TAREA ED03
Antonio Jimenez Sevilla

1. **¿Qué es validar y qué es verificar software?**

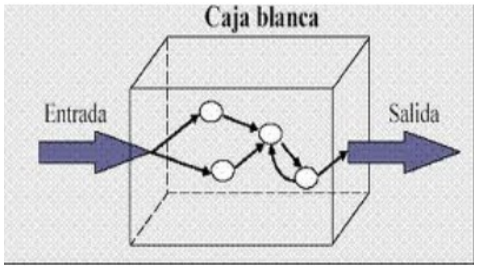
   Los términos de validación y verificación son usados en muchos campos, y comúnmente suelen ser confundidos, pero en realidad tiene actividades muy diferenciadas aunque se debe reconocer que cierto solape entre ambas actividades sí que hay.

   La validación: es el proceso de evaluación del sistema o de uno de sus componentes, para determinar si satisface los requisitos especificados. En resumen **es una actividad de, evaluar el resultado final, comprobar que cumple con lo inicialmente acordado.**

   La verificación: es la comprobación que un sistema o parte de un sistema, cumple con las condiciones impuestas. Con la verificación se comprueba si la aplicación se está construyendo correctamente. En resumen **comprobar los requisitos en cada una de las fases.**

2. **¿Qué son las pruebas de caja blanca?**

   Son un tipo de pruebas de software que se realiza sobre las funciones internas usando su lógica de aplicación.Va a analizar y probar directamente el código de la aplicación. Es necesario un conocimiento específico del código, para poder analizar los resultados de las pruebas.



3. **¿Qué son las pruebas de caja negra?**

   Es una prueba que se realiza desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. Aquí lo fundamental es comprobar que los resultados de la ejecución de la aplicación(sin tener que conocer la estructura ni su funcionamiento interno), son los esperados, en función de las entradas que recibe sin necesidad de conocer el proceso mediante el cual la aplicación obtiene los resultados.

4. **¿Qué diferencia hay entre las pruebas de caja blanca y las pruebas de caja negra?**

- Las pruebas de caja blanca realizan pruebas en la estructura del sistema.
- Las pruebas de caja negra para verificar que el requisito del sistema se cumple en consecuencia.

- Las pruebas de caja blanca requieren probadores altamente técnicos.
- El conocimiento técnico del probador no es muy esperado para las pruebas de caja negra
- Fácil de rastrear errores internos en pruebas de caja blanca.
- Fácil de realizar una prueba para ver cómo funcionará el sistema usando pruebas de caja negra.

5. ¿Cuáles son las recomendaciones para hacer las pruebas de caja blanca?

- **Cobertura de sentencias:** se han de generar casos de pruebas suficientes para que cada instrucción del programa sea ejecutada, al menos, una vez.
- **Cobertura de decisiones:** se trata de crear los suficientes casos de prueba para que cada opción resultado de una prueba lógica del programa, se evalúe al menos una vez a cierto y otra a falso.
- **Cobertura de condiciones**: se trata de crear los suficientes casos de prueba para que cada elemento de una condición, se evalúe al menos una vez a falso y otra a verdadero.
- **Cobertura de condiciones y decisiones**: consiste en cumplir simultáneamente las dos anteriores.
- **Cobertura de caminos**: es el criterio más importante. Establece que se debe ejecutar al menos una vez cada secuencia de sentencias encadenadas, desde la sentencia inicial del programa, hasta su sentencia final. La ejecución de este conjunto de sentencias, se conoce como camino. Como el número de caminos que puede tener una aplicación, puede ser muy grande, para realizar esta prueba, se reduce el número a lo que se conoce como camino prueba.
- **Cobertura del camino de prueba**: Se pueden realizar dos variantes, una indica que cada bucle se debe ejecutar sólo una vez, ya que hacerlo más veces no aumenta la efectividad de la prueba y otra que recomienda que se pruebe cada bucle tres veces: la primera sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo dos veces.

6. ¿En qué consiste la prueba del camino básico?

La prueba del camino básico es una técnica de prueba de la Caja Blanca propuesta por Tom McCabe.

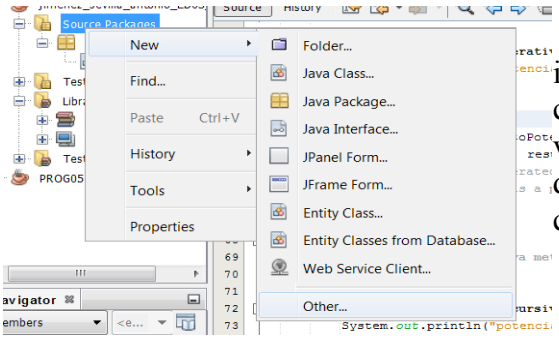
Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática. Los pasos que se siguen para aplicar esta técnica son:

- A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
- Se calcula la complejidad ciclomática del grafo.
- Se determina un conjunto básico de caminos independientes.
- Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

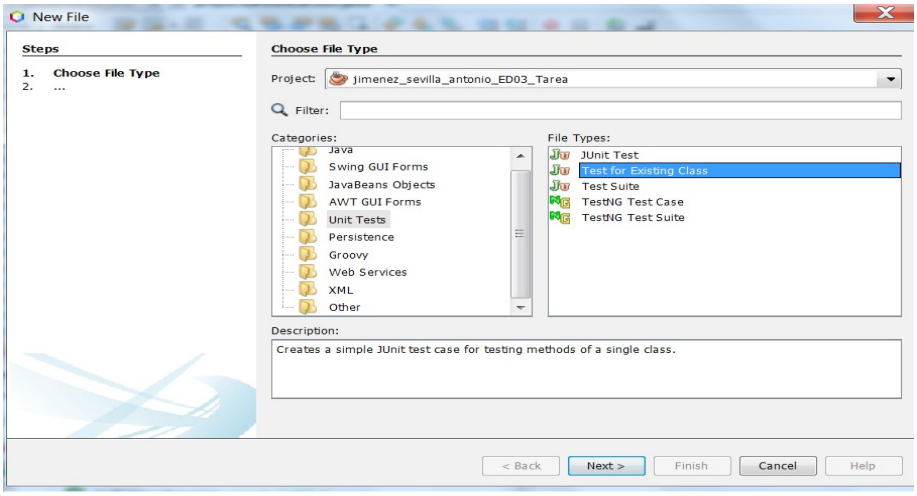
Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

7. **Se creará un ProyectoPotencia y en este ejercicio práctico se deberá verificar su funcionamiento mediante otras pruebas unitarias diferentes (tres): Mostrar las capturas de pantalla. Se deben realizar los siguientes apartados comentando los resultados y capturando las correspondientes pantallas de ejecución:**

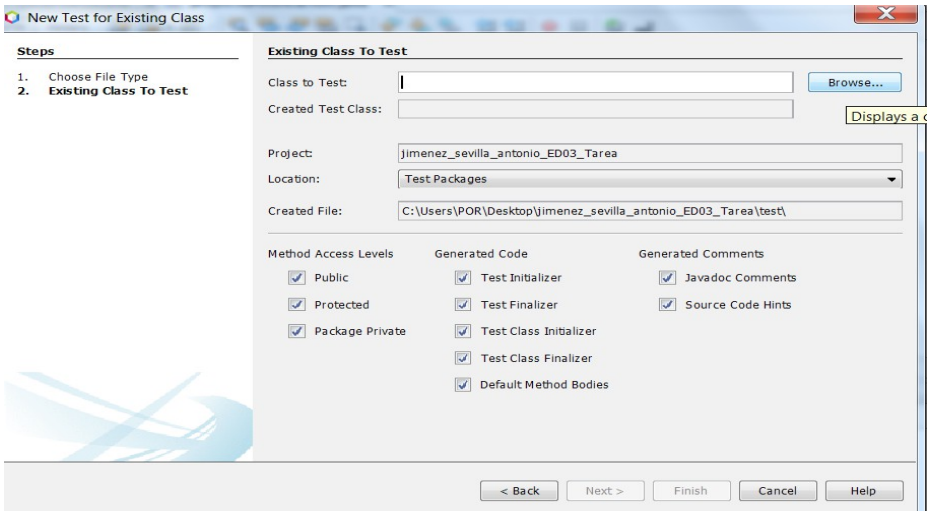


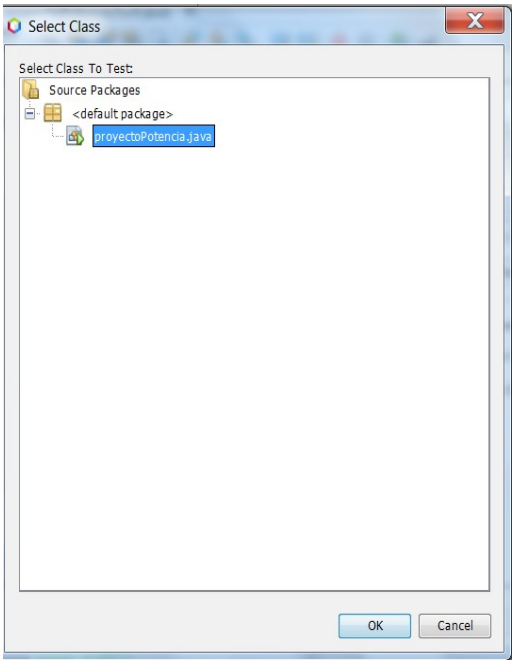
Una prueba unitaria de Java es **JUnit** para ello iremos al proyecto en la parte izquierda de Netbeans, y con el botón derecho del ratón se nos desplegará una ventana, posamos el ratón sobre New y veremos un desplegable en el que hay varias opciones, hacemos click con el botón izquierdo del ratón en Other...

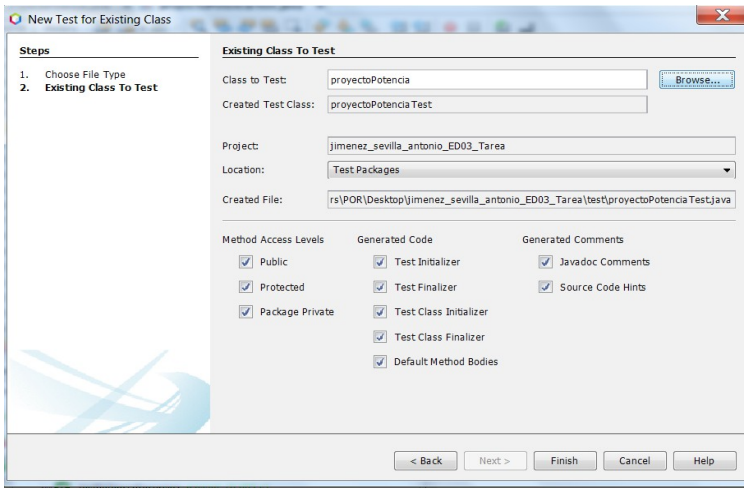
Se nos abre una ventana donde tenemos dos opciones una es categorías y la otra es el tipo de archivo (File Type) vamos con el ratón donde pone Unit Test y a la derecha en File Types nos aparecerán una serie de opciones, en nuestro caso usaremos Test for Existing File.



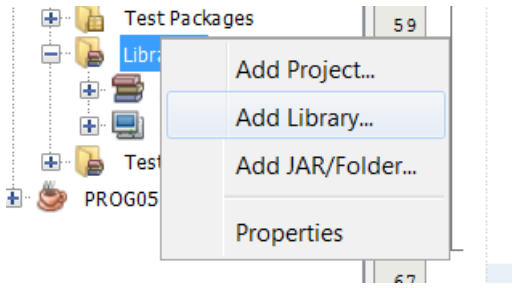
Una vez cliquemos nos aparecerá la siguiente ventana en la cual elegiremos la clase para realizar el test.

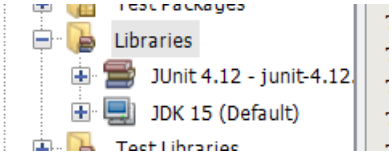


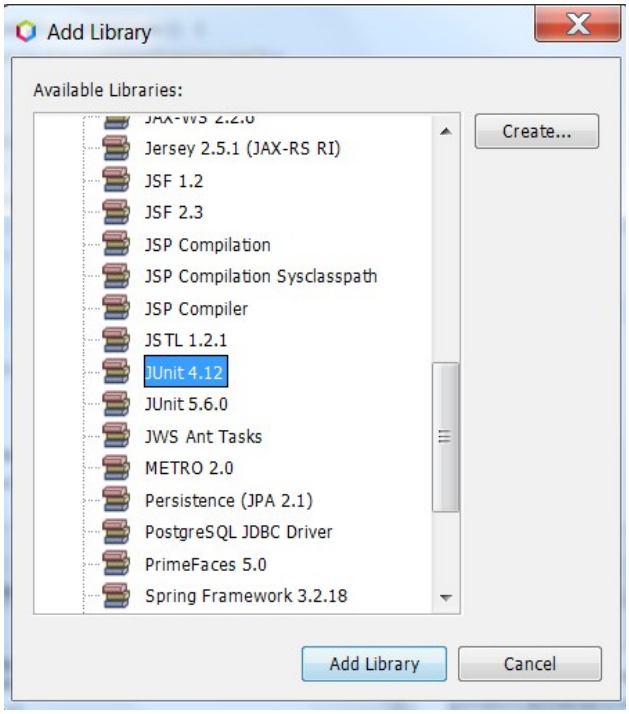




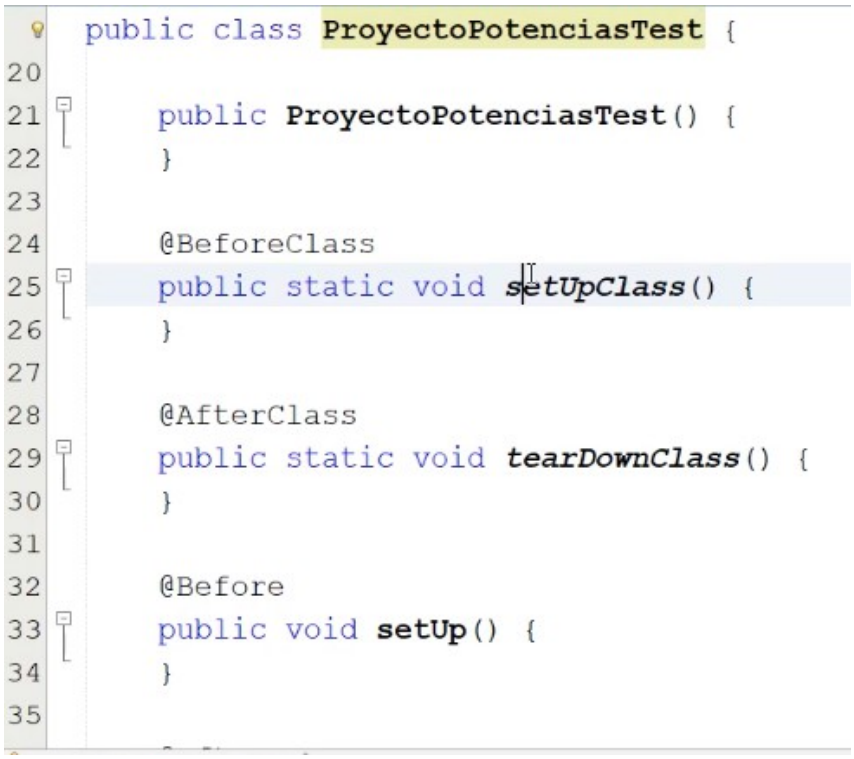
Antes de seguir, con Netbeans versión 12, hay que añadir a la librería Junit4 ya que sino, nos dará error y no nos dejará hacer alguna cosa.



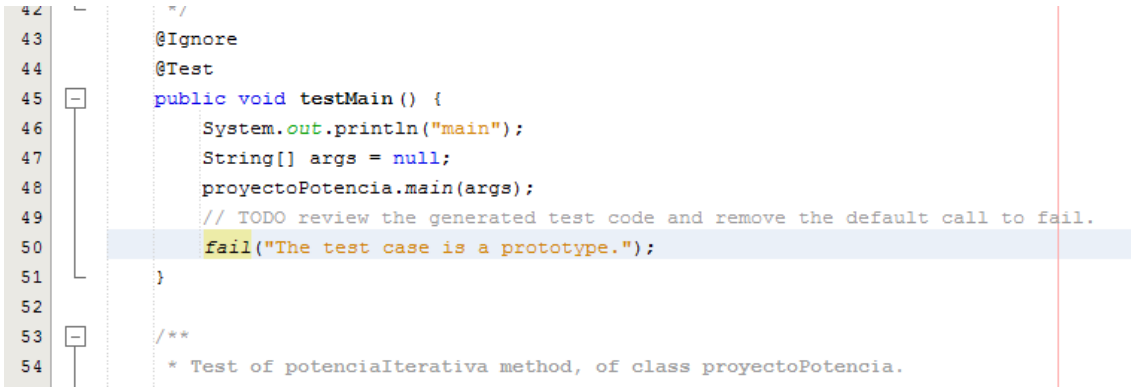




Una vez creado el test se nos abre una ventana en el que nos encontramos con una nueva clase proyectoPotenciasTest, setup test... esto es para la realización de pruebas antes y después de la clase.



Si seguimos bajando nos encontramos con el main, pero como no la vamos a usar pues ponemos @Ignore encima de @Test, para que ignore el Main.



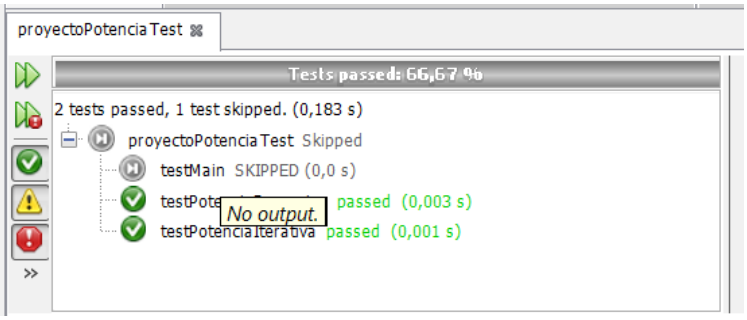
A continuación vamos a probar la potenciaIterativa la cual vamos a dar unos valores a “x” e “y” 2 para x, 3 para y. En expoResultado vamos a poner el resultado real de 2 elevado a 3. Importante poner las 2 barras en donde nos marca la palabra Fail, ya que sino nos dará error.

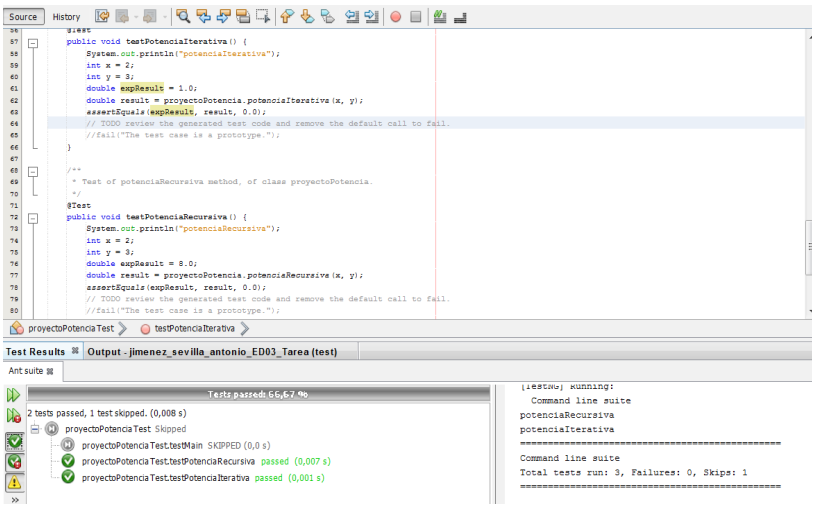
```
56      @Test
57      public void testPotenciaIterativa() {
58          System.out.println("potenciaIterativa");
59          int x = 2;
60          int y = 3;
61          double expResult = 8.0;
62          double result = proyectoPotencia.potenciaIterativa(x, y);
63          assertEquals(expResult, result, 0.0);
64          // TODO review the generated test code and remove the default call to fail.
65          //fail("The test case is a prototype.");
66      }
```

Con la potenciaRecursiva hacemos lo mismo.

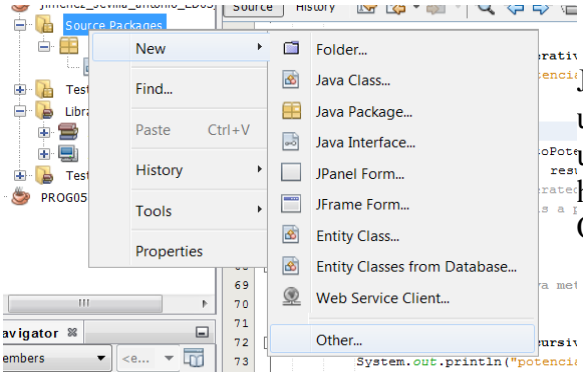
```
71      @Test
72      public void testPotenciaRecursiva() {
73          System.out.println("potenciaRecursiva");
74          int x = 2;
75          int y = 3;
76          double expResult = 8.0;
77          double result = proyectoPotencia.potenciaRecursiva(x, y);
78          assertEquals(expResult, result, 0.0);
79          // TODO review the generated test code and remove the default call to fail.
80          //fail("The test case is a prototype.");
81      }
```

Le damos a Run file y nos sale que 2 test han pasado y que uno ha sido esquivado.



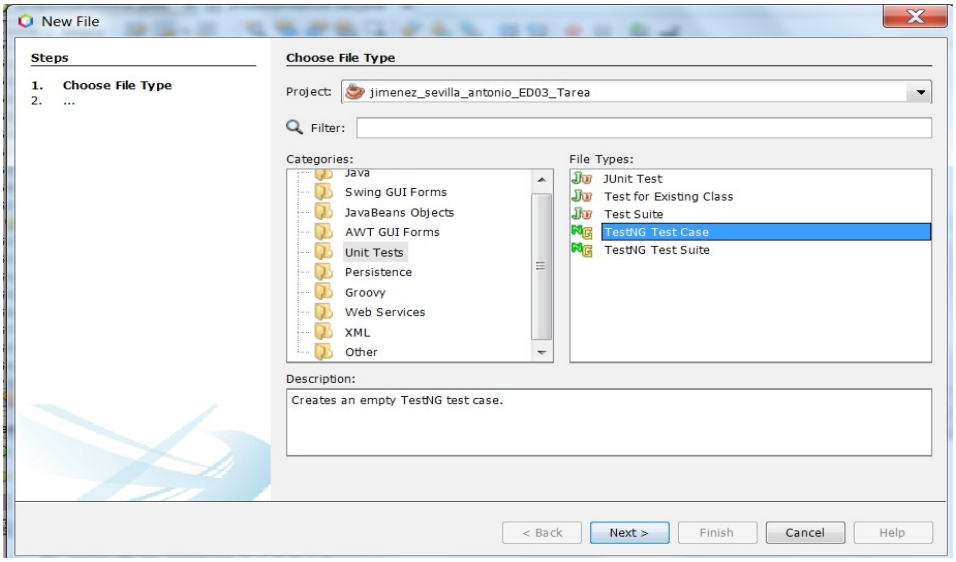


Otra prueba unitaria en Java es la de **testNG** que se realiza de la siguiente manera:

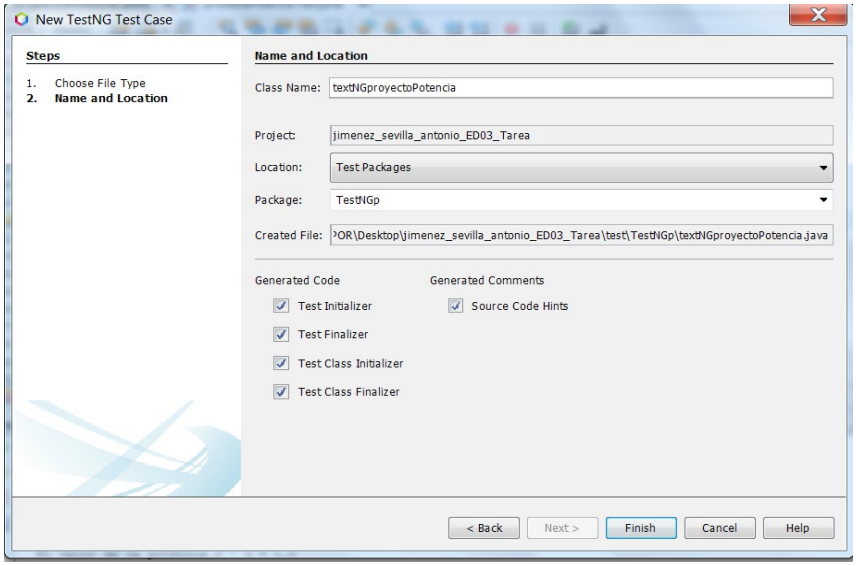


El proceso para abrir el test es parecido que JUnit, on el botón derecho del ratón se nos desplegará una ventana, posamos el ratón sobre New y veremos un desplegable en el que hay varias opciones, hacemos click con el botón izquierdo del ratón en Other...

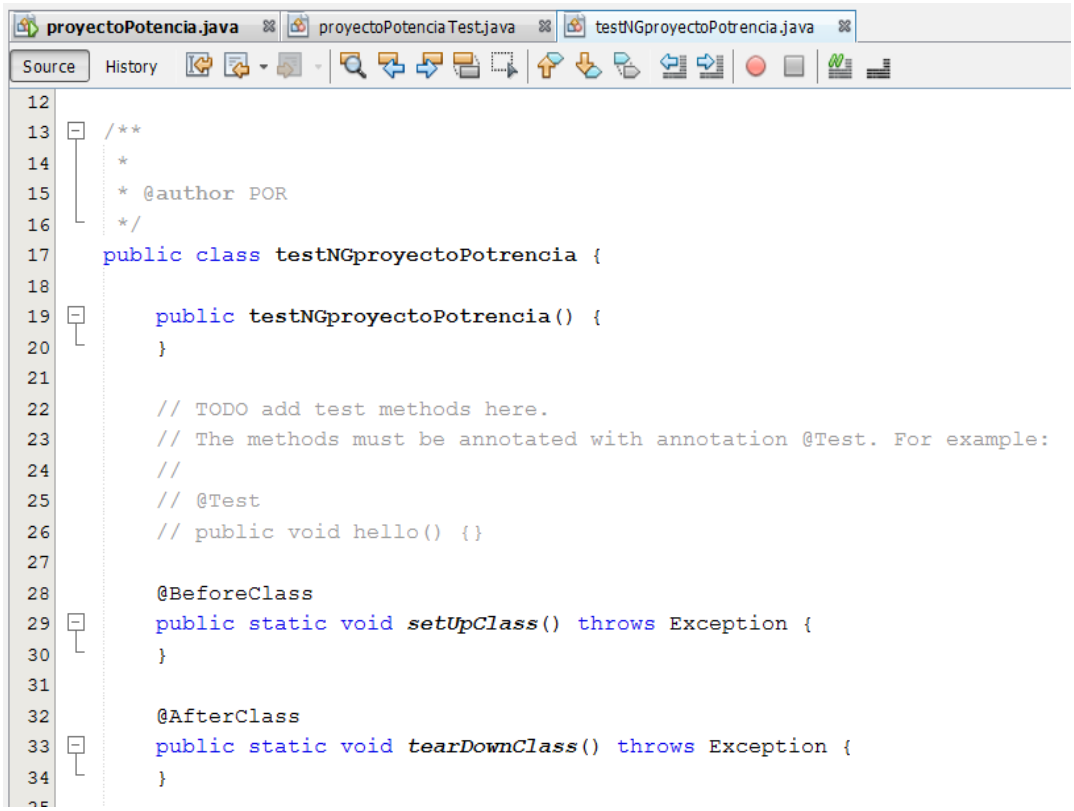
Se nos abre una ventana donde tenemos dos opciones una es categorías y la otra es el tipo de archivo (File Type) vamos con el ratón donde pone Unit Test y a la derecha en File Types nos aparecerán una serie de opciones, en este caso usaremos TestNG Test Case.



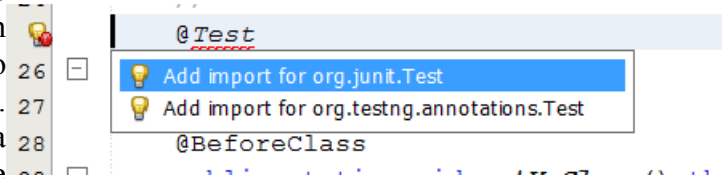
Y el nombre de la clase que vamos a querer hacer el test.



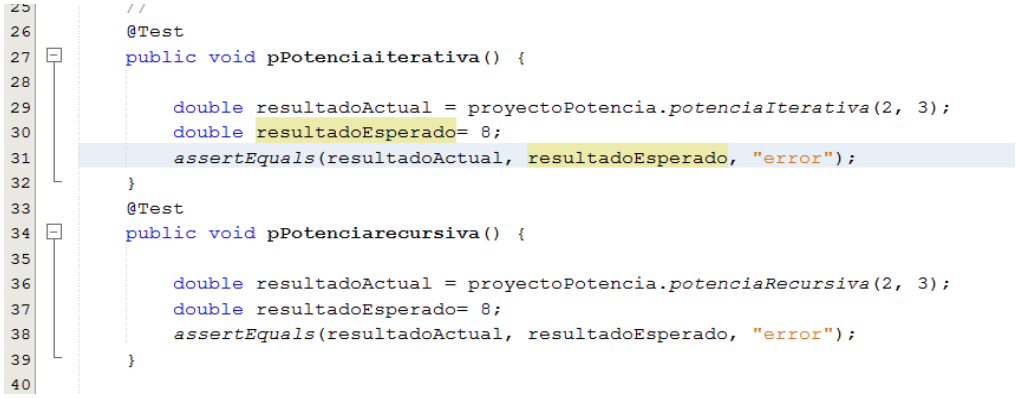
Se nos abre una ventana en la que nis encontramos una nueva clase testNGproyectoPotencia. Como vemos a simple vista parece igual que Junit, ya que tenemos también el setUpClass etc...



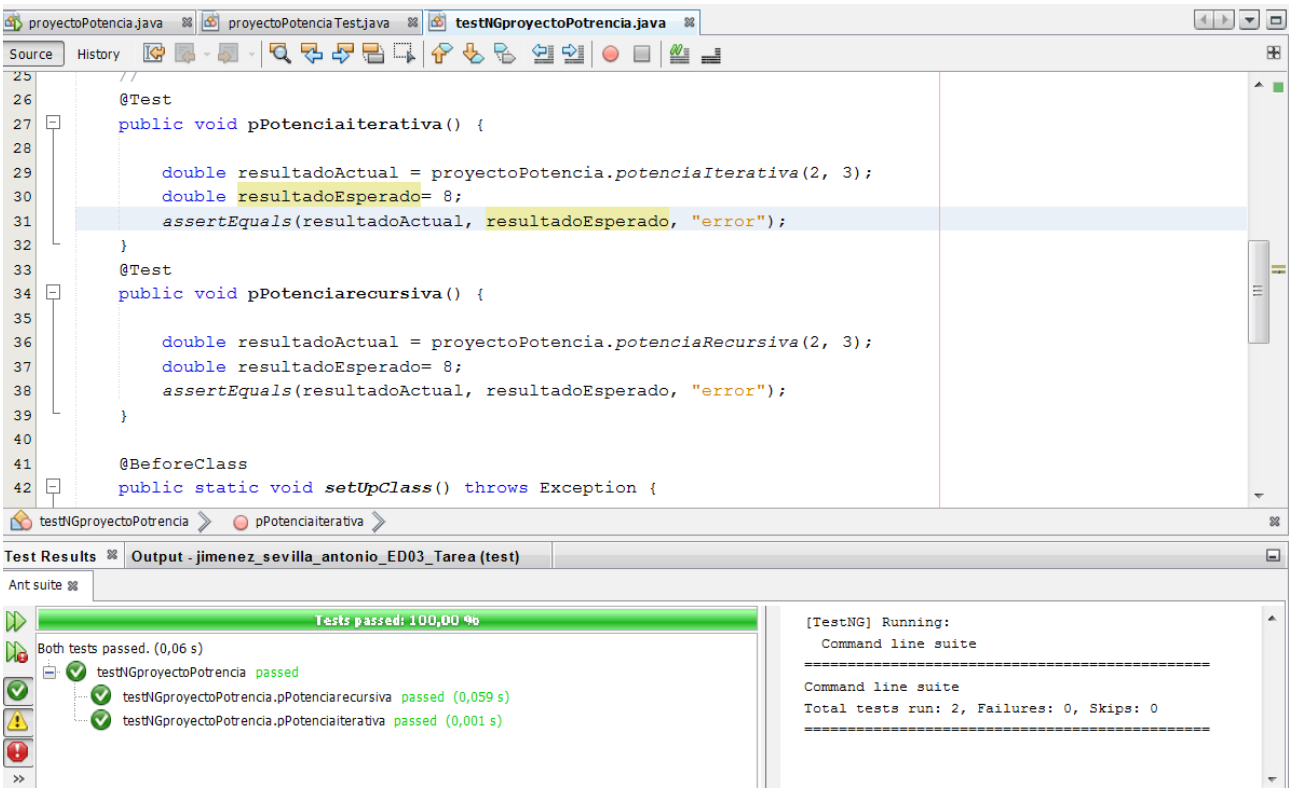
Aqui tenemos que en los comentarios ya tenemos un @Test y un public void, pero al estar puestos como comentarios pues no tienen ningún efecto. Con lo que eliminamos las barritas a @Test y a public void, no pedirá que importemos org.unit.Test.



Una vez importado lo que tenemos que hacer es crear una clase para proyectoIterativa. Colocamos dos atributos de tipo double, uno para el resultado actual y otro para el esperado. En el resultadoActual le diremos que es igual a nuestro proyectoPotencia y que lea nuestra clase potenciaItinerancia le damos los mismos valores de antes 2 y 3, en el otro atributo resultadoEsperado le damos el valor de 8 ya que es el resultado de la potencia de 2 elevado 3. Añadimos un assertEquals. Con potenciaRecursiva hacemos lo mismo

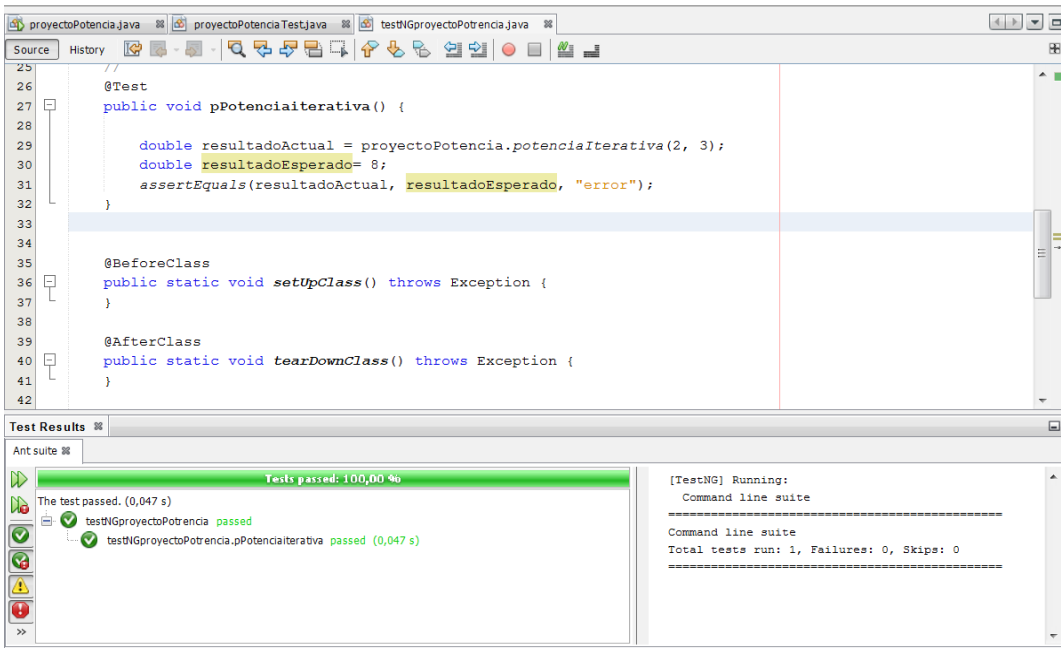


Le damos a run file y tenemos que lo pasa con un 100%.



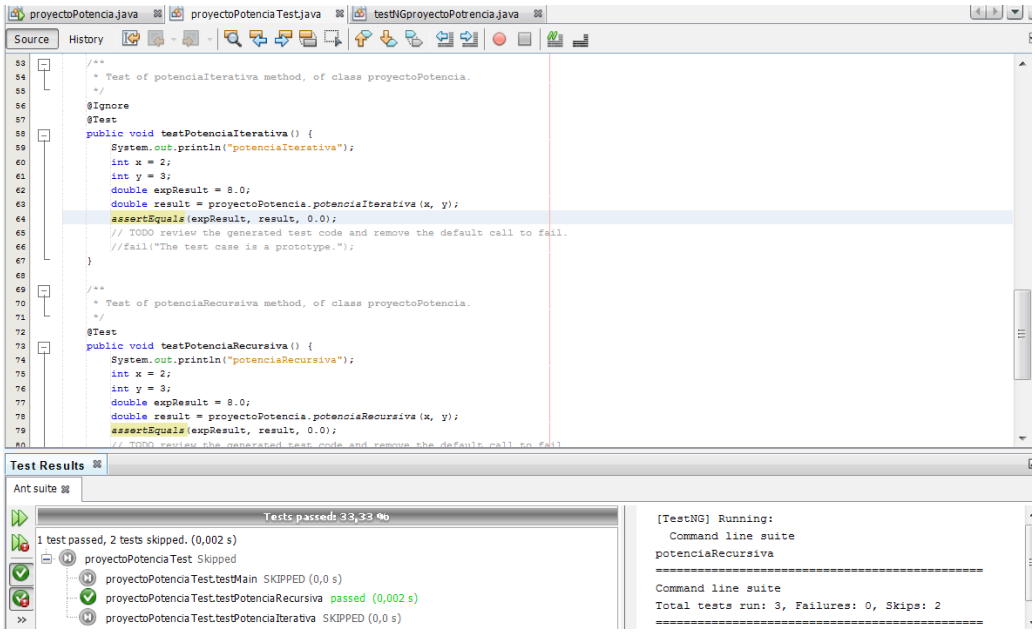
1. Diseña un caso de prueba que permita verificar el método PotenciaIterativa.

Mediante la prueba unitaria del **TestNG** he realizado la prueba del método Potencia Iterativa. He realizado la prueba de manera que solo he escrito el método para Potencia iterativa en TestNG. En la siguiente imagen vemos como.



2. Diseña un caso de prueba que permita verificar PotenciaRecursiva.

Para variar esta actividad la he realizado con el test **JUnit**, y para realizar lo que he hecho ha sido escribir encima de @Test de la PotenciaIterativa, ponemos @Ignore para que esquive el test y solo nos ejecute el test PotenciaRecursiva, que es el que queremos probar.



3. Indica el funcionamiento del método assert en la realización de pruebas. Justifica su utilización o no en este caso.

Con estos métodos podemos afirmar tipos primitivos, objetos y arreglos. Estos métodos están conformados de esta manera,**(valor real, valor esperado, mensaje en caso de falla)**, en el valor real es en donde vamos a afirmar que un valor será el que nosotros creemos, hay que tomar en cuenta que algunos varían con valor esperado y valor real.

Con lo que en nuestro caso el su utilización es muy importante puesto que permiten especificar el resultado real, el resultado esperado y mensaje de falla. Le decimos que 1 + 1= 2 para que pueda comparar el código escrito con los valores que le hallamos introducido y se el código correcto.

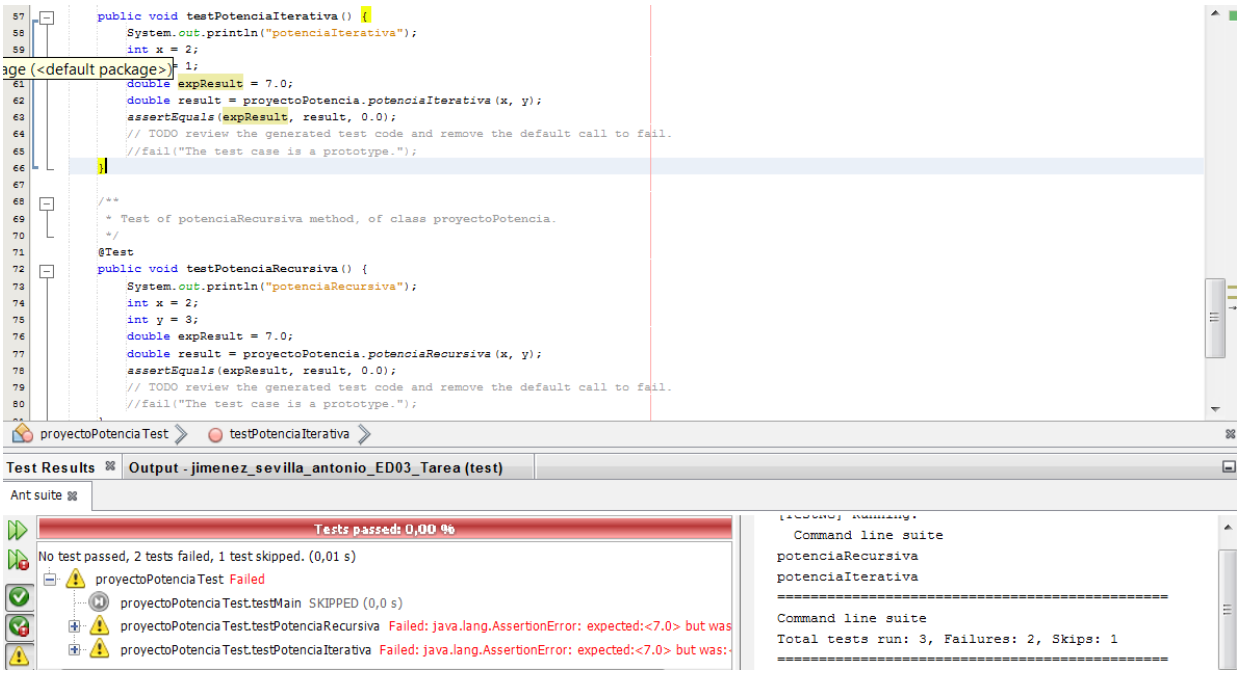
```
System.out.println("potenciaIterativa");
int x = 2;
int y = 3;
double expResult = 8.0;
double result = proyectoPotencia.potenciaIterativa(x, y);
assertEquals(expResult, result, 0.0);
```

En la imagen de la izquierda vemos que le damos un valor a x y otra y, en expoResult le indicamos que 2 elevado a 3 es igual 8. En result le decimos que nos haga la operación de las clases del código creado y que el valor esperado es el de (x, y).

Mediante la función asserEquals nos compara el resultado real de el esperado.

4. **Haz que uno de los tests falle. ¿Cómo sería?**

Yo lo que he realizado es escribir un valor para el resultado diferente al real, para comprobar que la operación mediante el código realizado es correcta.



Como se puede observar en la imagen inferior el test nos dice que falla, y que el valor esperado era 7 y que el resultado del código es 8 para los dos test. Con esto podemos estar seguros que el código está realizando bien las operaciones.