ED04 ENTORNOS DE DESARROLLO.

UT04 Optimización y documentación.

Profesora:Inmaculada D. Cerdeiriña del Tio.

Alumna: Cherry Reynoso Catalán

DAM Distancia. Entornos de Desarrollo

05 de Marzo del 2024

Cherry Reynoso Catalán

ÍNDICE

Cherry Reynoso Catalán

1. Intenta entenderlo y escribe el código JavaDoc que explique cada uno de los métodos que hay en el mismo, incrustándolo dentro del código (3 puntos).

El código con comentarios JAVADOC quedaría así:

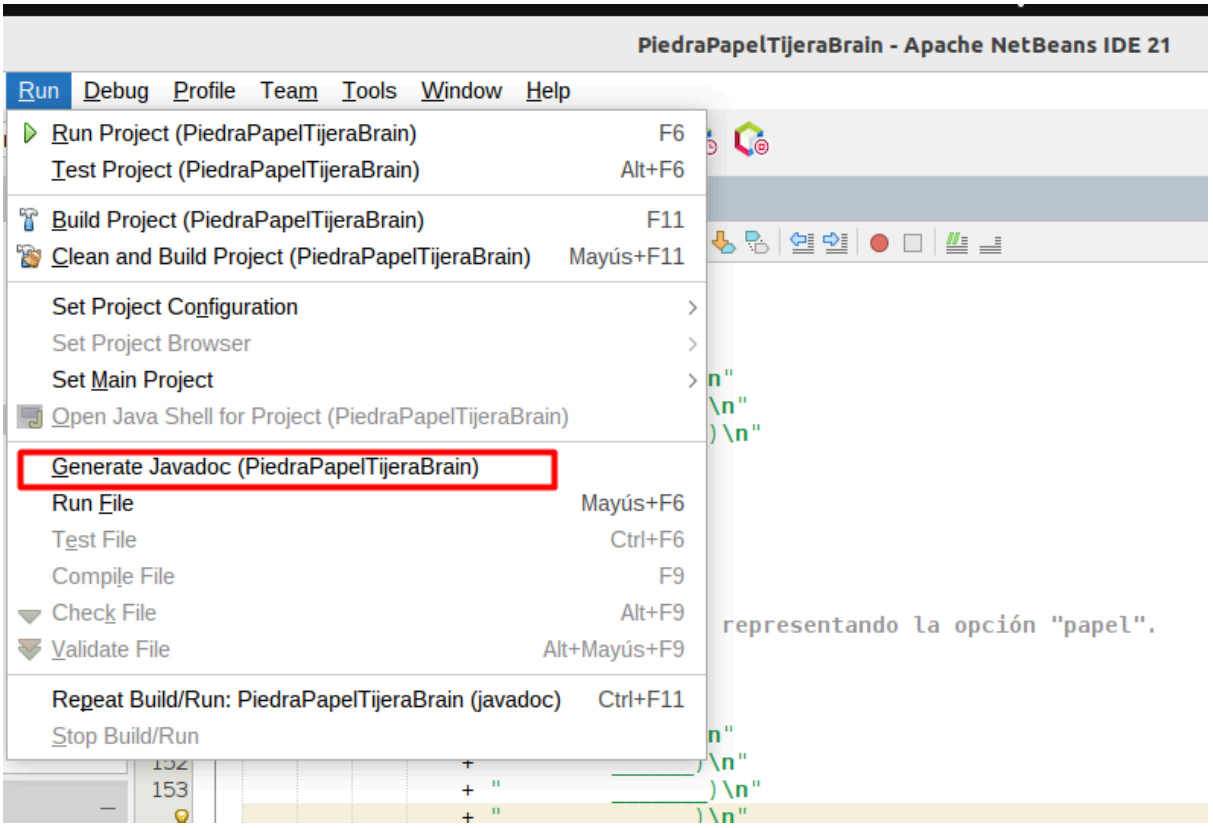
```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit
this template
 */


 package piedrapapeltijerabrain;

 import java.util.Scanner;

 /**
  * Esta clase replica el tradicional juego "Piedra, papel o tijeras"
  * por medio de 5 rondas mientras registra el tiempo y penaliza los fallos.
  * Se acumulan puntos en función de las victorias del jugador sobre el
ordenador.
  * @author cherry

  * */
 public class PiedraPapelTijeraBrain {
         /**
         * El método main contiene TODA la lógica del juego.
         * Comprende una serie de 5 rondas, entre el jugador y el ordenador.
         *Los fallos se contabilizan y se restan a los puntos finales.
         *Se contabiliza el tiempo y se muestra en pantalla.
         */
     public static void main(String[] args) {

         //Marcador de puntos: La variable chiquipuntos acumula los puntos
totales de cada ronda
         int chiquipuntos = 0;

         //Variables que definen jugador y bot.
         Scanner sc = new Scanner(System.in);
         String opcionJugador = "";
         String opcionBot = "";

         //Variable que determina si el jugador tuvo éxito o no
         int exitoEnProposito = 1;

         //Inicio del contador del tiempo
         long inicio = System.currentTimeMillis();

         /*
```

```
        Rondas del juego por medio de un bucle for.
        * La variable "propósito" es aleatoria y determina si el ordenador
intenta ganar o perder.
        */
        for (int i = 0; i < 5; i++) {

            int proposito = (int) Math.floor(Math.random() * 2 + 1);
            if (proposito == 1) {
                System.out.println("\n\tIntenta ganar");
            }
            if (proposito == 2) {
                System.out.println("\n\tIntenta perder");
            }

            //De forma aleatoria, el ordenador establece si juega con
tijera, papel o piedra a través de un condicional.
            int j = (int) Math.floor(Math.random() * 3 + 1);
            if (j == 1) {
                opcionBot = "tijera";
                System.out.println(opcionBot);
                tijera();
            }
            if (j == 2) {
                opcionBot = "papel";
                System.out.println(opcionBot);
                papel();
            }
            if (j == 3) {
                opcionBot = "piedra";
                System.out.println(opcionBot);
                piedra();
            }

            // El jugador introduce su jugada.
            do {
            System.out.println("Introduce tu jugada");
            opcionJugador = sc.nextLine();
            if (opcionBot.equals(opcionJugador)) {
            System.out.println("No tiene sentido que intentes empatar");
            }

            /* Compara las jugadas del jugador y del ordenador.
            Establece si ganó o perdió la partida el jugador añadiendo el
punto en la variable "chiquipuntos"
            */

            } while (opcionBot == opcionJugador);
            if (opcionJugador.equals("tijera") &&
(opcionBot.equals("papel"))) {
                exitoEnProposito = 1;
            }
            if (opcionJugador.equals("papel") &&
(opcionBot.equals("tijera"))) {
                exitoEnProposito = -1;
            }
```

```
            if (opcionJugador.equals("tijera") &&
(opcionBot.equals("piedra"))) {
                exitoEnProposito = -1;
            }
            if (opcionJugador.equals("piedra") &&
(opcionBot.equals("tijera"))) {
                exitoEnProposito = 1;
            }
            if (opcionJugador.equals("piedra") &&
(opcionBot.equals("papel"))) {
                exitoEnProposito = -1;
            }
            if (opcionJugador.equals("papel") &&
(opcionBot.equals("piedra"))) {
                exitoEnProposito = 1;
            }
            // si el propósito era perder, multiplica con -1 para cambiar el
signo
            if (proposito == 2) {
                exitoEnProposito *= -1;
            }
            //Si es ganar simplemente añade los puntos a "chiquipuntos"
            if (exitoEnProposito == 1) {
                chiquipuntos++;
            }
        }

        //finaliza el contador de tiempo
        long fin = System.currentTimeMillis();

        //calcula el tiempo de la partida
        double tiempo = (double) ((fin - inicio) / 1000);
        System.out.println("Has realizado el ejercicio en " + tiempo + "
segundos");

        //calcula los fallos y penalización
        int nFallos = 5 - chiquipuntos;
        System.out.println("Penalización: " + nFallos + " x 5s = " + nFallos
* 5);

        //calcula el tiempo final en razón de los fallos.
        double tiempoFinal = tiempo + nFallos * 5;
        System.out.println("Tu tiempo final es de " + tiempoFinal + "
segundos");
    }

    public static void tijera() {
        /**Imprime un código ASCII representandio la opción "tijera".
         */

        System.out.println("'''\n"
                + "    _______\n"
                + "---'   ____)____\n"
                + "          ______)\n"
                + "       __________)\n"
```

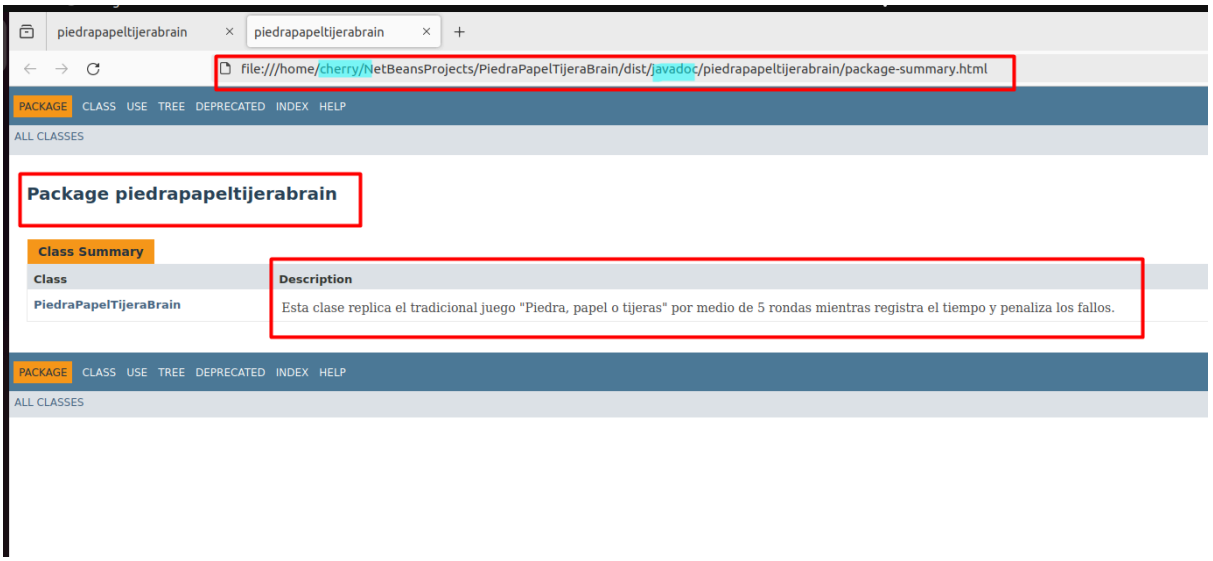
```
                + "       (____)\n"
                + "---.__(___)\n"
                + "''''");
    }

    public static void papel() {
        /**Imprime un código ASCII representando la opción "papel".
         */
        System.out.println("''''\n"
                + "     _______\n"
                + "---' ____)____\n"
                + "          _______)\n"
                + "          _______)\n"
                + "         _______)\n"
                + "---._____________)");
    }

    public static void piedra() {
        /**Imprime un código ASCII representando la opción "piedra".
         */
        System.out.println("''''\n"
                + "     _______\n"
                + "---'     ____)\n"
                + "       (_____)\n"
                + "        (_____)\n"
                + "        (____)\n"
                + "---.__(___)\n"
                + "''''");
    }
}
```

Una vez que hemos agregado los comentarios de los métodos y de las clases desde NetBeans, en el menú “Run” escogemos la opción de “Generate Javadoc”:



1.1 Generamos un javadoc desde el menú "Run". Cherry Reynoso Catalán.

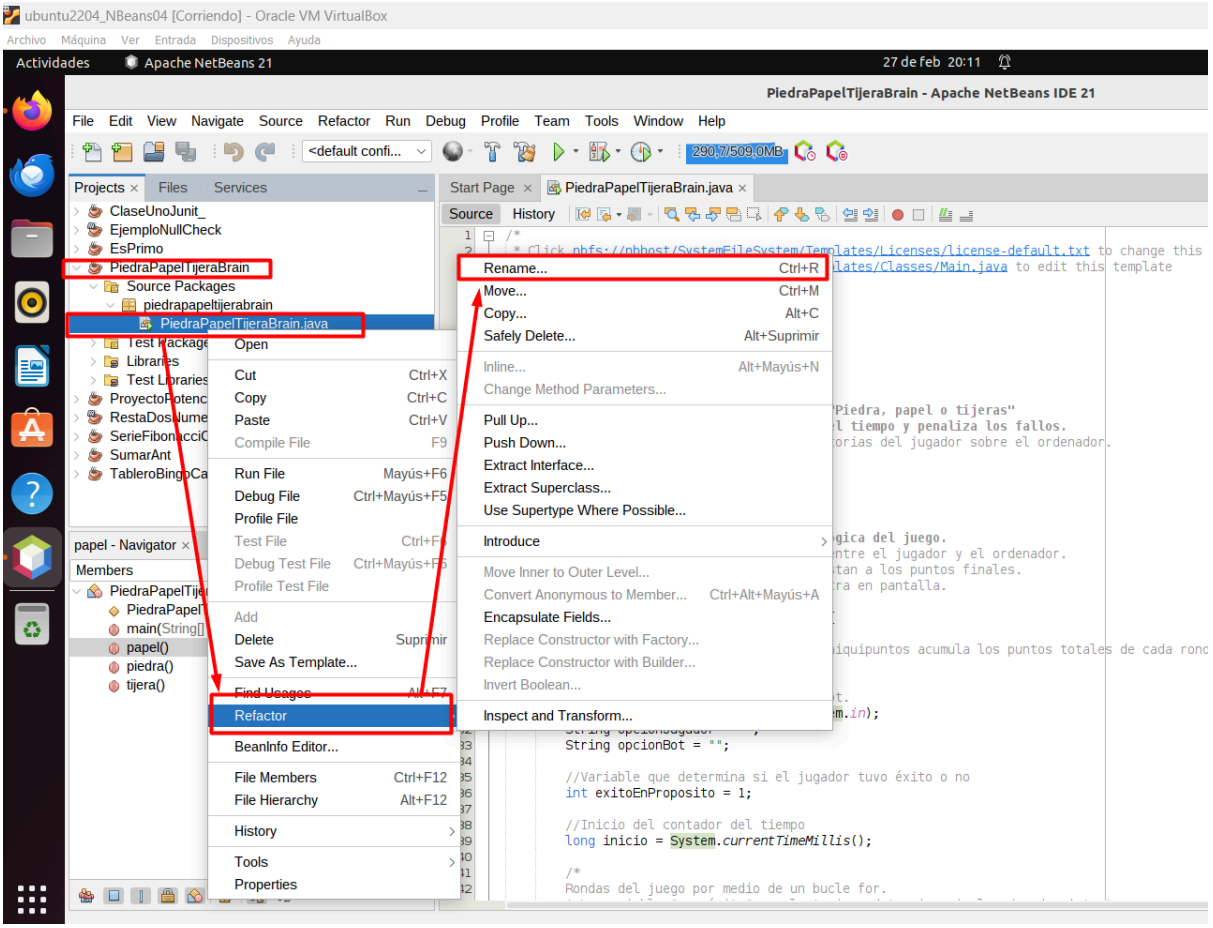
Esto nos abrirá el javadoc en nuestro navegador de preferencia donde podremos ver la clase de nuestro programa y la descripción que hemos hecho de ella:



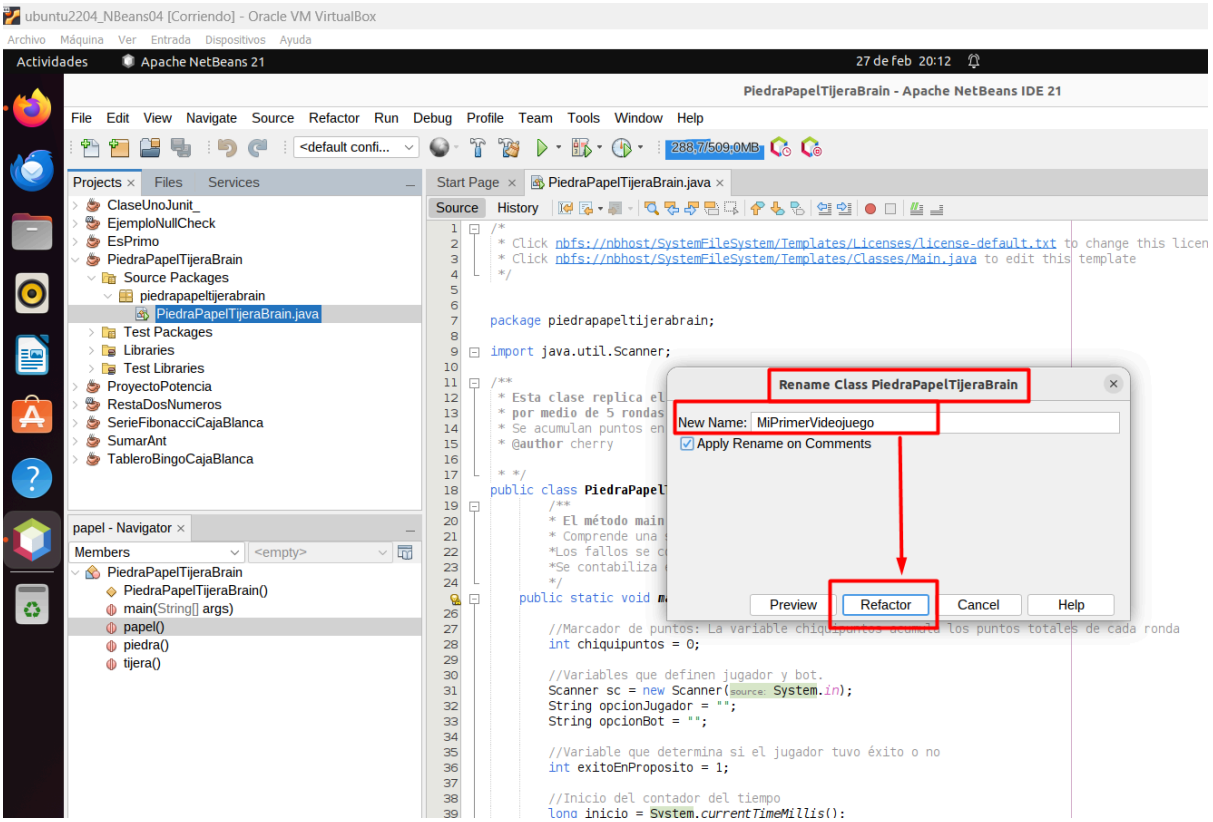
1.2 El javadoc que se genera nos muestra la descripción de la clase.Cherry Reynoso Catalán.

2. Refactoriza el programa para que se llame MiPrimerVideoJuego (3 puntos)

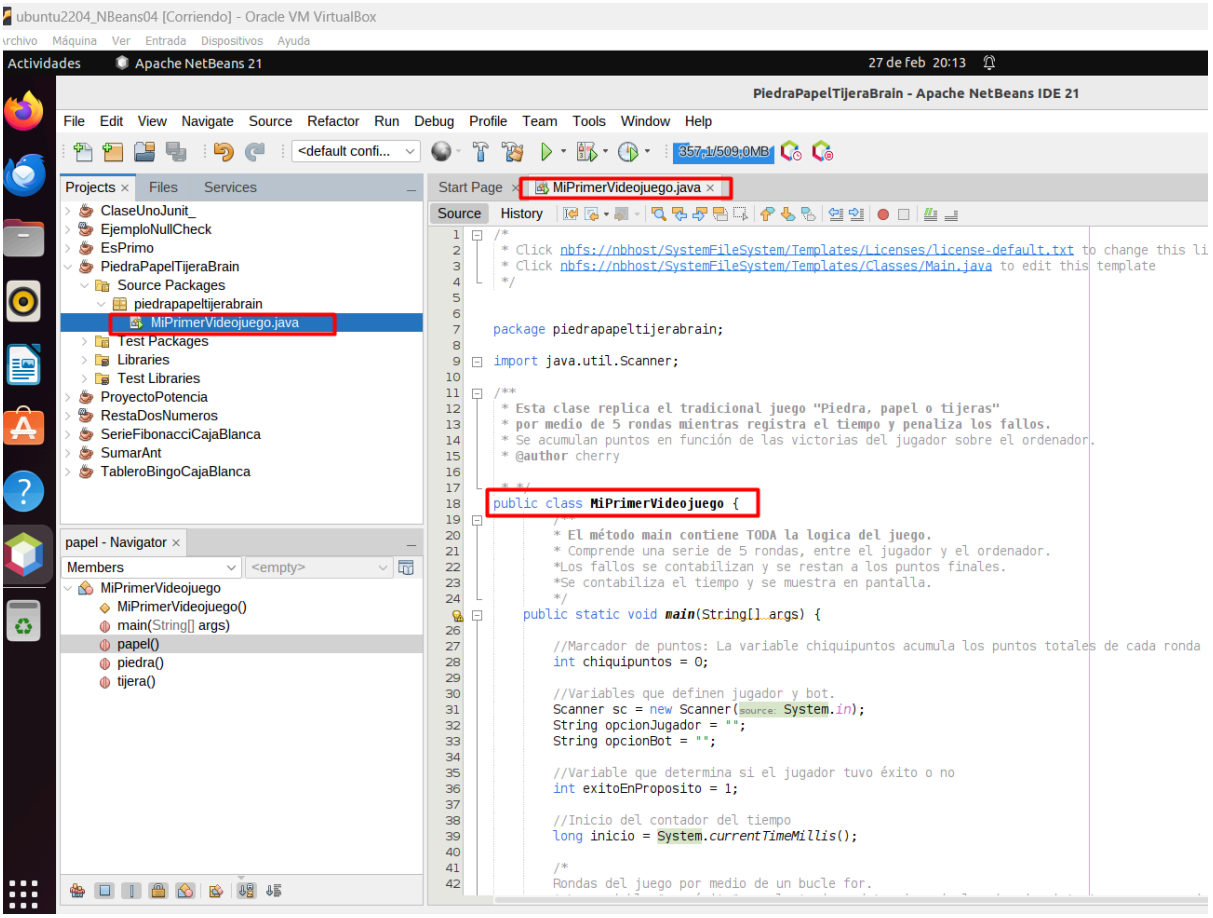
En nuestro proyecto, damos click derecho y en el menú contextual buscamos la opción “Refactor” y entre los desplegados escogemos la opción de “Rename”



1.3 En la opción Refactor cambiamos el nombre a nuestro proyecto.Cherry Reynoso Catalán.



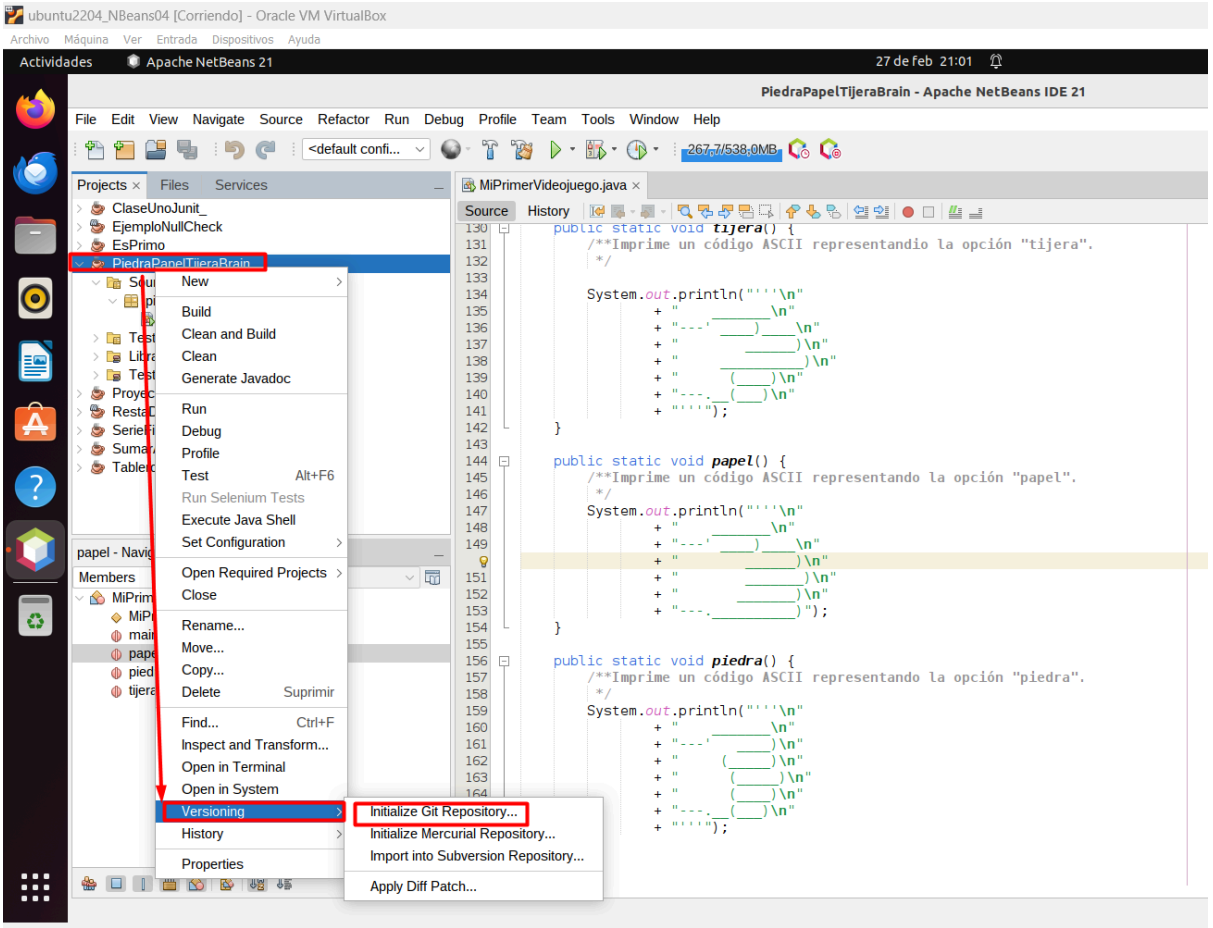
1.4 Nos saldrá una ventana para renombrar a nuestra clase. Cambiamos el nombre a "MiPrimerVideojuego" y dados click en el botón "Refactor". Cherry Reynoso Catalán



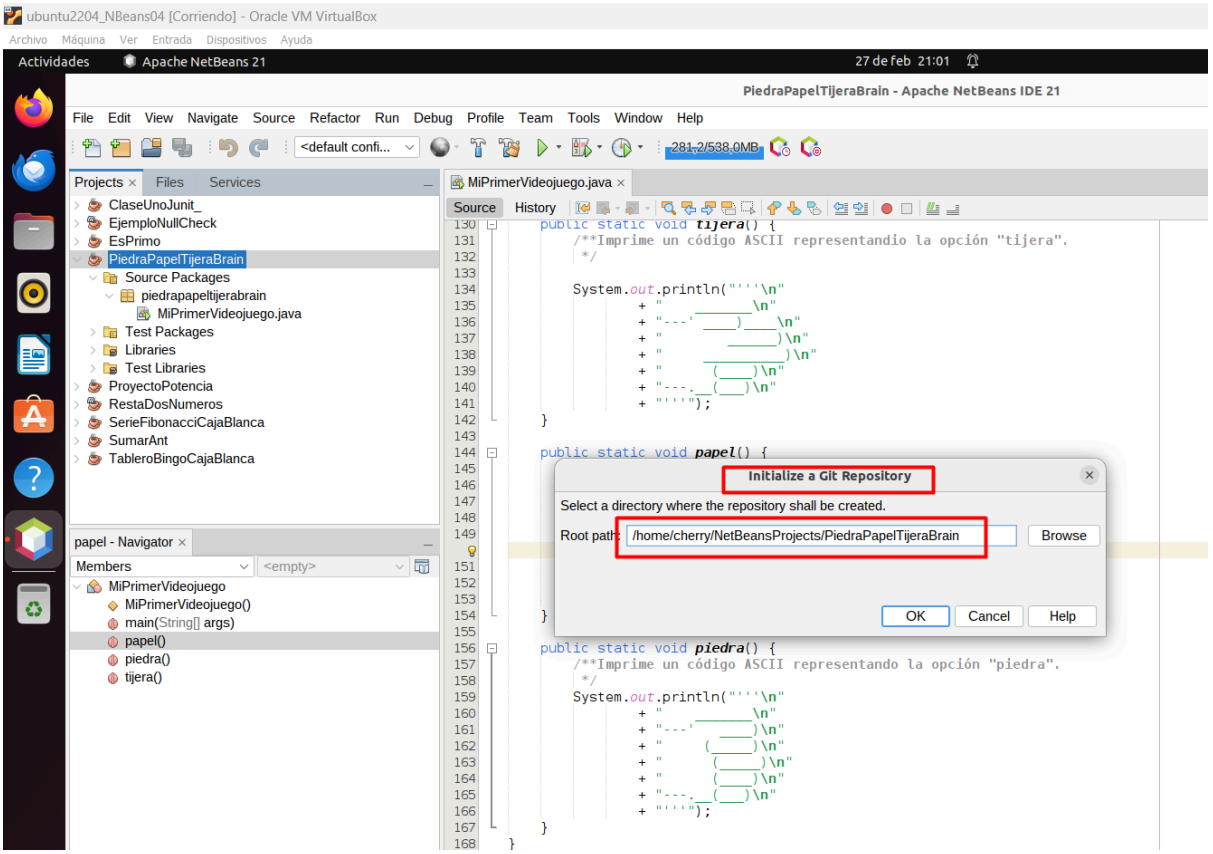
1.5 Podemos comprobar en nuestro código que la clase ha cambiado de nombre. Ahora es “Public Class MiPrimerVideojuego”. Cherry Reynoso Catalán.

3. Activa el repositorio Git, con Botón Derecho- Versioning – Git Repository- y usar Diff

Para poder guardar las versiones y cambios de nuestro código con git, primero debemos inicializar el repositorio:



1.6 Inicializamos nuestro repositorio local con git. Cherry Reynoso Catalán.

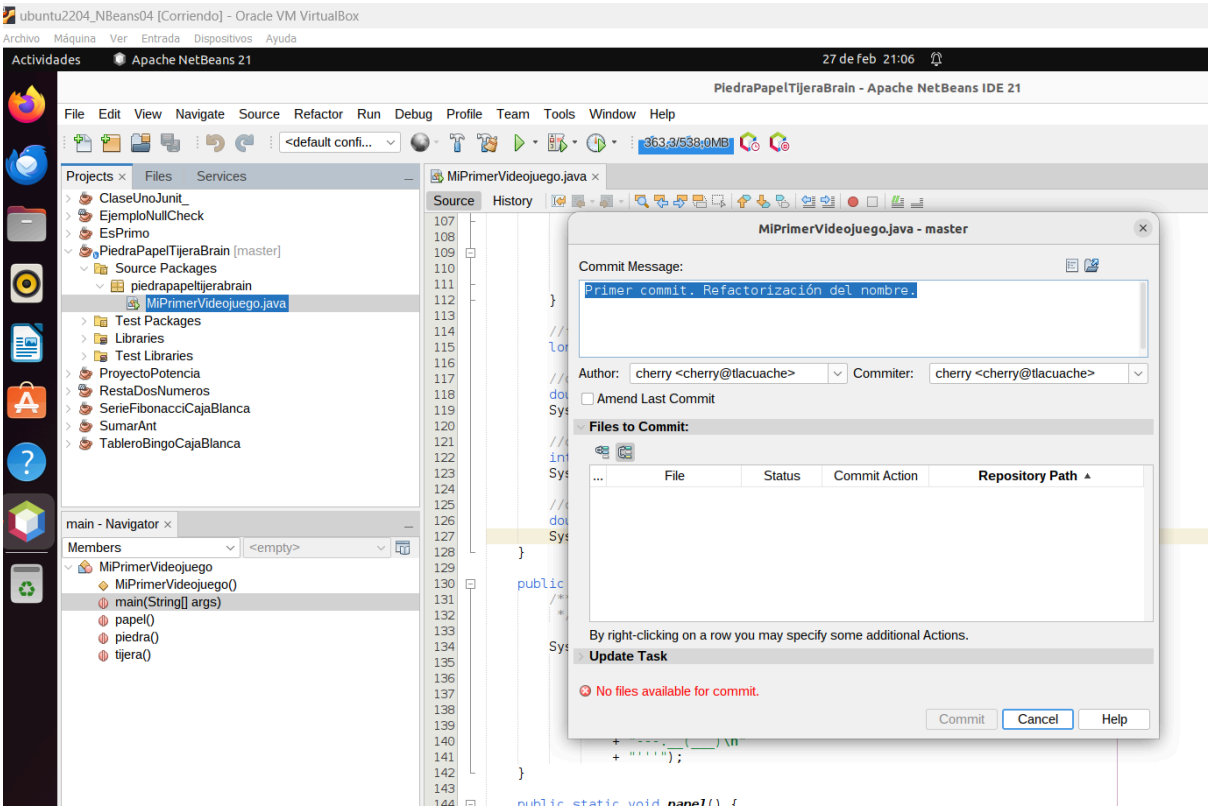


1.6 Decidimos la carpeta donde se alojará nuestro repositorio local. Cherry Reynoso Catalán.

Ya que hemos inicializado nuestro repositorio en Git, es fundamental realizar un proceso de **“add”** para llevar los cambios realizados a la *stage area* y luego un **“commit”** para guardar oficialmente esa versión con sus modificaciones. Este flujo nos permite gestionar de manera efectiva las diferentes versiones de nuestro proyecto, asegurando un registro preciso de todos los cambios presentes y futuros.

Este commit que hagamos generará a nuestra versión y a nuestra rama un “ID” con el que más adelante podremos mover nuestro marcador para poder retornar a versiones previas.

También es importante siempre agregar un mensaje en nuestro commit que nos permita identificar los cambios que se han hecho en el código en dicho commit.

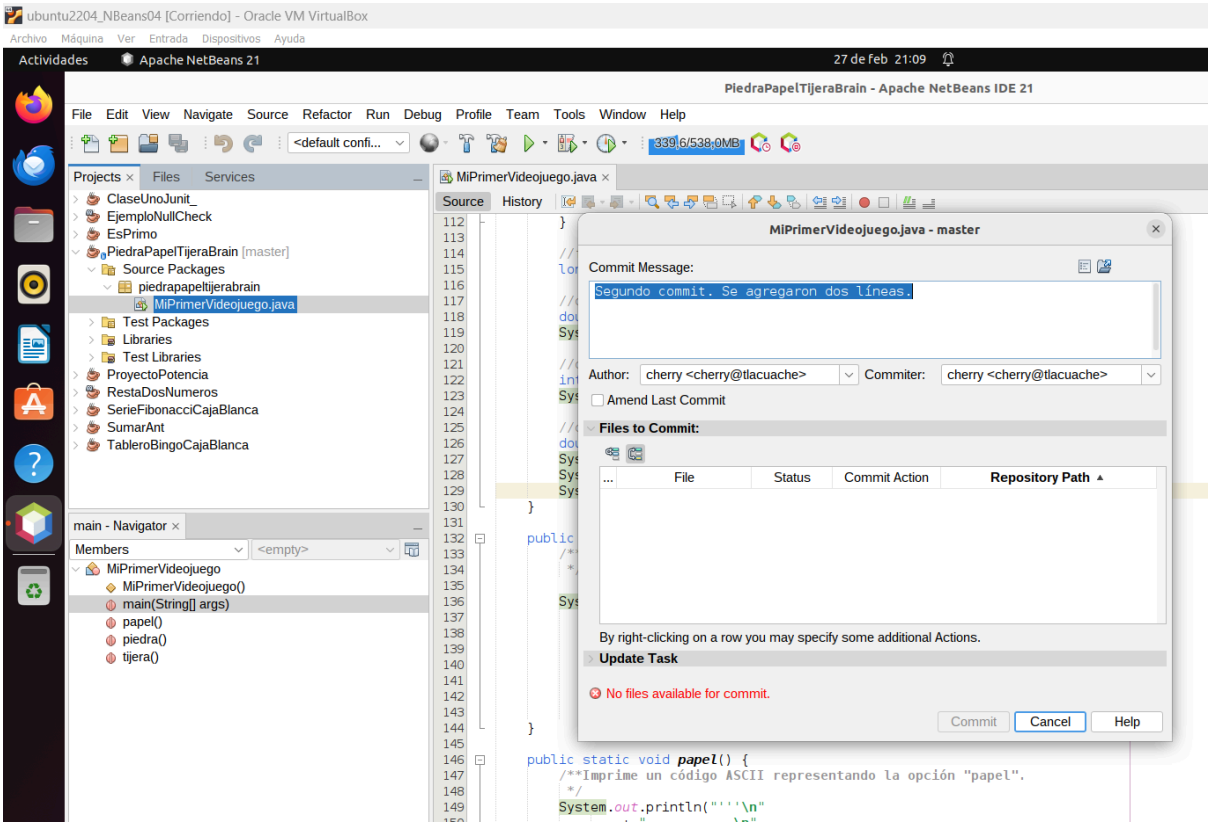


1.7 Primer commit: el cambio que guardamos es la refactorización y cambio de nombre de nuestra clase. Cherry Reynoso Catalán.

Si a nuestro código le agregamos la siguientes líneas:

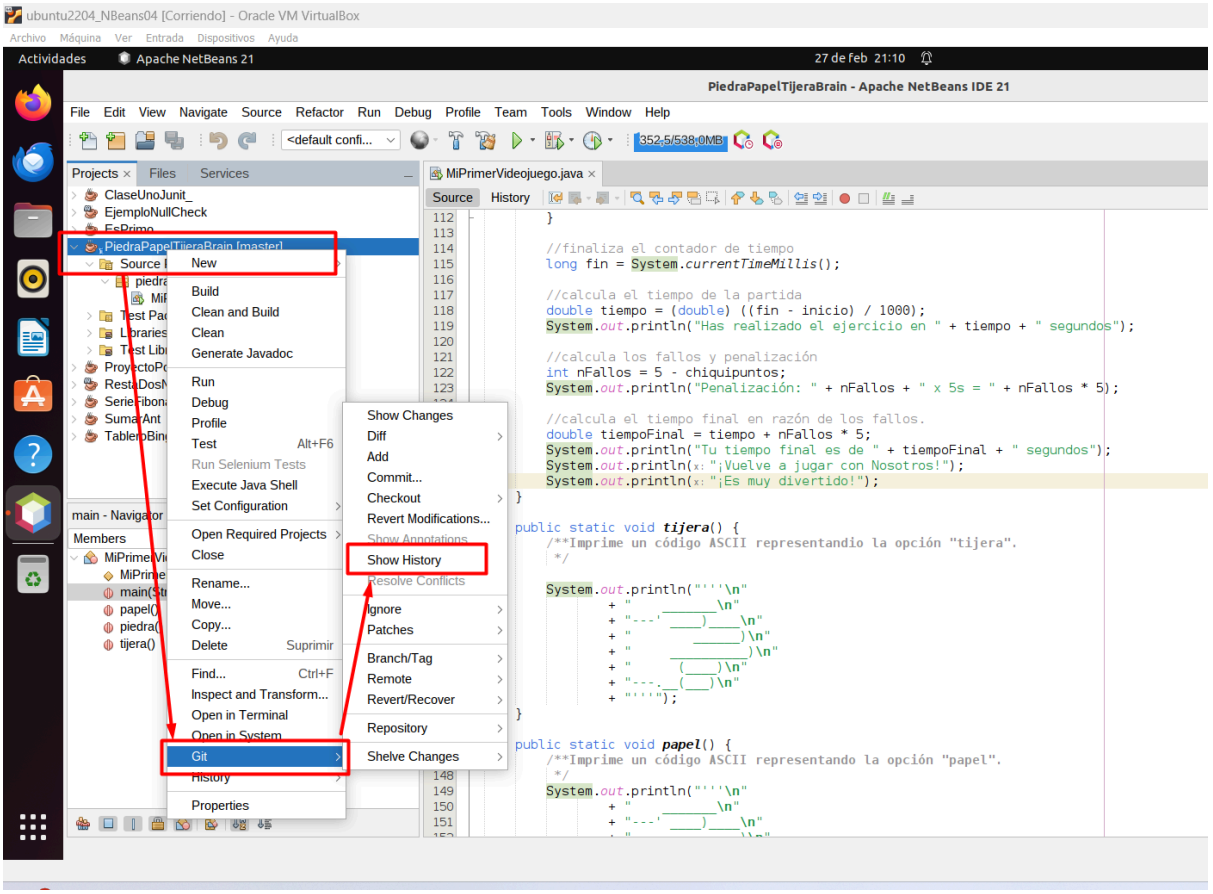
```
System.out.println("¡Vuelve a jugar con Nosotros!");
System.out.println("¡Es muy divertido!");
```

Deberemos hacer un nuevo **add** y un nuevo **commit** para guardar los cambios de versión:



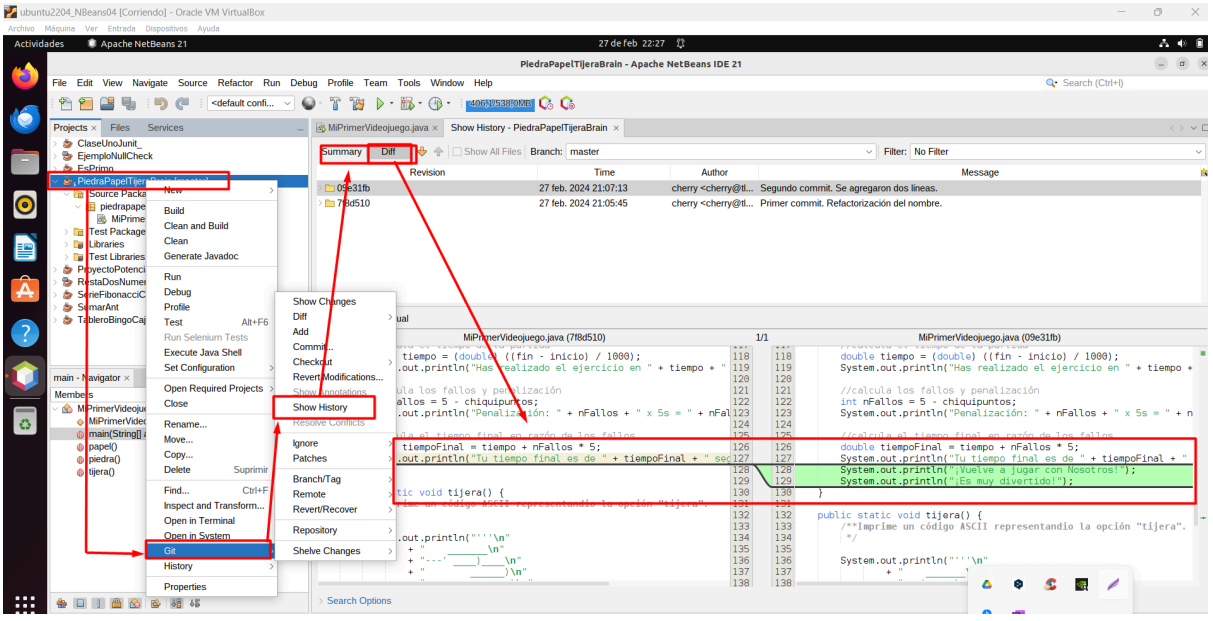
1.8 Segundo commit: guardamos los cambios realizados y agregamos un mensaje de lo que hemos modificado. Cherry Reynoso Catalán.

Una vez realizados ambos commit, podemos visualizar las diferencias de ambas versiones. Para ello damos click derecho en nuestro proyecto y en el menú de Git escogemos la opción “Show History”. Esto nos mostrará un historial de los commits que hemos realizado hasta el momento:

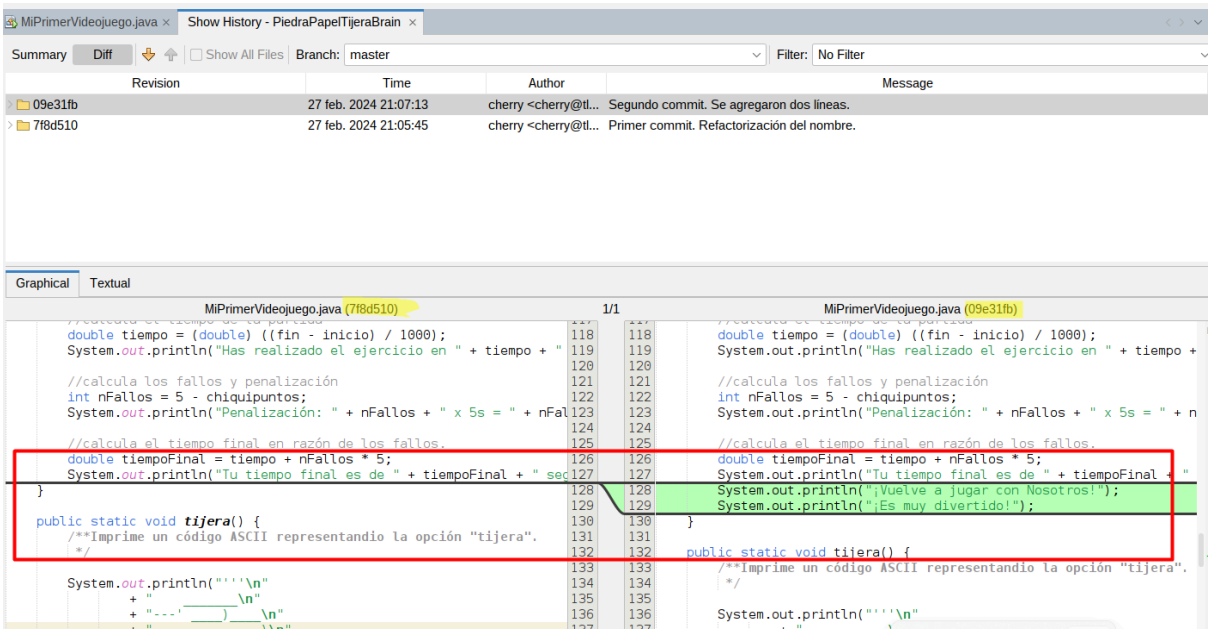


1.9 Escogemos la opción show history para ver el histórico de modificaciones realizadas enb nuestro código. Cherry Reynoso Catalán.

Y con Diff veremos las partes en las que nuestro código difiere según la versión:



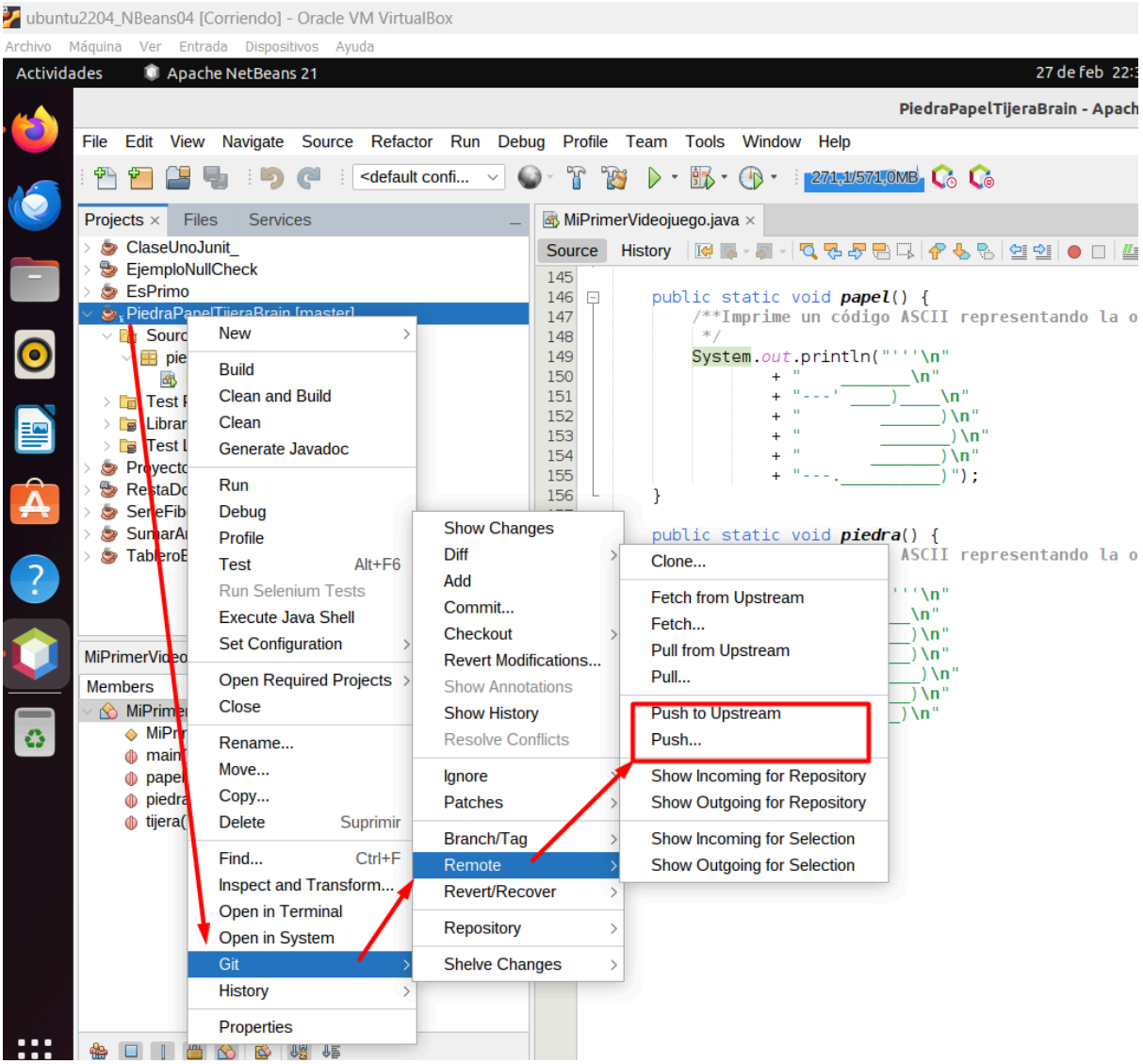
1.10 Con“Diff” nos mostrará exactamente las partes del código que ha cambiado entre una y otra versión. Cherry Reynoso Catalán.



1.11 En la opción “graphical” podemos ver estas diferencias de manera visual. también podemos ver el ID de cada commit que se está comparando. Cherry Reynoso Catalán.

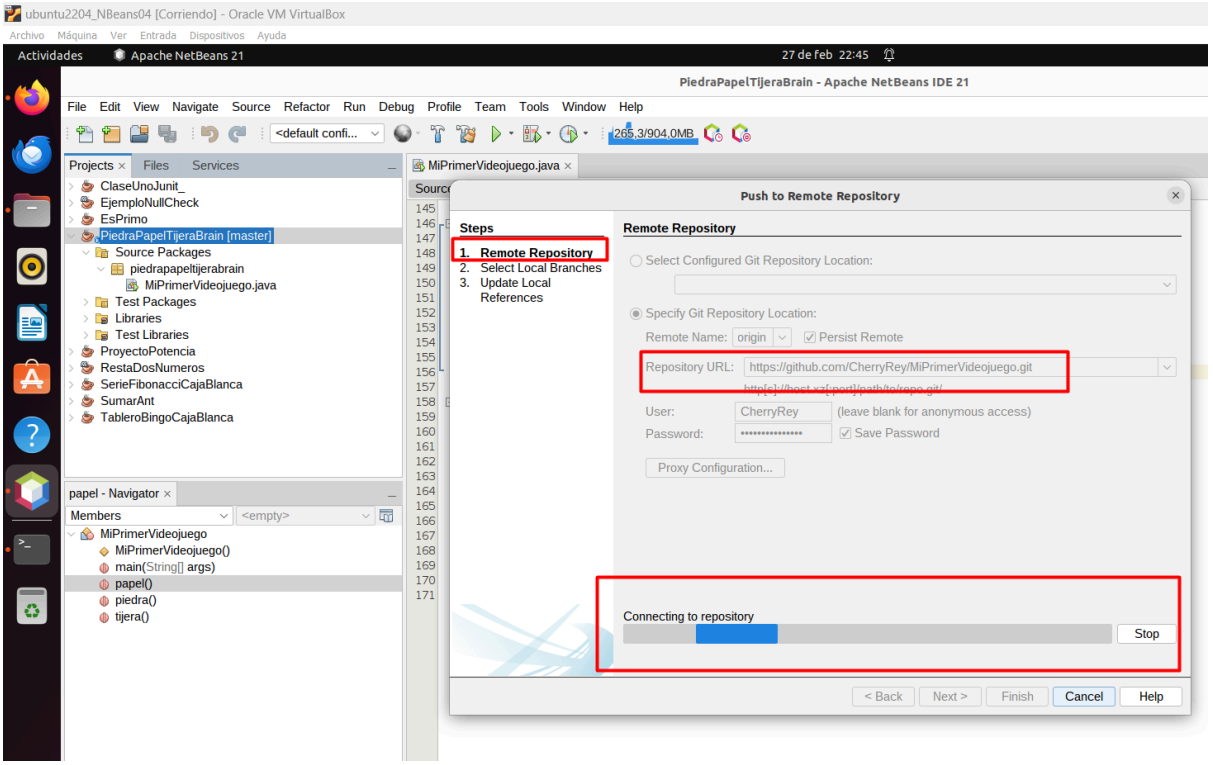
4. GitHub: Subir proyecto a repositorio remoto y agregar colaborador

Cuando usamos git lo que estamos haciendo es tener un repositorio de versiones en local. Para poder subir nuestro código a un repositorio remoto como GitHub debemos ir a click derecho de nuestro proyecto y en el menú de Git escoger la opción: *Remote → Push to upstream*

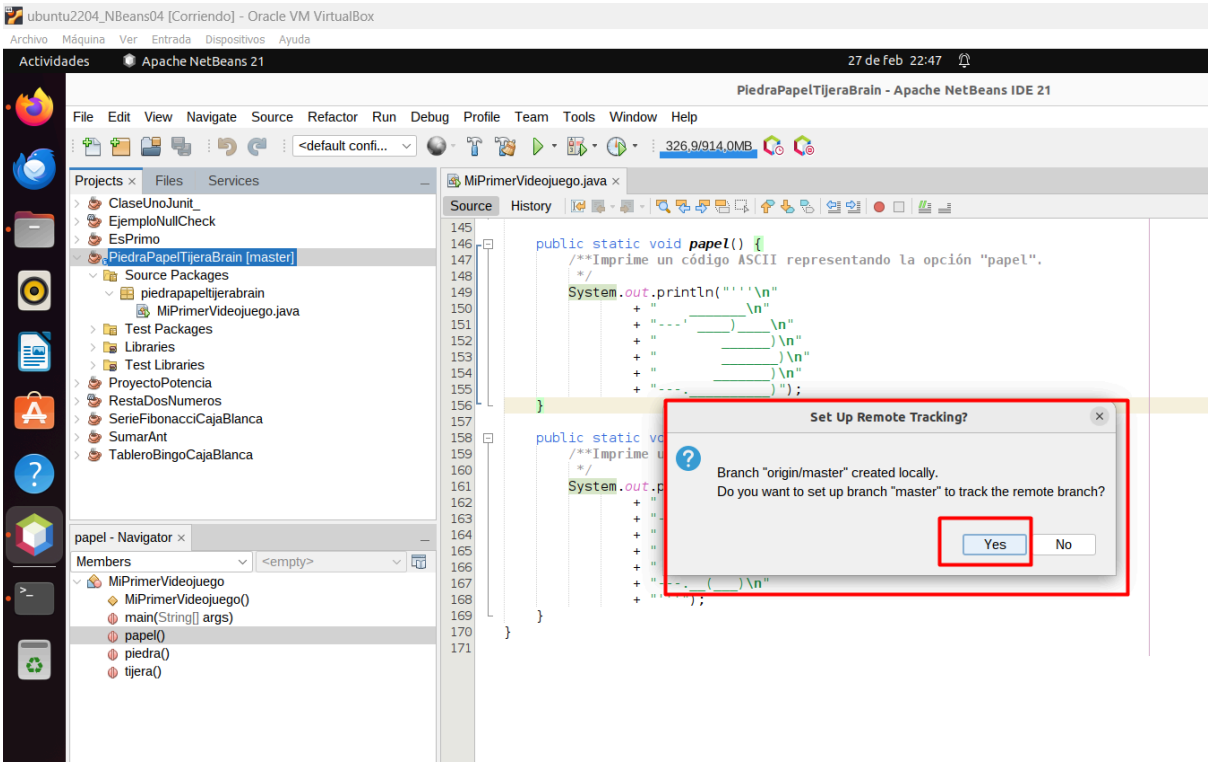


1.12 Menú de Git para enviar nuestro código aun repositorio remoto. Cherry Reynoso Catalán.

Nos aparecerá una nueva ventana donde indicaremos la URL de nuestro repositorio remoto y donde debemos también introducir nuestro usuario y la contraseña o token para poder subir el proyecto remotamente:

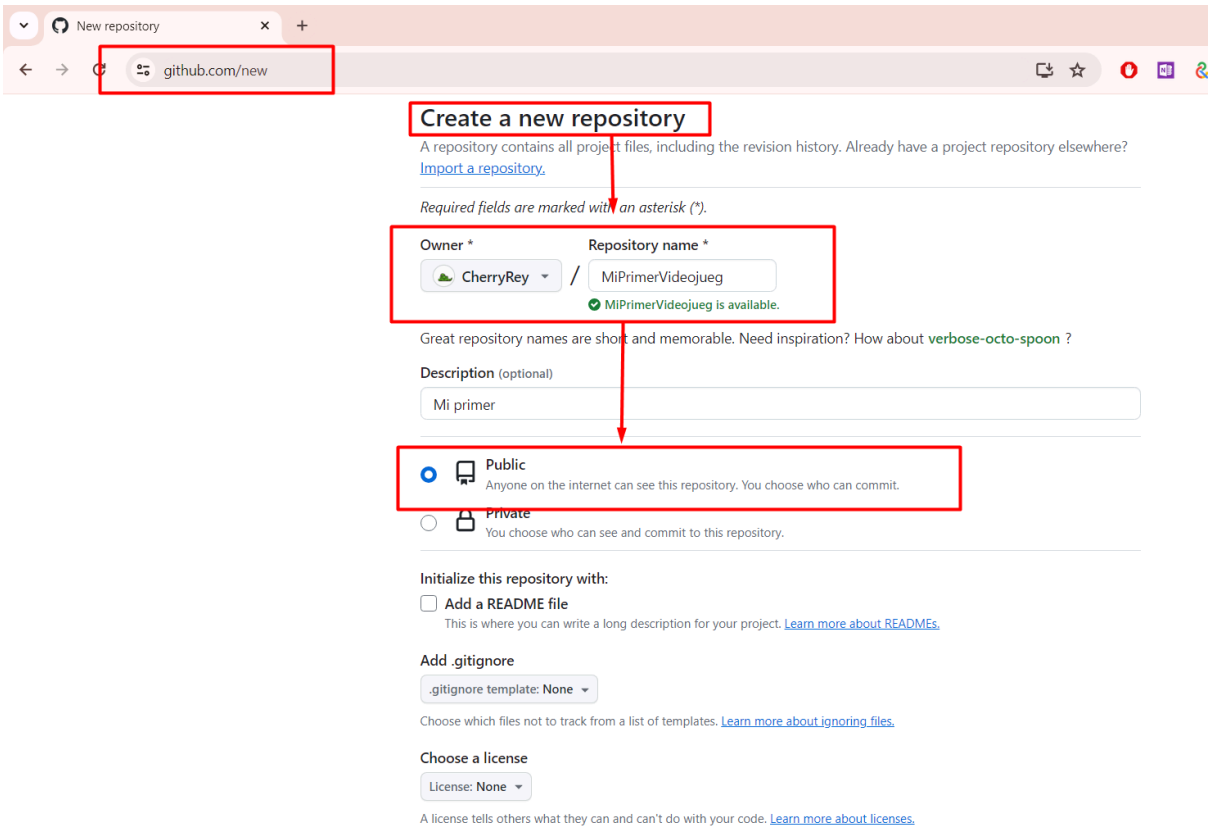


1.12 Conectando nuestro repositorio local con nuestro repositorio remoto. Cherry Reynoso Catalán.



1.13 Nuestra ramas origin/master estan sincronizadas en ambos repositorios. Cherry Reynoso Catalán.

Hay que destacar que antes de subir a GitHub, en nuestro GitHub debemos crear el repositorio donde subiremos todo lo que trabajemos en local y donde escogeremos si queremos que el repositorio sea público o privado para que otros puedan verlo y trabajar con él o no:



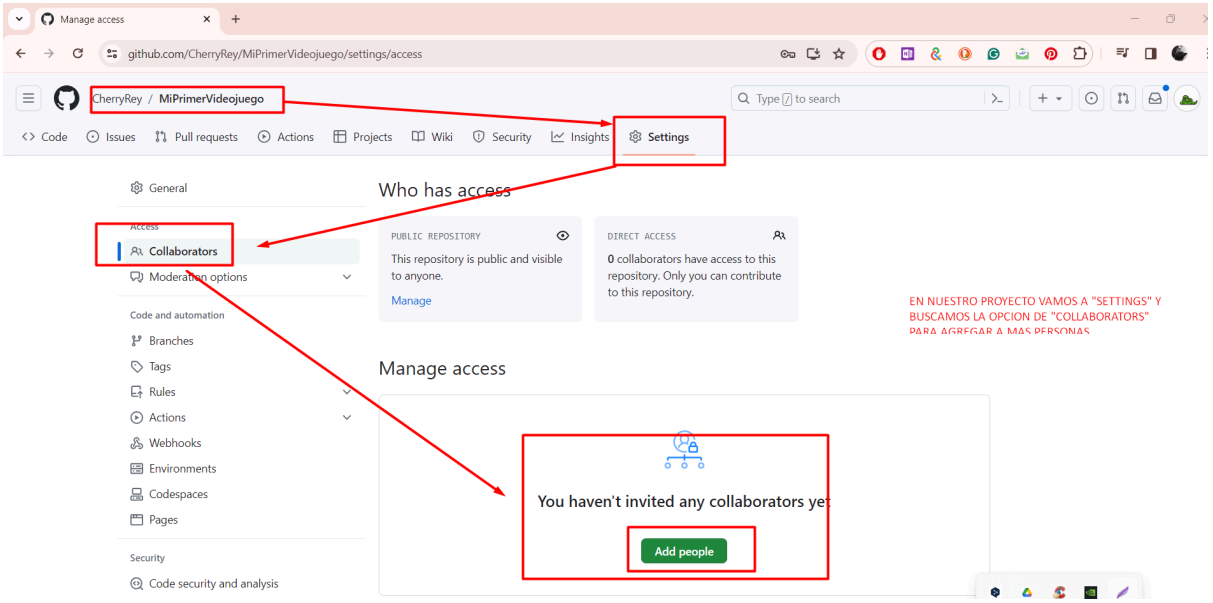
1.13 Antes de subir nuestro repositorio local a uno remoto, desde la página de Github(en este caso) deberemos crear el repositorio donde lo alojaremos e indicar las opciones de si es público o privado. Cherry Reynoso Catalán.

Desde el siguiente enlace podemos comprobar que el repositorio ha sido subido satisfactoriamente:

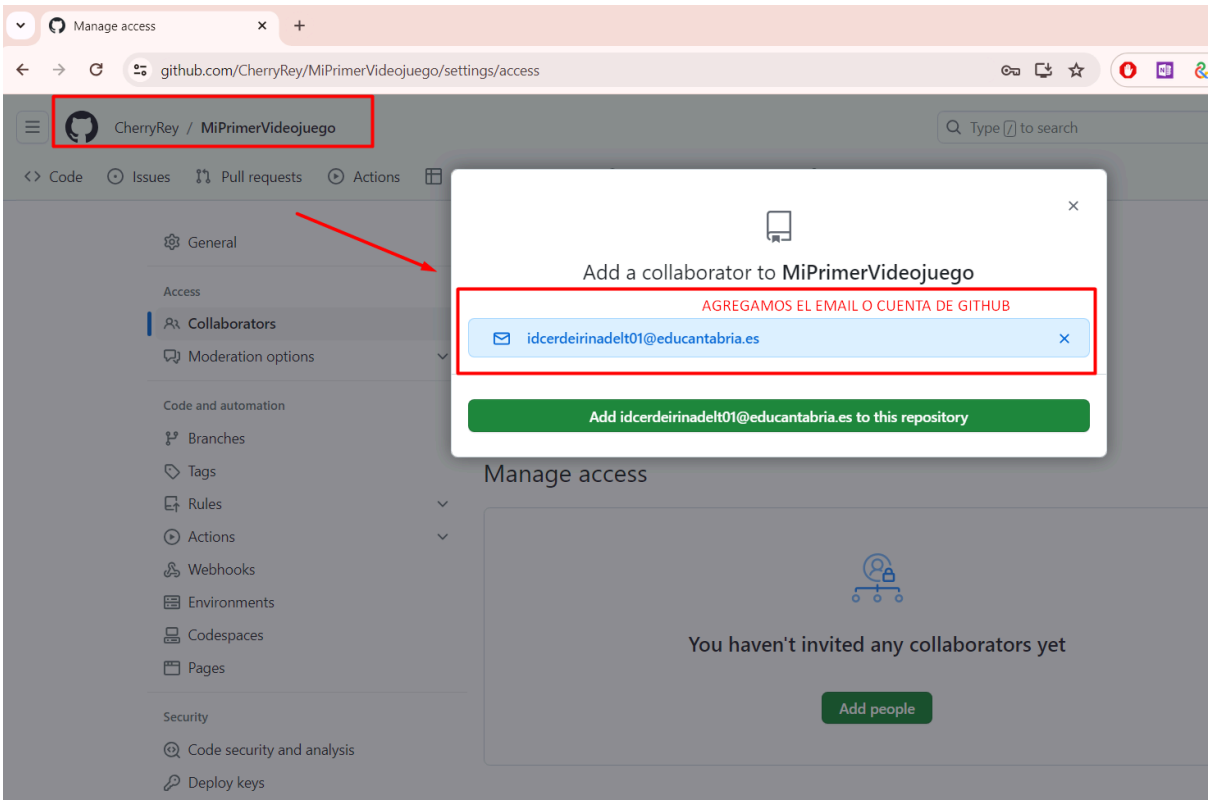
https://github.com/CherryRey/MiPrimerVideojuego.git

También podemos invitar a nuestro repositorio remoto a otras personas o "colaboradores". Esto lo hacemos desde la página de nuestro proyecto en *GitHub → Settings → Collaborators → Add people*[1]*:*

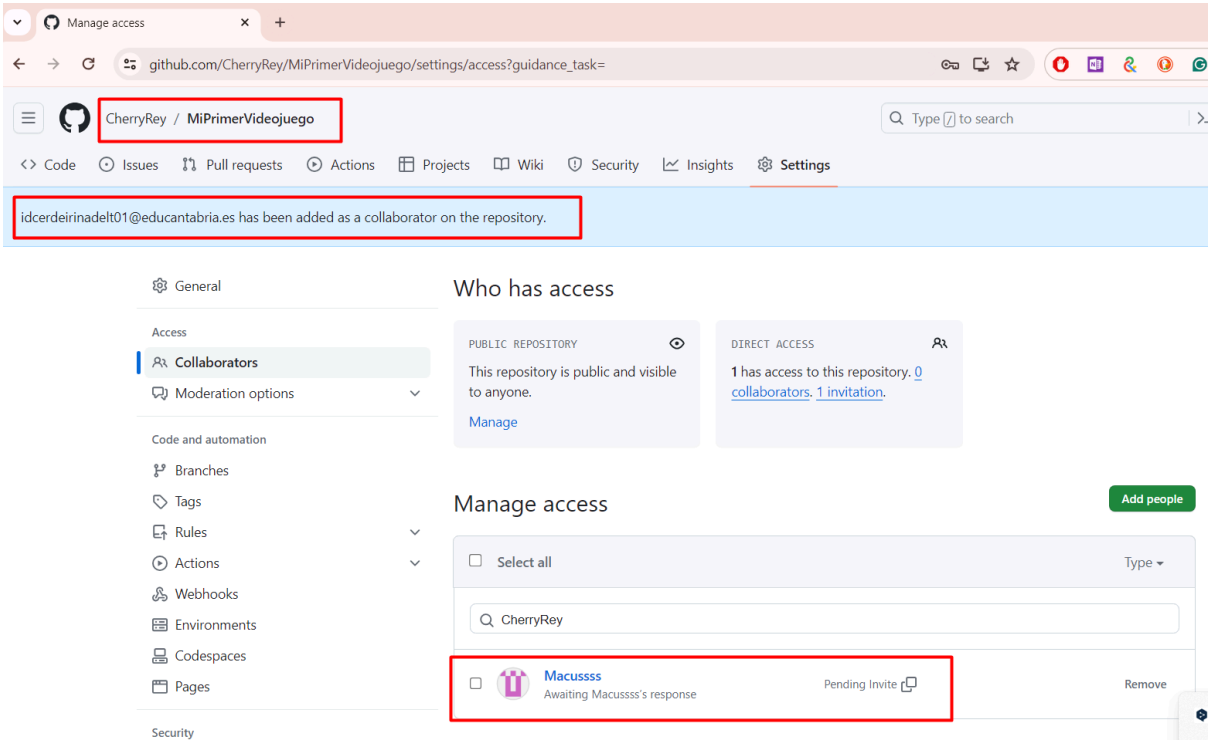
1 https://docs.github.com/es/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-access-to-your-personal-repositories/inviting-collaborators-to-a-personal-repository



1.14 Podemos compartir, invitar y añadir colaboradores en GitHub desde Settings. Cherry Reynoso Catalán.



1.15 Agregamos un email en especifico para enviar la invitaciónal colaborador. Cherry Reynoso Catalán.



1.16 Nuestra invitación ha sido enviada y queda pendiente de que el colaborador la acepte. Cherry Reynoso Catalán.

Fuentes Consultadas:

- https://docs.github.com/es/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-access-to-your-personal-repositories/inviting-collaborators-to-a-personal-repository