

JUNIT PRACTICA



PABLO ARANCIBIA

Índice

Practica (Repaso)

- 2.1-¿Cuál es la estructura que debe tener las pruebas?
- 2.2-¿Cómo es la sintaxis de JUnit?
 - 2.2.1-Assertions
- 2.3- Recomendaciones generales y buenas practicas
- 2.4-¿Qué criterio utilizar para elegir los métodos a testear?
- 2.5-¿Para que sirve ignorar algunas pruebas?
- 2.6- Ejercicio practico introductorio



Estructura generales de las pruebas:

- 1) Setup / Configuración (@Before, @BeforeClass)
- 2) Ejecución (@Test)
- 3) Verificación (Assertions)
- 4) Limpieza (@After, @AfterClass)

Sintaxis de las pruebas

Las anotaciones que nosotros tenemos en JUnit son:

@Test: Con esto indicamos que es una prueba

@Before: Se utiliza para marcar un metodo que se ejecutará antes de cada prueba

@After: se usa para marcar que un metodo se ejecuta despues de cada prueba

@BeforeClass: Se utiliza para marcar que ese metodo se ejecuta antes que todas las pruebas

@AfterClass: Se utiliza para marcar un método que se ejecutará una vez después de todas las pruebas en la clase

@Ignore: Indicamos que esta prueba tiene que ignorarse y no se debe ejecutar.

¿Qué son los assertions?

- Son sentencias en un test, que verifican si una condición es verdadera. Sirve para realizar comprobaciones y asegurarse que ciertas suposiciones sean validas.

Es importante que al realizar las pruebas unitarias utilicemos las assertions correctas al metodo que estamos probando.

Por ejemplo: Si estamos sumando $2+2$, deberíamos comprobar que el resultado sea 4 y no que sea nulo.

¿Cuales son los diferentes tipos de assertions?

- `assertEquals(expected, actual)`: Compara si el valor `expected` es igual al valor actual.
- `assertTrue(condition)`: Verifica si una condición dada es verdadera.
- `assertFalse(condition)`: Verifica si una condición dada es falsa.
- `assertNull(object)`: Verifica si un objeto dado es nulo.
- `assertNotNull(object)`: Verifica si un objeto dado no es nulo.
- `assertSame(expected, actual)`: Verifica si el valor `expected` es el mismo objeto que el valor actual.
- `assertNotSame(unexpected, actual)`: Verifica si el valor `unexpected` no es el mismo objeto que el valor actual.

Recomendaciones generales y buenas practicas

- Nombres descriptivos: Que hará el test.
- Separación de preocupaciones: Simplificarlo y separar cada test.
- Cobertura de código: Testear casos límite y casos de error.
- Aserciones claras: Utilizar segun corresponda
- Datos de prueba independientes: No depender de datos externos
- Minimizar las dependencias externas: Utilizar herramientas para simular objetos si son necesarios
- Ejecución rápida y aislada: No deben ser pesadas para ejecutarse ya que se deben repetir varias veces en el desarrollo.
- Mantenimiento de las pruebas: Mantén tus pruebas actualizadas y relevantes.

¿Cuál es el criterio que tenemos que tener para elegir que pruebas realizar?

Como recomendación general, para elegir las pruebas que debemos realizar es tener en cuenta las funcionalidades críticas o principales que generen mayor impacto en tu aplicación.

Ej Calculadora:

Es mas importante revisar que la calculadora sume correctamente, a que el mensaje que muestre como respuesta se vea bien.

¿Para que sirve ignorar las pruebas?

Normalmente podemos ignorar las pruebas convenientemente en dos casos particulares:

- Cuando la función no está implementada completamente (no está terminada la funcionalidad)
- Cuando el programador no comprende completamente que es lo que debería realizar la función. (falta de comprensión del negocio)

Preguntas frecuentes

1. ¿Cómo testear un método que no tiene retorno?
2. ¿Se deben testear los Servicios?
3. ¿Cuáles son las fallas más comunes que se cometen en JUnit?



¿Cómo Testear un método que no tiene retorno?

En el caso de que tengamos un método void, no esperamos que nos devuelva un resultado. Por lo que lo que debemos probar es el compartamiento, y verificar mediante los Asserts

¿Se deben testear los servicios?

Para saber si debemos testear los servicios, podemos responder a algunas de las siguientes preguntas;

- Los métodos que vamos a testear, ¿se encuentran dentro de los servicios?
- ¿Son los métodos de los servicios, importantes para el negocio? ¿Requieren prioridad?

¿Cuáles son las fallas más comunes al realizar test unitarios?

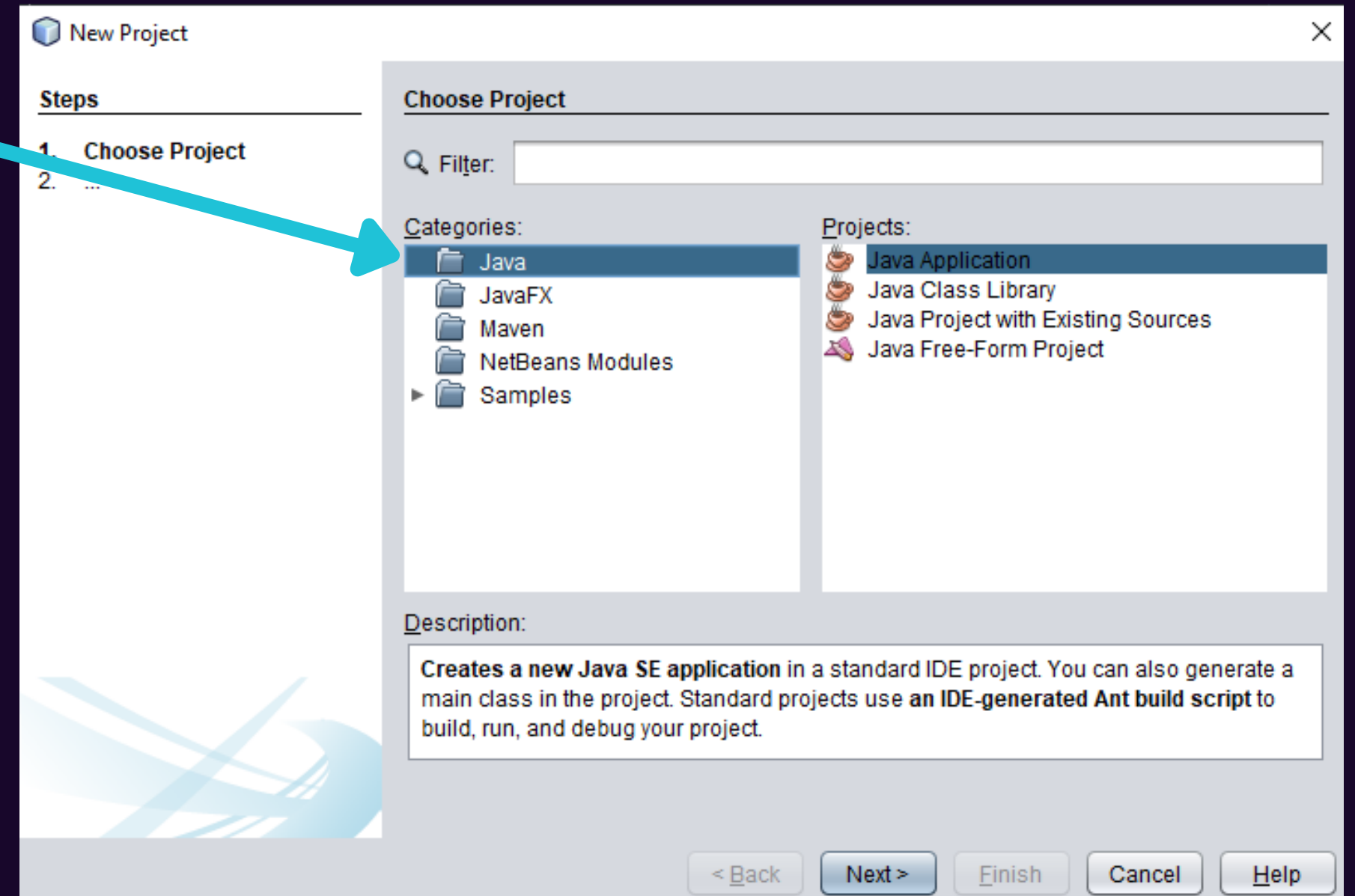
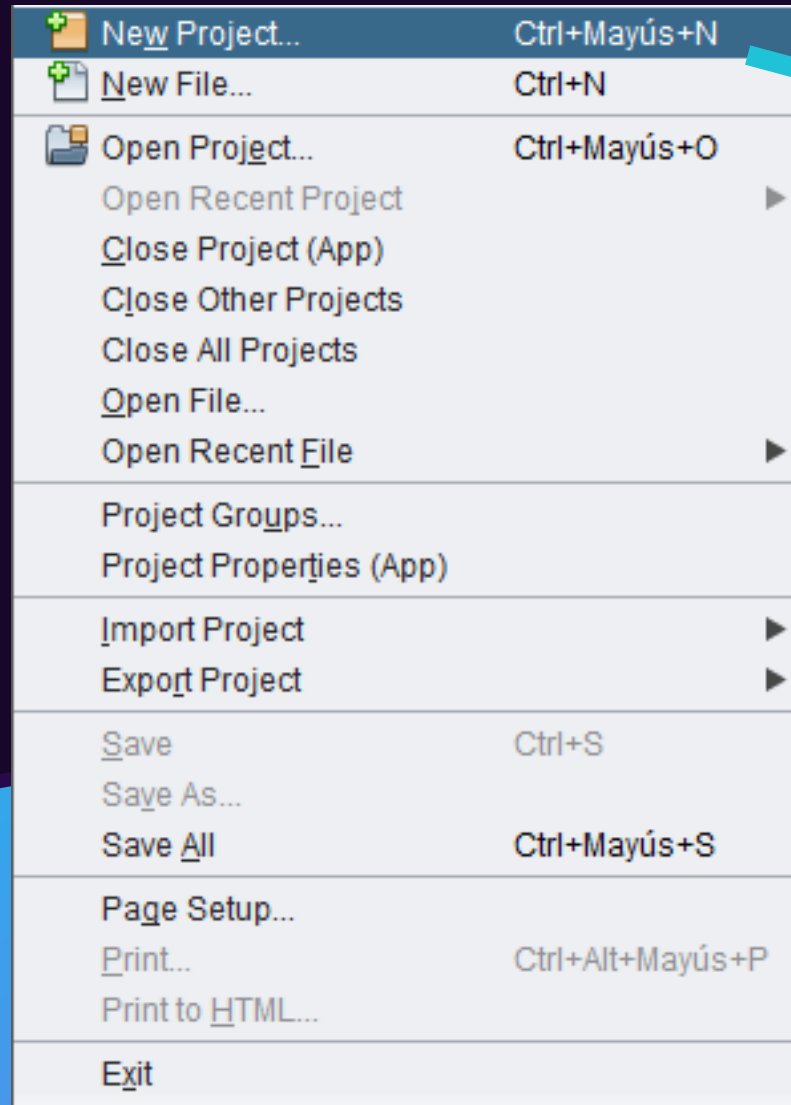
1. Pruebas incompletas
2. Pruebas dependientes
3. Falta de aislamiento
4. Usar erróneamente los Asserts
5. Falta de mantenimiento

Ejercicio Practico

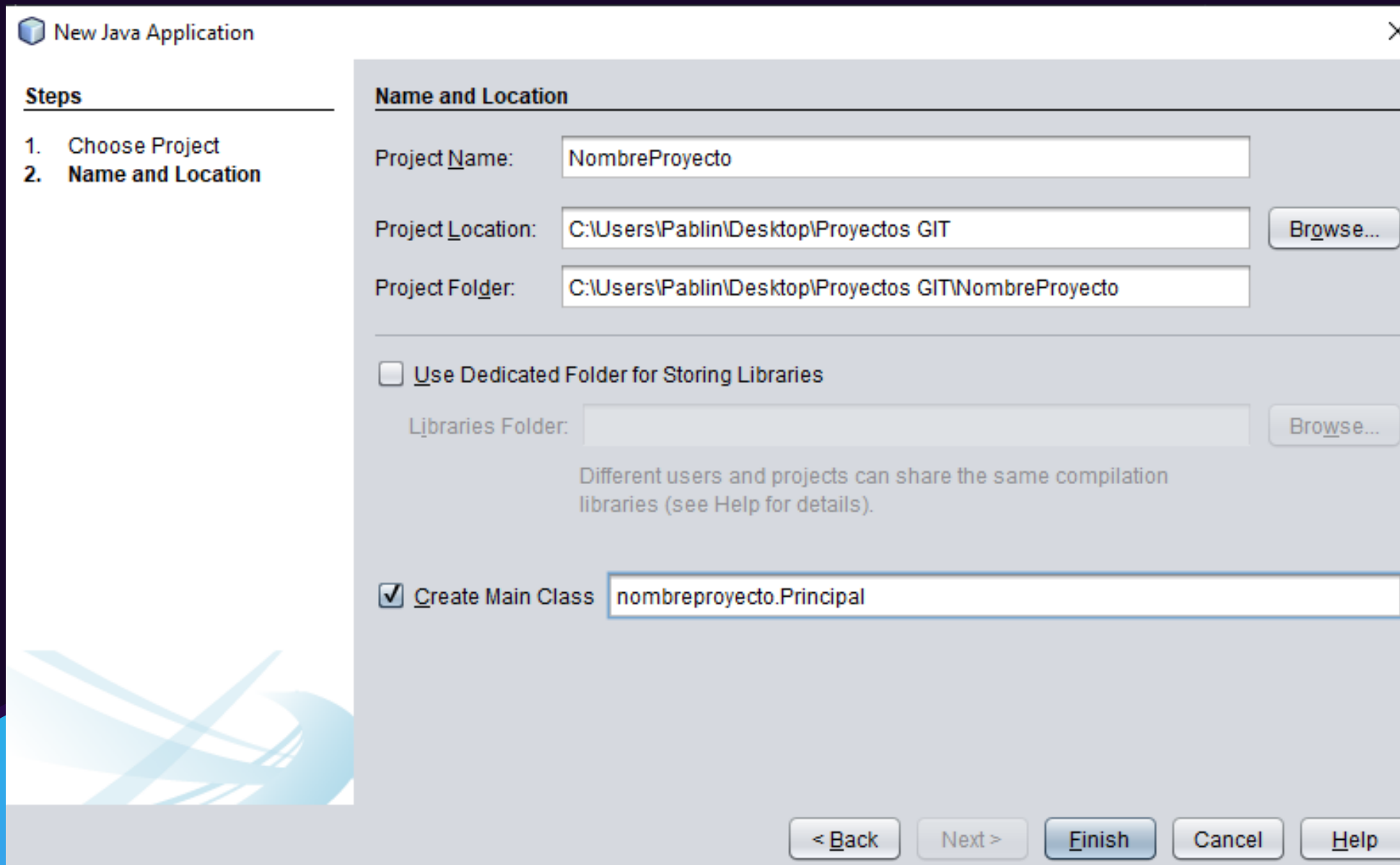
1. Librerías y dependencias
2. Configuración inicial (Before, BeforeClass)
3. Estructura de un Test (Test)
4. Verificaciones (Assertions)
5. Limpieza y liberación de recursos (After, AfterClass)
6. Testear un System.out
7. Ejemplo de ejercicio de Electrodomésticos
8. Testear un Scanner



Librerías y dependencias NetBeans 8.2



Librerías y dependencias NetBeans 8.2



New Java Application

Steps

1. Choose Project
2. Name and Location

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

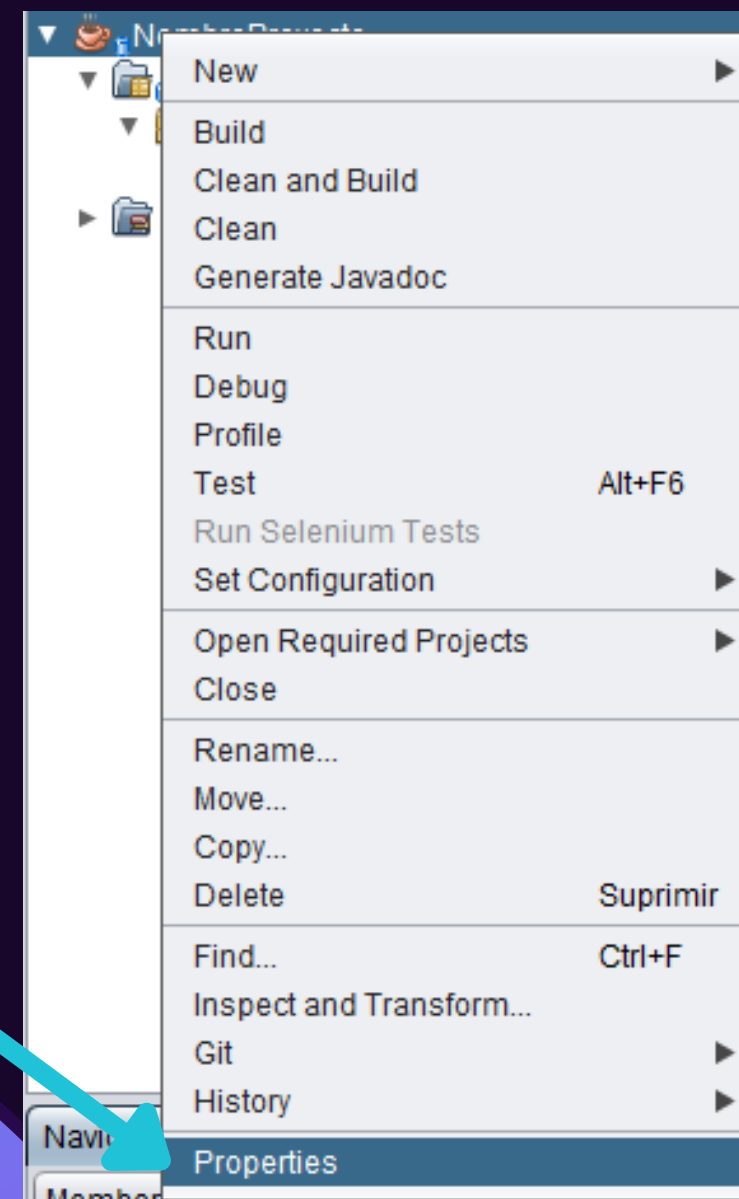
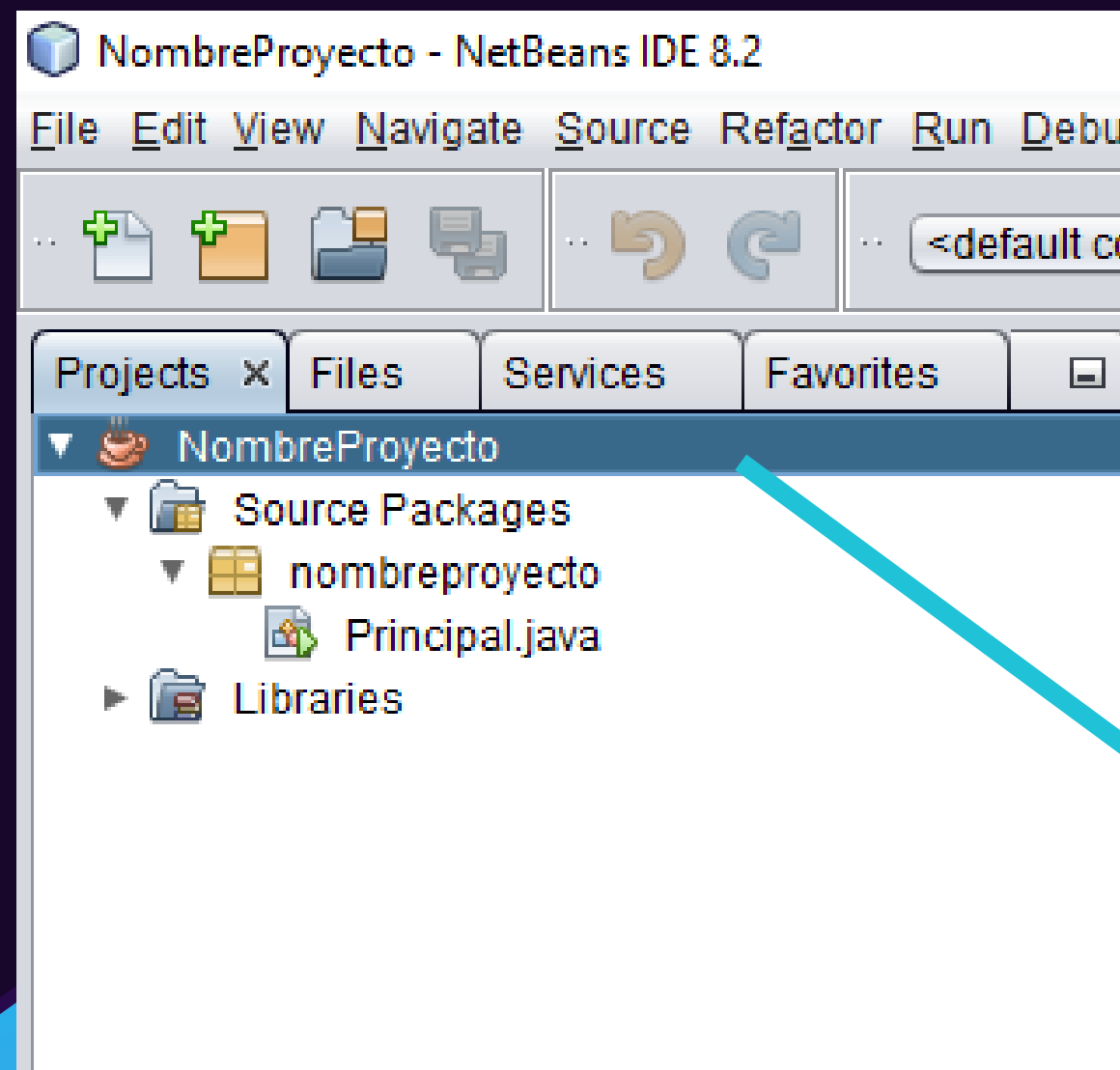
< Back Next > Finish Cancel Help

Project name: Nombre del proyecto

Project Location: Ubicacion del proyecto

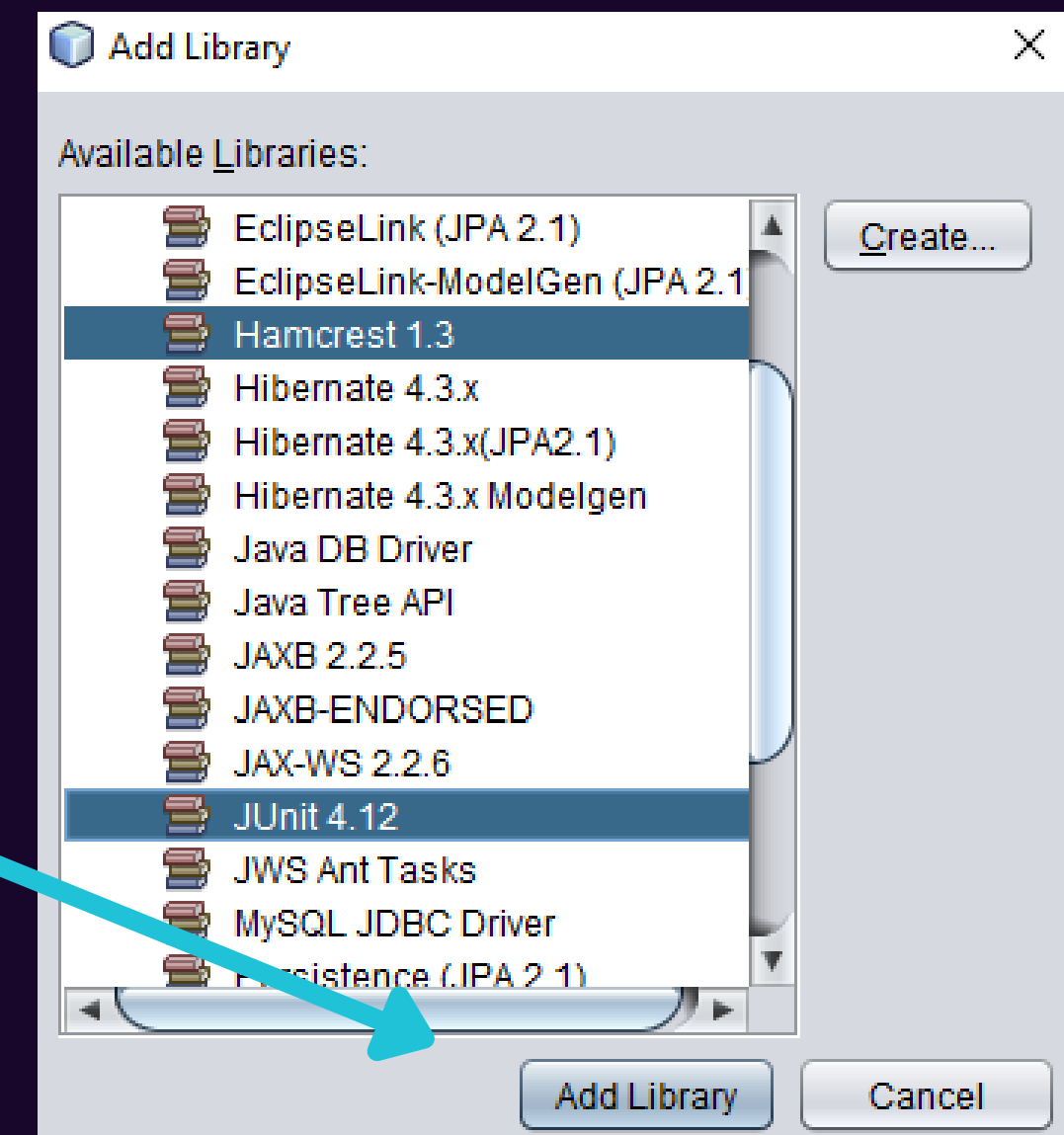
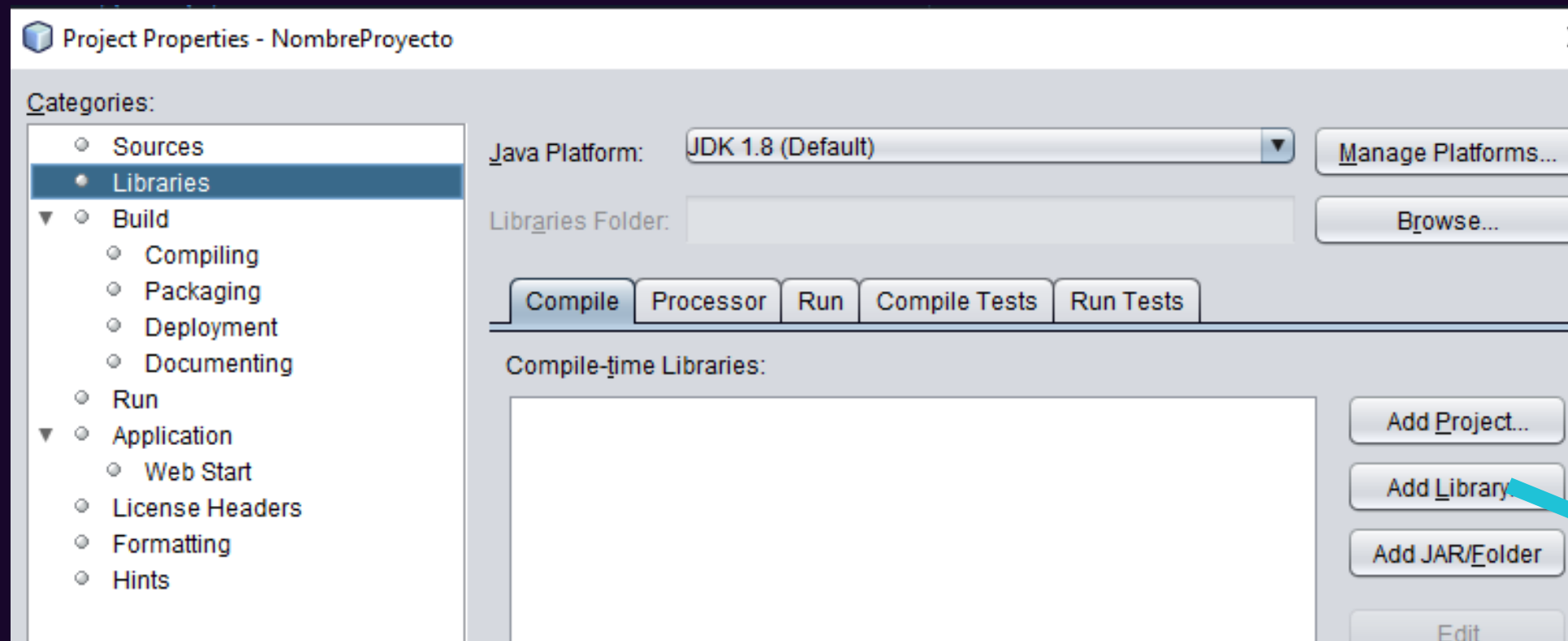
Project Folder: Carpeta donde estarán los archivos del proyecto

Librerías y dependencias NetBeans 8.2: Agregar JUnit

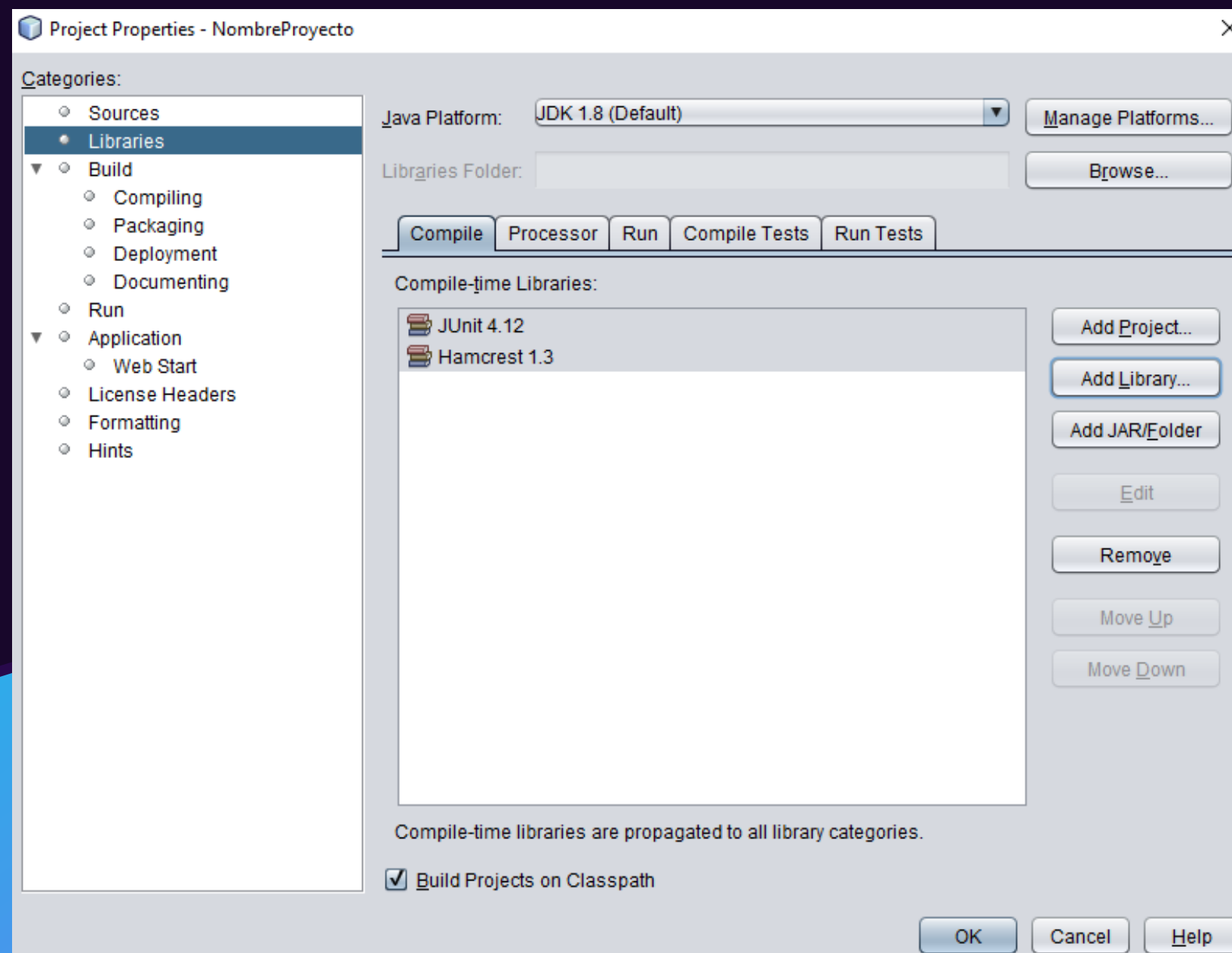


- Click derecho en el proyecto.
- Apretar en "Properties"
- Apretar donde dice "Add Library"
- Agregar Hamcrest 1.3 y JUnit 4.12
- Luego le damos a "Ok"

Librerías y dependencias NetBeans 8.2: Agregar JUnit

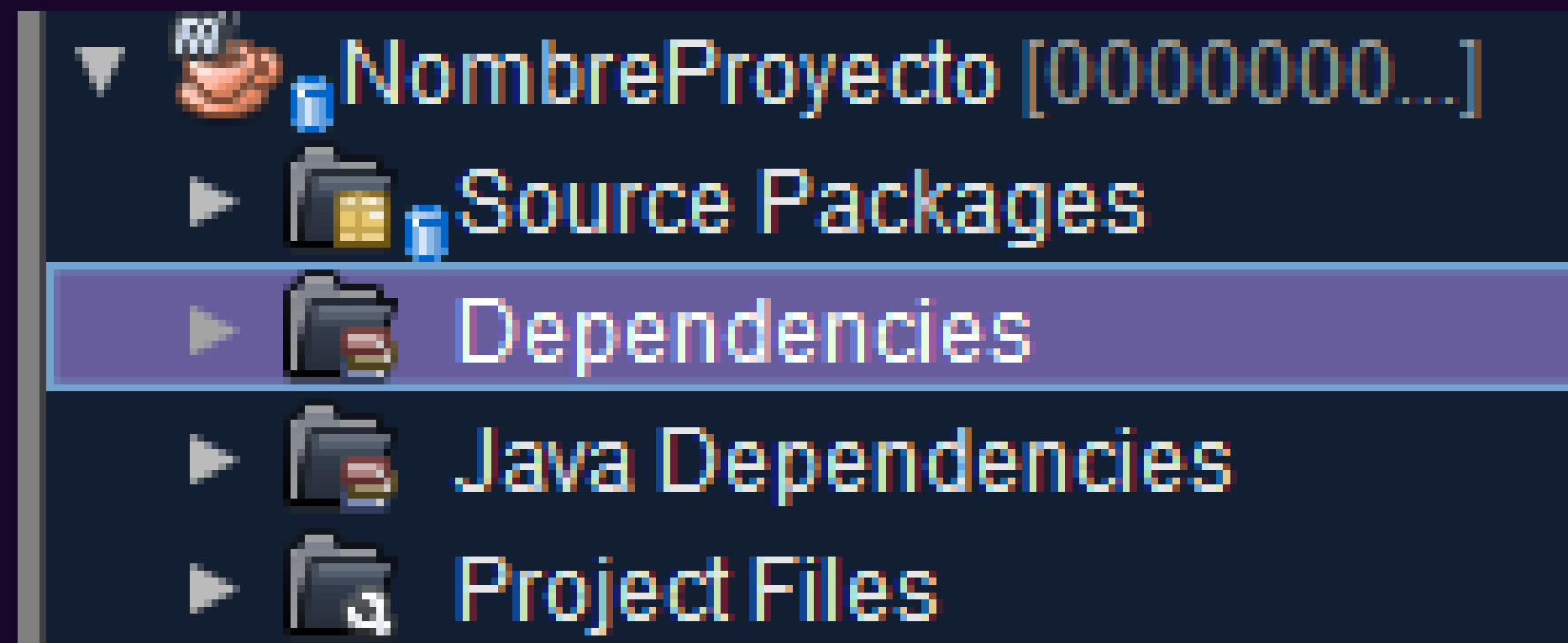


Librerías y dependencias NetBeans 8.2: Agregar JUnit



Una vez realizados los pasos, ya tendremos las librerías listas para poder hacer los test unitarios en NetBeans 8.2

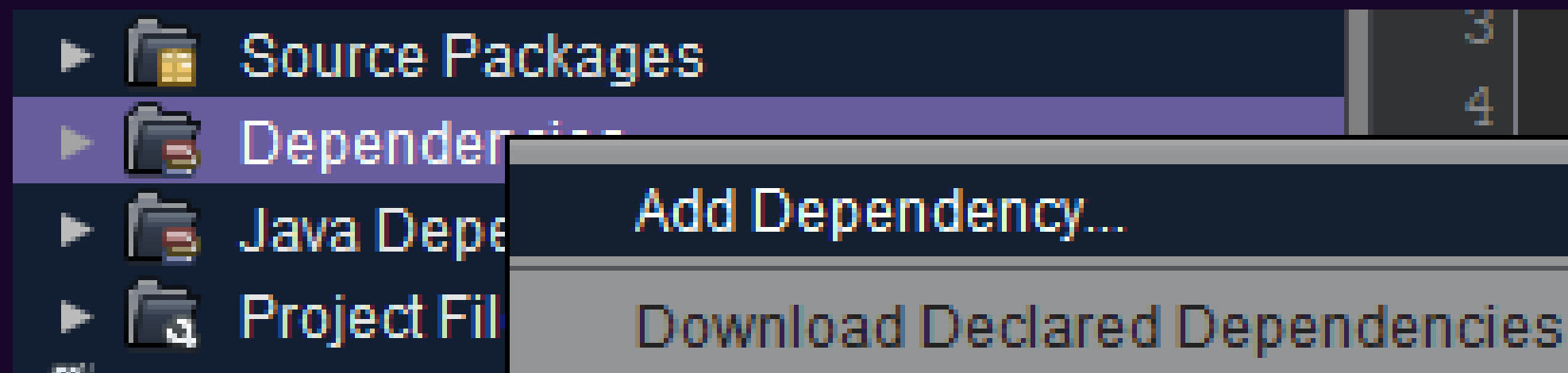
Librerías y dependencias NetBeans 18 con Maven



Una vez creado el proyecto en maven, seguiremos los siguientes pasos para tener JUnit 4:

- Click derecho en Dependencies
-

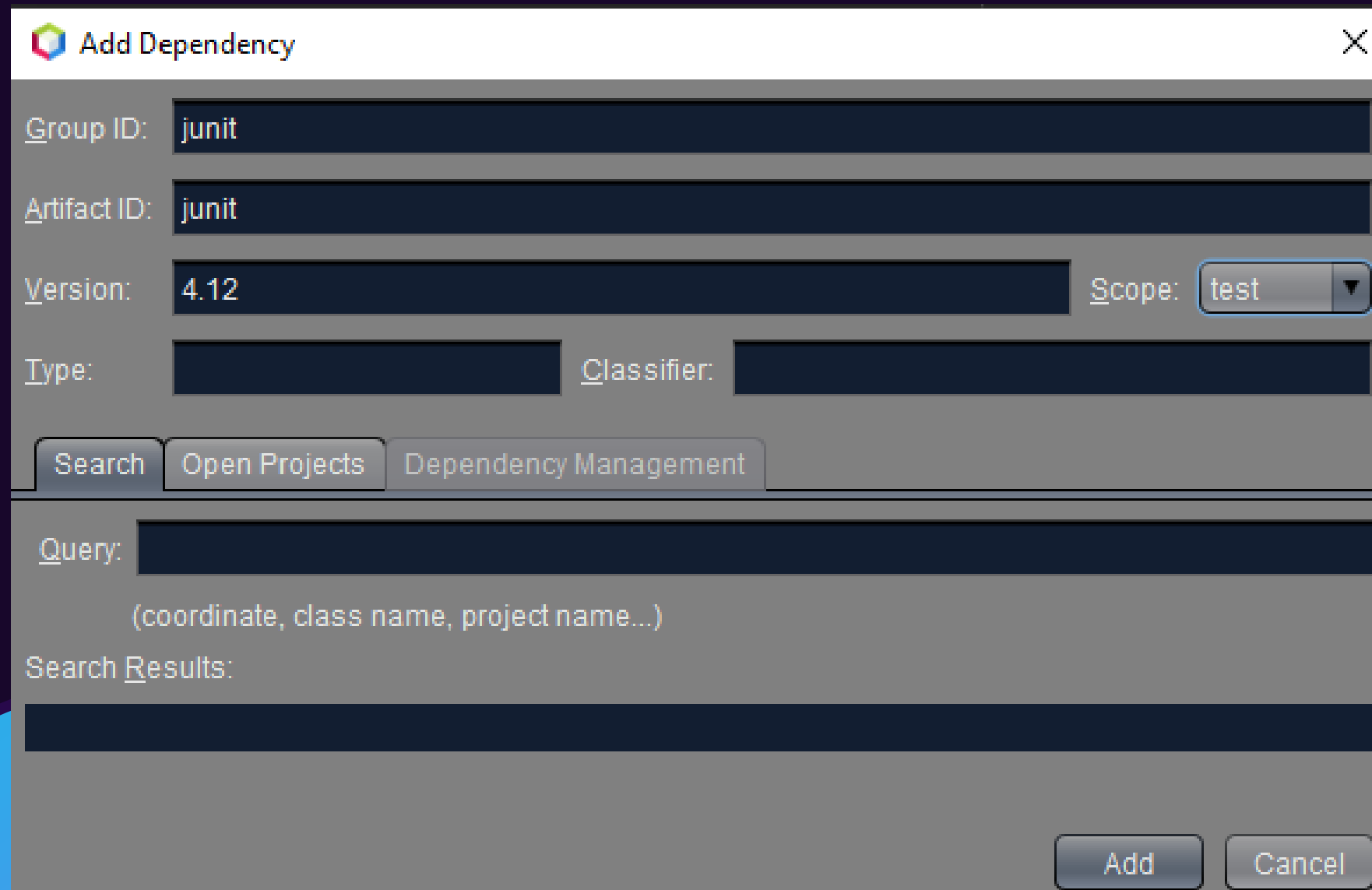
Librerías y dependencias NetBeans 18 con Maven



Vamos a agregar una nueva dependencia.

Agregaremos en ArtifactID y GroupID "jun"
En la version pondremos "4.12"

Librerías y dependencias NetBeans 18 con Maven



Add Dependency

Group ID:

Artifact ID:

Version: Scope:

Type: Classifier:

Query:

(coordinate, class name, project name...)

Search Results:

Agregaremos en ArtifactID y
GroupID "junit"
En la version pondremos "4.12"

De esta manera ya tendremos
cargado JUnit para realizar los test.

**¡MUCHAS
GRACIAS!**



PABLO ARANCIBIA

Bibliografia complementaria

Tutoriales de JUnit:

- [Videos de Egg](#)
- Material complementario de otros canales: [\(1\)](#) , [\(2\)](#) (Creditos a sus respectivos autores)

Documentación de JUnit:

- JUnit4: <https://junit.org/junit4/>
- JUnit5: <https://junit.org/junit5/docs/current/user-guide/>

NOTA: Como recomendación, siempre es bueno leer documentación propia de lo que vayamos aprendiendo.