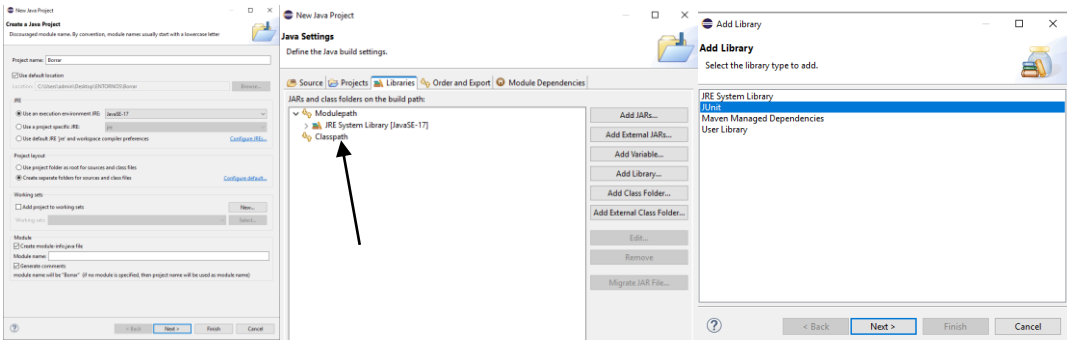
DISEÑO DE PRUEBAS CON JUnit

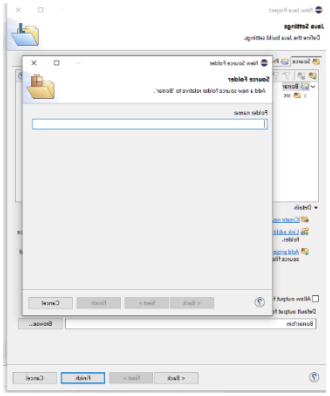
Junit es un Framework Open Source para la automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tare de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad.

1. Añadir la librería y configurar proyecto

Al crear el proyecto java, en el classpath debemos añadir la librería jUnit



Una vez añadida la librería seguimos creando el proyecto. Vamos a añadir además de la carpeta src otra para los test



A partir de aquí sólo nos queda añadir nuestras clases (no main) e implementar los test para probar que los métodos retornan las salidas debidas. Por cada clase que queremos probar tendremos una clase dónde irán los test que prueban sus métodos. Por cada método a probar tendremos un métodoTest en la correspondiente clase de test

2. Uso de JUnit para implementar los test

Hagamos un proyecto para probar una función suma y otra sumaPositivos. Haremos test para ver si hemos codificado bien los métodos de esa clase

Ejemplo:

1. Genera un nuevo proyecto java con las carpetas src y test habiendo añadido al classpath Junit
2. Añade en la carpeta src una clase SumasVarias
3. Añade dos métodos:
   suma (int, int) ->int
   sumaPositivos(int, int) -> -1 si los números no son ambos positivos o la suma si lo son

4. Añade en la carpeta(source folder) test un jUnit Test Case
5. Implementa el método que testeará nuestra clase

Una prueba Junit es un método contenido en una clase que solo se usa para realizar pruebas. Esto se llama una clase de prueba. Para definir que un determinado método es un método de prueba, lo implementamos con la *@Testanotación.*

Utilizamos un método de aserción, para comparar un resultado esperado con el resultado real. Estas llamadas a métodos normalmente se denominan aserciones o afirmaciones.

Definición de métodos de prueba

La siguiente tabla ofrece una descripción general de las anotaciones más importantes en JUnit para las versiones 4.x y 5.x. Todas estas anotaciones se pueden utilizar en los métodos.

JUnidad 4	Descripción
`import org.junit.*`	Declaración de importación para usar las siguientes anotaciones.
`@Test`	Identifica un método como método de prueba.
`@Before`	Ejecutado antes de cada prueba. Se utiliza para preparar el entorno de prueba (por ejemplo, leer datos de entrada, inicializar la clase).
`@After`	Ejecutado después de cada prueba. Se utiliza para limpiar el entorno de prueba (p. ej., eliminar datos temporales, restaurar valores predeterminados). También puede ahorrar memoria limpiando costosas estructuras de memoria.
`@BeforeClass`	Ejecutado una vez, antes del inicio de todas las pruebas. Se utiliza para realizar actividades que requieren mucho tiempo, por ejemplo, para conectarse a una base de datos. Los métodos marcados con esta anotación deben definirse `static` para que funcionen con JUnit.
`@AfterClass`	Ejecutado una vez, una vez finalizadas todas las pruebas. Se utiliza para realizar actividades de limpieza, por ejemplo, para desconectarse de una base de datos. Los métodos anotados con esta anotación deben definirse `static` para que funcionen con JUnit.
`@Ignore` o `@Ignore("Why disabled")`	Marca que la prueba debe ser deshabilitada. Esto es útil cuando se ha cambiado el código subyacente y el caso de prueba aún no se ha adaptado. O si el tiempo de ejecución de esta prueba es demasiado largo para incluirlo. Es una buena práctica proporcionar la descripción opcional, por qué la prueba está deshabilitada.
`@Test (expected = Exception.class)`	Falla si el método no lanza la excepción nombrada.
`@Test(timeout=100)`	Falla si el método tarda más de 100 milisegundos.

Hay muchas otras anotaciones, pero algunas de las más comunes son las siguientes.

- @Before identifica un método que debe ejecutarse antes de cada método de prueba en la clase. Por lo general, se usa para actualizar o restablecer el estado necesario para que los métodos de prueba se ejecuten correctamente.
- @After identifica un método que debe ejecutarse después de cada método de prueba en la clase. Se puede utilizar para restablecer variables, eliminar archivos temporales, etc.
- @Ignore especifica que no se debe ejecutar un método de prueba.
- @BeforeClass identifica un método que debe ejecutarse una vez antes de ejecutar cualquier método de prueba.
- @AfterClass identifica un método que debe ejecutarse una vez después de ejecutar todos los métodos de prueba.

Afirmar declaraciones

JUnit proporciona métodos estáticos para probar ciertas condiciones a través de la `Assertions` clase. Estas declaraciones de aserción generalmente comienzan con `assert`. Permiten especificar el mensaje de error, el resultado esperado y el real.
Un método de aserción compara el valor real devuelto por una prueba con el valor esperado. Lanza un `AssertionException` si la comparación falla.

La siguiente tabla ofrece una descripción general de estos métodos. Los parámetros entre corchetes [] son opcionales y de tipo String.

Statement	Description
fail([message])	Let the method fail. Might be used to check that a certain part of the code is not reached or to have a failing test before the test code is implemented. The message parameter is optional.
assertTrue([message,] boolean condition)	Checks that the boolean condition is true.
assertFalse([message,] boolean condition)	Checks that the boolean condition is false.
assertEquals([message,] expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertEquals([message,] expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message,] object)	Checks that the object is null.
assertNotNull([message,] object)	Checks that the object is not null.
assertSame([message,] expected, actual)	Checks that both variables refer to the same object.
assertNotSame([message,] expected, actual)	Checks that both variables refer to different objects.