*

Identificador	*PR012*

Nombre	*Prueba de enseñar los eventos*
Entrada	*Objeto de tipo eventos que contiene todos los eventos guardados por el sistema*
Prueba	*Comprueba que el método devuelve false al no enseñar todos los eventos*
Salida	*void*

Identificador	*PR013*
Nombre	*prueba de enseñar los usuarios*
Entrada	*Objeto de tipo usuarios que contiene todos los usuarios guardados por el sistema*
Prueba	*Comprueba que el método devuelve false al no enseñar todos los usuarios*
Salida	*void*

Identificador	*PR014*
Nombre	*Prueba de mostrar perfil de un usuario*
Entrada	*Objeto de tipo usuario que contiene los datos correctos de un evento*
Prueba	*Prueba que el perfil de usuario se imprima correctamente*
Salida	*void*

5. Pruebas de validación

Identificador de la prueba	*PV1*
Nombre	*Comprobar el registro correcto del usuario*
Descripción de la prueba	*Comprobar que cuando el usuario introduce su información en el menú de registrarse se cree un usuario y se guarda en el fichero de texto*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se guarda el usuario en el fichero de texto*
Requisitos validados	*RF01 ha sido validado*

Identificador de la prueba	*PV2*
Nombre	*Comprobar la manera de aceptar claves de usuarios*

Descripción de la prueba	*Comprobar que cuando el usuario introduce su clave para registrarse el sistema enseña mensajes de error cuando la clave no cumple tener al menos longitud seis, un dígito, una letra mayúscula que no sea igual que el nombre ni sea clave.*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*se mostrara un mensaje de error indicando lo que no cumple la clave*
Requisitos validados	*RF02 ha sido validado*

Identificador de la prueba	*PV3*
Nombre	*Comprobar el login correcto del usuario*
Descripción de la prueba	*Comprobar que cuando el usuario introduce su información correcta en el menú de login puede entrar y que si la información introducida no coincide se puede intentar de nuevo*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario entra a su cuenta o vuelve a introducir su información*
Requisitos validados	*RF03 ha sido validado*

Identificador de la prueba	*PV4*
Nombre	*Comprobar la accesibilidad del sistema*
Descripción de la prueba	*Comprobar que los eventos pueden ser visualizados por un usuario registrado o no registrado*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se representan los eventos a ambos tipos de usuarios*
Requisitos validados	*RF04 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV5*
Nombre	*Comprobar los campos opcionales del perfil del usuario*

Descripción de la prueba	*Comprobar que cuando se registra el usuario se muestran campos opcionales de nombre apellidos y otra información para rellenarse*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se representan los campos opcionales para ser rellenados*
Requisitos validados	*RF05 ha sido validado*

Identificador de la prueba	*PV6*
Nombre	*Comprobar la modificación o la eliminación correcta de los perfiles de los usuarios*
Descripción de la prueba	*Comprobar que un usuario registrado puede modificar su perfil o eliminarlo*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se elimina o se modifica el perfil del usuario*
Requisitos validados	*RF06 ha sido*

Identificador de la prueba	*PV7*
Nombre	Visualización de eventos
Descripción de la prueba	*Comprobar que un usuario registrado o no puede visualizar los eventos en la forma especificada en el ERS*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se Muestran los eventos en el formato especificado en el ERS*
Requisitos validados	*RF07 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV8*
Nombre	Visualización de usuarios
Descripción de la prueba	*Comprobar que un usuario registrado puede visualizar los usuarios en la forma especificada en el ERS*

Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se Muestran los usuarios en el formato especificado en el ERS*
Requisitos validados	*RF08 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV9*
Nombre	Seguimiento de usuario
Descripción de la prueba	*Comprobar que un usuario registrado puede seguir a otros usuarios registrados*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario registrado sigue a otros usuarios*
Requisitos validados	*RF09 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV10*
Nombre	Creación de eventos
Descripción de la prueba	*Comprobar que un usuario registrado patrocinador o no puede crear eventos patrocinados o no según el tipo de usuario*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario registrado cree el evento del tipo correcto según su tipo y se guarda*
Requisitos validados	*RF010 ha sido validado*

Identificador de la prueba	*PV11*
Nombre	Modificación de eventos
Descripción de la prueba	*Comprobar que un usuario registrado puede modificar sus eventos*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario registrado modifica su evento*
Requisitos validados	*RF011 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV12*

Nombre	Eliminación de eventos
Descripción de la prueba	*Comprobar que un usuario registrado puede eliminar sus eventos*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario registrado elimina sus eventos*
Requisitos validados	*RF012 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV13*
Nombre	Inscripción en eventos
Descripción de la prueba	*Comprobar que un usuario registrado puede inscribirse en un evento*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El usuario registrado se inscribe en el evento*
Requisitos validados	*RF013 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV14*
Nombre	Búsqueda de eventos
Descripción de la prueba	*Comprobar que un usuario registrado o no puede buscar eventos por un filtro proporcionado por el sistema*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Se muestra al usuario registrado o no los resultados de la búsqueda*
Requisitos validados	*RF014 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV15*
Nombre	Modificación de cuenta de usuario
Descripción de la prueba	*Comprobar que un usuario registrado puede modificar su cuenta*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Los campos cambiados en la cuenta se guardan*
Requisitos validados	*RF015 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV16*
Nombre	Eliminación de la cuenta del usuario
Descripción de la prueba	*Comprobar que un usuario registrado puede eliminar su cuenta*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*La cuenta se elimina correctamente*
Requisitos validados	*RF016 ha sido validado*

Identificador de la prueba	*PV17*
Nombre	Búsqueda de usuarios
Descripción de la prueba	*Comprobar que un usuario puede buscar a otros usuarios*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*Los usuarios buscados se deben mostrar al usuario según el formato del ERS y la resulta de aplicar el filtro*
Requisitos validados	*RF017 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV18*
Nombre	Comentarios en eventos
Descripción de la prueba	*Comprobar que un usuario registrado puede comentar en eventos*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El comentario del usuario se guarda en el evento*
Requisitos validados	*RF018 no ha sido validado por fallo en el sistema*

Identificador de la prueba	*PV19*
Nombre	Comentarios en perfiles de usuarios
Descripción de la prueba	*Comprobar que un usuario registrado puede comentar en el perfil de otro usuario*
Componentes involucrados	*View, Controller y modelo*
Resultado esperado	*El comentario del usuario se guarda en el perfil del otro usuario*

Requisitos validados	*RF019no ha sido validado por fallo en el sistema*

ANEXO

Se incluirá aquí los listados de las pruebas unitarias creadas con JUNIT

```
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import Controlador.Controller;
import Model.Usuario;
import Model.Usuarios;

import java.util.Scanner;


public class ControllerTest {

	@Before
	public void setUp() throws Exception {

		Controller c = new Controller();

	}


	@Test
	public void testLimpiarConsola() {
```

```
        Controller c = new Controller();
        int exp_val = 0;
        int real_val = c.limpiarConsola();
        assertEquals(exp_val, real_val);

    }

    @Test
    public void testInicio() {

        Controller c = new Controller();
        int real_val = c.inicio();
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the number you chose in the menu
again: ");
        int exp_val = s.nextInt();
        if (exp_val < 1 || exp_val > 4){
            exp_val = 0;
        }
        assertEquals(exp_val, real_val);

    }

    @Test
    public void testMenuPrincipal() {
```

```
            Controller c = new Controller();
            int real_val = c.menuPrincipal();
            Scanner s = new Scanner(System.in);
            System.out.println("Enter the number you chose in the menu
again: ");
            int exp_val = s.nextInt();
            switch(exp_val){
            case 1:
                  exp_val = 20;
                  break;
            case 2:
                  exp_val = 5;
                  break;
            case 3:
                  exp_val = 6;
                  break;
            case 4:
                  exp_val = 7;
                  break;
            case 5:
                  exp_val = 3;
                  break;
            }
            assertEquals(exp_val, real_val);
      }

      @Test
```

```
        public void testBuscarUsuarios() {

                Controller c = new Controller();
                Usuarios lu = new Usuarios();
                Usuario u = new Usuario();
                u = c.buscarUsuarios(lu);
                assertNull(u);

        }

}
```

```
import static org.junit.Assert.*;

import java.util.GregorianCalendar;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import Model.Evento;

public class EventoTest {


    /*
```

```
     * Para utilizar el constructor Evento hay
     * que hacerlo public en la clase Usuario.
    */

        @Before
        public void setUp() throws Exception {
                int id = 10;
                String autor = "Saad";
                String nombre = "Code Testing";
                String lugar = "Salon";
                byte ambito = 1;
                int day = 5;
                int month = 12;
                int year = 2015;
                String descripcion = "Having fun testing code";
                GregorianCalendar date = new
GregorianCalendar(day,month,year);
                Evento event = new Evento(id, autor, nombre, lugar, ambito, day,
month, year, descripcion);


        }




        @Test
        public void testGetID() {


                int id = 10;
```

```
        String autor = "Saad";
        String nombre = "Code Testing";
        String lugar = "Salon";
        byte ambito = 1;
        int day = 5;
        int month = 12;
        int year = 2015;
        String descripcion = "Having fun testing code";
        GregorianCalendar date = new
GregorianCalendar(day,month,year);
        Evento event = new Evento(id, autor, nombre, lugar, ambito, day,
month, year, descripcion);

         int actID = event.getID();
         assertEquals(id, actID);
    }

    @Test
    public void testGetNombre() {

        int id = 10;
        String autor = "Saad";
        String nombre = "Code Testing";
        String lugar = "Salon";
        byte ambito = 1;
        int day = 5;
        int month = 12;
        int year = 2015;
```

```
        String descripcion = "Having fun testing code";
        GregorianCalendar date = new
GregorianCalendar(day,month,year);
        Evento event = new Evento(id, autor, nombre, lugar, ambito, day,
month, year, descripcion);

         String actName = event.getNombre();
         assertEquals(nombre, actName);
    }

    @Test
    public void testGetLugar() {

        int id = 10;
        String autor = "Saad";
        String nombre = "Code Testing";
        String lugar = "Salon";
        byte ambito = 1;
        int day = 5;
        int month = 12;
        int year = 2015;
        String descripcion = "Having fun testing code";
        GregorianCalendar date = new
GregorianCalendar(day,month,year);
        Evento event = new Evento(id, autor, nombre, lugar, ambito, day,
month, year, descripcion);

         String actPlace = event.getLugar();
```

```
            assertEquals(lugar, actPlace);
        }

        @Test
        public void testGetFecha() {

            int id = 10;
            String autor = "Saad";
            String nombre = "Code Testing";
            String lugar = "Salon";
            byte ambito = 1;
            int day = 5;
            int month = 12;
            int year = 2015;
            String descripcion = "Having fun testing code";
            GregorianCalendar date = new
GregorianCalendar(year,month,day);
            Evento event = new Evento(id, autor, nombre, lugar, ambito, day,
month, year, descripcion);

            GregorianCalendar actDate = event.getFecha();
            assertEquals(date, actDate);
        }

}
```

```
import static org.junit.Assert.*;
```

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import Model.Eventos;
import Model.Usuario;
import Model.Usuarios;
import View.View;

public class ViewTest {

	@Before
	public void setUp() throws Exception {
	}


	@Test
	public void testLimpiarPantalla() {

		View v = new View();
		v.limpiarPantalla();
		System.out.println("@@@@@@@@ This should be the only message shown @@@@@@@@");

	}
```

```
	@Test
	public void testMostrarInicio() {
		View v = new View();
		v.mostrarInicio();
		System.out.println("@@@@@@@@ Initial menu should be printed by now @@@@@@@@");
	}

	@Test
	public void testMostrarLogin() {
		View v = new View();
		v.mostrarLogin();
		System.out.println("@@@@@@@@ The Login meu must be printed by now @@@@@@@@");
	}

	@Test
	public void testMostrarEventos() {
		View v = new View();
		Eventos es = new Eventos();
		Eventos es2 = new Eventos();
		int p = 1;
		boolean real_val = v.mostrarEventos(es, es2, p);
		boolean exp_val = false;
		assertEquals(exp_val, real_val);
	}

	@Test
```

```
	public void testMostrarUsuarios() {
		View v = new View();
		Usuarios us = new Usuarios();
		int p = 1;
		boolean exp_val = false;
		boolean real_val = v.mostrarUsuarios(us, p);
		assertEquals(exp_val, real_val);
	}

	@Test
	public void testMostrarPerfil() {
		View v = new View();
		Usuario u = new Usuario();
		v.mostrarPerfil(u);
		System.out.println("@@@@@@@@ The default user must be
printed @@@@@@@@");
	}




}
```