Práctica Refactorización Y Documentación

Entornos de Desarrollo
Zahira Zamora Camacho
Carlos Ignacio Domínguez Merchán
Febrero 2013

Índice

Objetivos de la práctica

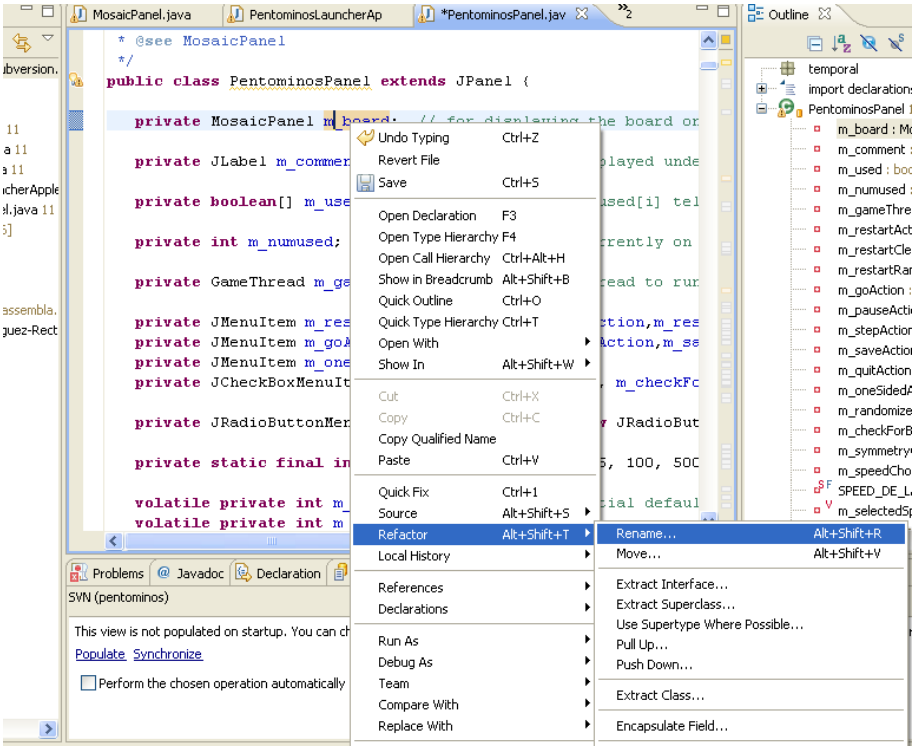
Se pide mejorar ciertas partes del código de un proyecto que resuelve el problema del pentominó, para ello usaremos la herramienta de desarrollo Eclipse y sus funciones de refactorización. Además para mayor legibilidad se incluirán comentarios JAVADOC que exportaremos junto con el proyecto *pentominos* refactorizado. Como sistema de comunicación entre programadores utilizaremos un repositorio SVN (*https://subversion.assembla.com/svn/refactorizacion_pentominos*). El objetivo principal es familiarizarnos con las funciones de refactorización que Eclipse nos brinda y con el uso de comentarios JAVADOC.

Desarrollo de la práctica

1.- Opciones de refactorización de Eclipse

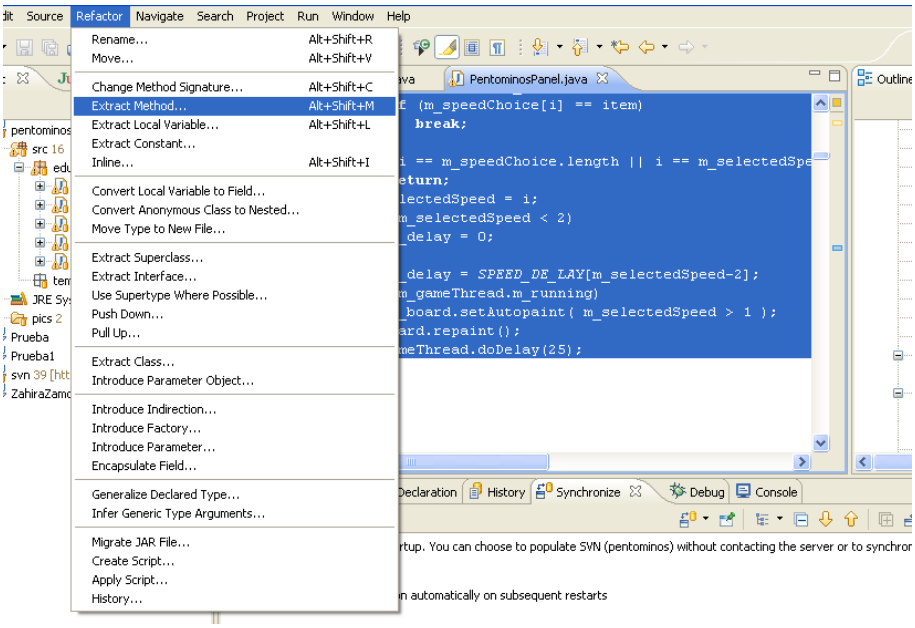
1.1.- Renombrado de nombres en todo el proyecto

Es una de las funciones más interesantes ya que con un sólo cambio nos permite modificar todas las referencias a ese nombre, ya sea variable, método, etc. Nos posicionamos encima del nombre, por ejemplo de una variable. Después tenemos varias opciones. Una de ellas es con botón derecho Refactor → Rename… Otra es Alt + Shift + R Lo cambiamos y pulsamos Intro.

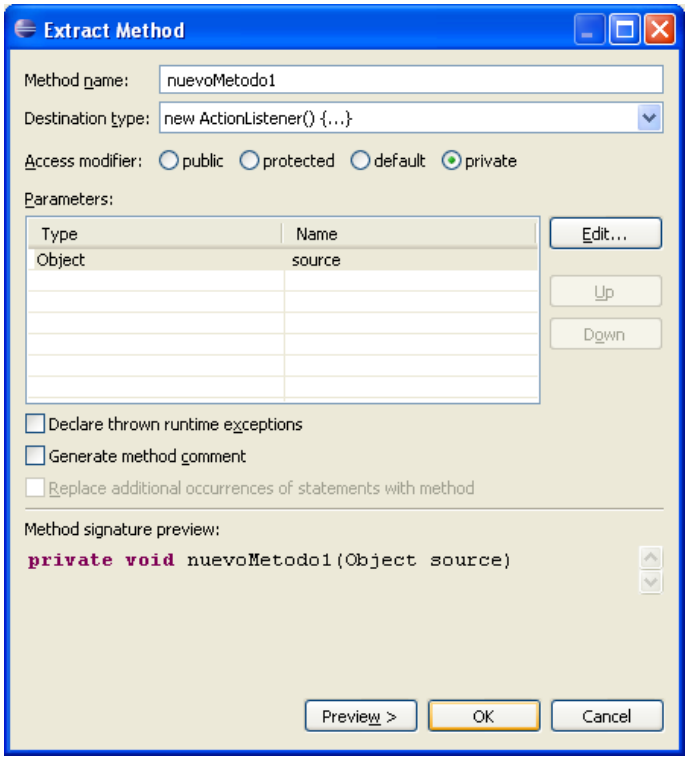


1.2.- División de métodos en partes

Cuando tenemos un método demasiado largo podemos dividirlo en partes, marcando las líneas de código a extraer después Refactor → Extract Method o bien Alt+Shift+M

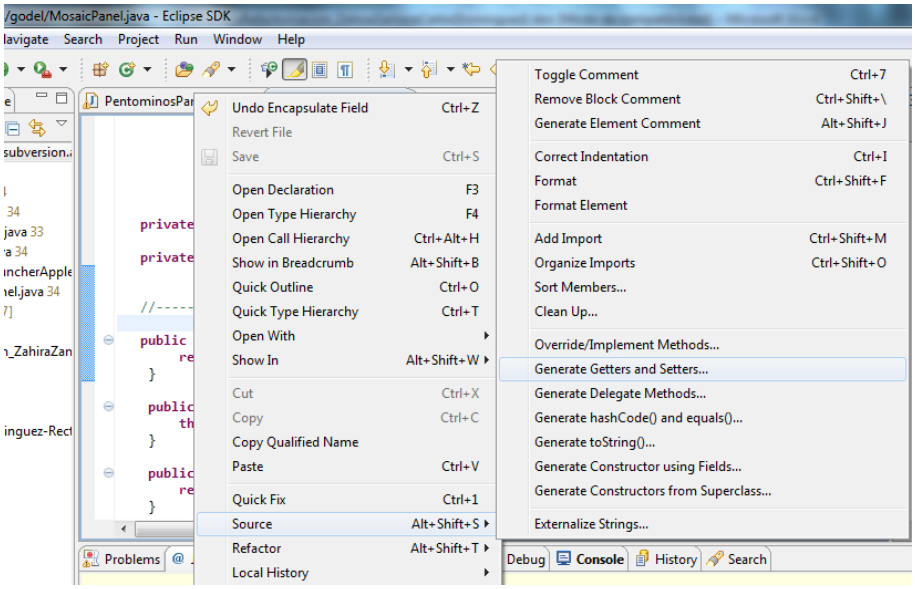


Elegiremos un nombre para el nuevo método. Usando la opción Preview podemos ver los cambios antes de que se hagan.

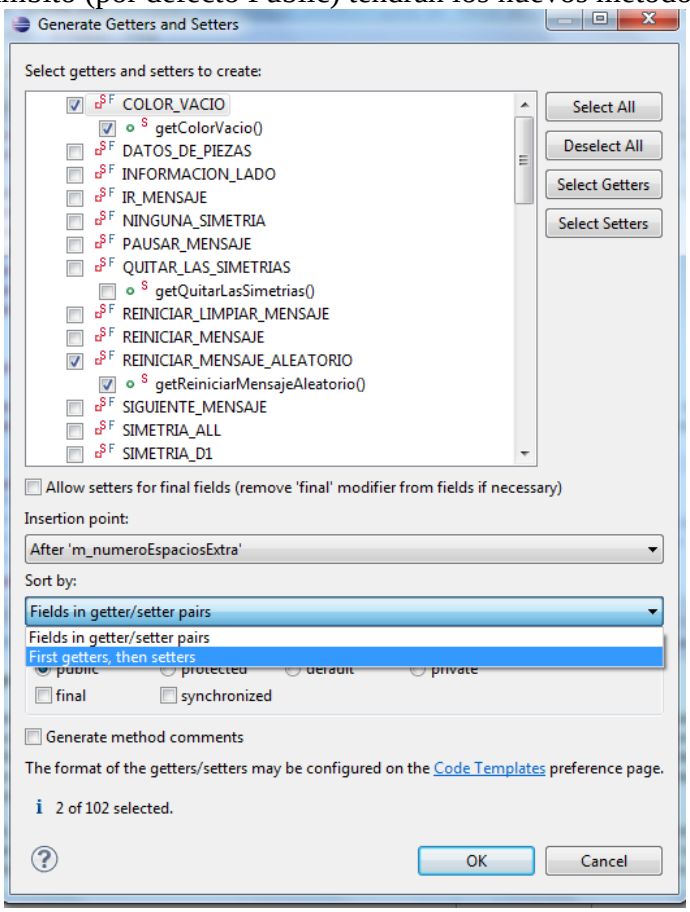


1.3.- Creación automática de Getters and Setters

Una forma rápida de crear los métodos get y set para los atributos de nuestras clases es situarnos en el código de nuestra clase pulsar botón derecho del ratón Source →Generate Getters and Setters

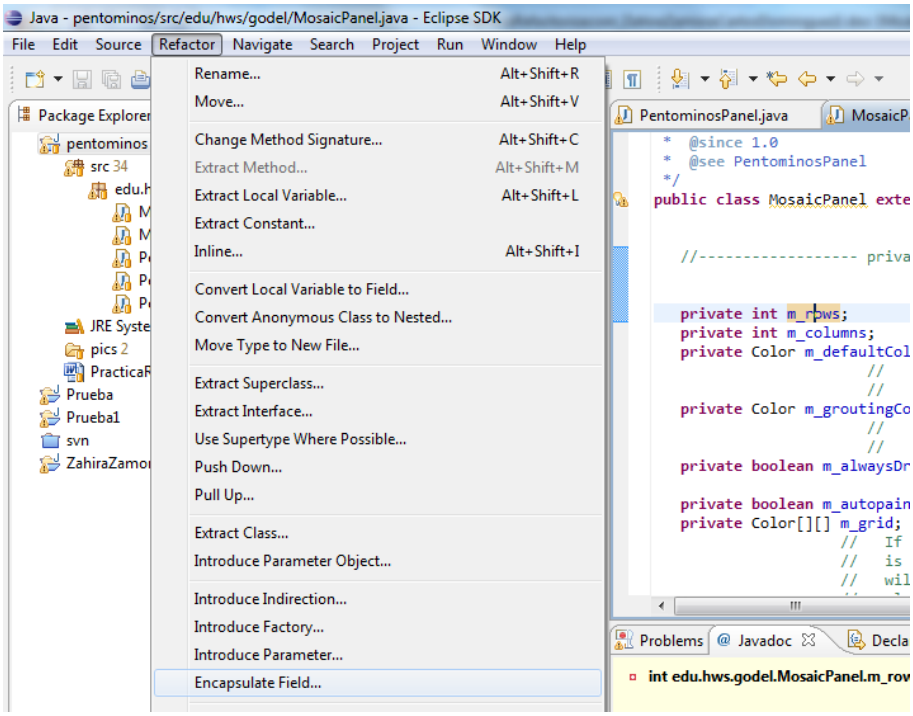


En la siguiente ventana podremos elegir que get y set de cada atributo queremos crear y en qué orden, además de que ámbito (por defecto Public) tendrán los nuevos métodos.

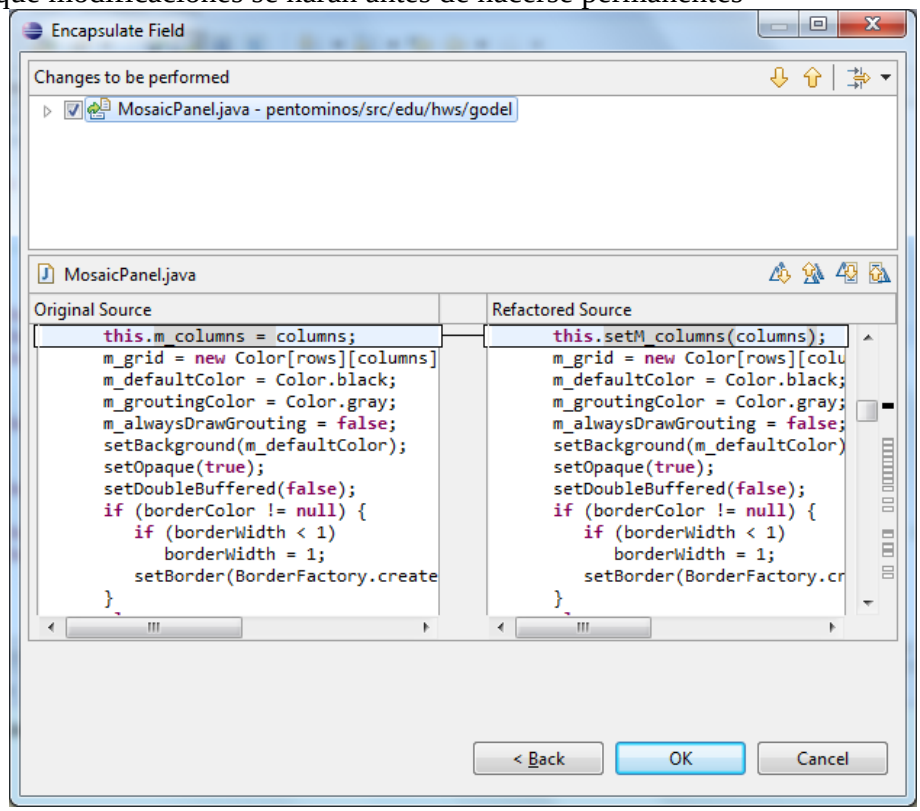


1.4.- Encapsular atributos

Si lo que necesitamos es encapsular atributos para ser usados mediante métodos set() y get() usaremos la opción Refactor → Encapsulate Field, posicionándonos antes sobre el atributo a encapsular.

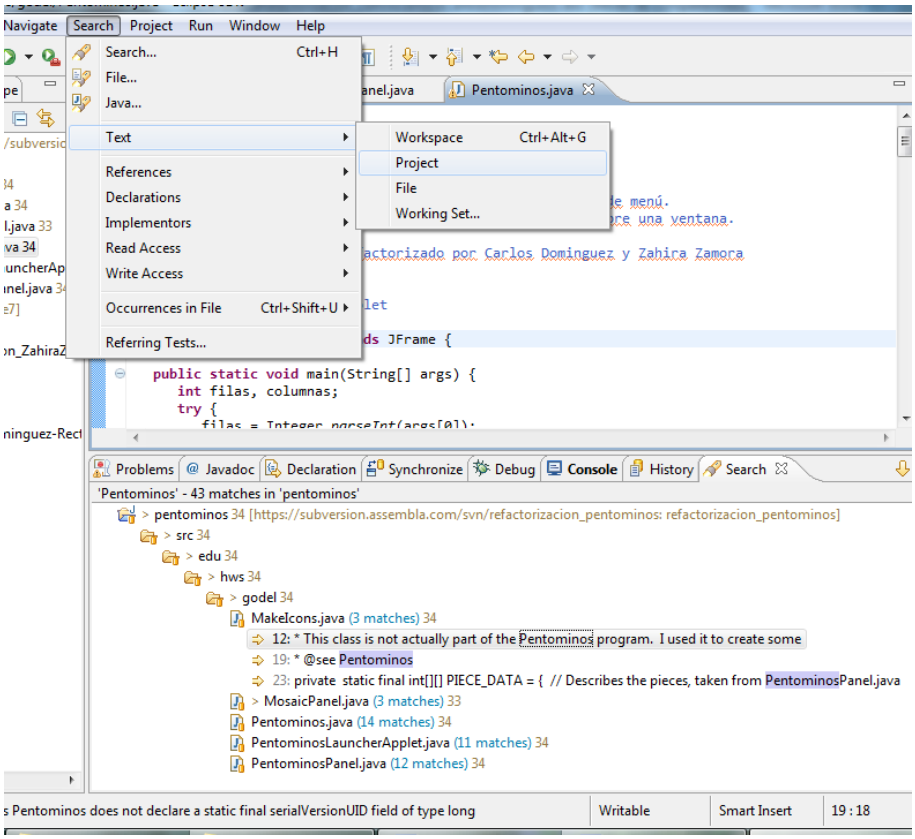


Podemos ver que modificaciones se harán antes de hacerse permanentes



1.5.- Coincidencias de un atributo, variable, método, clase…

Para saber en cuantos sitios de nuestro proyecto tenemos coincidencias, nos posicionamos encima del término en nuestro código y vamos a Search → Text → Project.
En la ventana de debajo se nos abre una nueva pestaña Search con los resultados y podemos saltar cómodamente entre cada coincidencia esté en la clase que esté.



2.- Refactorización del proyecto

Hemos hecho las siguientes modificaciones al proyecto pentominos:

- Los atributos de las clases comienzan con el prefijo m_ . Los atributos estáticos empiezan por M_ . Los atributos finales y estáticos no tienen prefijo, pero están en mayúsculas. Lo hemos realizado mediante la opción *Rename* (ver 1.1).
- En el PentominosPanel.menuHandler cada acción se ha realizado en un método aparte, en vez de dentro del if encadenado. Hemos usado la opción *Extract Method* (ver 1.2).
- El método PentominosPanel.doSaveImage se ha dividido en dos: Una parte pide un fichero y comprueba que no existe, la otra parte utiliza ese fichero para grabar la imagen. Se ha realizado también con la opción *Extract Method.*
- Los atributos privados de MosaicPanel no se acceden directamente, sino através de los Getters y Setters. Realizado mediante la opción automática de *creación de Getters y Setters* (ver 1.3) y después hemos usado la opción de refactorización *Encapsulate Fields* (ver 1.4).
- Las clases Pentominos y PentominosPanel se han traducido al español, tanto nombres de métodos como variables, atributos y parámetros. Realizado mediante la opción *Rename.*
- Todas las clases forman parte del paquete edu.hws.godel. Creamos un nuevo *package* y arrastrando las clases al nuevo paquete.

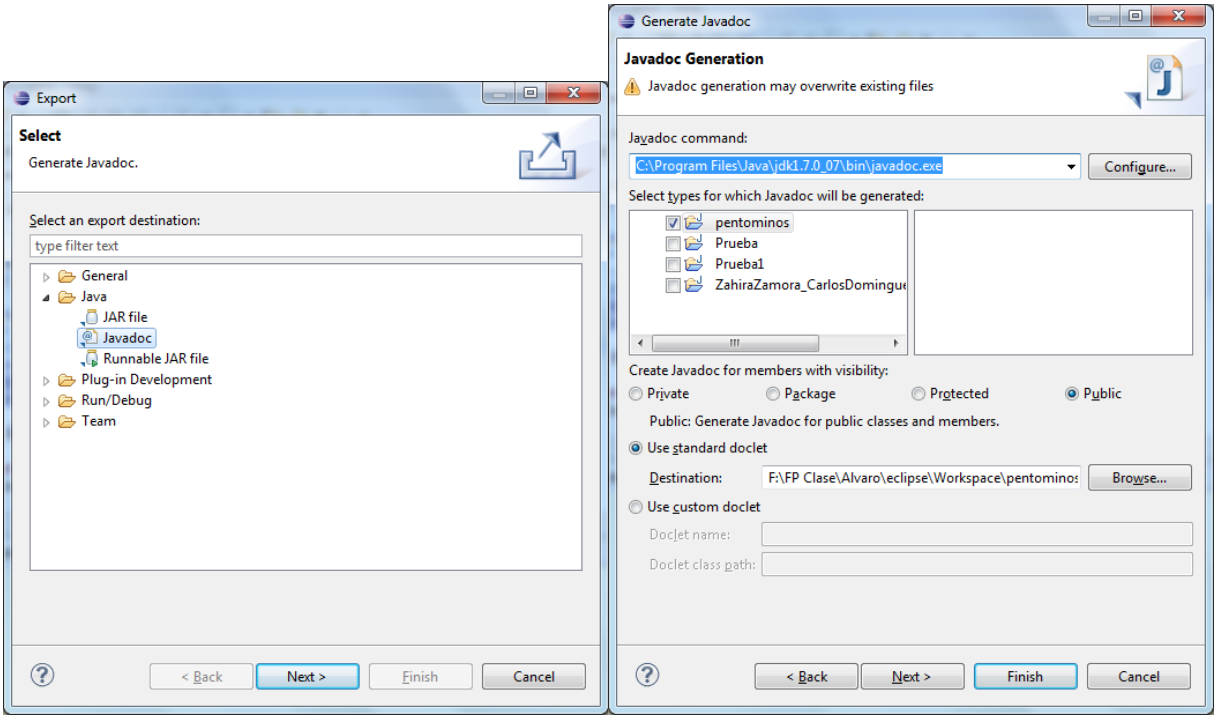
3.- Documentación del proyecto pentominos (Javadoc)

La adhesión de comentarios Javadoc enriquece los proyectos Java para su legibilidad y comunicación con otros grupos de desarrolladores.
En cada clase se pueden añadir diferentes etiquetas (tag) al estilo HTML, como @author (que se ha añadido a todas las clases) así como referencias @see a otras clases que se usen o por la que sea usada. Además para cada método se han añadido las etiquetas @param para todos sus atributos, si los métodos tienen retorno @return y por último, cada método tendrá una referencia (@link) a algún método que sea llamado dentro de él. Si no llama a ningún método, tendrá una referencia a un método que lo llame.

```
public boolean getAlwaysDrawGrouting() {
   return m_alwaysDrawGrouting;
}


/**
 *  Set the number of rows and columns in the grid.  If the value of
 *  the preserveData parameter is false, then the color values of all
 *  the rectangles in the new grid are set to null.  If it is true,
 *  then as much color data as will fit is copied from the old grid.
 *  @param rows int
 *  @param columns int
 *  @param preserveData boolean
 *  @see #forceRedraw()
 *
 */
public void setGridSize(int rows,
      int columns, boolean preserveData) {
   if (rows > 0 && columns > 0) {
      Color[][] newGrid = new Color[rows][columns];
      if (preserveData) {
         int rowMax = Math.min(rows,this.getM_rows());
         int colMax = Math.min(columns,this.getM_columns());
         for (int r = 0; r < rowMax; r++)
```

Una vez que hemos terminado la inclusión de comentarios Javadoc, generaremos la documentación automáticamente yendo a Export → Java → Javadoc, después marcaremos nuestro proyecto y Finish.



La documentación por defecto se nos guardará en una carpeta llamada doc dentro de nuesto proyecto en el workspace. En esta carpeta tendremos una serie de archivos Html, y como archivo de inicio abriremos en Index.html para abrir la documentación de nuestro proyecto.