|  |  |
| --- | --- |
| Présentation de la Javadoc - OpenClassrooms  JAVADOC EN JUNIT  Repositorio: git@github.com:mikelbarbe/AD\_2\_JavaDoc.git | Autores   * Mikel Barberena GitHub: mikelbarbe * Almudena   GitHub: AlmudenaGB   * Beatriz Waliño   GitHub: Beatriz Wali   * Adolfo   GitHub: AdolMF   * Juan Carlos   GitHub: Jcrev95 |

**Descripción**

En esta actividad en grupo, trabajaremos sobre un repositorio de GitHub compartido entre los integrantes de este. Cada integrante trabajara sobre su propia rama de desarrollo creando diferentes clases de operaciones matemáticas en el IDE IntelliJ.

Cada clase deberá de documentarse con etiquetas JavaDoc, para una vez terminadas todas las clases, generaremos la documentación en formato HTML a partir del código de Java.

También explicaremos como se crea la documentación en diferentes casos especiales. Y para finalizar realizaremos distintas pruebas unitarias del proyecto, es decir cada compañero, probara la clase creada por otro y verificara que funciona correctamente.

**Metodología: Pasos Realizados, Uso de GIT y Herramientas Empleadas**

**1. Creación y configuración del proyecto Calculadora. Los pasos fueron:**

Creación de proyecto en IntelliJ y creación de un nuevo repositorio en GitHub para gestionar el proyecto de la nueva actividad.

**Herramientas utilizadas son para la creación del proyecto en IntelliJ:**

**IntelliJ :** IDE utilizado para el desarrollo del código Java.

**GITBash**: interfaz de línea de comandos para usar Git en Windows

**2. Creación del Repositorio Remoto en GitHub**

Creación del repositorio en GitHub para compartir el proyecto con los compañeros de trabajo.

Se configura el repositorio local con el remoto de GitHub.

**Herramientas empleadas para la creación del repositorio remoto:**

**GitHub**: para almacenar el repositorio y permitir que todos los miembros del equipo colaboren y sincronicemos los cambios.

**GITBash**: interfaz de línea de comandos para usar Git en Windows

**3. Implementación de las Clases**

Implementación de diferentes clases que correspondan a cálculos matemáticos. Cada clase tiene métodos que se encargan de una operación específica, como suma, resta, multiplicación, módulo y cociente.

**Herramientas empleadas para la implementación de las Clases**

Para realizar las clases utilizamos la herramienta **IntelliJ** para escribir y compilar el código en Java.

**4. Pruebas Unitarias**

Se han creado pruebas unitarias usando JUnit para validar la correcta ejecución de los métodos de las clases.

**Herramientas empleadas para las pruebas unitarias**

Las herramientas empleadas son **Junit** que es un framework utilizado para escribir y ejecutar pruebas unitarias en Java.

**IntelliJ**  utilizado para escribir las pruebas.

Los cambios que se han ido haciendo se han registrado en el repositorio local y luego se suben a GitHub.

1. **Uso de Gitbash**:

Los comandos utilizados en este punto son:

git add . para agregar todos los archivos modificados al área de preparación.

git commit -m "Mensaje" para realizar el commit con los cambios.

git push origin main para subir los cambios al repositorio remoto en GitHub.

**6. Colaboración en Equipo**

Se comparte el repositorio con los compañeros de equipo para poder colaborar en el proyecto. Se clona el repositorio, se va realizando la actividad y luego se sincroniza los cambios con el repositorio remoto en GitHub.

Los comandos utilizados en este punto son:

git pull origin main para traer los últimos cambios realizados por los compañeros.

git push origin main para subir sus propios cambios al repositorio.

**8. Finalización del Proyecto**

El proyecto ha sido completado y las pruebas unitarias han pasado correctamente.

Problemas y soluciones

**Documentación de Casos Especiales**

1. **¿Qué ocurre cuando alguno de los parámetros de entrada es cero?**
   * **Suma y Resta**: En el caso de los métodos de suma y resta, si alguno de los parámetros de entrada es cero, la operación se realizaría sin problemas. La suma o resta no se ve afectada por el cero.
   * **Multiplicación**: Si alguno de los parámetros de entrada es cero, el resultado será siempre cero, ya que cualquier número multiplicado por cero es cero (n \* 0 = 0).
   * **División**: Si uno de los parámetros de entrada de la división es cero, el comportamiento varía dependiendo del contexto:
     + División de un número entre cero: Esto genera una excepción matemática (división por cero) y debe ser manejado adecuadamente en el código para evitar errores en tiempo de ejecución (ArithmeticException).
     + División de cero entre cualquier número: El resultado será cero (0 / a = 0), siempre que ‘a’ no sea cero.
2. **Supongamos que la calculadora no puede utilizar números negativos ¿Cómo lo documentaríamos?**

Si la calculadora no puede aceptar números negativos, se debe documentar en cada uno de los métodos. En caso de que se intente pasar un número negativo como parámetro, el método debería arrojar una excepción o devolver un mensaje de error indicando que los números negativos no son permitidos.

/\*\*

\* Suma dos números enteros.

\* @param a Primer número entero. Debe ser mayor o igual a cero.

\* @param b Segundo número entero. Debe ser mayor o igual a cero.

\* @return Resultado de la suma.

\* @throws IllegalArgumentException Si alguno de los números es negativo.

\*/

public int sumaEnteros(int a, int b) {

if (a < 0 || b < 0) {

throw new IllegalArgumentException("Los números no pueden ser negativos.");

}

return a + b;

}

1. **¿Qué ocurriría si la división no da un número exacto?**

En el caso de la división que no resulta en un número exacto, el comportamiento dependerá de cómo se haya implementado la operación:

* + Para divisiones de números reales (tipo double), el resultado será el valor más preciso posible dentro de los límites del tipo de dato double.
  + Para divisiones de números enteros, el resultado será un número entero truncado, ya que Java no devuelve decimales en una división de enteros. Por ejemplo, 7 / 2 devolverá 3, truncando el valor decimal.

/\*\*

\* Realiza la división de dos números enteros.

\* @param a Dividendo.

\* @param b Divisor.

\* @return Resultado de la división truncado a número entero.

\* @throws ArithmeticException Si se intenta dividir por cero.

\*/

public int dividirEnteros(int a, int b) {

if (b == 0) {

throw new ArithmeticException("No se puede dividir entre cero.");

}

return a / b; // Truncará el valor decimal.

}

1. **¿Qué ocurriría con la potencia si los números pasados son muy grandes?**

En operaciones de potencia, cuando la base o el exponente son demasiado grandes, se podría producir un desbordamiento de memoria.

Si el resultado excede el rango del tipo double, se podría devolver Infinity (positivo o negativo), o si excede el rango de long o int, se podría producir un error de desbordamiento.

/\*\*

\* Calcula la potencia de un número con un exponente.

\* @param base La base de la potencia.

\* @param exponente El exponente al que se elevará la base.

\* @return Resultado de la potencia. Si el valor es demasiado grande, se puede devolver 'Infinity'.

\* @throws ArithmeticException Si ocurre un desbordamiento o error en el cálculo.

\*/

public double potencia(double base, int exponente) {

double resultado = Math.pow(base, exponente);

if (Double.isInfinite(resultado)) {

throw new ArithmeticException("El resultado de la potencia es demasiado grande.");

}

return resultado;

}