**Patrones de diseño.**

En la elaboración del proyecto no implementamos ningún patrón de diseño de forma explícita, pero al analizar el código que realizamos para poder ver una apertura para la implementación de algún patrón de diseño, encontré que el patrón Singleton es el que más se vincula.

Para poder realizar la implementación del patrones es necesario comprender como está pensado nuestro proyecto. Para ello daré una breve explicación que ayudara a poder definir el tipo de problema que presenta y como lo resolvimos, de esta manera será más fácil identificar porque los dos patrones de diseño presentados se pueden implementar en el proyecto.

**Problema planteado y solución del proyecto.**

En la elaboración del proyecto el equipo pensó en realizar la implementación del fractal de Mandelbrot de forma paralela, aprovechando que se tiene la posibilidad en Java, pero en un determinado momento decidimos añadir mas fractales de tal manera que pudiéramos disponer de varios fractales para la creación de un menú de estos.

**Patrón de diseño que se pudo implementar.**

* **Nombre:**

Singleton.

* **Problema:**

Se utiliza cuando es necesario el tener solo un objeto de una instancia, es decir, restringir la creación de los objetos, ya sea para conservar la integridad del programa o la integridad de los datos.

* **Solución:** 
  + Tener como atributo una instancia de la misma clase.
  + El modificador de acceso del constructor deberá ser privado.
  + Creación del método que obtenga la instancia de la clase, cuando se ejecute la creación del objeto se cree el objeto del la clase, y condicionar para que cuando se quiera crear una segunda instancia del objeto se devuelve la ya creada en la primer llamada al método.

**Tabla

Descripción generada automáticamente**

**Diagrama – Singleton**

Al tener el problema que se debe resolver es más sencillo el poder identificar los patrones y en este caso se tiene que las instancias deben ser únicas, lo que para nuestra implementación funciona adecuadamente debido a que solo se crea una instancia de cada clase.

**Análisis de la implementación de Singleton.**

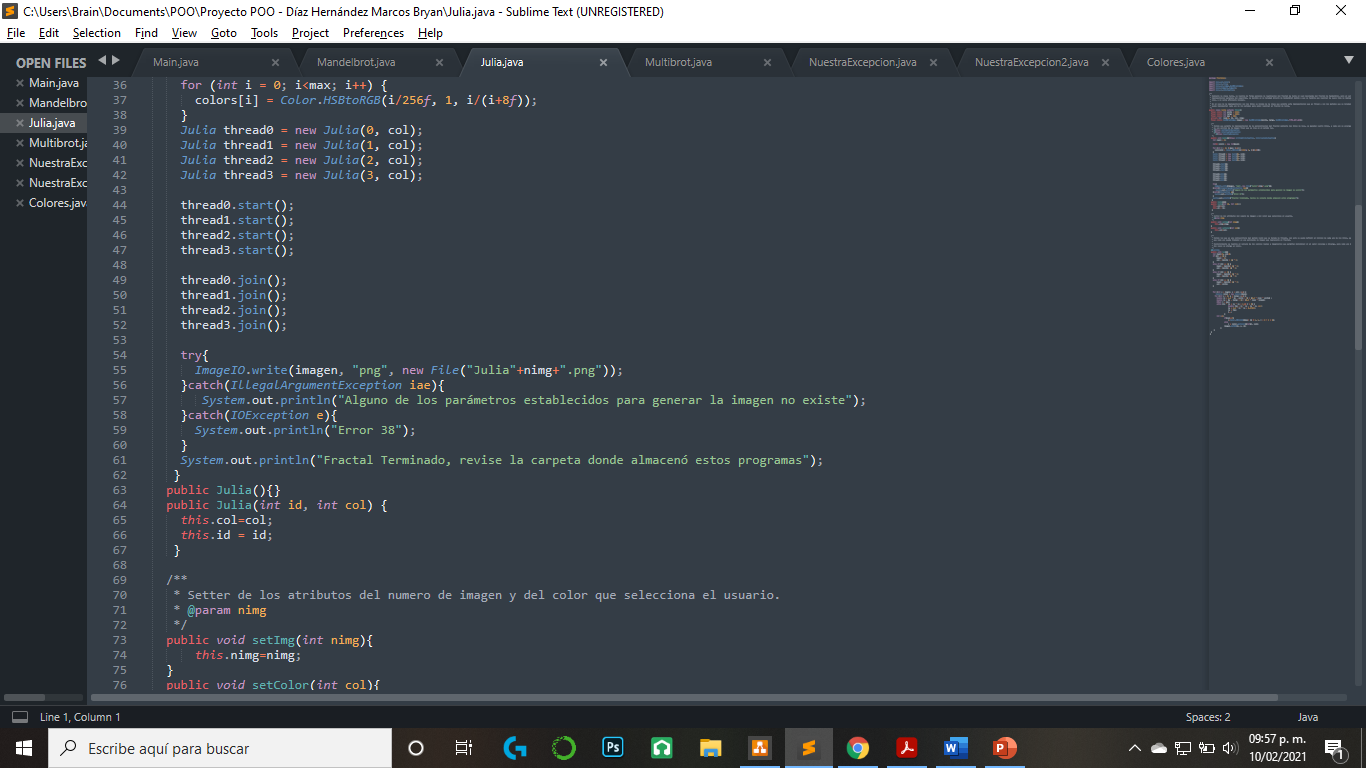
Pero el punto principal de esto es el saber porque se pudo utilizar Singleton y no otros patrones, y la respuesta es porque estamos utilizando hilos para la implementación, y porque de igual manera estamos utilizando Java.

* **Problema.**

El uso de hilos en Java permite el utilizar los métodos de sincronización entre los hilos, lo que significa que es posible modificar el comportamiento de estos en cualquier momento de la ejecución. Pero lo importante para el patrón de diseño es la creación de una única instancia pero lo que puede suceder es que los hilos al ejecutarse puedan crear una instancia de forma simultanea lo que nos dejaría con dos instancias y donde una no se ocuparía o en el caso de nuestra implementación el trabajo realizado en esa implementación resulte en un error, debido a que no se realice correctamente el grafico.

* **Solución.**

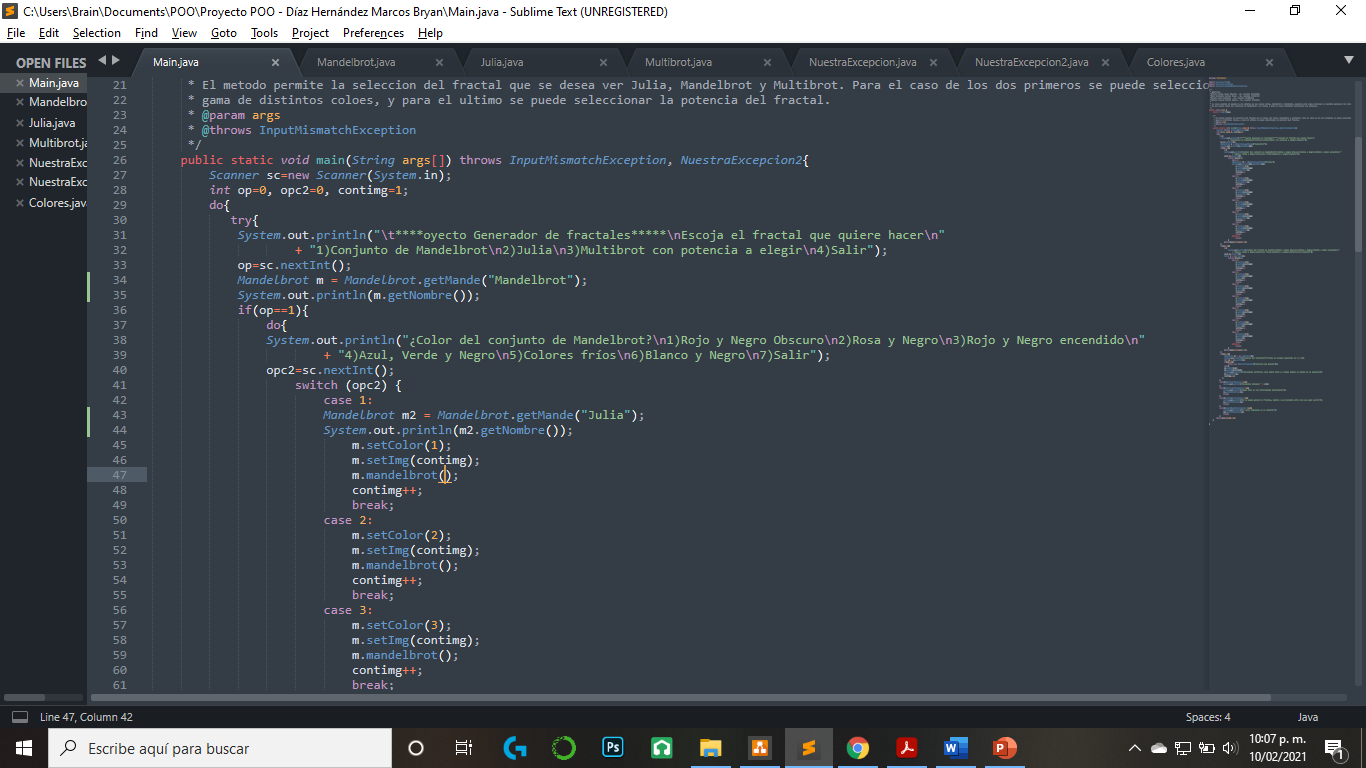
Para la solución de este tipo de errores causados por los hilos, existen dos formas, una es la implementación de los método de sincronización, como lo es Notify(), que indica cuando un hilo ha terminado su ejecución y permite que los demás comiencen, esto es cuando se tiene una sección definida donde un solo hilo puede comenzar a realizar la ejecución del código como lo es el OpenMP.



Sincronización por método Join()

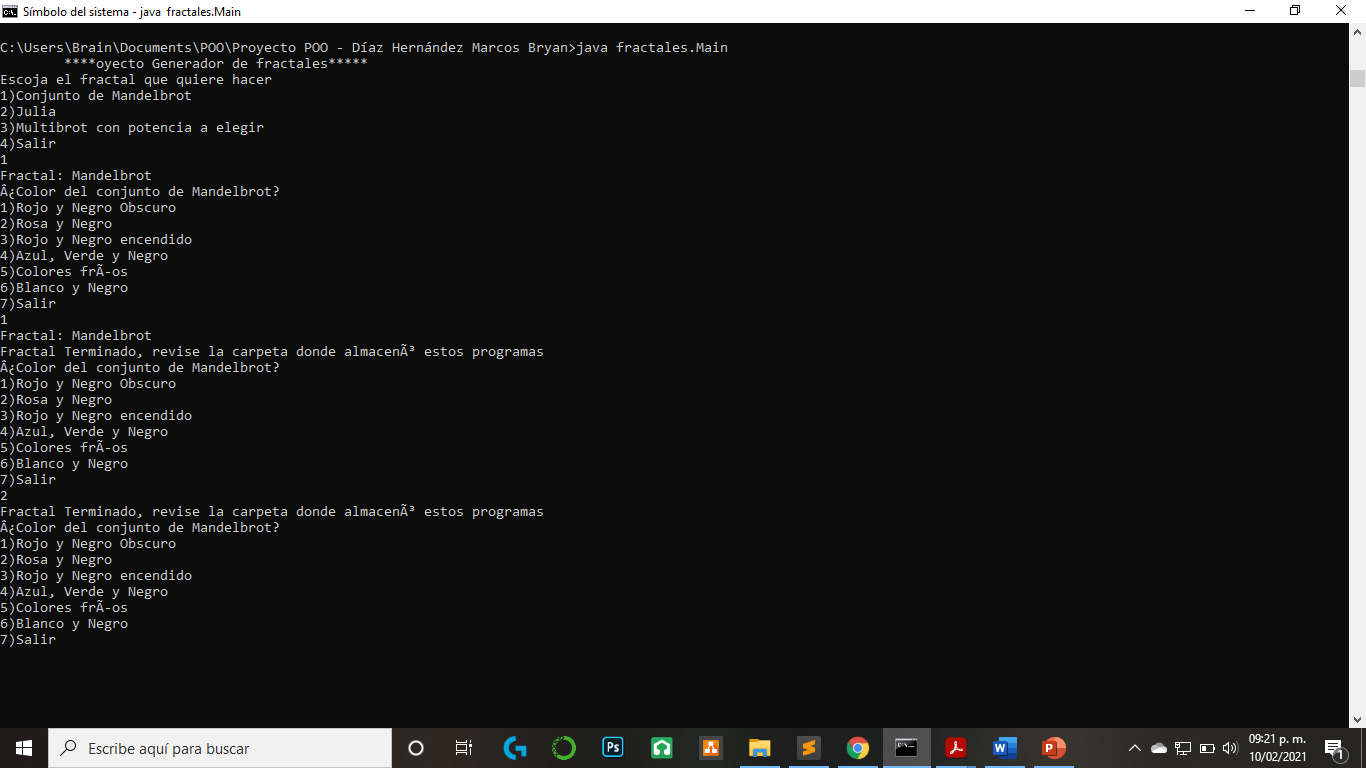
Por medio de los métodos es posible condicionar a los hilos para que solo uno de estos cree el objeto de la clase Mandelbrot y cuando los demás tengan que realizar su implementación reciban el mismo objeto que creo el primer hilo, ya que con Singleton se evita la creación objetos que no son necesarios.

La segunda forma de realizar esta coordinación de los hilos en caso de que no se utilice Java o se quiera implementar de otra forma, es la creación de la instancia antes de que los hilos comiencen, de esa manera ya no entra la condición de carrera entre estos y trabajan sobre la misma instancia.



Creación de la instancia.

En el caso de nuestro programa, lo que sucedía es que cada vez que el usuario seleccionaba ver el fractal de Mandelbrot, se creaba la instancia nuevamente cada vez que se seleccionaba la opción, pero al utilizar el patrón Singleton, esto ya no sucede y se reutiliza el objeto creado.



Implementación del Singleton.

* UML – Clases.

Diagrama

Descripción generada automáticamente

Singleton en la clase Mandelbrot.

Para el caso de cada fractal se puede implementar el patrón de diseño, debido a que permite que solo se esté creando una instancia de forma única, y no genera ningún error en la creación del respectivo fractal.

* **Desventajas.**

Si bien es posible realizar esta implementación en cada una de las clases de los fractales esto no termina siendo lo mejor, debido a que se tienen instancias Singleton múltiples, esto puede generar que en un determinado momento estas instancias creen un problema por compartir el mismo ámbito.

Finalmente el patrón a mi parecer es el más adecuado debido a que usamos hilos, de lo contrario no creo que sea los mas adecuado, ya que con otros patrones es posible que se puedan obtener mejores resultados, pero siempre y cuando el patrón se pueda adaptar a la implementación con la que se está trabajando.

**Referencias:**

* Allen H. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*. United States of America: Prentice Hall.
* Steve Holzner. (2006). *Design Patterns For Dummies*. Indianápolis, Indiana: Wiley.