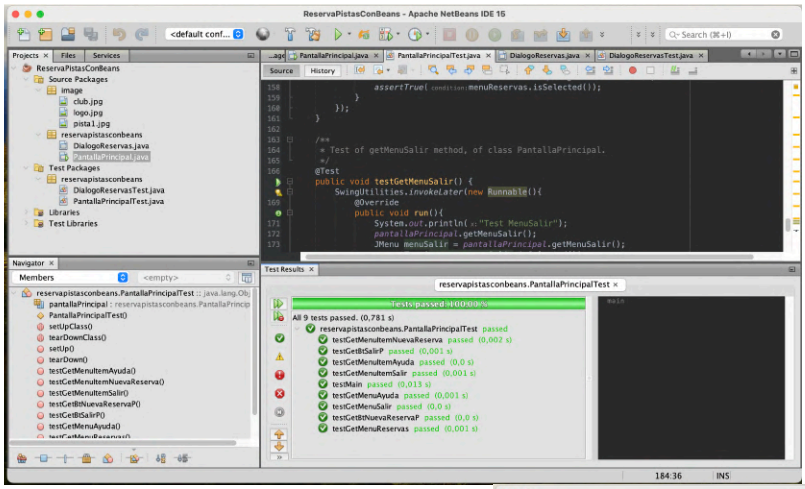
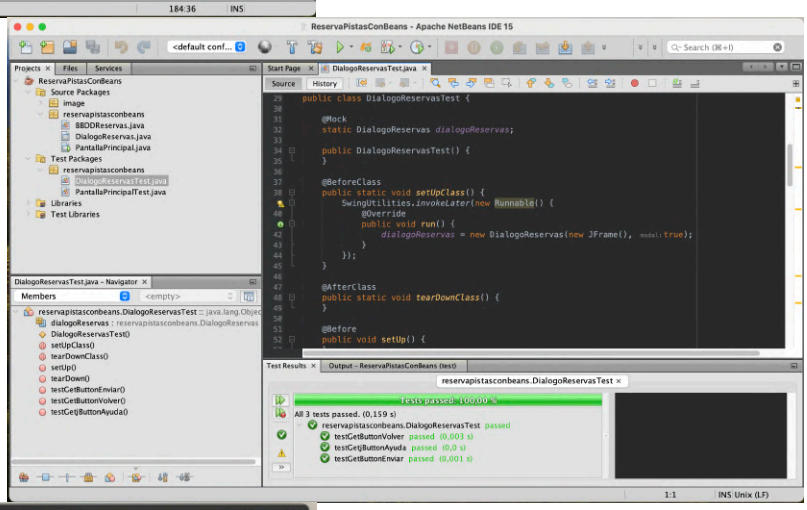
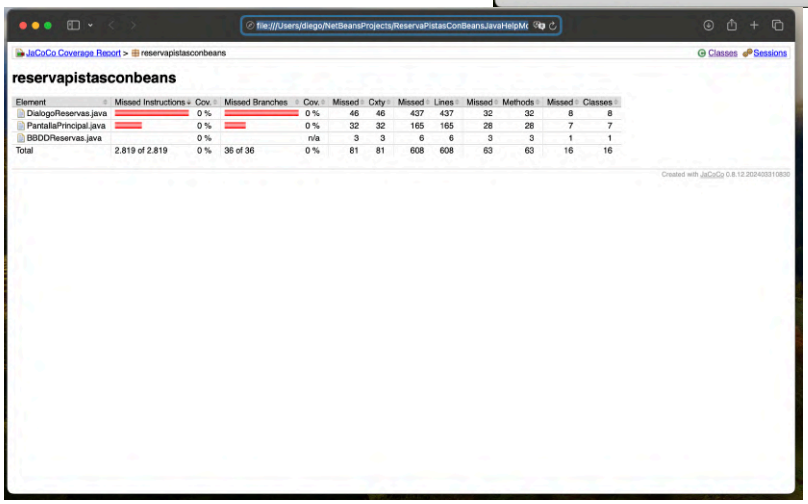
Tarea 8 para Desarrollo de Interfaces







Diego Manuel Carrasco Castañares

Detalles de la tarea de esta unidad.

Enunciado.

A partir de los recursos de la unidad 8, realiza las pruebas unitarias de tu aplicación de reservas teniendo en cuenta las siguientes observaciones:

La mayoría de los métodos son de tipo "actionPerformed". Estos se pueden simular invocando al método doClick() del elemento en cuestión (tienes un ejemplo más adelante).

Es necesario tener claro lo que se explica en el siguiente enlace: https://www.thegeekyway.com/edt_java/, ya que la simulación de los eventos la vamos a gestionar en el hilo Event Dispatching Thread (EDT), y del test se va a encargar el hilo principal (Main Thread). Por eso, para que no haya problemas, la clase de tests debemos crearla haciendo uso de SwingUtilities:

```
public class ReservaTest {

    private static Reservas reserva;

    public ReservaTest() {
    }

    @BeforeClass
    public static void setUpClass() {
        // Leer https://www.thegeekyway.com/edt_java/
        SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            reserva = new Reservas(new InitialJFrame(), true);
        }
    });
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() throws InterruptedException {
        // El hilo principal espera a que el EDT termine de inicializar
        Thread.sleep(1000, 0);
    }

    @After
    public void tearDown() {
    }

    // TODO add test methods here.
    // The methods must be annotated with annotation @Test. For example:
    //
    @Test
    public void rButtonCongresoActionPerformedTest() {
        //Reservas reserva = new Reservas(new InitialJFrame(), true);
        JRadioButton radio = reserva.getrButtonCongreso();
        radio.doClick();

        assertEquals(reserva.getjLabelDia().isEnabled(), true);
```

- Emplea el framework Mockito. Lo recomendable es hacerlo para probar la funcionalidad del botón Reservar, ya que, aunque ahora este botón no hace nada, si formara parte de una aplicación completa dicho botón se encargaría de conectar con el servidor y guardar la información de la reserva en una base de datos. Este proceso es el que hay que "mockear", recogiendo el evento y simular que "todo se ha guardado correctamente".

- Investiga cómo se puede revisar el porcentaje de cobertura de código (testeado), y aplícalo a tus pruebas.

Criterios de puntuación. Total 10 puntos.

1. Realizados tests unitarios para las principales partes de la aplicación. 3 puntos.

2. Realizado test empleando mockito para aislar a la clase y salvar las dependencias con otras clases. 3 puntos.

3. Incorporado plugin de cobertura de pruebas en el código y revisado porcentaje para las clases de la aplicación de reservas. 2 puntos.

4. Documento explicativo de todo el proceso (en formato .pdf e incorporando pantallazos). 2 puntos.

Indicaciones de entrega.

Una vez realizada la tarea debes entregar:

- Un directorio llamado **léeme** donde incluyas el documento detallando el desarrollo que has seguido para la realización de la tarea (incluye pantallazos).
- Un directorio llamado **proyecto** con el proyecto completo, incluyendo las clases de tests, las cuales deben pasar, es decir, deben tener éxito.

Realiza un fichero comprimido con las siguientes pautas:

apellido1_apellido2_nombre_SIGxx_Tarea

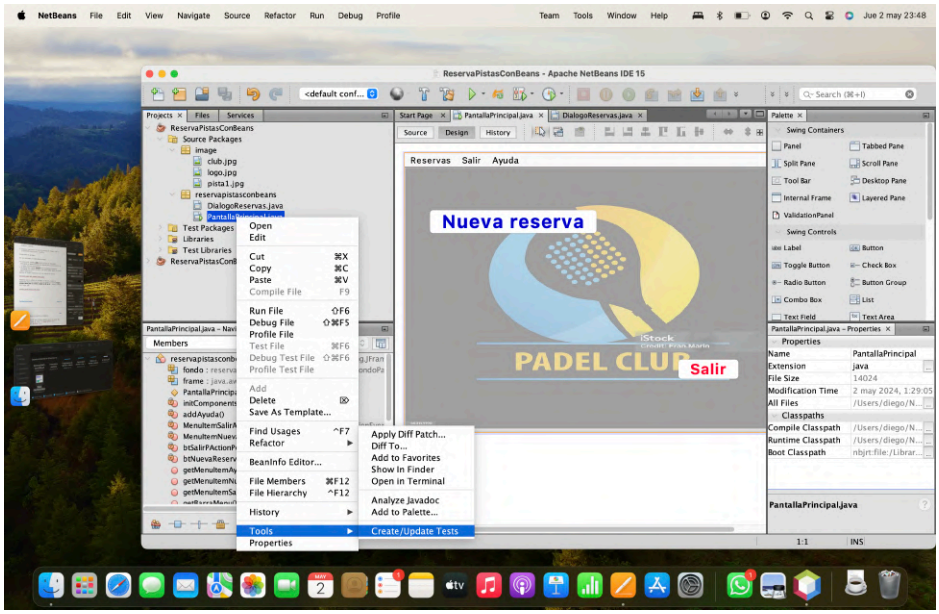
Asegúrate que el nombre no contenga la letra ñ, tildes ni caracteres especiales extraños. Así por ejemplo la alumna **Begoña Sánchez Mañas para la octava unidad del MP de DI** , debería nombrar esta tarea como…

sanchez_manas_begona_DI08_Tarea

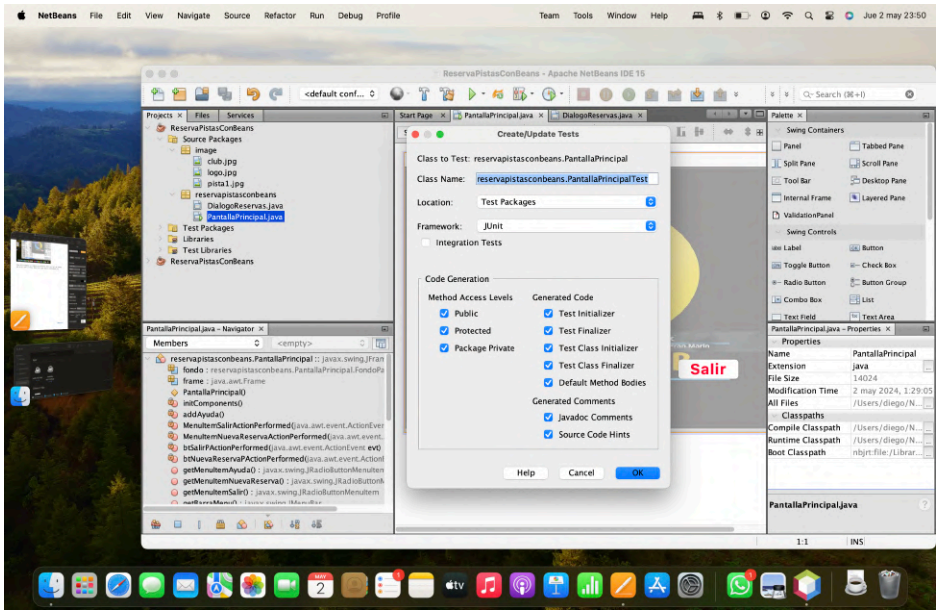
Empezaremos abriendo el proyecto en NetBeans.

Lo primero que haremos será crear los métodos get en las clase sobre la que queremos hacer los test, ya que los métodos de los botones son privados.

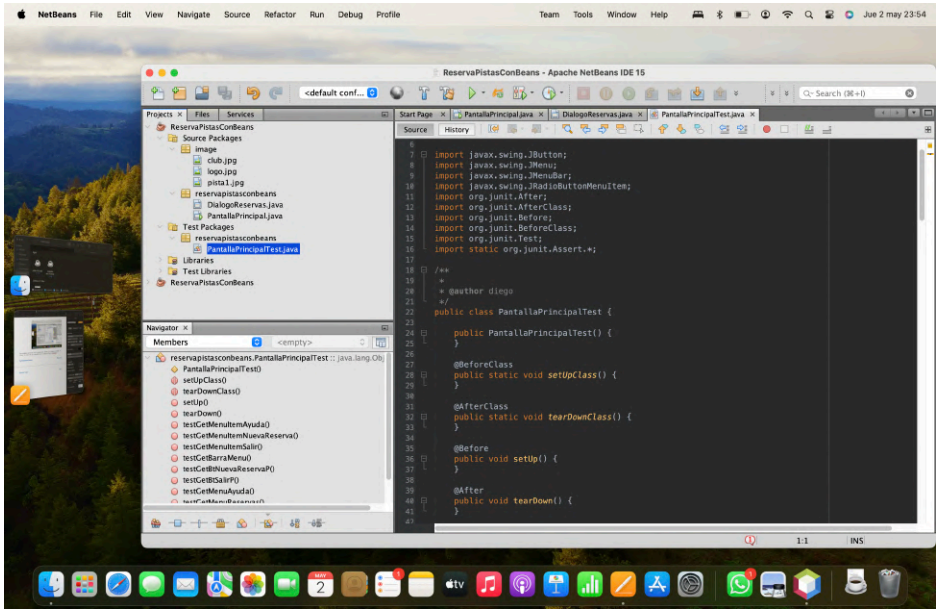
Tras ello haremos clic con el botón derecho sobre la clases que queremos testar, PantallaPrincipal en mi caso, luego en tools ==> create/update test.



Tras ello elegimos el nombre que queremos darle al test de prueba, que en mi caso lo voy a dejar tal como está y confirmamos ok.



Como podemos apreciar en la siguiente imagen ya tenemos creada la clase, la cual nos ha guardado en el paquete Test Packages.

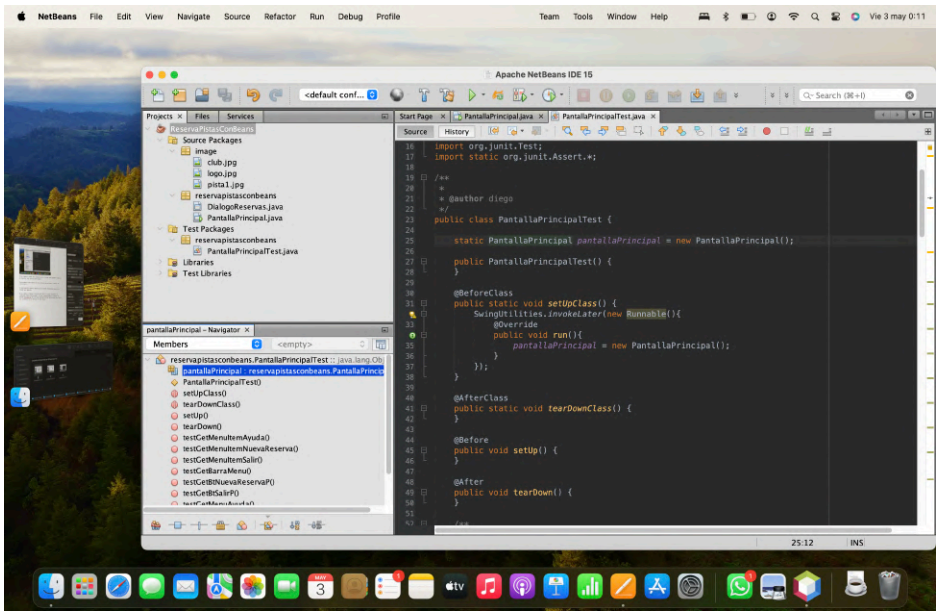


En el vamos a crear una instancia de la clase que queremos testar, en nuestro caso PantallaPrincipal.

Para poderla usar dentro de los métodos necesitamos crearla de tipo static.

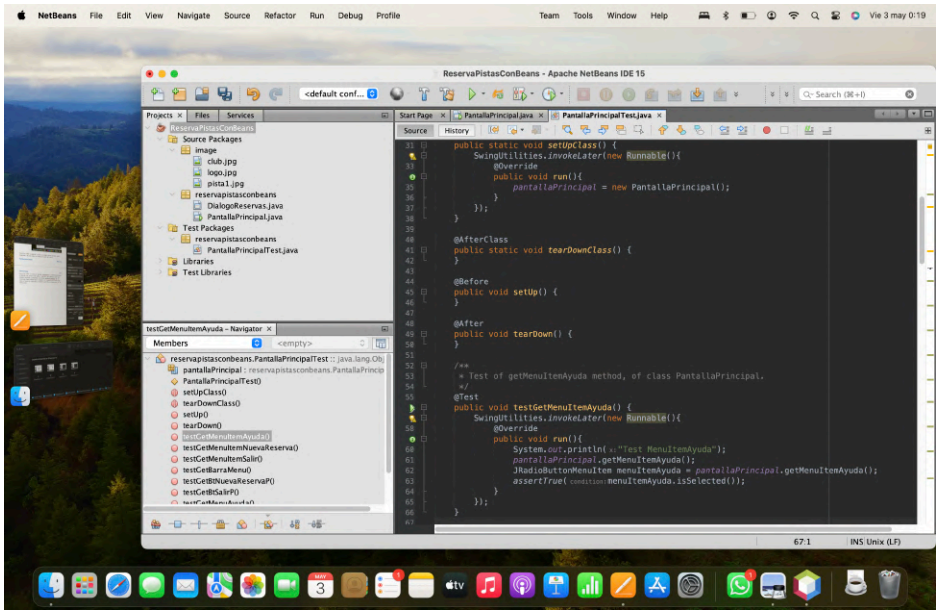
A continuación vamos a modificar el método setUpClass que se ha creado por defecto.

En el vamos a simular la ejecución del hilo para ejecutar la interface PantallaPrincipal, lo cual conseguiremos llamando al método invokeLater de la clase SwingUtilities declarando para ello una instancia en el método run del mismo.



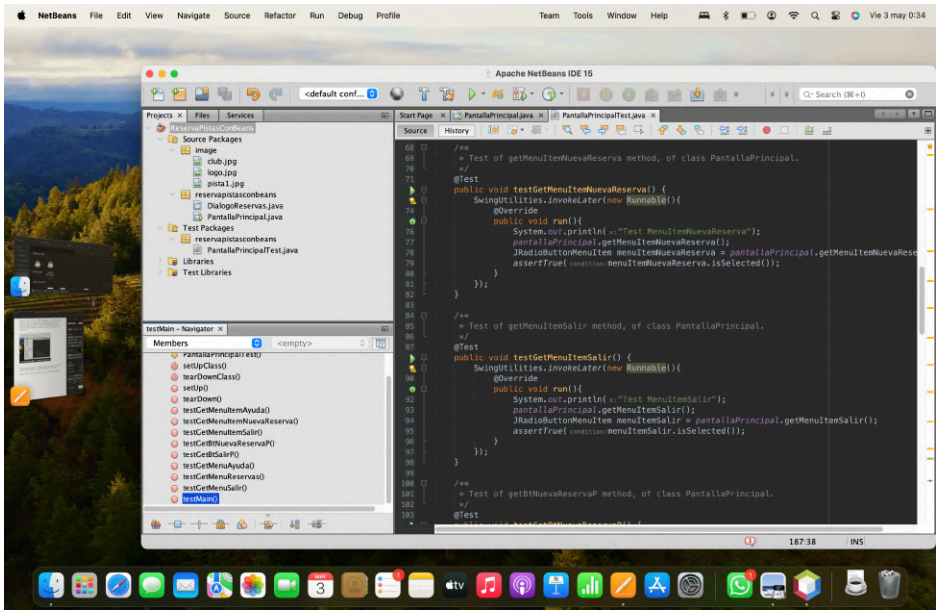
Tras ello vamos a empezar a crear el primer método de test para comprobar, en este caso, el botón de ayuda del menú.

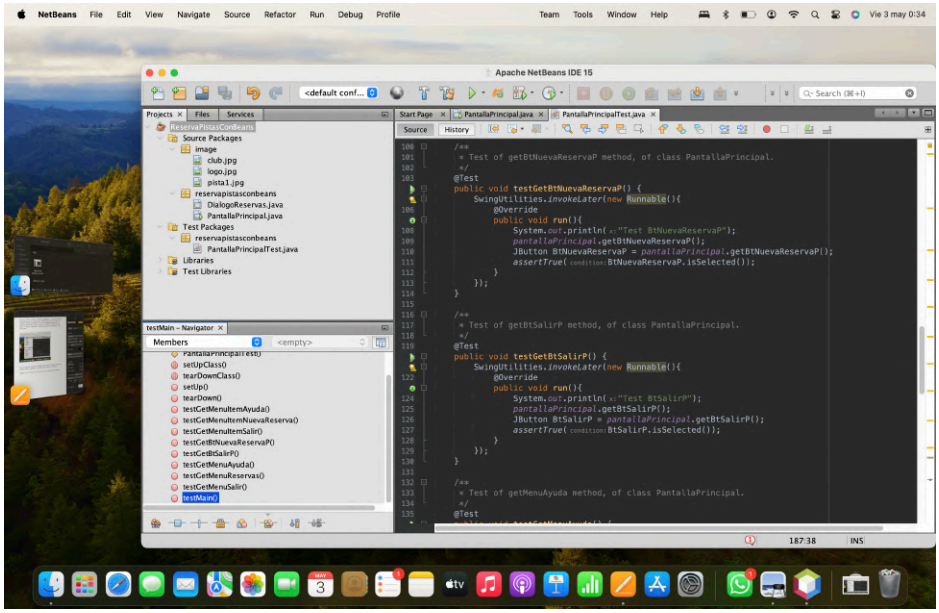
Para ello vamos a llamar al método invokeLater y sobreescribiremos el método run, en el cual vamos a imprimir por pantalla que test estamos haciendo, llamaremos ala método getMenuItemAyuda de la interface PantallaPrincipal, creamos una instancia del botón que vamos a testar y, mediante el método assertTrue comprobamos si se selecciona.

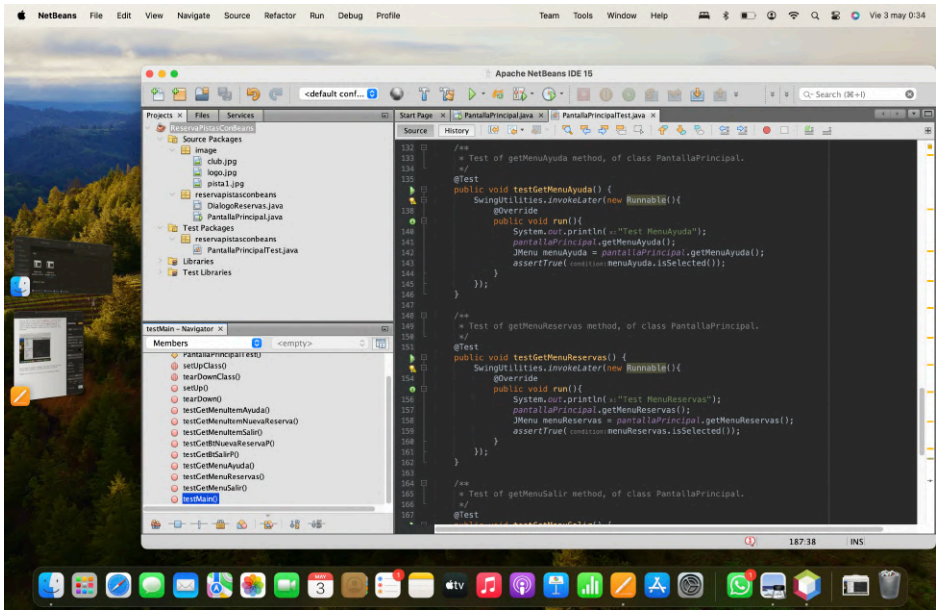


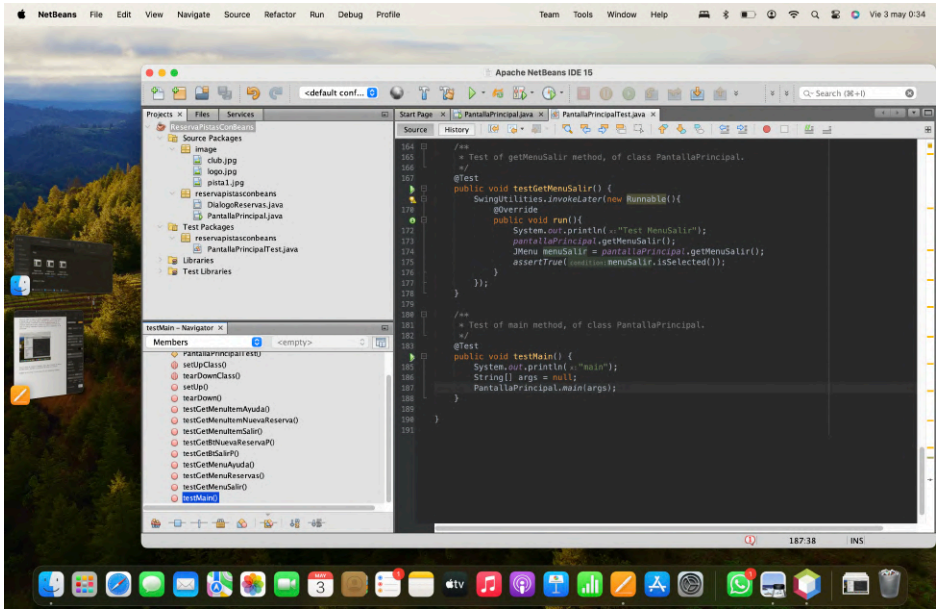
Para el resto de botones lo haremos igual, pero llamando al método de cada botón y haciendo la instancia del botón en cuestión.

Podemos ver el código de cada método en las siguientes imágenes.



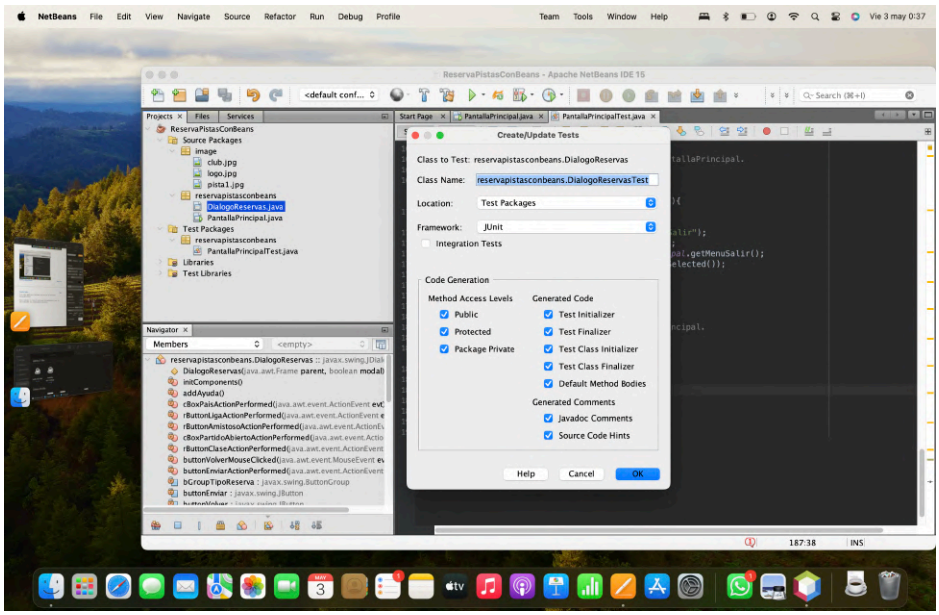




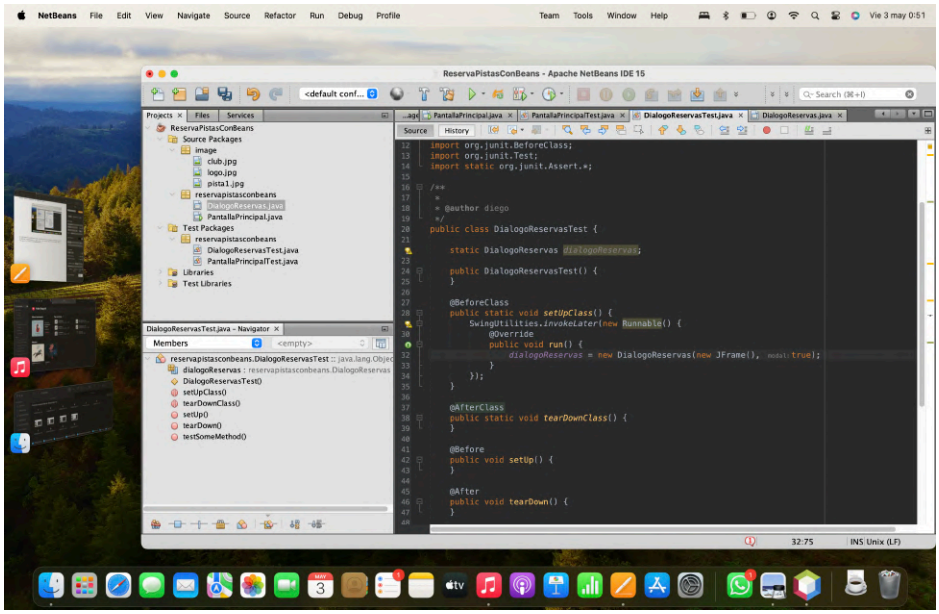


Y con esto damos por finalizado los test de la interface PantallaPrincipal.

Ahora crearemos la clase de test para DialogoReservas, para lo cuál seguiremos los mismos pasos anteriores (clic en botón derecho sobre la case que queremos crear el test ==> Tools ==> Create/ Update Tests, elegimos el nombre y ok).



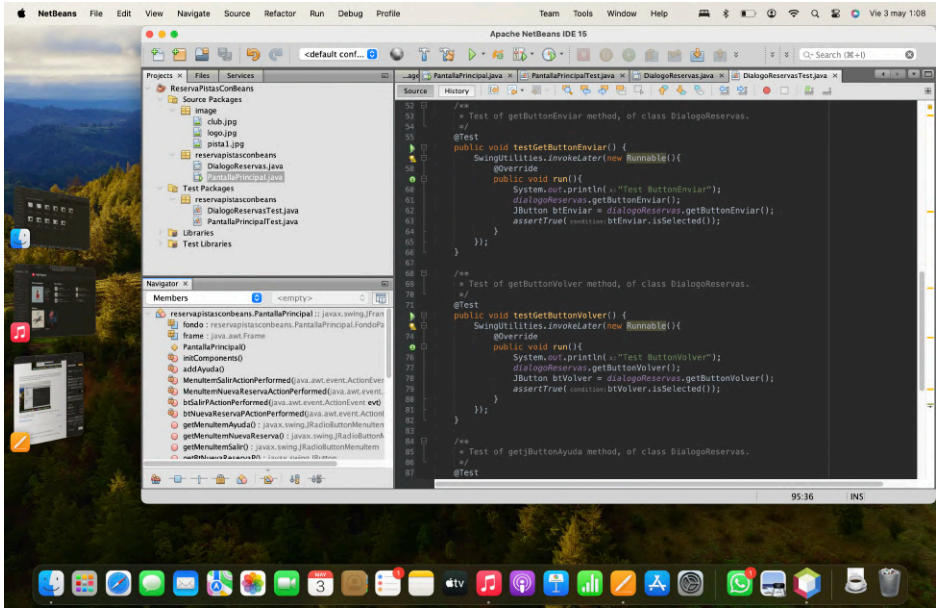
A continuación crearemos una instancia de la clase y modificaremos el método setUpClass al igual que los pasos anteriores pero esta vez para la clase DialogoReservas.

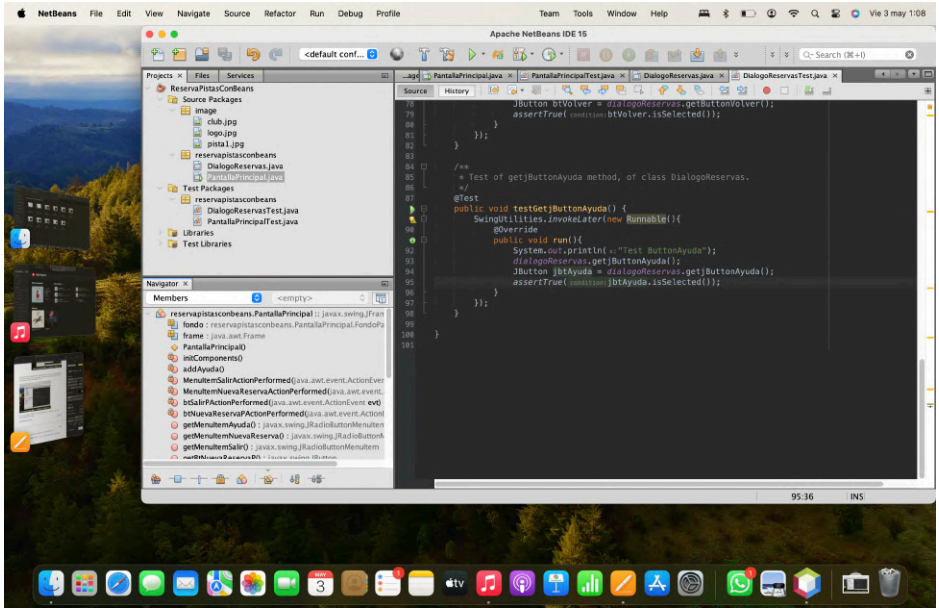


Tras ello vamos a crear los tres método para comprobar que los tres botones funcionan.

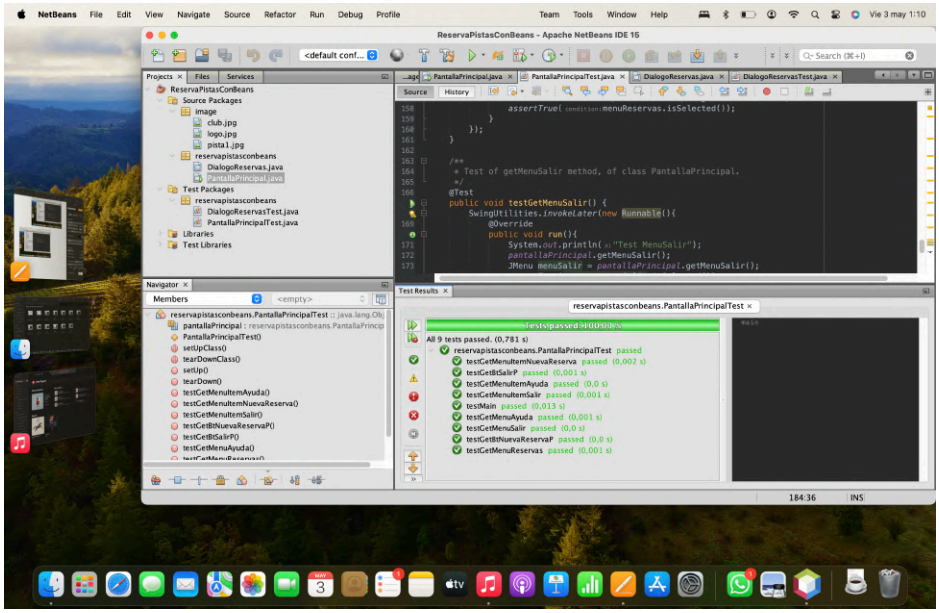
Esto lo haremos igual que que para la clase anterior.

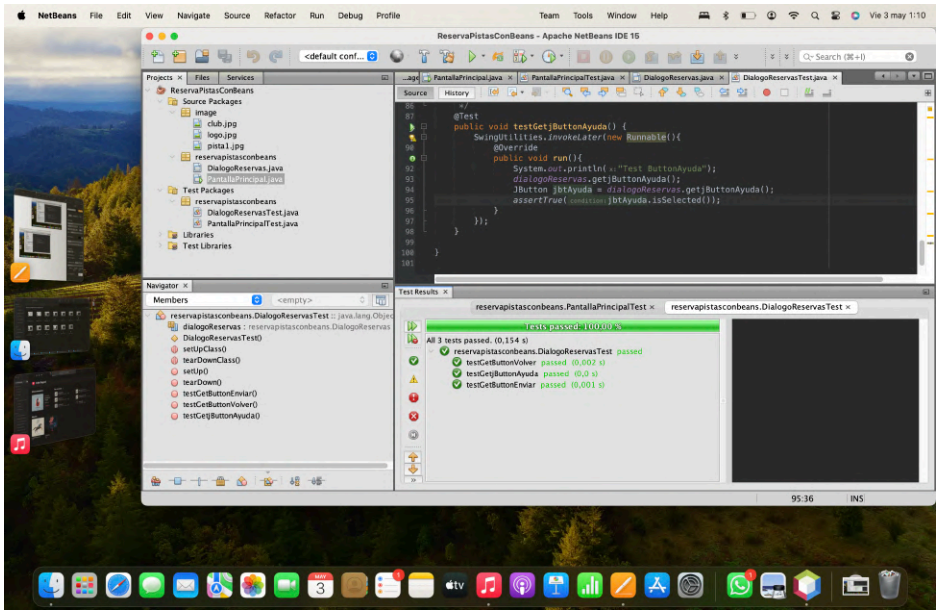
A continuación podemos ver el código de cada uno de los métodos.





En las siguientes imágenes podemos ver la salida de las dos clases de test donde puede verse todos los test válidos.



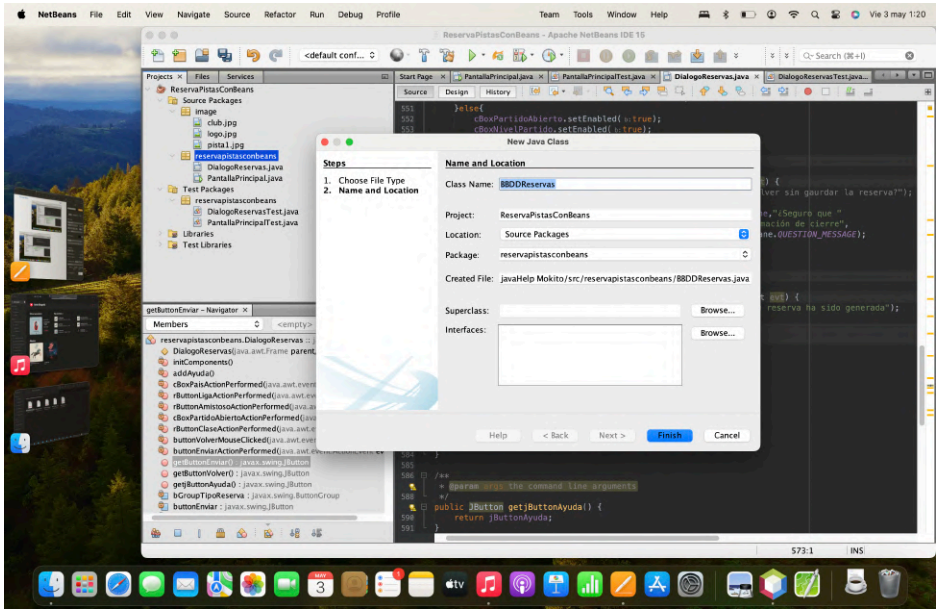


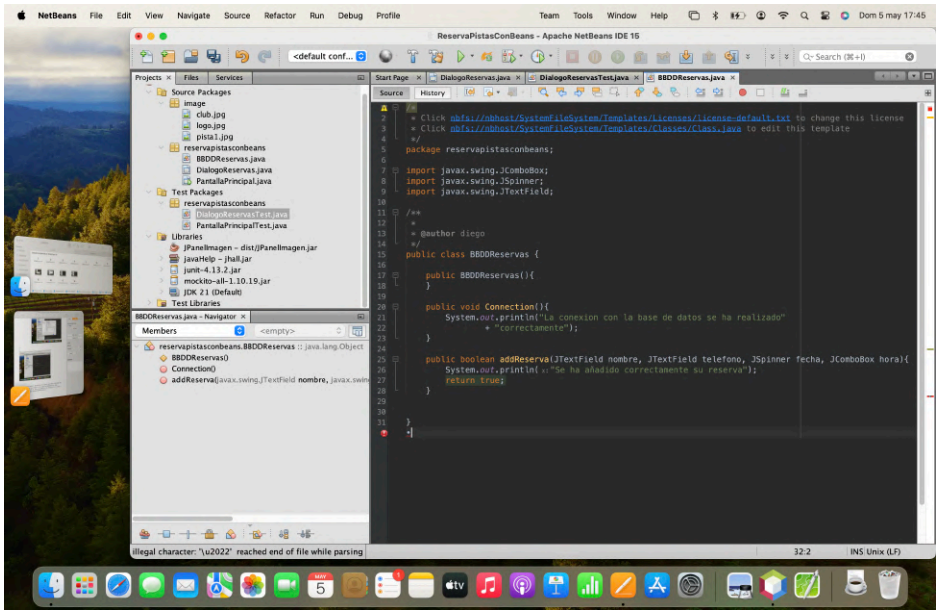
Ahora vamos a proceder a realizar la parte que corresponde a mockito.

Para ello lo primero que haremos será añadir la librería de mockito.

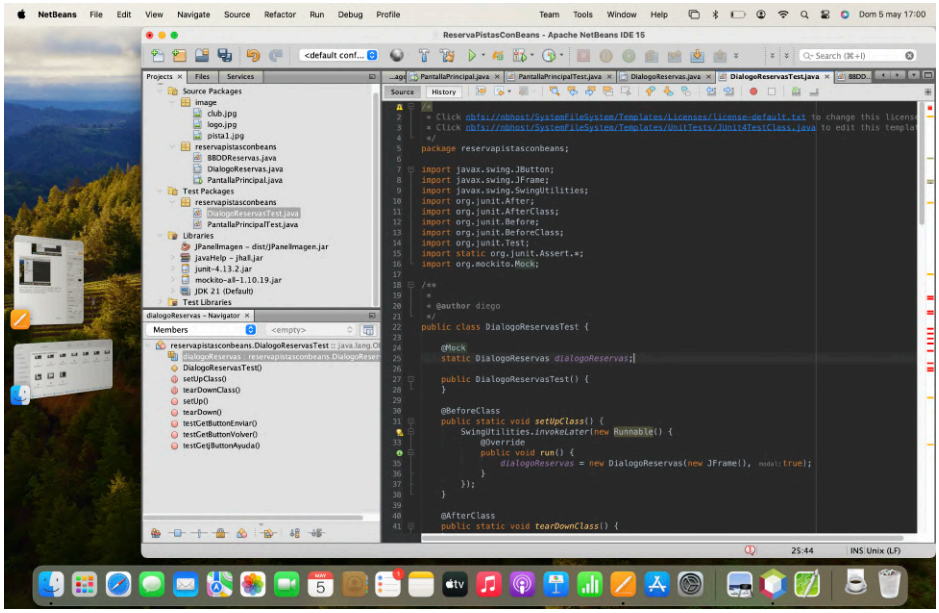
Tras ellos vamos a crear una nueva clase llamada BBDDReservas que contendrá dos método públicos, uno llamado connection y el otro addReserva.

Esto lo haremos para simular la conexión a la base de datos y el añadir una nueva reserva y con ello “probar” si la conexión y la actualización de datos se ha realizado correctamente.





A continuación vamos a encapsular el objeto que vamos a probar que para el caso que nos ocupa es la clase DialogoReservas, que haremos añadiéndole la anotación @Mock a la instancia e la clase DialogoReservas que hicimos anteriormente.



A continuación vamos a modificar el método que creamos del botón enviar para adaptarlo a la prueba que tenemos que hacer con mockito.

Para comprobar si realmente se hace la actualización de los datos a la base de datos debemos de tener rellenos los campos de nombre y teléfono, por lo que debemos acceder a los campos correspondiente.

Para poder acceder a los datos que ha introducido el cliente debemos crear los métodos get de dichos campos (nombre, teléfono, prefijo, fecha de la reserva y hora de la misma, en este caso).

Tras ello vamos a acceder a los campos correspondientes de la clase DialogoReservas y vamos a instanciar todos los campos que vamos a comprobar.

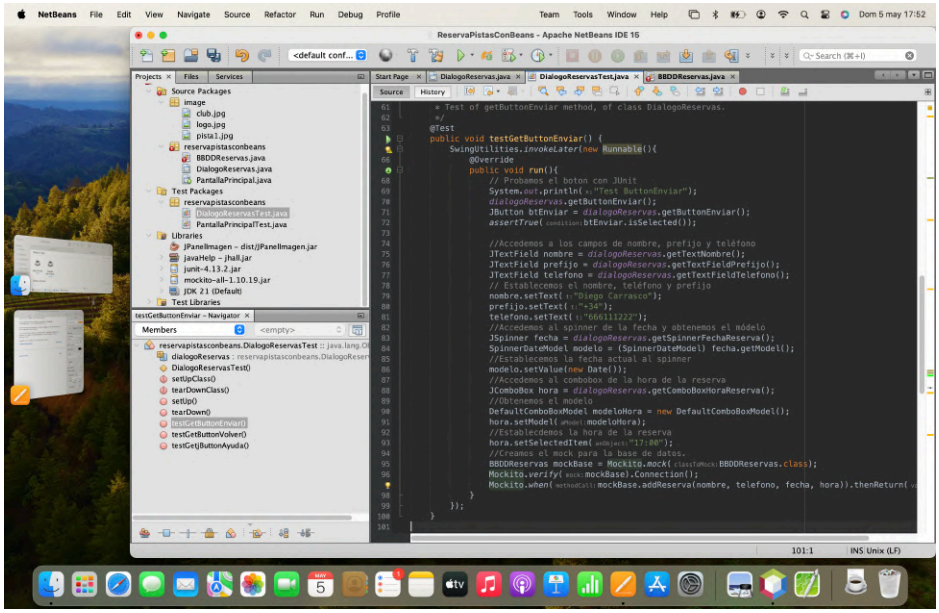
Como hemos podido ver anteriormente hemos creado un método Connection, que simula la conexión a la base de datos y un método addReserva que simula la actualización de la base de datos con los datos de los campos que le hemos pasado.

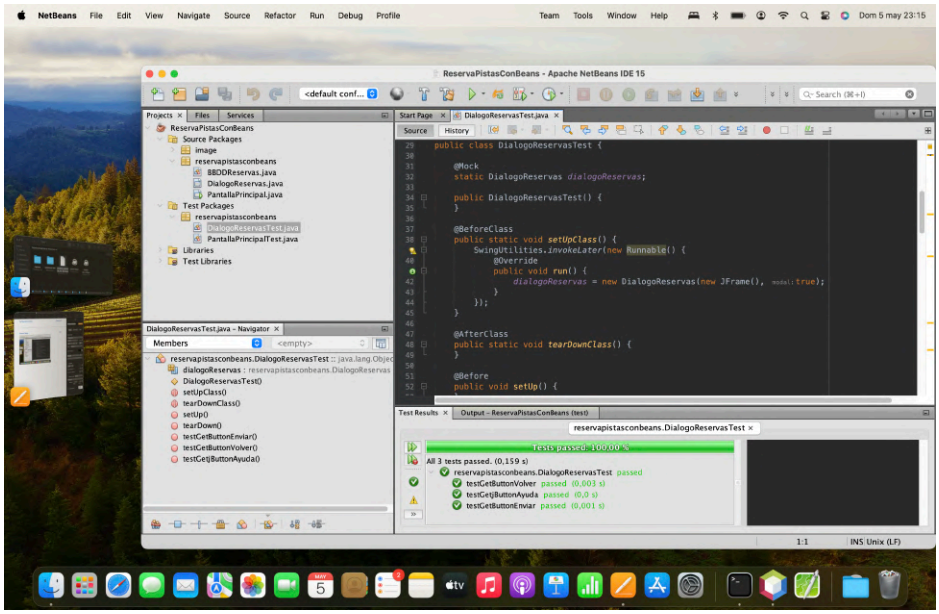
Para comprobar dichos métodos crearemos el mockBase correspondiente a la clase BBDDReservas.

Posteriormente aplicaremos el método verify al método Connection para comprobar dicho método

Y por último aplicaremos el método when al método addReserva, le paremos los campos correspondientes y verificaremos si nos devuelve true.

Podemos ver el código en la siguiente imagen.





La parte de comprobar el porcentaje de código testado me ha costado sudores hacerla, ya que apenas he encontrado información en internet.

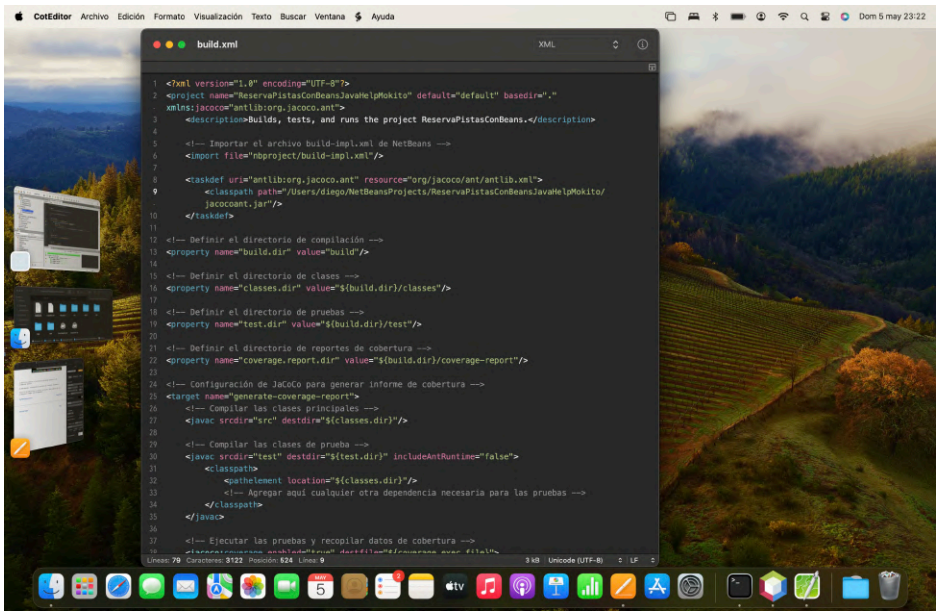
He descargado jacoco desde su página oficial (jacoco.org).

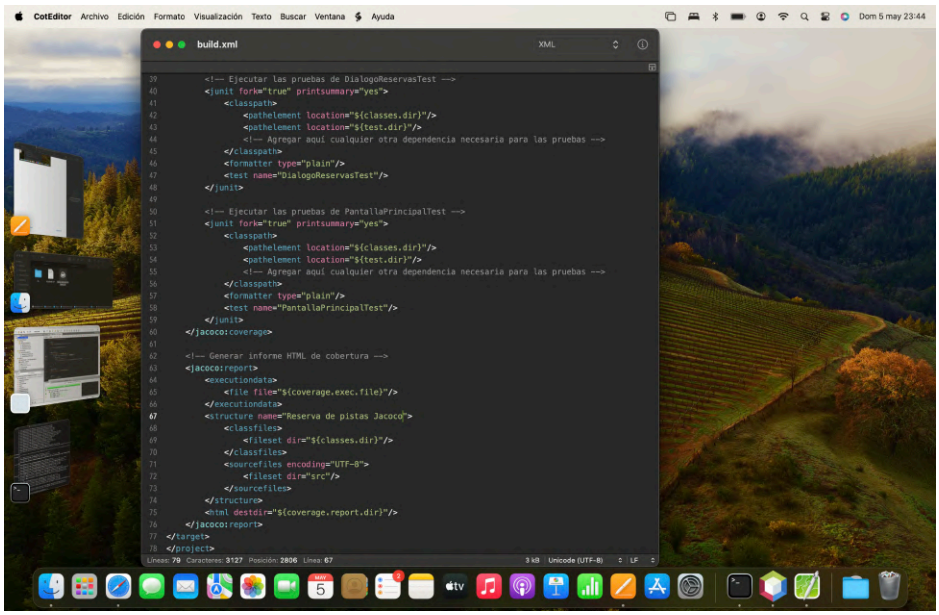
Dentro del archivo me venían los jar que necesitamos para comprobar la cobertura, que podemos ver en la siguiente imagen.

Estos jar los vamos a incluir como librerías en el proyecto.

Lo siguiente que haremos será modificar el archivo build.xml.

El código de dicha clase podemos verlo en las siguientes imágenes.



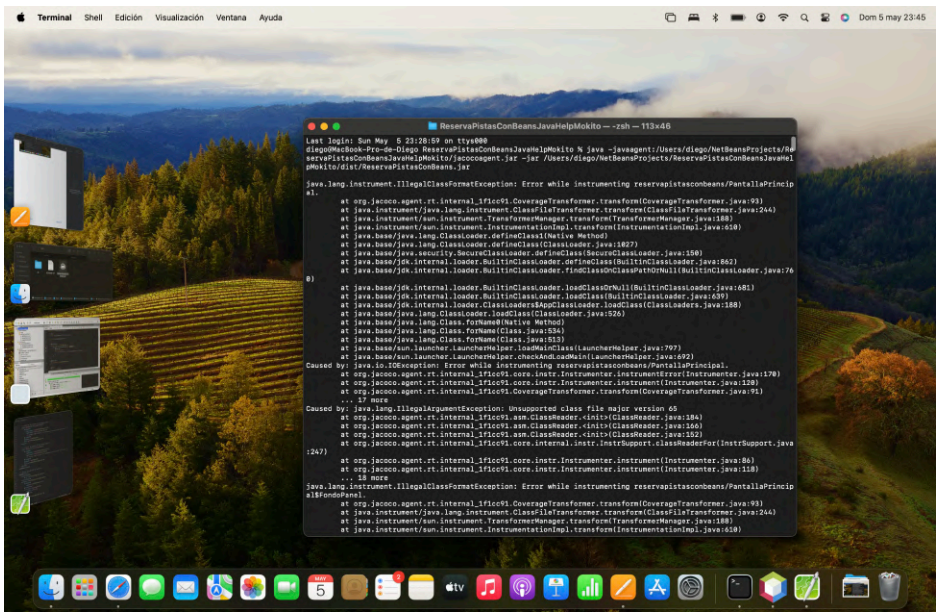


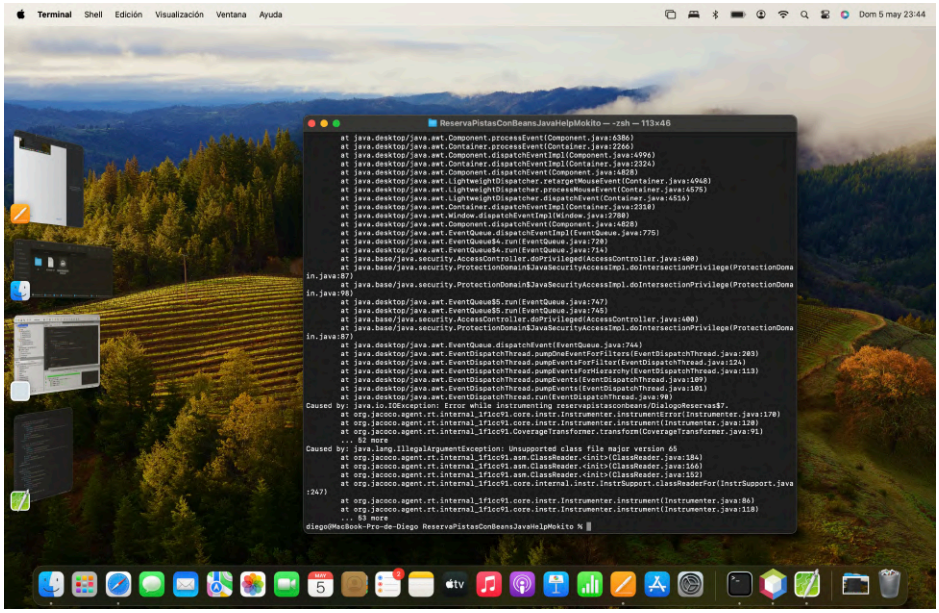
Tras ello vamos ejecutar nuestro programa con el agente de jacoco.

Para ello lanzamos el comando:

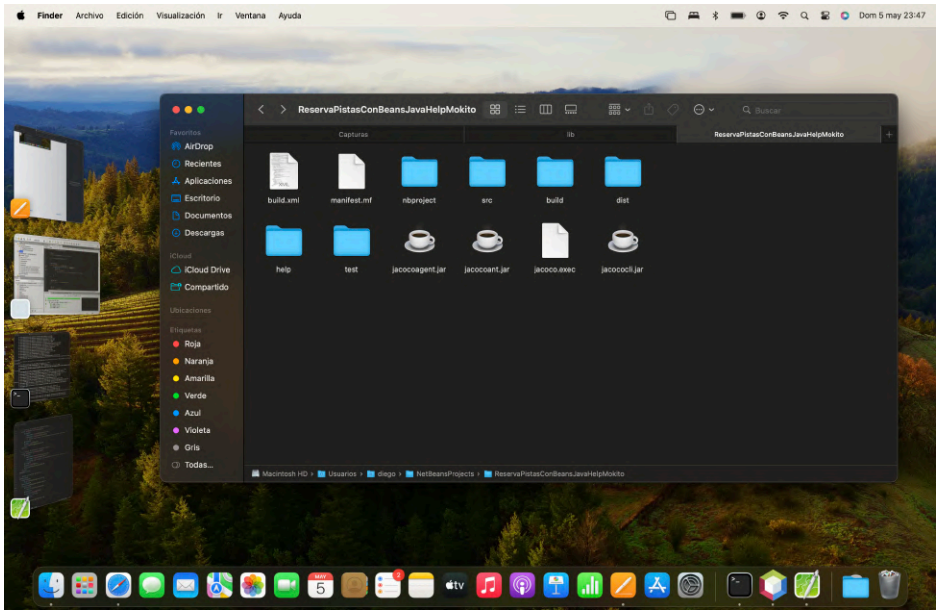
java -javaagent:/Users/diego/NetBeansProjects/ReservaPistasConBeansJavaHelpMokito/jacocoagent.jar -jar /Users/diego/NetBeansProjects/ReservaPistasConBeansJavaHelpMokito/dist/ReservaPistasConBeans.

En las siguiente imágenes podemos ver el comando ejecutado.





Tras ello podemos ver como nos ha guardado en el directorio de nuestro proyecto el archivo generado jacoco.exec.

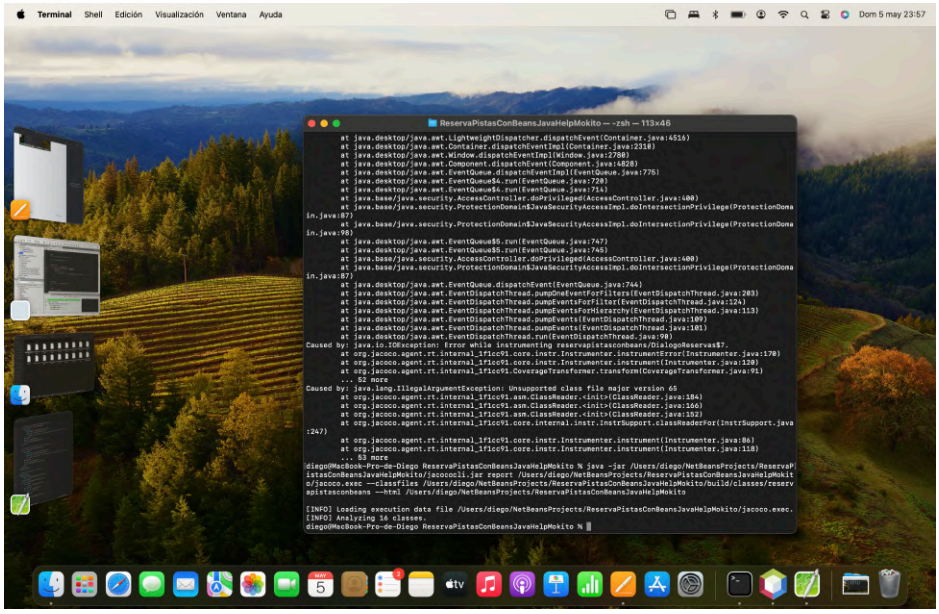


Posteriormente vamos a ejecutar el comando:

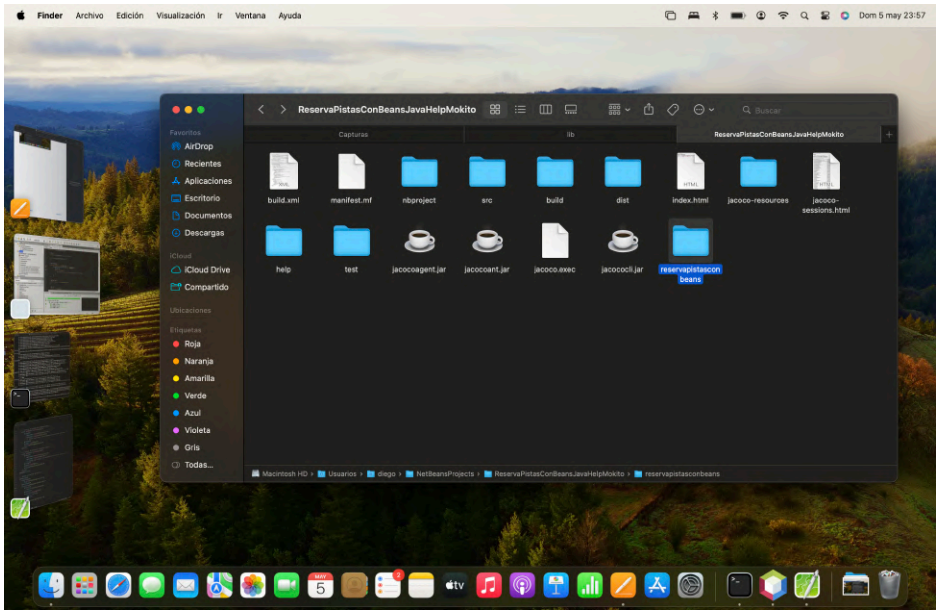
java -jar /Users/diego/NetBeansProjects/ReservaP istasConBeansJavaHelpMokito/jacococli.jar report /Users/Diego/ NetBeansProjects/ReservaPistasConBeansJavaHelpMokito/jacoco. exec --classfiles /Users/diego/NetBeansProjects/ ReservaPistasConBeansJavaHelpMokito/build/classes/ reservapistasconbeans –html /Users/Diego/NetBeansProjects/ ReservaPistasConBeansJavaHelpMokito.

Este comando lo que ha es usar el jar jacoco.cli para generar el informe de cobertura del código en formato HTML a partir del archivo jacoco.exec que ha generado anteriormente.

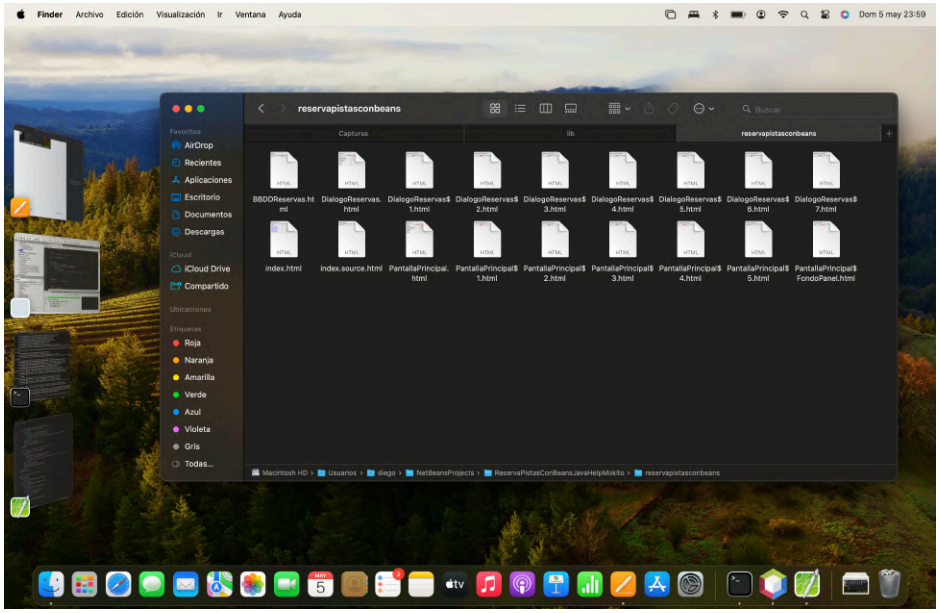
Es decir, este comando ejecuta la herramienta JaCoCo CLI para generar un informe de cobertura de código en formato HTML utilizando los datos del archivo jacoco.exec y los archivos de clase ubicados en /Users/diego/NetBeansProjects/ReservaPistasConBeansJavaHelpMokito/build/classes/reservapistasconbeans. El informe generado se guardará en /Users/Diego/NetBeansProjects/ReservaPistasConBeansJavaHelpMokito en formato HTML.



En la siguiente imagen podemos ver la carpeta que se ha creado, llamada reservaspistasconbeans.



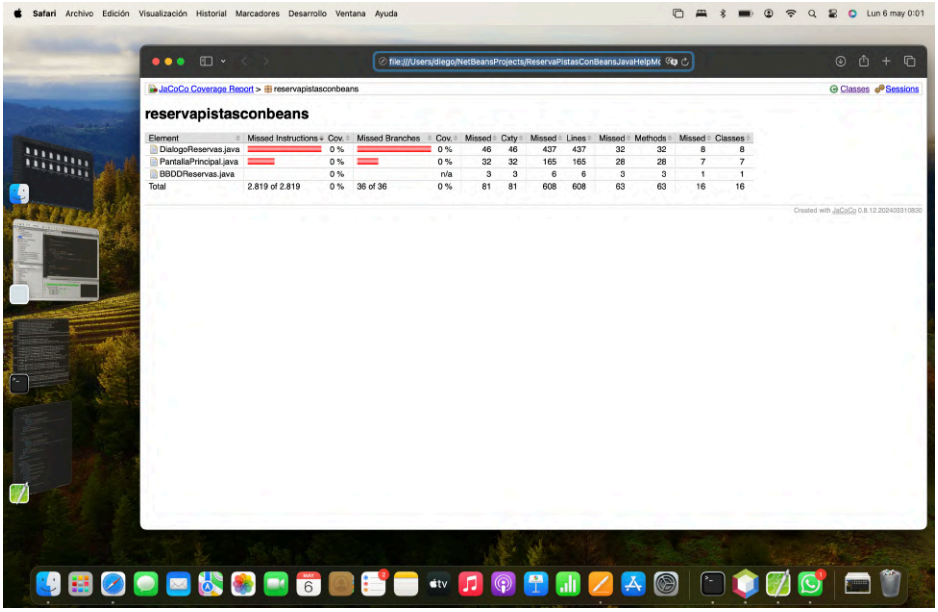
Y el contenido de dicha carpeta es el que se puede apreciar en la siguiente imagen.



Cada uno de esos archivos html es el informe de cobertura correspondiente a cada una de clases guardadas en nuestro proyecto.

En la siguiente imagen podemos ver como tengo abierto el archivo índex.source.html.

En el se puede apreciar que el porcentaje de código testado es 0%, esto ya no he podido solucionarlo, los demás errores (espacio de nombres, configuración del archivo build.xml, los comando que ejecutar etc) si los he conseguido solucionar.



Todos los demás archivos xml van adjunto en el proyecto para que puedan ser comprobados.

Con esto damos por finalizada la tarea.