MIEMBROS

* Laura Moreno París GII
* Borja Rodríguez González GII
* Rubén López Fraile GII
* Alejandro Parrilla Monrocle GII
* Jesús Álvarez Tenorio GII

|  |  |
| --- | --- |
| práctica  Ampliación de Ingeniería de Software | Descripción breve  La práctica de la asignatura de Ampliación de Ingeniería del Software pretende poner en práctica el conjunto de conocimientos implicados en dicha asignatura. Por ello, aunque la práctica es única, será llevada a cabo por partes y en base a los temas que abarca la asignatura. |

Contenido

[1. INTRODUCCIÓN 2](#_Toc445135643)

[2. PLANIFICACIÓN 2](#_Toc445135644)

[2.1. Identificar errores 3](#_Toc445135645)

[2.2. Establecer requisitos 4](#_Toc445135646)

[2.3. Diagrama de clases 4](#_Toc445135647)

[2.4. Corregir código según los errores identificados 5](#_Toc445135648)

[2.5. Realizar prototipo de baja fidelidad 6](#_Toc445135649)

[2.6. Añadir nueva funcionalidad 6](#_Toc445135650)

[2.7. Realizar interfaz 8](#_Toc445135651)

[2.8. Pruebas 8](#_Toc445135652)

[2.9. Documentar todo el proceso 12](#_Toc445135653)

[2.10. Presentación 12](#_Toc445135654)

[3. Evaluación del proyecto 13](#_Toc445135655)

# 1. INTRODUCCIÓN

En esta práctica tenemos que gestionar el mantenimiento de un software, en este caso una calculadora muy básica la cual tenemos que corregir y añadir funcionalidad nueva (mantenimiento evolutivo). Para ello lo primero que tenemos que hacer es realizar planificar las tareas y asignarlas a cada miembro del grupo.

# 2. PLANIFICACIÓN

En primer lugar tenemos que identificar las tareas que encontramos y posteriormente realizar la planificación en cuanto a tiempo estimado mediante un diagrama Pert.

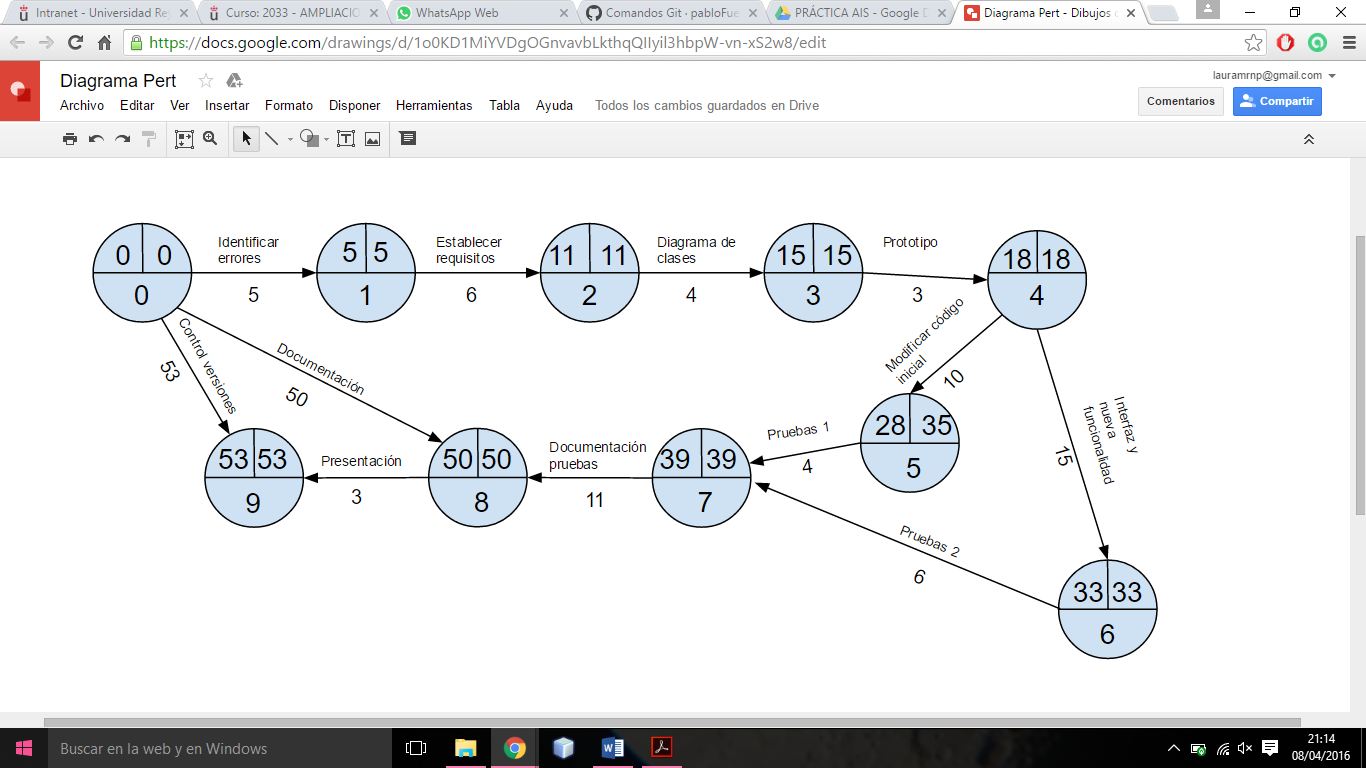
Las tareas que hemos encontrado son las siguientes:

1. Identificar errores
2. Establecer requisitos
3. Diagrama de clases
4. Realizar prototipo
5. Modificar código inicial
6. Añadir nueva funcionalidad e interfaz
7. Pruebas 1
8. Pruebas 2
9. Documentación de pruebas
10. Presentación
11. Documentar todo el proceso
12. Control de versiones

En cuanto al reparto de dichas actividades tenemos lo siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Tarea | Participantes | Tarea | Participantes |
| 1 | Borja Rodríguez | **6** | Alejandro Parrilla y Rubén López |
| 2 | Rubén López | **7** | Alejandro Parrilla |
| 3 | Laura Moreno | **8** | Rubén López |
| 4 | Jesús Á., Laura M. y Borja R. | **9** | Laura, Borja, Rubén y Alejandro |
| 5 | Alejandro Parrilla | **10** | Rubén López |

El diagrama de Pert, llevado a cabo por Laura, resultante de las anteriores actividades según su realización hasta la fecha fijada por el grupo para finalizar la primera parte de la práctica es el siguiente:



En este diagrama hemos calculado los tiempos en función de días, puesto que hemos presupuesto que nuestro producto debería estar realizado antes del día 18 de Abril, por tanto contamos con 53 días.

A los sucesos los hemos asignado números desde el 1 hasta el 10 y las actividades tienen el nombre completo de la actividad que se va a llevar a cabo.

Nos encontramos con que, hasta el suceso número 5 tenemos una precedencia lineal de las actividades. En este suceso tiene lugar una precedencia divergente puesto que para la iniciación tanto de la modificación del código inicial como de la nueva funcionalidad es necesario que se haya completado la realización del prototipo. Para estas dos actividades tenemos que realizar las pruebas correspondientes a cada una de ellas por lo tanto su actividad sucesora, para cada caso, serán pruebas 1 y pruebas 2. Ambas pruebas finalizarán con la documentación de todas ellas.

El camino crítico de este diagrama es el que implica a los sucesos 1, 2, 3, 4, 5, 7, 8, 9 y 10, lo que quiere decir que las actividades implicadas en esos sucesos tienen una holgura de 0 días. Sin embargo, la actividad consistente en realizar la modificación del código inicial tiene una holgura de 4 días. REVISAR

## 2.1. Tabla con tareas

Los números de las tareas son los que hemos definido en el apartado de para la creación del diagrama de Pert.

|  |  |  |  |
| --- | --- | --- | --- |
| Tarea | Precedencia | Tiempo | Recursos humanos |
| Identificar errores | \_ | 5 | Borja Rodríguez |
| Establecer requisitos | 1 | 6 | Rubén López |
| Diagrama de clases | 2 | 4 | Laura Moreno |
| Prototipo | 3 | 3 | Alejandro Parrilla |
| Modificar código inicial | 4 | 10 | Borja Rodríguez  Jesús Álvarez  Laura Moreno |
| Añadir funcionalidad e interfaz | 4 | 15 | Alejandro Parrilla  Borja Rodríguez |
| Pruebas 1 | 5 | 4 | Rubén López  Alejandro Parrilla |
| Pruebas 2 | 6 | 6 | Rubén López |
| Documentación pruebas | 7, 8 | 11 | Rubén López |
| Presentación | 9 | 3 |  |
| Documentación | - | 50 |  |
| Control de versiones | \_ | 53 |  |

## 2.2. Identificar errores

En un primer lugar, se llevará a cabo la identificación de los errores que presenta el código aportado. Para ello, enumeraremos los siguientes errores encontrados:

1. Nos encontramos que tras cada línea donde se ha querido insertar un salto de línea, en el código, sólo aparece “n” en lugar de “\n”.
2. En el menú inicial, si se inserta un carácter distinto a 1-5 (distintas opciones que nos permite la aplicación), falla y se cierra. En su lugar debería notificar el error y permitir recuperarse.
3. Una vez seleccionada una de las primeras opciones, en el apartado de introducir un operando, tanto el primero como el segundo, no se comprueban los valores de entrada, lo que permite la inserción de caracteres no numéricos como operandos de la calculadora.
4. En los apartados de introducir operando, al seleccionar cancelar se finaliza la aplicación con error. Sin embargo, debería mostrar de nuevo el menú inicial con las operaciones.
5. En el menú principal, ocurre lo mismo que en el error anterior, pero además este botón de cancelar carece de funcionalidad, ya que para salir sería con introducir el valor 5. Por lo que habría que quitar dicho botón o este sustituirlo por “Salir” y quitar esta opción, la 5, de las opciones dadas.
6. De nuevo en el apartado de introducir un apartado, si pulsas sin haber insertado nada, la aplicación se cierra con un error en lugar de esperar a que se inserte algún valor para el operando.
7. Por último, al pulsar sobre  se notifica un error y esto no debería ser así, sino que se debería cerrar la aplicación sin error alguno.

## 2.3. Establecer requisitos

**1. Requisitos Funcionales (**funciones que realizará el sistema, dando soporte al usuario**)**

* **RF01** El cliente requiere de una aplicación con interfaz gráfica que permita llevar a cabo operaciones aritméticas básicas, donde se respete en todo momento un jerarquía de operaciones con y sin paréntesis.
* **RF02** Como requisito básico e inicial, la calculadora debe realizar las funcionalidades aritméticas más básicas (suma, resta, multiplicación y división).
* **RF03** El cliente desea poder ampliar la funcionalidad básica de la aplicación permitiendo poder cumplir una jerarquía de operaciones a través de paréntesis.
* **RF04** La aplicación evaluará las expresiones tras esperar a que el usuario las introduzca de manera completa.
* **RF05** La aplicación debe impedir que se introduzcan letras que puedan dar lugar a errores en la aplicación.
* **RF06** Por su parte, en caso de error, la aplicación debe mostrar por pantalla el error cometido y permitir al usuario recuperarse del mismo, introduciendo otro carácter que sí sea válido o retrocediendo al estado anterior que dio lugar al error.
* **RF07** La aplicación debe tener consistencia, evitando que el usuario cometa errores tales como salir de la aplicación cuando no lo desee u operar con caracteres inválidos.

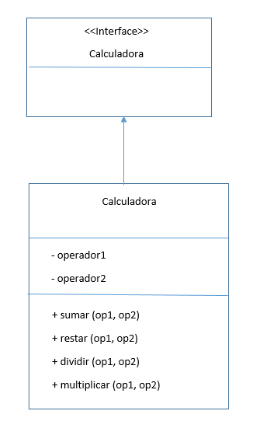
**2. Requisitos No Funcionales (**restricciones de diseño y de implementación**)**

* **RNF01** El lenguaje de programación que se utilizará en el desarrollo de la aplicación será el lenguaje Java.
* **RNF02** La distribución de los botones de la calculadora en la interfaz gráfica estará basada en la misma distribución que podemos encontrar en la calculadora de Windows o en los teclados de los ordenadores.
* **RNF03** Los colores utilizados en la interfaz deberán ser claros y sobrios.
* **RNF04** El cliente tiene como requisito el añadido del icono de la Universidad Rey Juan Carlos en la propia interfaz gráfica de la aplicación.

## 2.4. Diagrama de clases

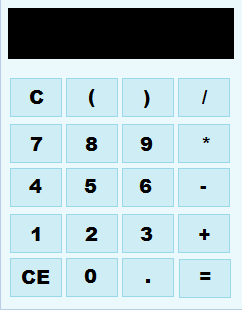
Usamos el diagrama de clases antes de la implementación de nuestra calculadora  para visualizar las relaciones establecidas entre las clases. En este caso tendremos una clase llamada calculadora y otra interfaz, que será una interfaz gráfica.

En la clase calculadora tendremos dos atributos, que serán dos operadores y cuatro métodos que serán las operaciones aritméticas de sumar, restar, dividir y multiplicar.



## 2.5. Realizar prototipo de baja fidelidad

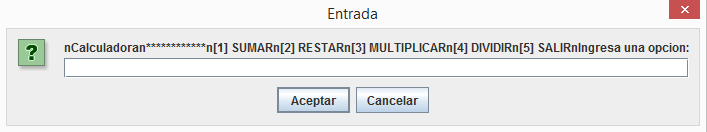
El prototipo de bajo fidelidad se realizará para poder presentar a los usuarios de la aplicación nuestra calculadora de forma visual y que nos permita realizar posibles modificaciones en la aplicación de manera más cómoda y sencilla.

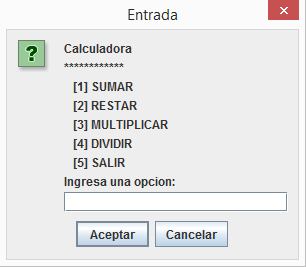
**

## 2.6. Corregir código según los errores identificados

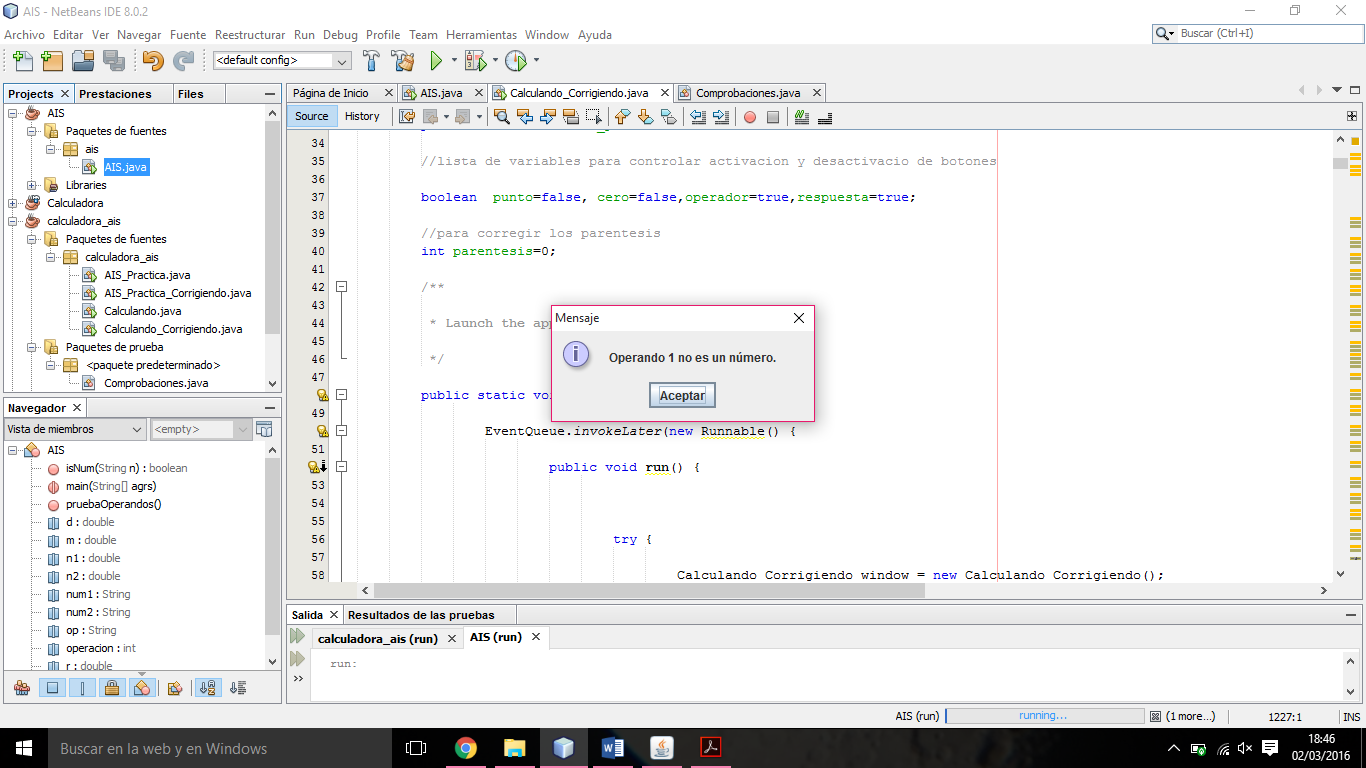
A continuación, se trabajará en la depuración y corrección del código, con el fin de dejarlo sin errores.

* **Error de salto de línea.** *En el primer pantallazo veremos como se muestra dicha aplicación sin haber solucionado el error, mientras que en el segundo pantallazo ya se ha solucionado cambiando el código, lo cual nos permite diferenciar las partes del menú de manera más clara.*



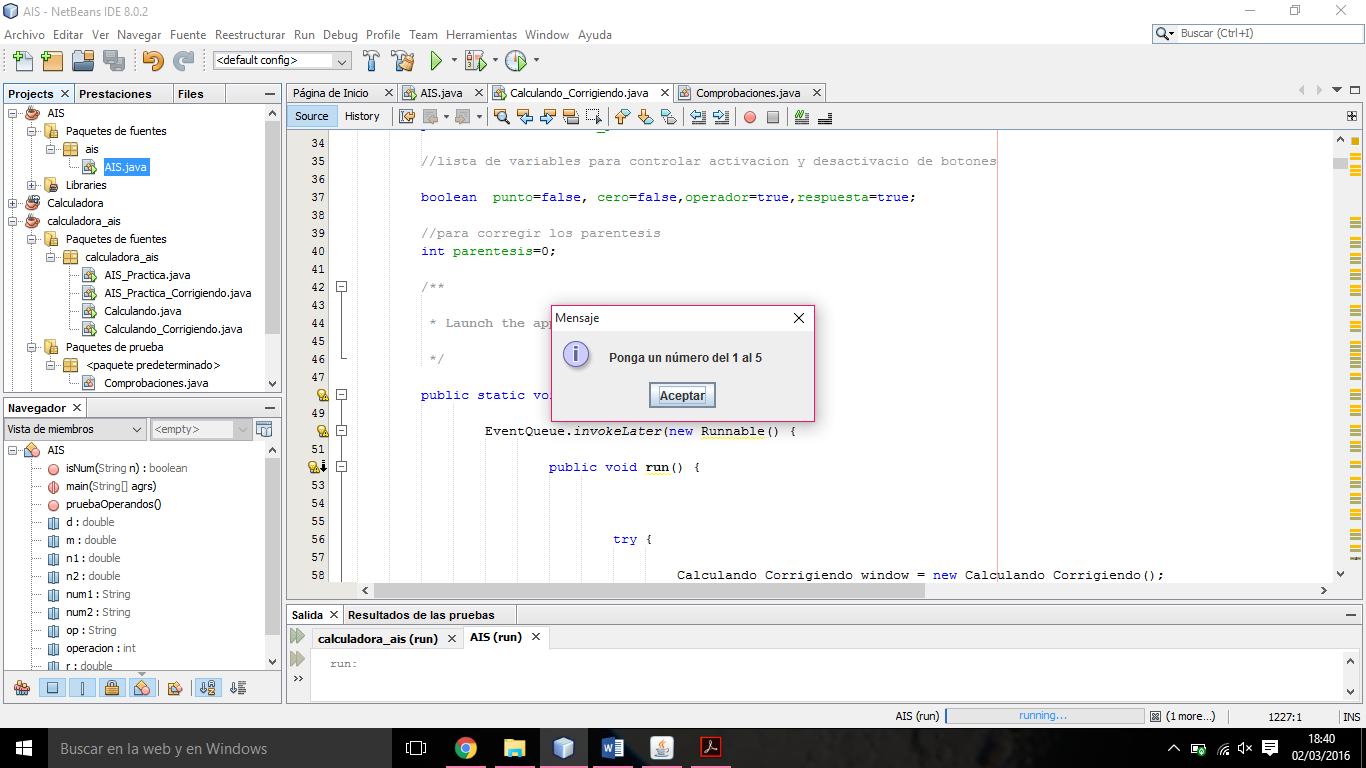


* **Error en la introducción del carácter del menú principal.** *En este caso, si en el menú principal introducimos algo distinto a un número del 1 al 5 nuestra calculadora se cierra y da un error. Para corregir este fallo hemos añadido una restricción para que cuando no se seleccione la opción adecuada, la aplicación nos informe de qué es lo que hemos hecho mal. Con esta corrección quedaría de la siguiente manera si introduzco, por ejemplo, una a:*



* **Error en la inserción de caracteres no numéricos como operandos.** *Aquí nos encontramos con que una vez seleccionada la operación a realizar, nuestra calculadora nos pide dos operandos y no comprueba que ambos sean numéricos. Para solventar este problema hemos añadido una restricción, similar a la del error anterior, que nos notifique cuando lo que introducimos no es un número y nos lo vuelva a pedir de nuevo. La operación no podrá realizarse hasta que ambos operandos sean caracteres numéricos.*

*Si en el primer operando introducimos una letra, la aplicación nos muestra el siguiente mensaje:*

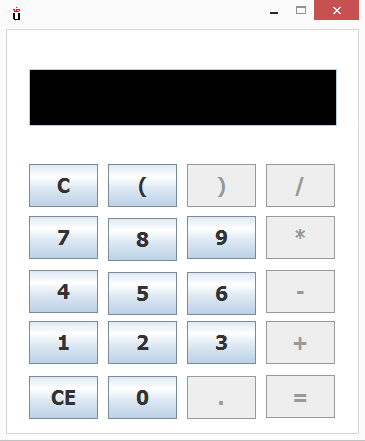


## 2.7. Añadir nueva funcionalidad

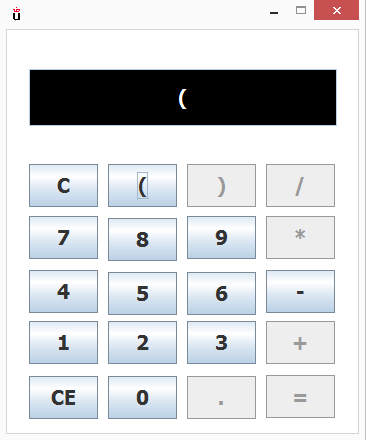
Una vez realizado el prototipo y de manera contigua a la corrección del código inicial, se ampliará la funcionalidad básica para dar lugar a una que incluya una jerarquía de operaciones.

Para poder realizar esta nueva funcionalidad, en primer lugar, con el fin de prevenir errores, se ha decidido el uso de botones que podrán estar activados o desactivados. A continuación diferenciaremos entre distintos casos, en todos estos casos también están activos siempre el botón de borrar el elemento anterior *CE* y el de borrar todo *C*:

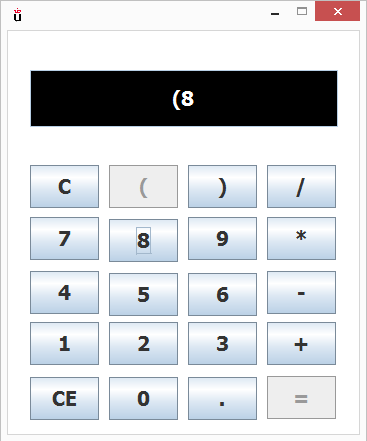
* **Inicio**: inicialmente solo podremos tener activos los botones de dígitos y el paréntesis de apertura, mientras que el resto estarán desactivados. Así evitamos que el usuario de la aplicación empiece, por ejemplo, con */* o con *)* lo cual no tendría sentido.



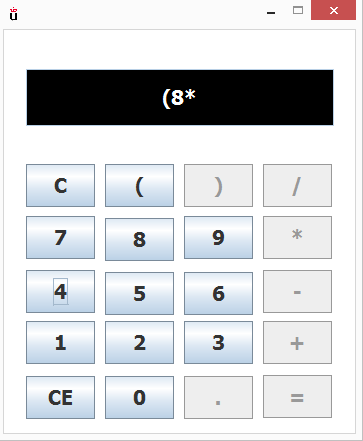
* **Tras *(*** : como es lógico, tras *(* solo podremos tener activos los dígitos y el símbolo *–* para el caso en el que queramos incluir un valor negativo.



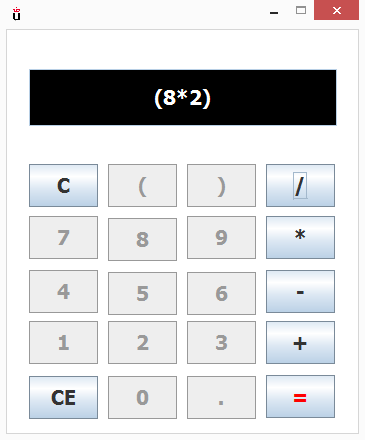
* **Tras dígito:** este es el caso en el que más botones hay activos, ya que tras un dígito podremos añadir otro dígito, un símbolo de operación o *)* en caso en el que haya un paréntesis de apertura *(* previo, ocasionalmente también podrá estar activo el símbolo *=*, pero este caso lo trataremos después.



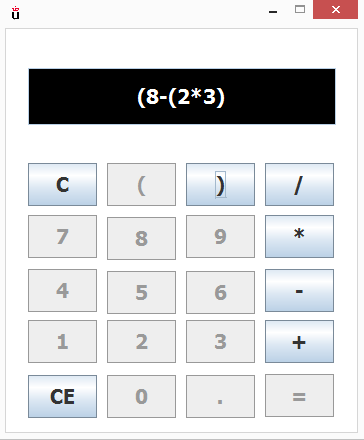
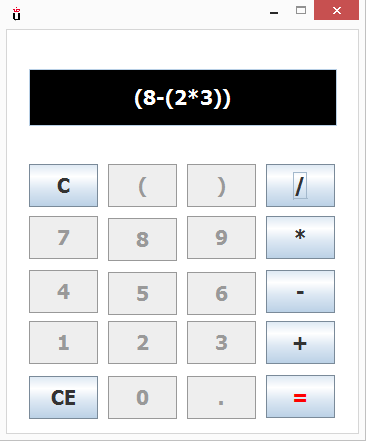
* **Tras símbolo de operación:** por último, en este caso dispondremos activos los botones de dígitos y la apertura de paréntesis *(.*



* **Tras *)* :** en este caso estarán activos los símbolos de operaciones *+, -, \** y */*, ocasionalmente también podrá estar activo el símbolo *=*, pero este caso lo trataremos después.



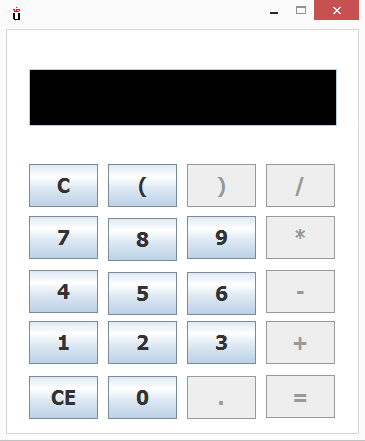
Por otro lado, necesitamos identificar los paréntesis y garantizar que estén equilibrados antes de devolver el resultado final. Con el fin de prevenir errores en la utilización de la aplicación, “obligaremos” a que el usuario cierre todos los paréntesis que abrió. Para garantizar esto, como podremos observar en la primera imagen, el símbolo “=” no se nos permitirá utilizarlo hasta tener equilibrados los paréntesis (como se observa en la segunda imagen).

Para ello, dispondremos de una variable entera “paréntesis” que nos permitirá saber si se cumple el equilibrio de paréntesis, ya que esta variable se incrementará en uno por cada paréntesis de apertura y se disminuye en uno por cada paréntesis de cierre, por lo que cuando el valor de dicha variable sea igual a cero, sabremos que la operación tiene tantos paréntesis de apertura como de cierre.

## 2.8. Realizar interfaz

Finalizado la tarea anterior, se implementará la interfaz adecuada, con apariencia de una calculadora como la realizada en el prototipo de baja fidelidad, para poder llevar a cabo las funciones requeridas de manera eficiente y cómoda por los usuarios finales de la aplicación.



# 3. Control de versiones

## 3.1. Herramienta usada

La herramienta que hemos usado para realizar el control de versiones es GitHub. En primer lugar nos tenemos que crear un repositorio instalando Git en nuestro ordenador.

## 3.2. Conflicto

Puesto que durante la realización del proyecto no nos encontramos con ningún conflicto, hemos tenido que provocarlo en dos situaciones.

**Primer conflicto**

**Segundo conflicto**

# 4. Pruebas

Una vez finalizado todo el proceso de codificación, se realizarán una serie de pruebas sobre la propia aplicación con el objetivo de advertir posibles errores no solventados y comprobar que se cumplen las funcionalidades básicas requeridas por los usuarios de la aplicación. Estas funciones básicas se pueden enumerar de la siguiente manera:

* Suma
* Resta
* División
* Multiplicación

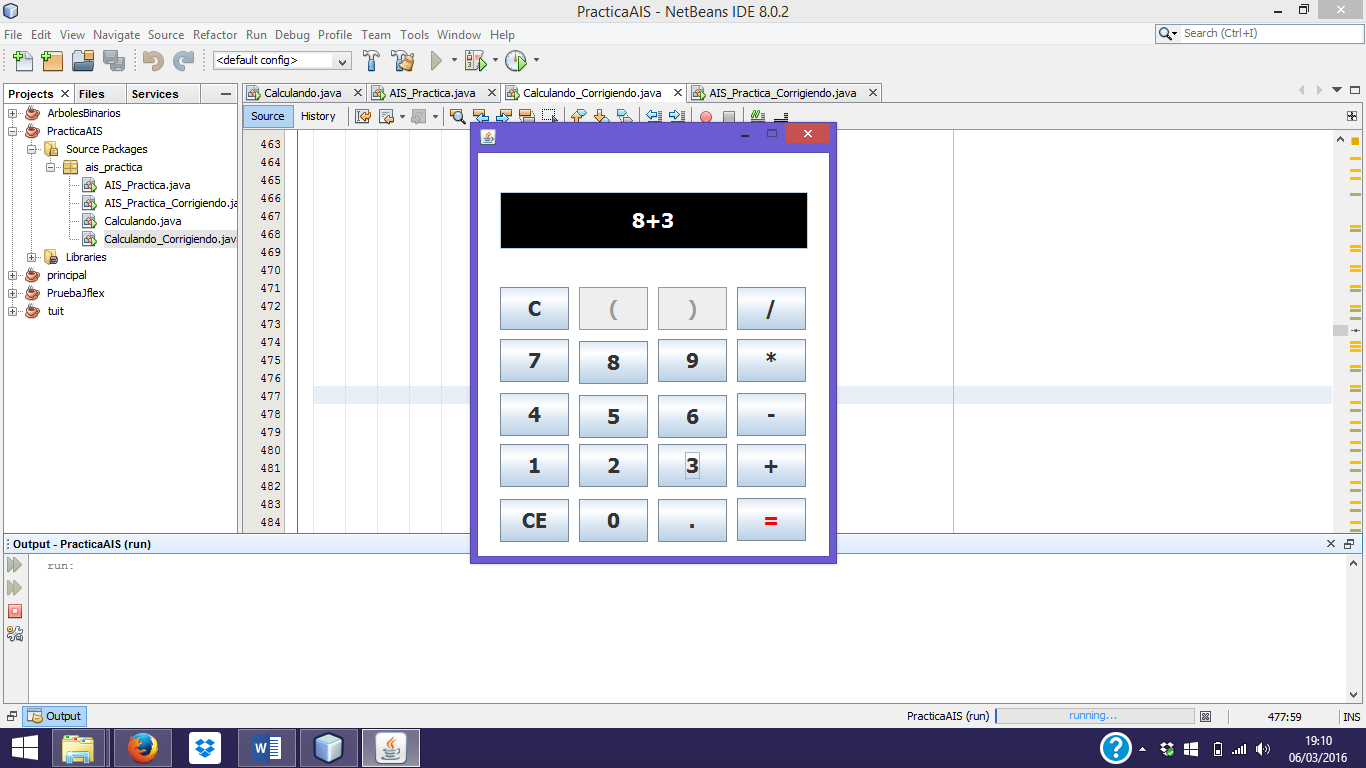
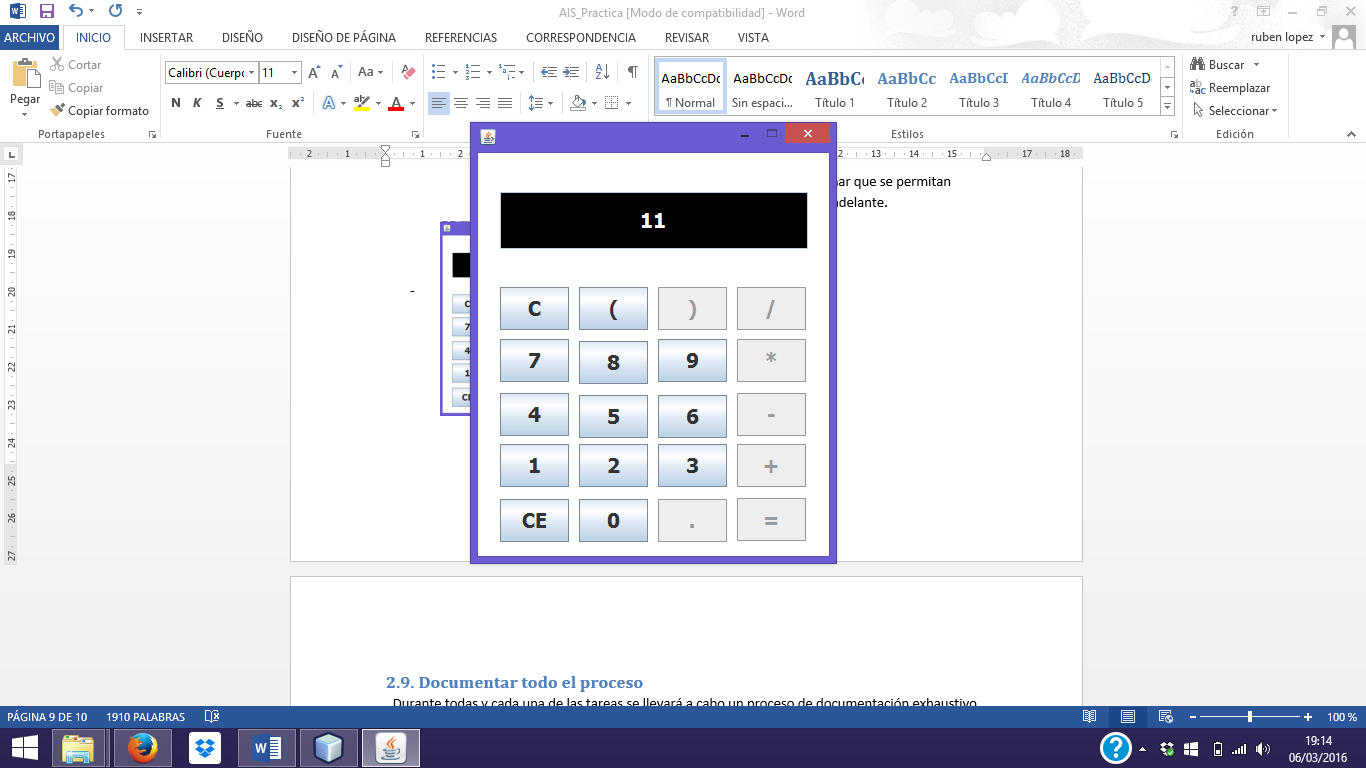
Si esta tarea se realiza con éxito, podremos finalmente presentar el producto final a los usuarios. En caso contrario, se volverá a tareas anteriores para arreglar errores de código o completar funciones del mismo.

Las pruebas que hemos realizado sobre nuestra calculadora consisten en realizar una serie de operaciones aritméticas con el objetivo de comprobar si son realizadas con éxito y en el aseguramiento de que nuestros operandos no pueden ser caracteres alfabéticos. Éste último está completamente asegurado en nuestra interfaz, puesto que no admite la posibilidad de pulsar otro carácter que no sea un digito.

En cuanto a la comprobación de las distintas operaciones tenemos lo siguiente:

#### Suma sin paréntesis

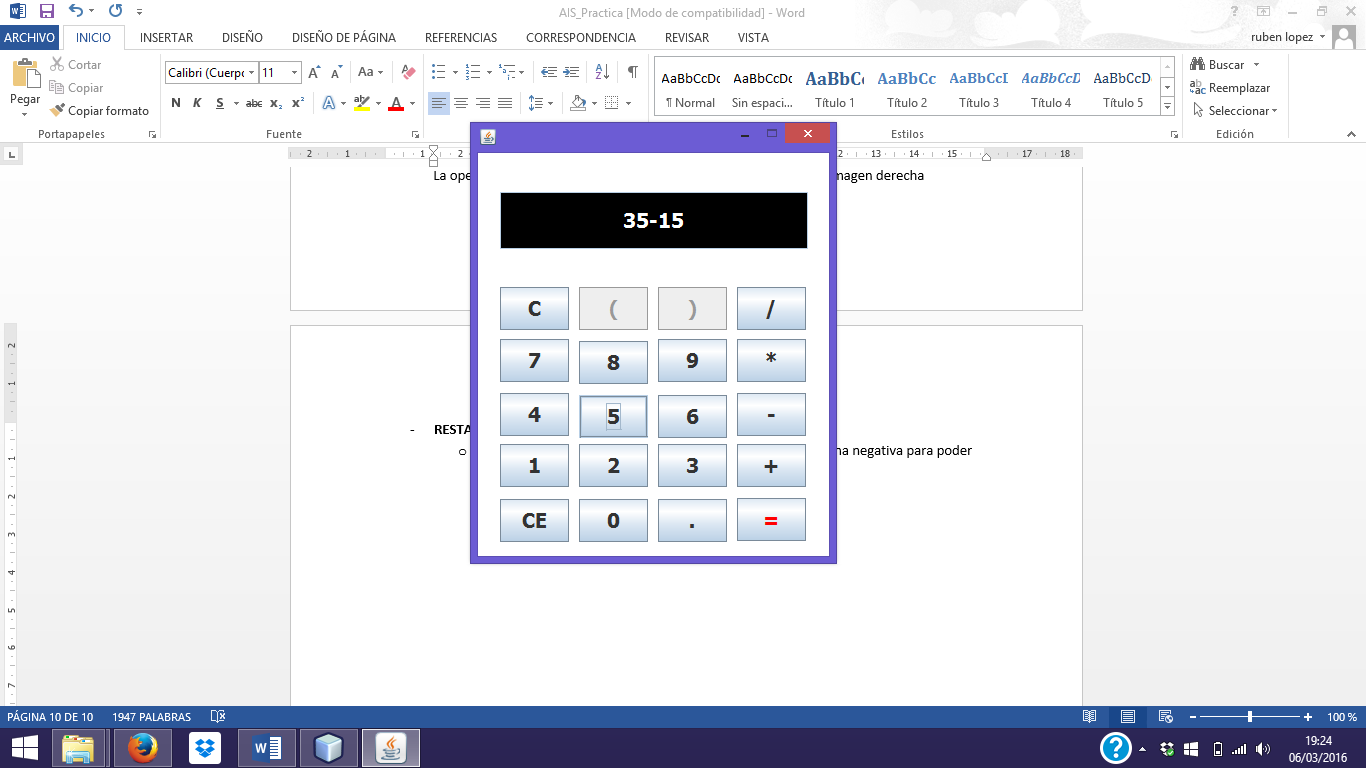
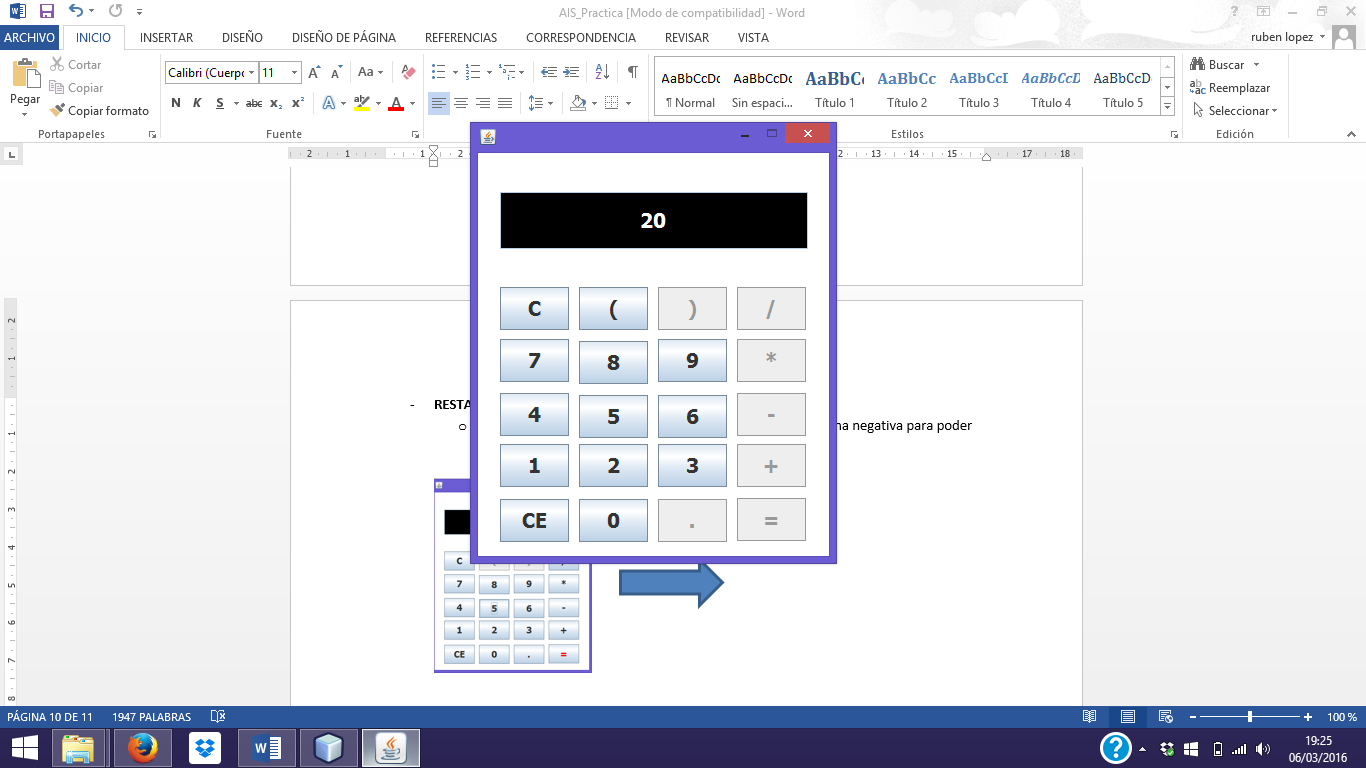
En esta y en las sucesivas pruebas sobre operaciones aritméticas únicamente evaluaremos que la operación se realiza con éxito, sin estimar que se permitan introducir caracteres inválidos. Esas pruebas se harán más adelante.

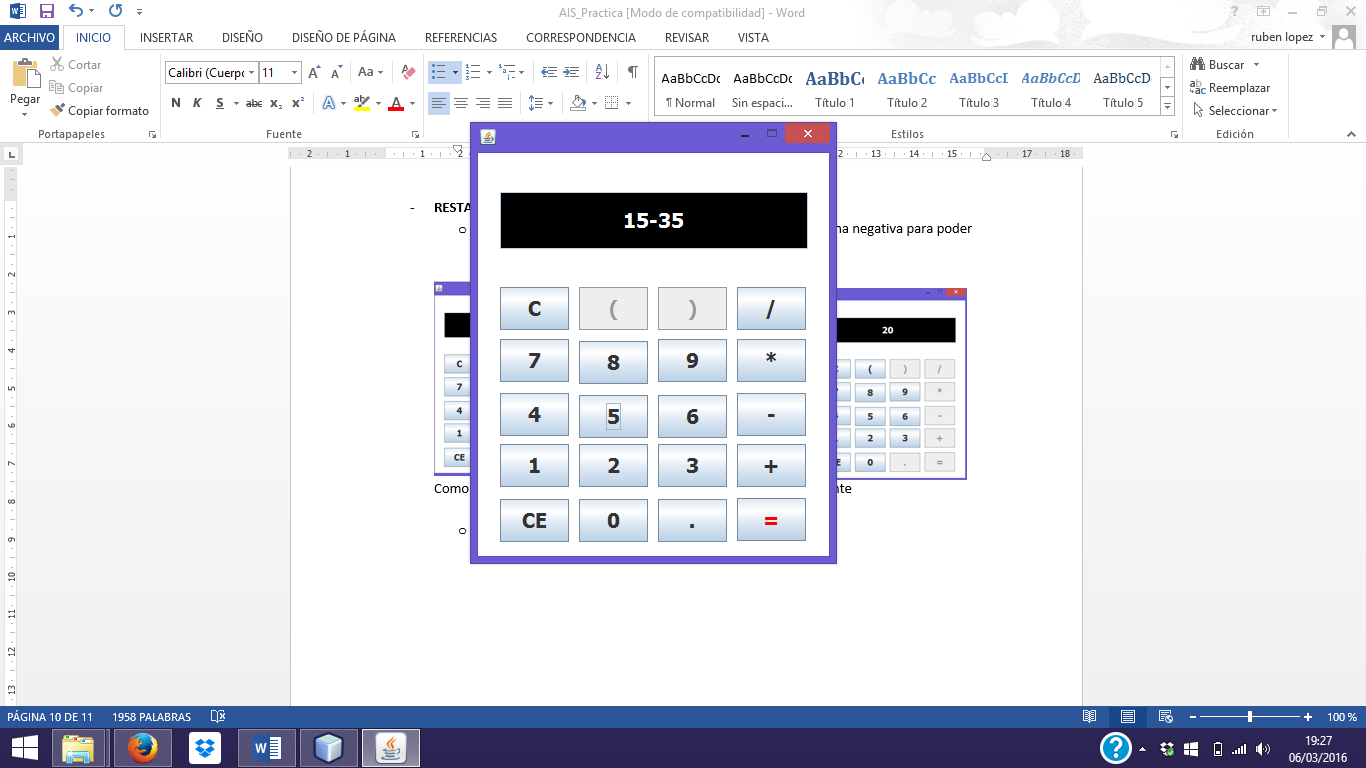
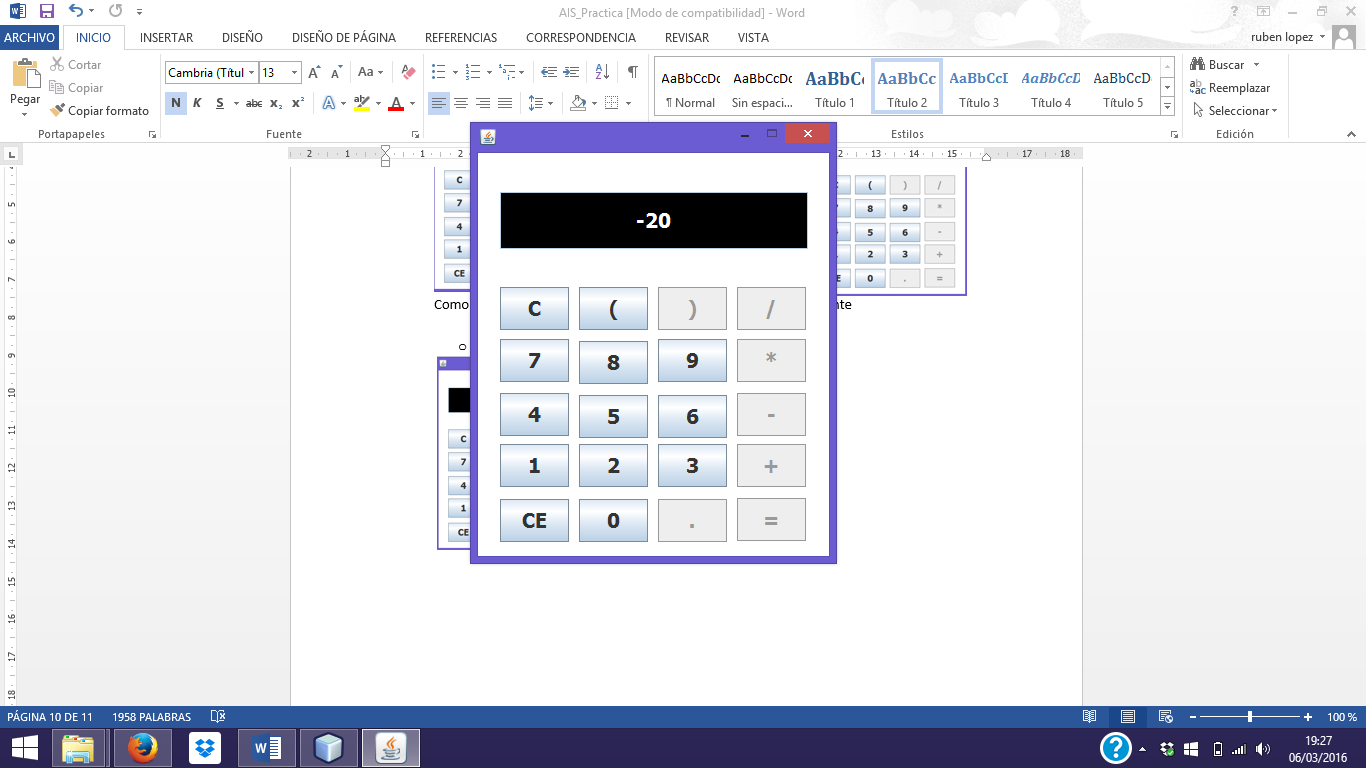
La operación se realiza correctamente, como bien se aprecia en la imagen derecha

#### Resta sin paréntesis

En este caso mostraremos dos casos. Una resta positiva y una negativa para poder observar si mantiene los criterios matemáticos.

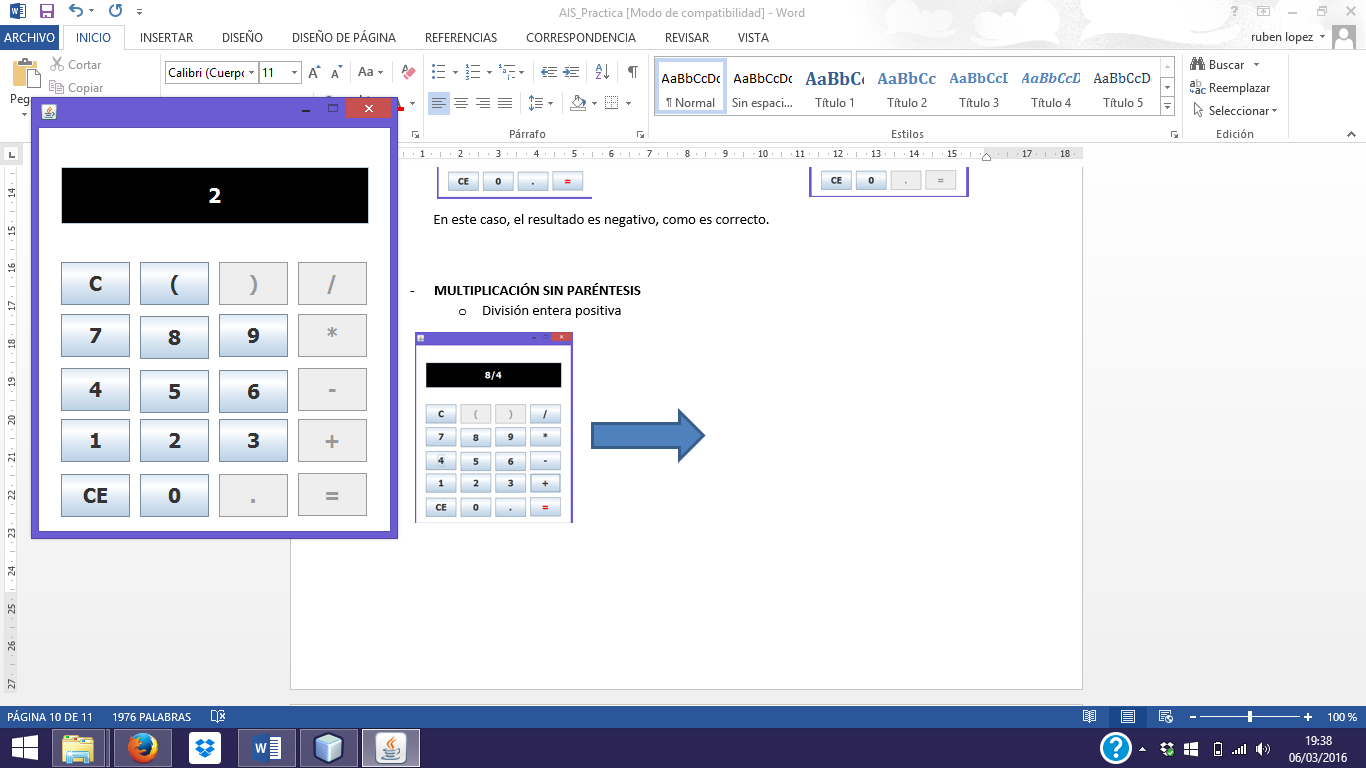
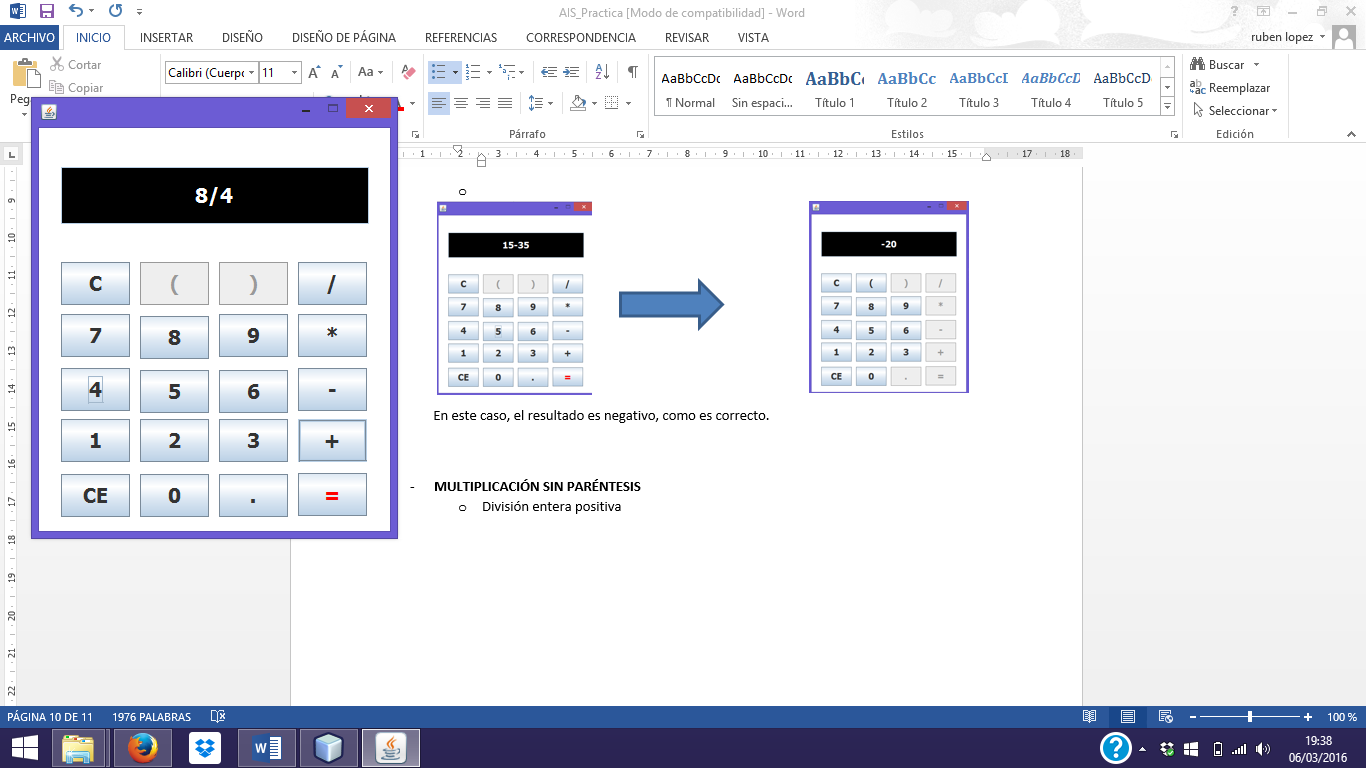
Como podemos ver, le resta positiva se ha completado correctamente.

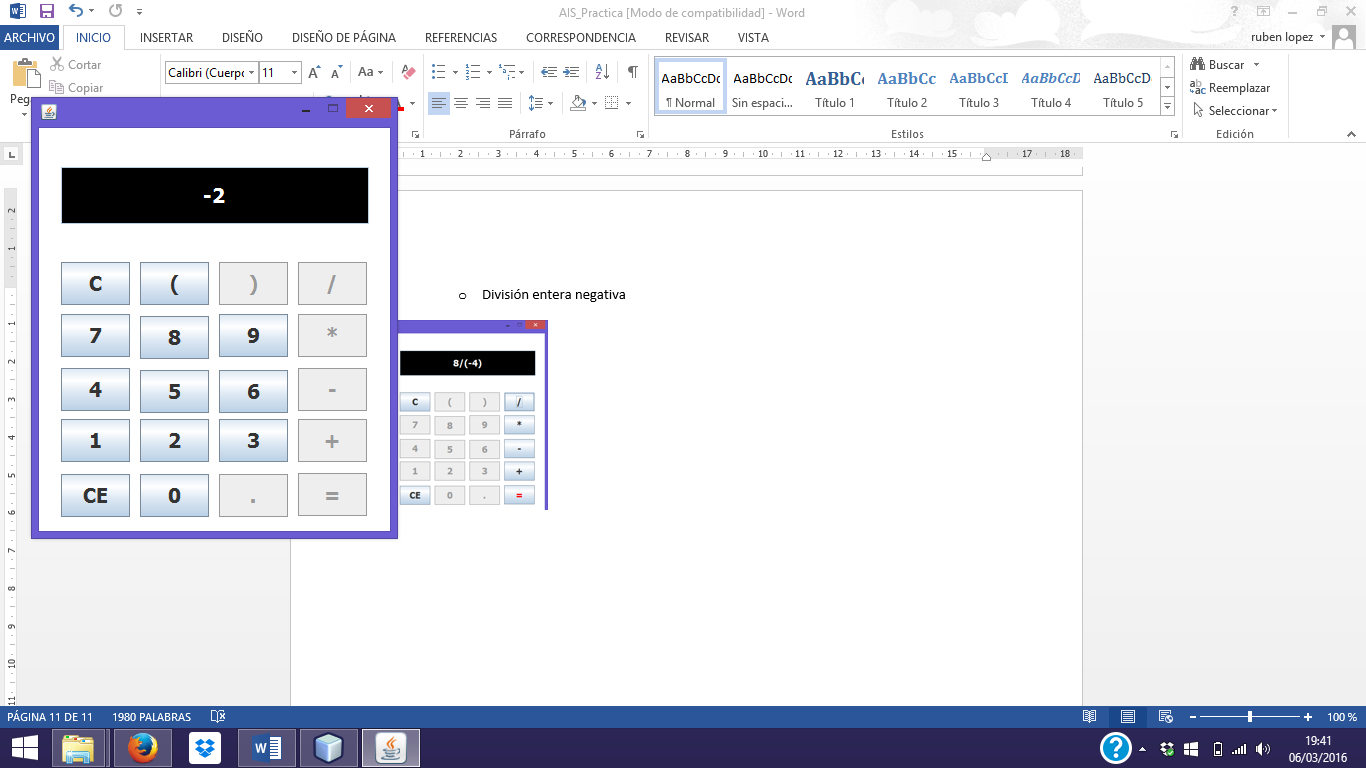
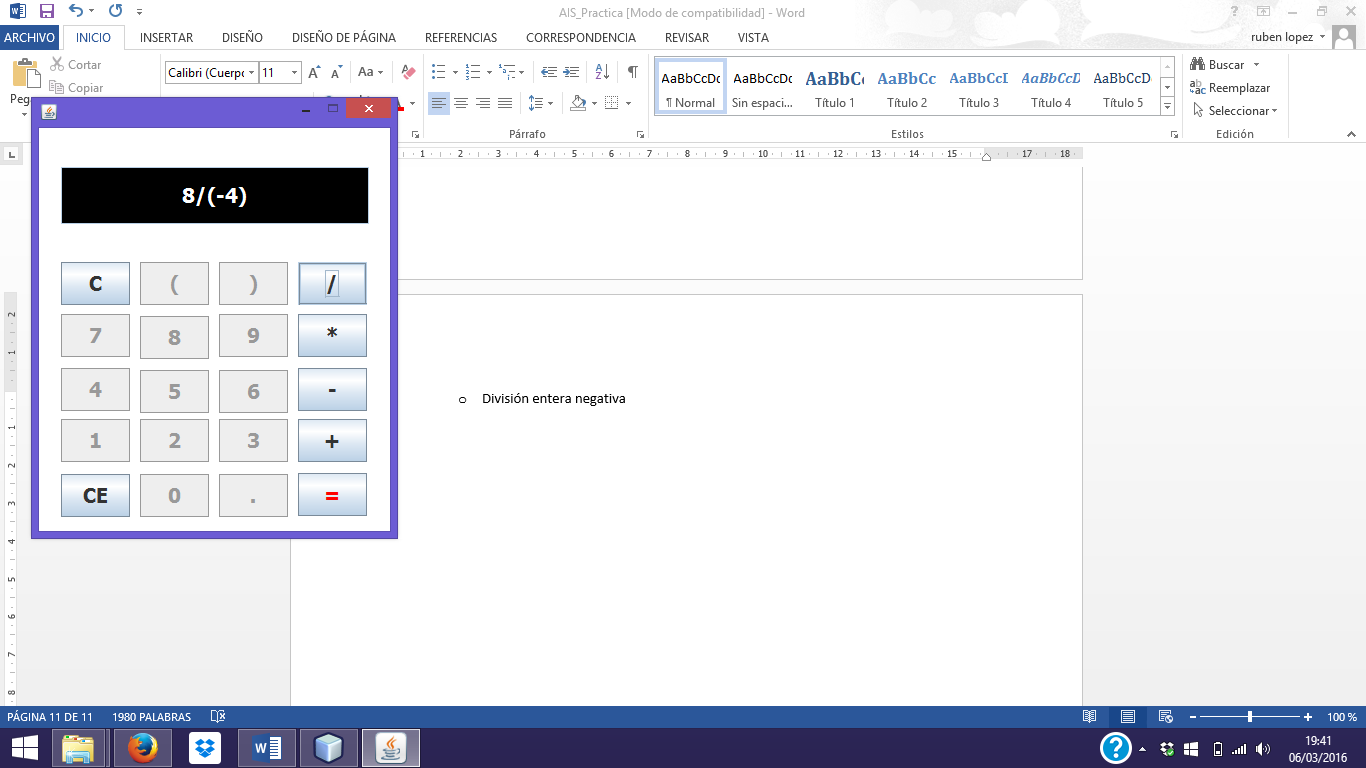
En este caso, el resultado es negativo, como es correcto.

#### División sin paréntesis

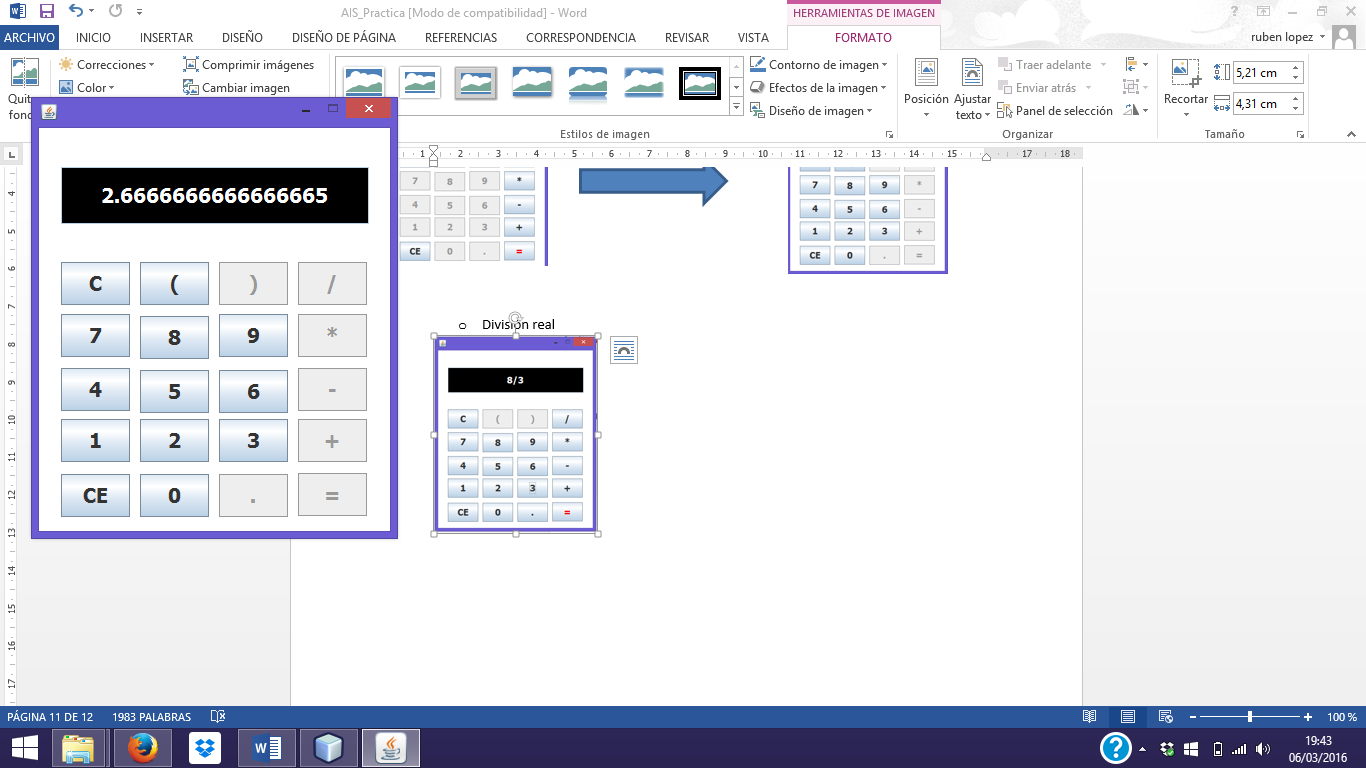
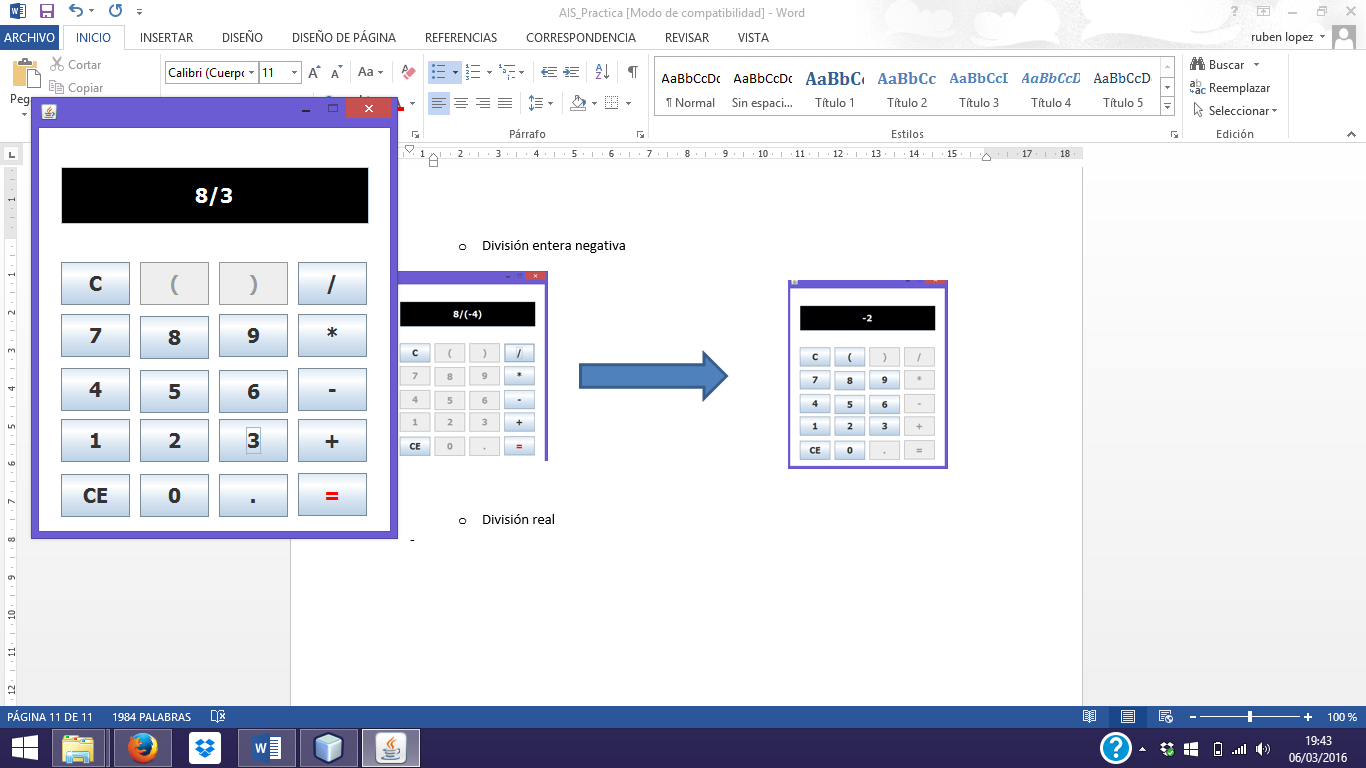
**División entera positiva**



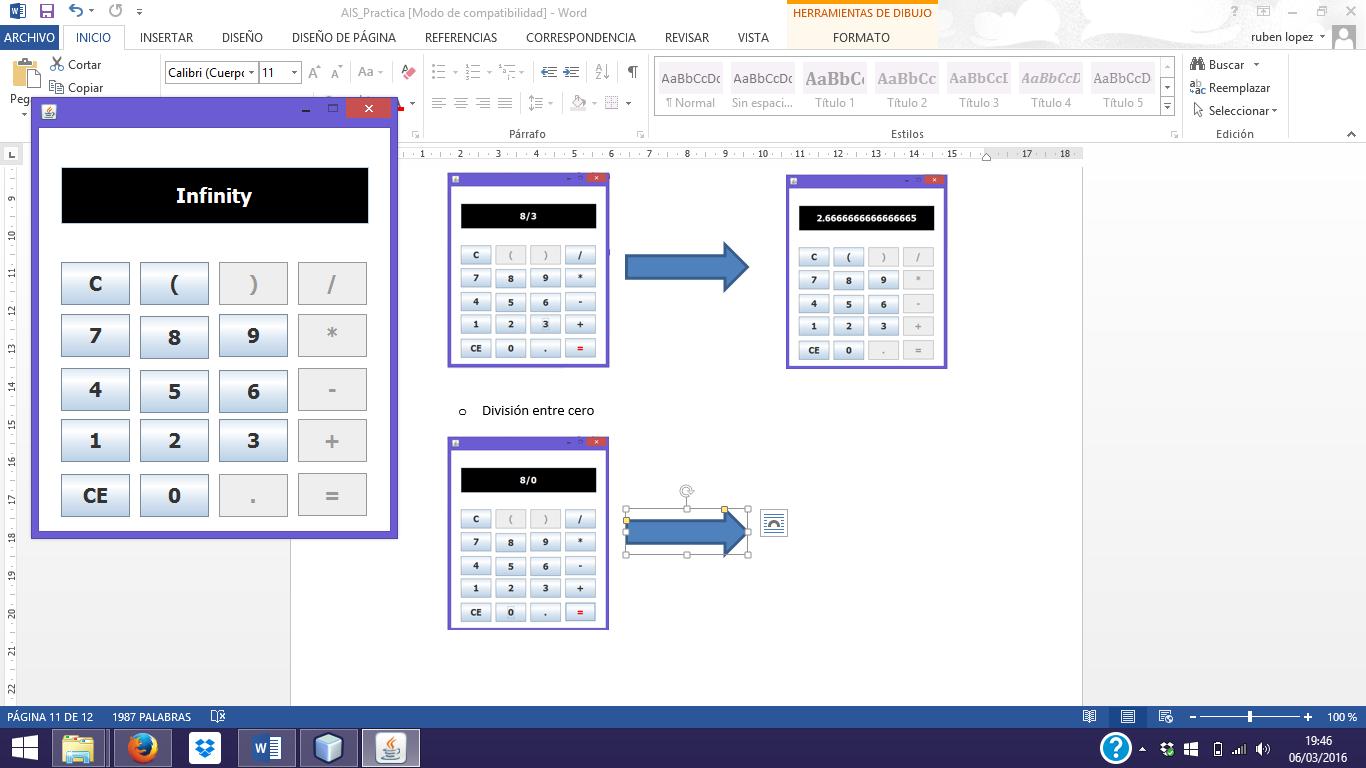
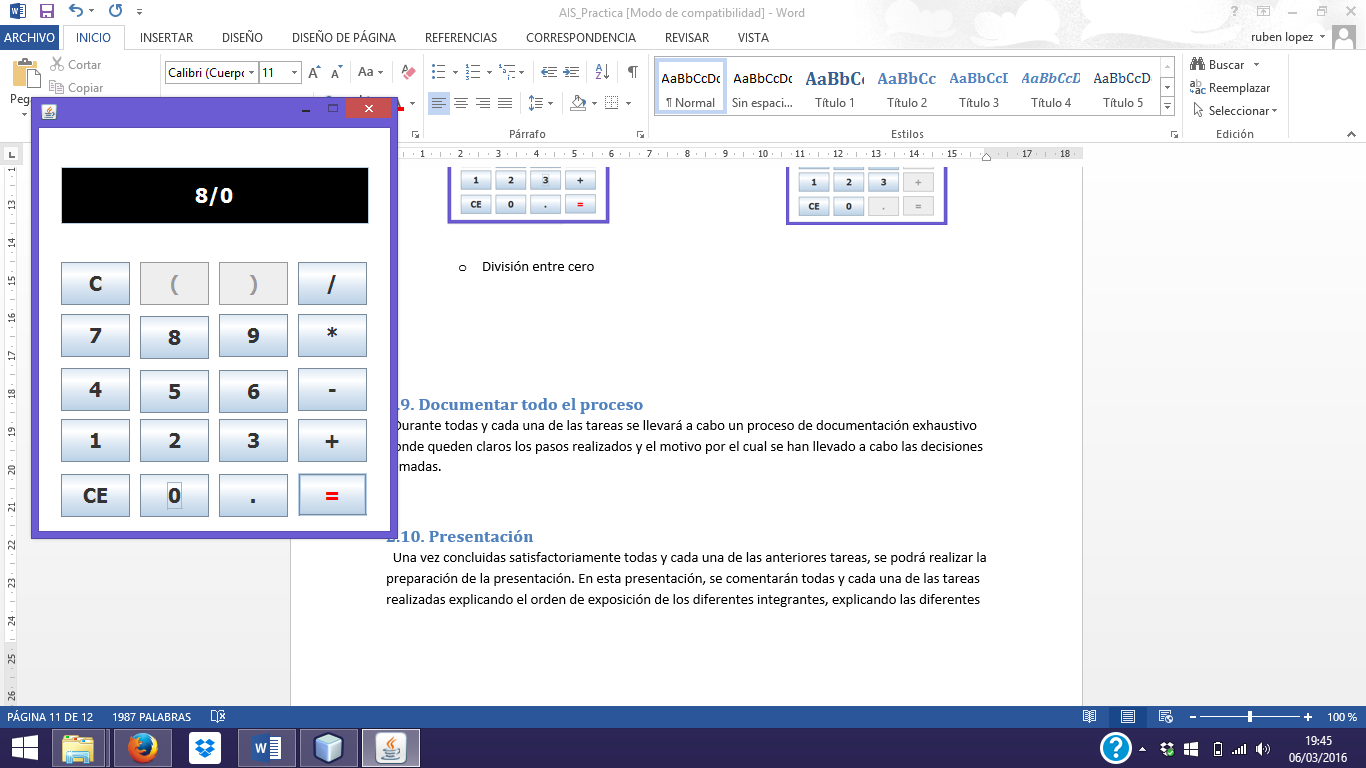
**División entera negativa**



**División real**



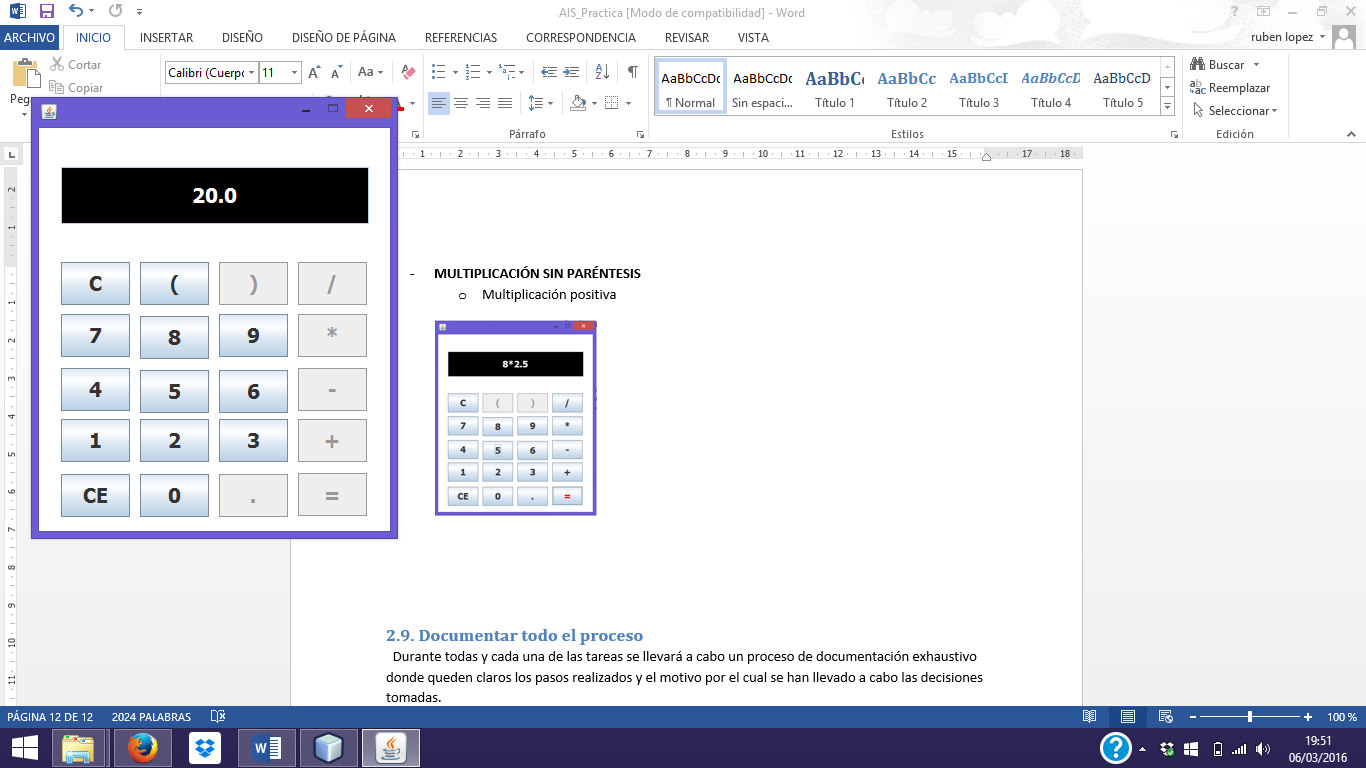
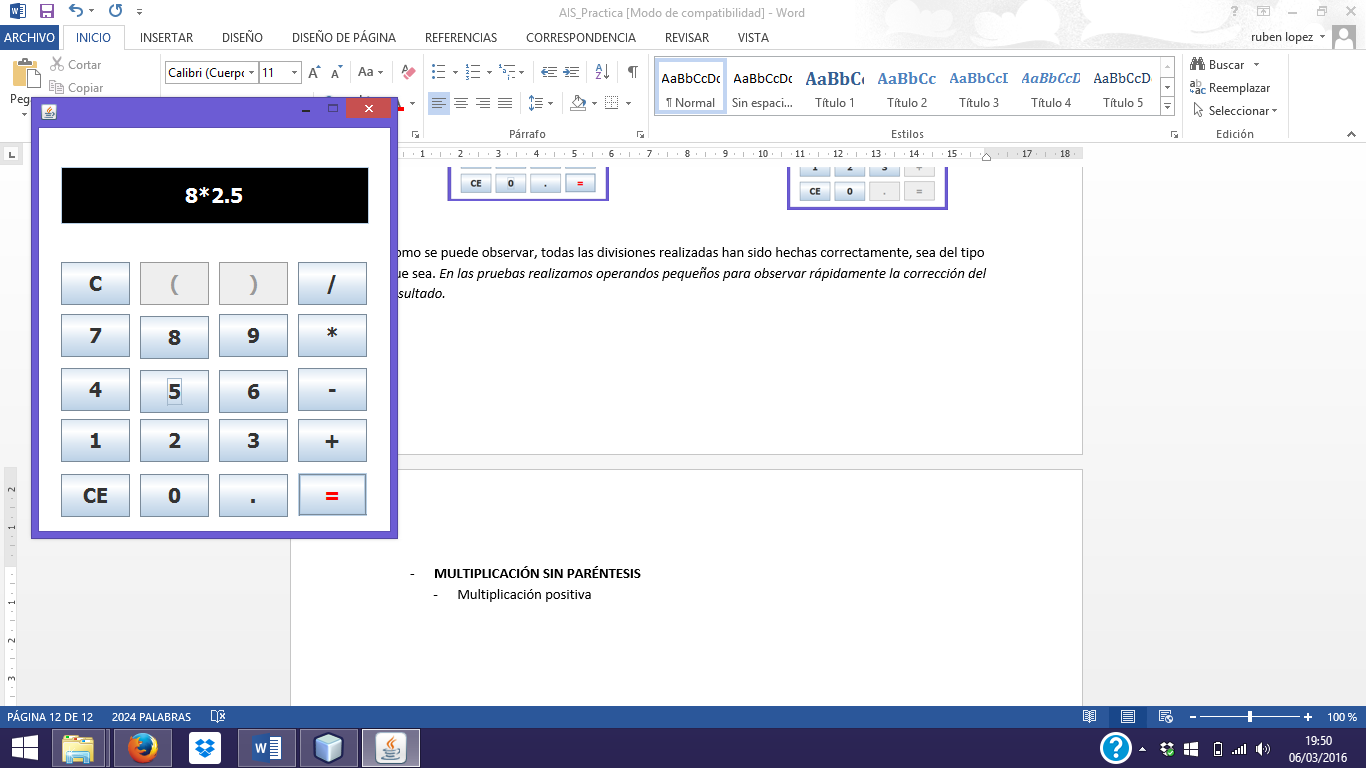
**División entre cero**



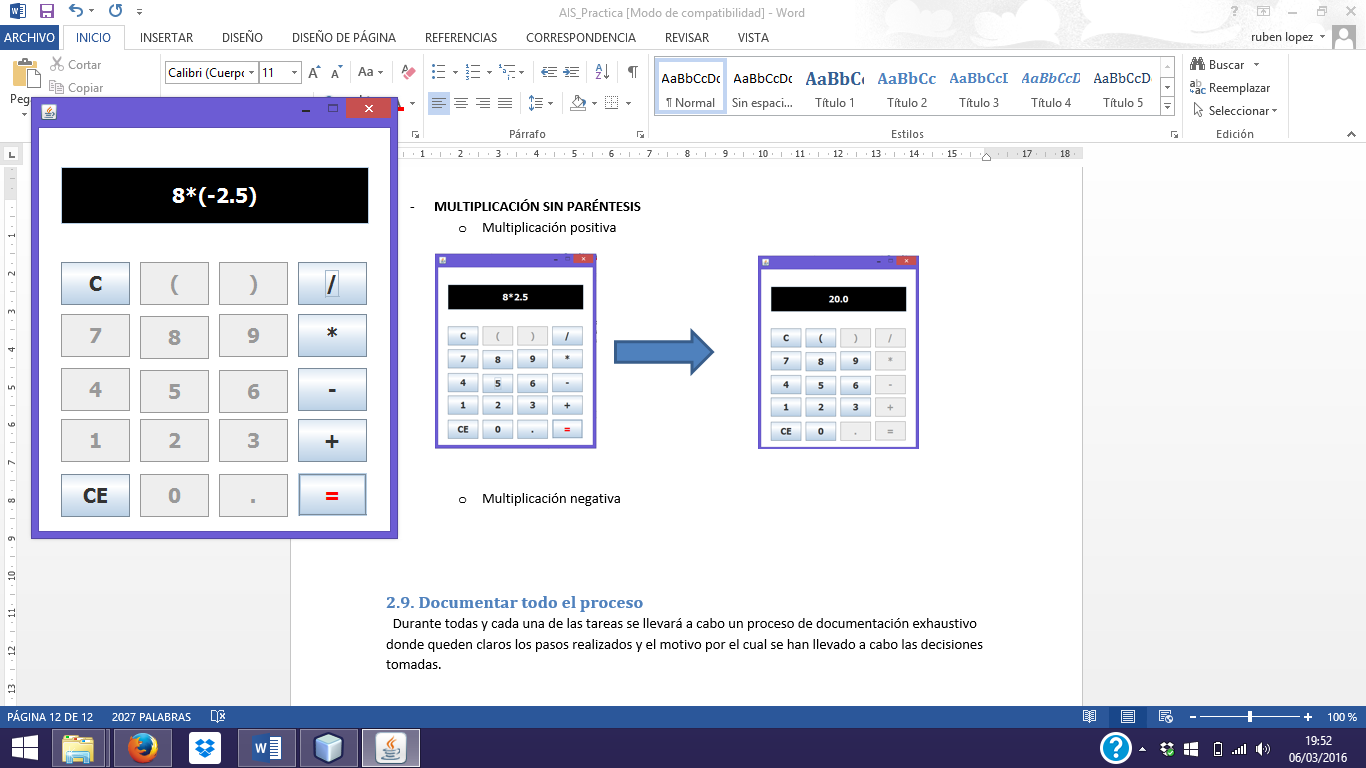
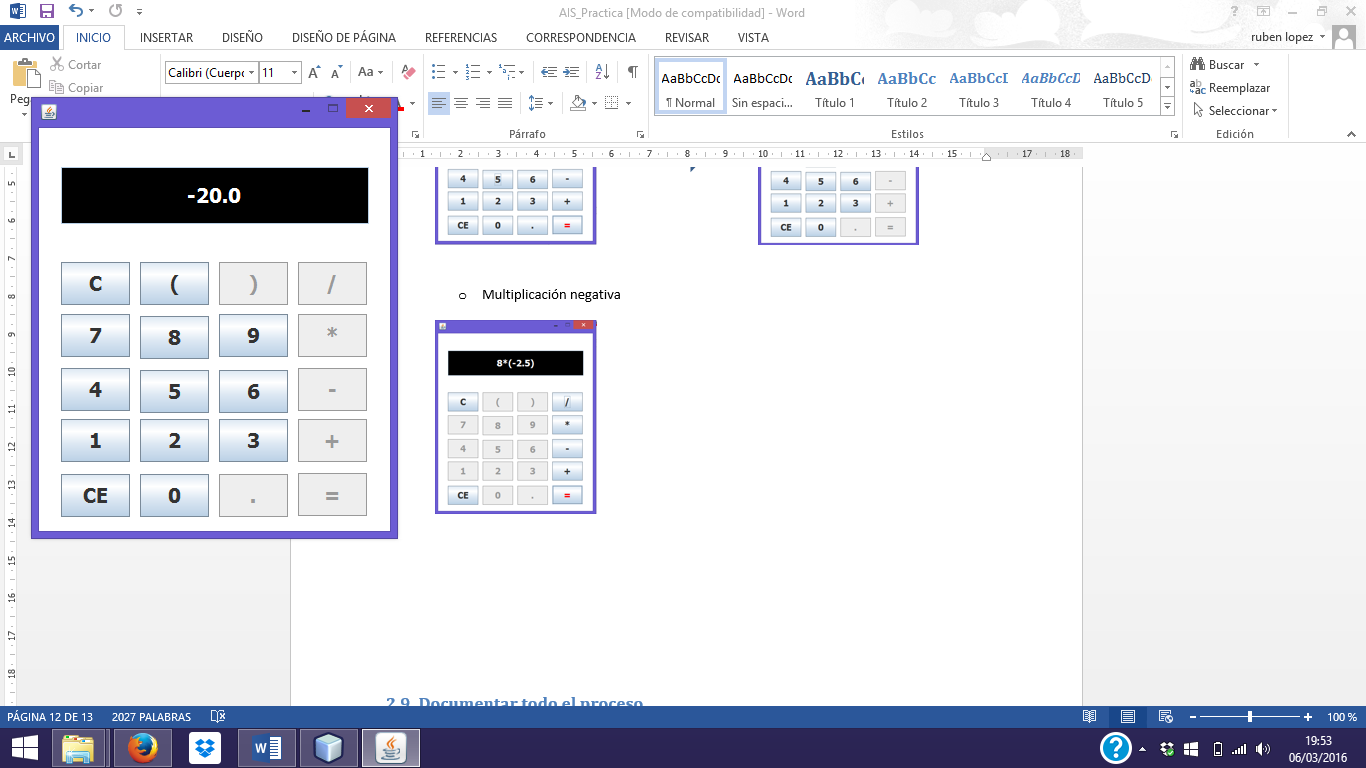
Como se puede observar, todas las divisiones realizadas han sido hechas correctamente, sea del tipo que sea. *En las pruebas realizamos operandos pequeños para observar rápidamente la corrección del resultado.*

#### Multiplicación sin paréntesis

**Multiplicación positiva**



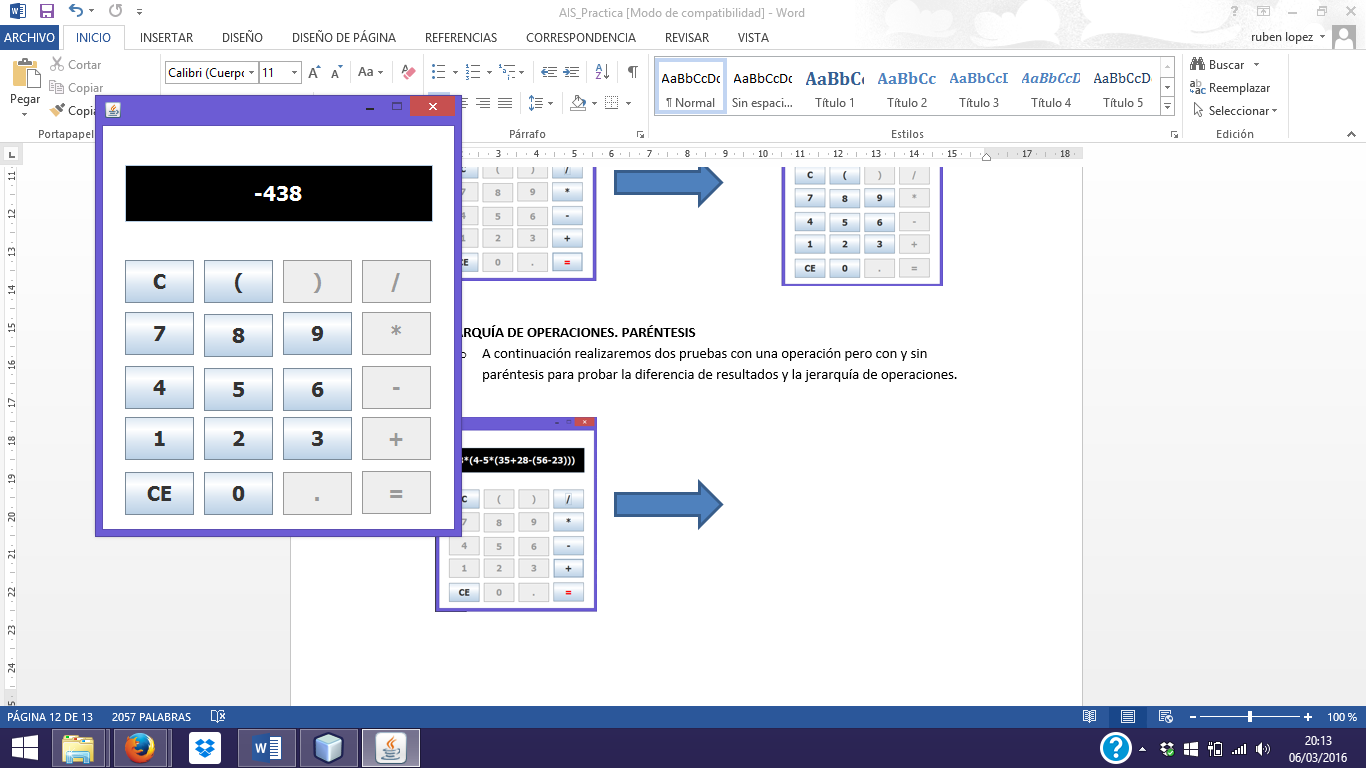
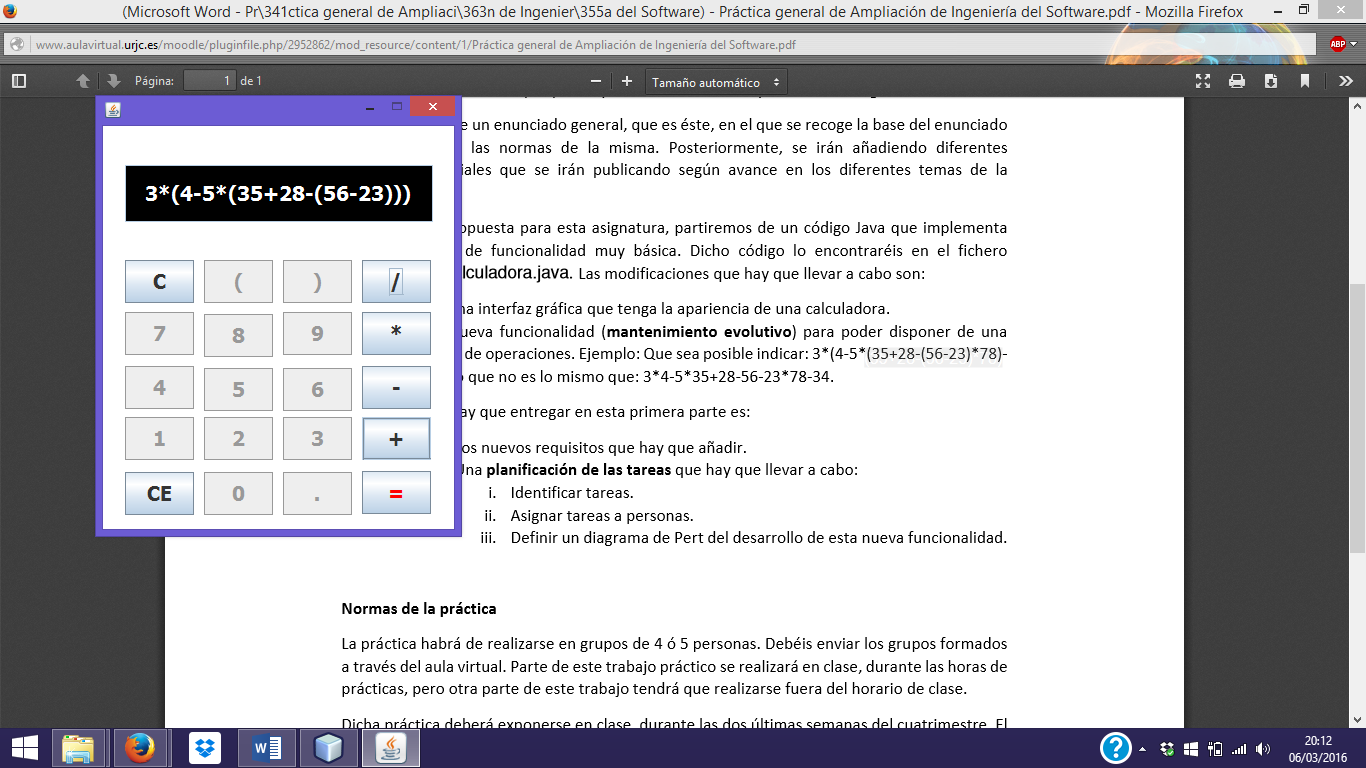
**Multiplicación negativa**

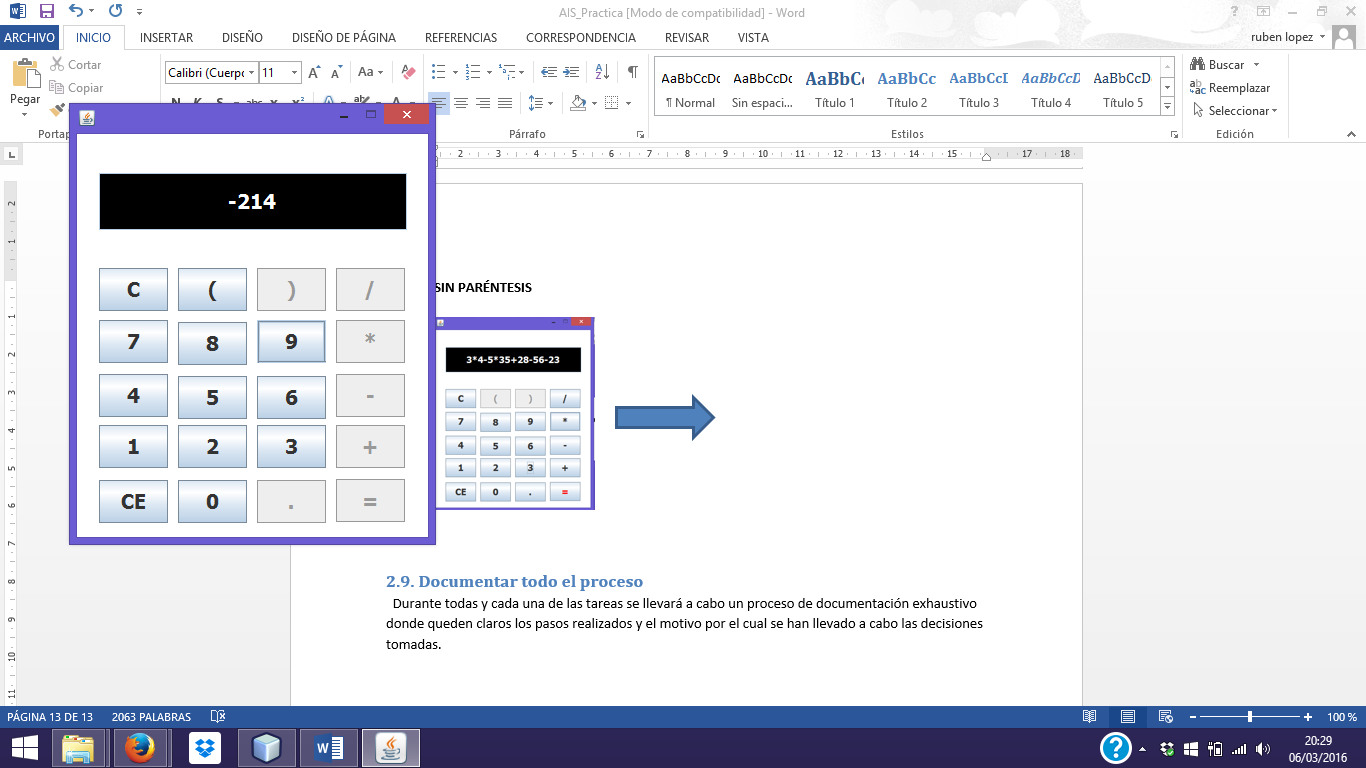
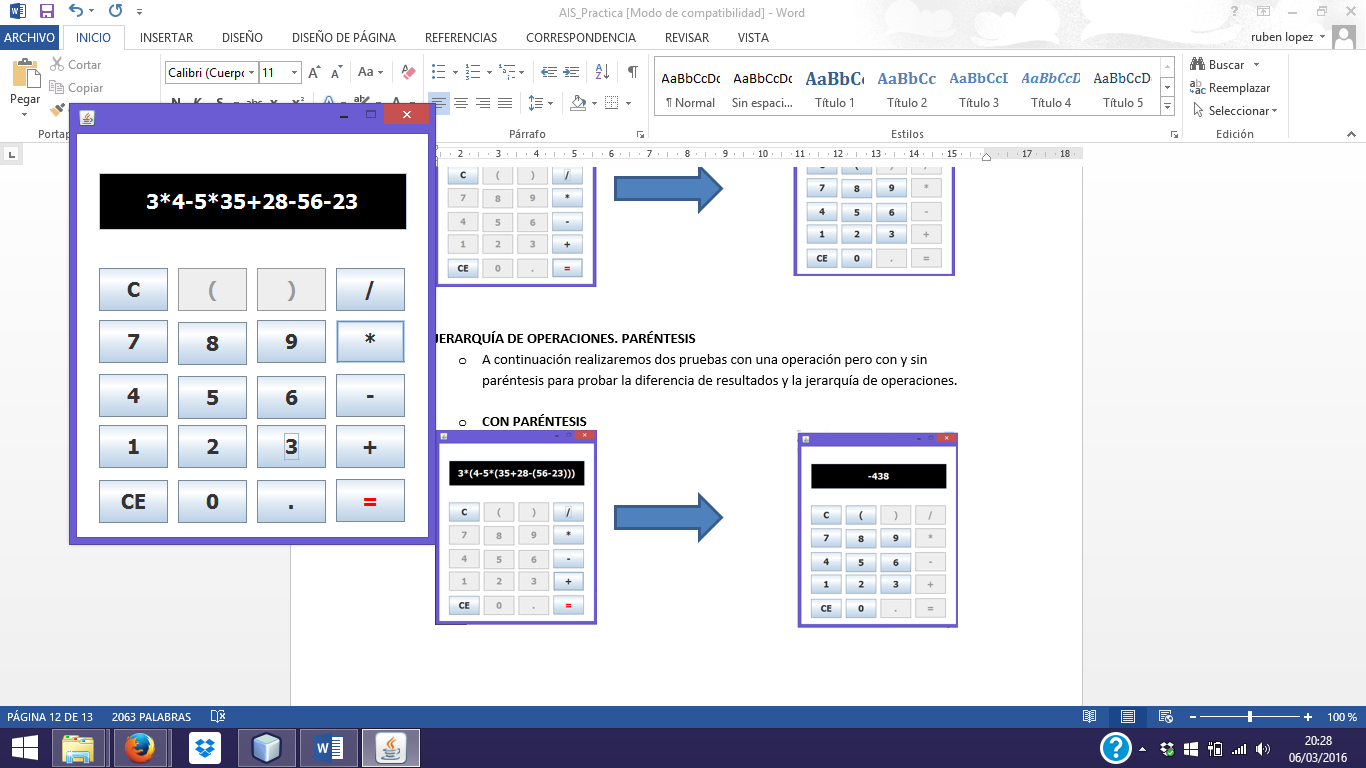
#### Jerarquía de operaciones. Paréntesis.

A continuación realizaremos dos pruebas con una operación pero con y sin paréntesis para probar la diferencia de resultados y la jerarquía de operaciones.

**Con paréntesis**



**Sin paréntesis**



## 

YO QUITARÍA ESTO

## Documentar todo el proceso

Durante todas y cada una de las tareas se llevará a cabo un proceso de documentación exhaustivo donde queden claros los pasos realizados y el motivo por el cual se han llevado a cabo las decisiones tomadas.

## Presentación

Una vez concluidas satisfactoriamente todas y cada una de las anteriores tareas, se podrá realizar la preparación de la presentación. En esta presentación, se comentarán todas y cada una de las tareas realizadas explicando el orden de exposición de los diferentes integrantes, explicando las diferentes decisiones tomadas, cómo se han implementado cada una de ellas y cuánto tiempo se ha dedicado a realizar.

# 4. Evaluación del proyecto

En este apartado queremos recoger cómo se ha llevado a cabo nuestro proyecto, es decir, qué tiempos se han empleado realmente en la realización de cada tarea y compararlos con los tiempos que habíamos estimado en el comienzo con el diagrama de Pertt. De esta forma  vamos a comprobar, en primer lugar, si los plazos previstos se han cumplido, y en segundo, ayudándonos de lo anterior, podemos detectar cuál ha sido la implicación del equipo en la práctica.

En primer lugar vamos a ver cuál ha sido el tiempo real empleado en cada tarea.

CAMBIAR TIEMPOS

|  |  |
| --- | --- |
| Tarea | Tiempo real empleado (días) |
| Identificar errores | 1 |
| Establecer requisitos | 1 |
| Diagrama de clases | 1 |
| Realización prototipo | 3 |
| Modificación código inicial | 1 |
| Añadir funcionalidad e interfaz | 4 |
| Pruebas 1 | 3 |
| Pruebas 2 | 1 |
| Documentación de pruebas | 2 |
| Presentación | 15 |
| Documentación de todo el proyecto |  |
| Control de versiones |  |

REVISAR AL FINAL DEL PROYECTO

A continuación vamos a comparar estos tiempos con los que habíamos estimado en el diagrama de Pertt. Así vemos que tareas como identificación de errores, establecimiento de requisitos, realización de diagrama de clases, modificación de código, prototipo y presentación no han variado sus tiempos, es decir, se han cumplido los tiempos previstos para estas actividades críticas. Sin embargo, observamos que la tarea de añadir funcionalidad se ha llevado a cabo un día antes de lo que teníamos previsto. Lo mismo ocurre con la realización de la interfaz, en este caso hemos conseguido realizarlo tres días antes. Para las pruebas tenemos que la tarea ha finalizado con un día de antelación. Y por último, la documentación, al haber realizado las tareas anteriores en menor tiempo, también nos ha ocupado cuatro días menos de los previstos.

Con esto podemos deducir algunas conclusiones. En primer lugar, podemos pensar que estos tiempos previstos han sido demasiado largos, es decir, hemos hecho una previsión a la alza en cuanto a los días teniendo en cuenta que podrían aparecer problemas durante la realización de esta primera parte del proyecto. Con problemas nos referimos a otros proyectos al mismo tiempo.

En segundo lugar, lo que podemos concluir ante la disminución de los tiempos es la implicación del equipo de trabajo ya que éste ha puesto todo su esfuerzo para que el proyecto finalizase antes de los plazos previstos.