ENTORNOS DE DESARROLLO – UF2 (ACTIVIDAD B)

Define el concepto de refactorización.

La refactorización consiste en someter el código de un programa a una limpieza que provocará que este sea más limpio, con un diseño más simple y que sea más fácil de arreglar y de modificar, en definitiva, un código mejorado. Dicho esto, hay que destacar que la refactorización no nos lleva a arreglar errores ni a incorporar funcionalidades, alterna la estructura del código sin cambiar su comportamiento externo y una cosa, a tener muy en cuenta es que, si durante una refactorización se ha cambiado el comportamiento del software o web, es que se ha generado un error o bug.

Los objetivos de la refactorización son:

* Mejorar la facilidad de comprensión del código.
* Cambiar su estructura y diseño.
* Eliminar código muerto.
* Facilitar el mantenimiento en el futuro.

¿La refactorización incluye cambios en el comportamiento del software? ¿Si/No y por qué?

Como ya he mencionado en el anterior punto, la refactorización no supone incluir cambios en el comportamiento del software, como tampoco supone arreglar errores ni incorporar funcionalidades nuevas al programa, únicamente se trata de hacer una limpieza en dicho código para su mejor comprensión. Si provocáramos cambios en el comportamiento sería porque se ha generado un error o bug.

¿Cuáles son las limitaciones que existen para aplicar la refactorización y cuáles son los patrones de refactorización más usados?

Las limitaciones que existen a la hora de realizar una refactorización serían:

* Los proyectos suelen estar condicionados por el aspecto económico.
* La refactorización no puede facturarse a un cliente inmediatamente.
* El programador que aplica la refactorización puede no ser el mismo que el creador del código, este hecho aumenta el riesgo de introducir nuevos errores.
* Las modificaciones del código han de poder probarse sin esfuerzo, por lo que hay que tener preparadas unas pruebas funcionales o unitarias.
* Si se trata de un código antiguo en el que ha trabajado muchos programadores, algunos de ellos noveles, podemos encontrarnos con un auténtico lío cuya mejora puede costar cara.
* A veces no podemos cambiar la interfaz de usuario y se complica la refactorización.

En cuanto a los patrones más usado mencionaré los siguientes:

* **Renombrado.** Este patrón es el que nos hace cambiar el nombre de cualquier elemento del código, ya sea un paquete, una clase, un método o un campo, por uno que sea más significativo para la función que esté realizando.
* **Sustituir bloques de código por un método.** Este patrón es que nos hace cambiar un bloque de código para que este se haga desde un método. La utilización de métodos incorpora muchas ventajas, como puede ser la reutilización de su código tantas veces se quiera, únicamente invocándolo por su nombre.
* **Campos encapsulados.** Es aconsejable que cada campo definido en una clase cuente con su método getter y su método setter. Esto nos permitirá acceder al campo o modificarlo si se requiere.
* **Mover la clase.** Con el objetivo de no duplicar código ya generado, si fuera necesario, se podría mover una clase de un paquete a otro, o de un proyecto a otro. Este hecho nos llevaría a tener que actualizar todo el código fuente para su concordancia con la nueva localización de la clase.
* **Borrado seguro.** Este patrón nos indica que debemos comprobar que, si hemos borrado algún elemento del código, borremos, también, todas las referencias que se hacían a él.
* **Cambiar parámetros del proyecto.** Nos permite añadir nuevos parámetros a un método y cambiar los modificadores de acceso.
* **Extraer la interfaz.** Crea una interfaz de los métodos “public” no estáticos seleccionados en una clase o interfaz.
* **Mover del interior a otro nivel.** Consiste en mover una clase interna a un nivel superior en la jerarquía.

Indicar la diferencia entre un analizador de código estático y uno dinámico.

Cuando se realiza un código debemos asegurarnos que este funciona, por ello es preciso analizarlo. Cuando recurrimos al análisis de código, lo estamos sometiendo a un proceso de revisión para evaluarlo, hecho que supone la búsqueda de problemas de funcionamiento para poder mejorarlo.

Existen dos herramientas creadas para esa función:

* **Análisis estático.** Cuenta con la particularidad de que se hace sin la necesidad de ejecutar el código. Esto proporciona que la búsqueda de errores se haga en un momento de escritura prematuro, lo que nos ahorra mucho tiempo en fases futuras del proyecto. El problema es que puede darnos errores que en realidad no lo son y solo se verían durante la ejecución del código.
* **Análisis dinámico.** Aquí, el análisis se realiza mientras el código se está ejecutando. Este necesita más tiempo que el análisis estático y necesita un proceso completo de testeo. El análisis dinámico permite ver muchos errores que quedan ocultos en un análisis estático.

¿Para qué sirve el control de versiones? ¿Cuál es la estructura de las herramientas de control de versiones?

El control de versiones es un sistema que nos permite registrar los cambios realizados sobre un archivo o conjunto de archivos, de manera que nos será posible recuperar cualquiera de las versiones por la que ha pasado. Los sistemas de control de versiones han ido evolucionando y se pueden clasificar en tres tipos:

* **Sistemas de control de versiones locales.** Almacenaban los archivos en una base de datos, no como archivos independientes. De esta forma, cuando se recurría a un proyecto solo se tenía una copia, eliminando la posibilidad de confundir o eliminar versiones. Se llevaba a cabo en el mismo ordenador del desarrollador y esto imposibilitaba una forma eficiente de compartir código.
* **Sistemas de control de versiones centralizados.** Con el objetivo de que varios desarrolladores trabajen con un mismo proyecto de manera más eficiente se creó este sistema de control de versiones. Aquí los cambios y versiones se almacenaban en un servidor al cual podían acceder todos los que intervenían en el proyecto. Planteaba un problema: ¿cómo trabajar varios usuarios sobre un mismo archivo?
* **Sistemas de control de versiones distribuidos.** En esta versión se optó por darle a cada desarrollador una copia local de todo el proyecto, creando una red distribuida de repositorios, en la que cada desarrollador podía trabajar de manera aislada pero teniendo un mecanismo de resolución de conflictos mucho más elegante que su versión anterior. Gracias a este sistema de control de versiones, cada desarrollador puede trabajar a su ritmo, almacenar los cambios a nivel local y mezclar los conflictos únicamente cuando se requiera. Otro punto a favor es que, como cada usuario tiene una copia, la pérdida de datos es muchísimo menos grave porque su recuperación es posible.

En cuanto a la estructura podemos decir que el sistema de control de versiones está formado por un conjunto de componentes:

* **Repositorio.** Es el lugar de almacenamiento de los datos de los proyectos. Suele ser un directorio en algún ordenador.
* **Módulo.** En un directorio específico del repositorio. Puede identificar una parte del proyecto o ser el proyecto por completo.
* **Revisión.** Es cada una de las versiones parciales o cambios en los archivos o repositorio completo. La evolución del sistema se mide en revisiones. Cada cambio se considera incremental.
* **Etiqueta.** Información textual que se añade a un conjunto de archivos o a un módulo completo para indicar alguna información importante.
* **Rama.** Revisiones paralelas de un módulo para efectuar cambios sin tocar la evolución principal. Se suele emplear para pruebas o para mantener los cambios en versiones antiguas.

También se pueden mencionar los siguientes elementos:

* **Tronco.** Es el camino de cambio principal.
* **Cabeza.** Es la última versión del tronco.
* **Delta.** Son los cambios de una versión respecto a otra.

Define qué es un repositorio.

Se trata de un lugar digital creado para almacenar información también digital de forma centralizada. Estos datos almacenados se podrán distribuir y podrán tener acceso público o privado. Los repositorios suelen contar con sistemas de respaldo y mantenimiento preventivo y correctivo, hecho que permite recuperar la información en caso de que el hardware se estropee (preservación digital).

Existen tres tipos de repositorios:

* **Repositorios institucionales.** Creados por la propia organización en la que se va a emplear para depositar, usar y preservar la producción científica y académica que estos generan.
* **Repositorios telemáticos.** Son los creados por un grupo de investigadores, una institución, etc. que reúnen documentos relacionados con un área temática específica.
* **Repositorio de datos.** Se trata de los repositorios que almacenan, conservan y comparten los datos de las investigaciones.

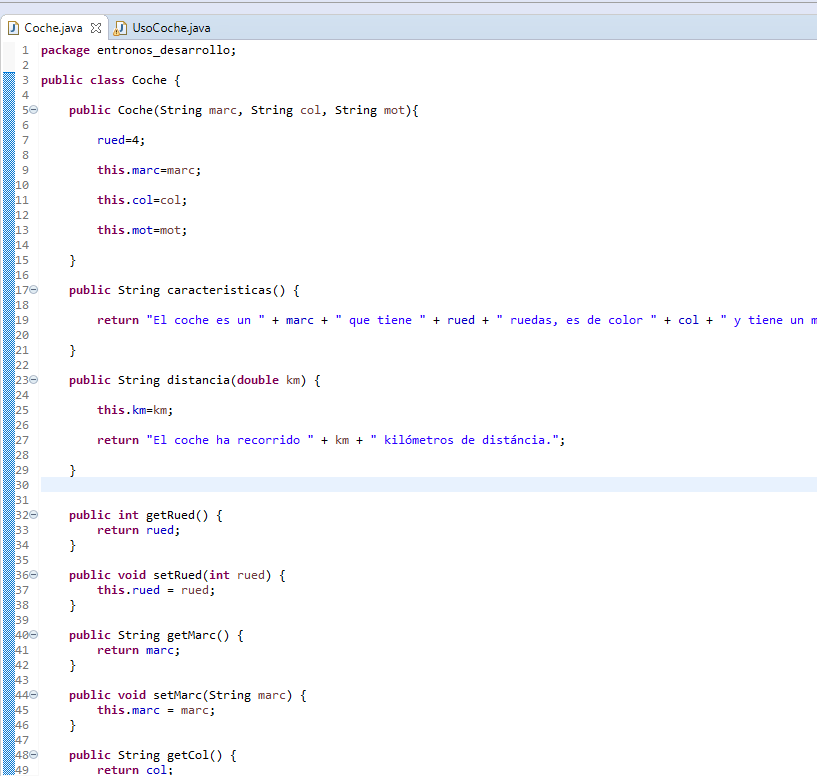
Aplicar refactorización a un programa.

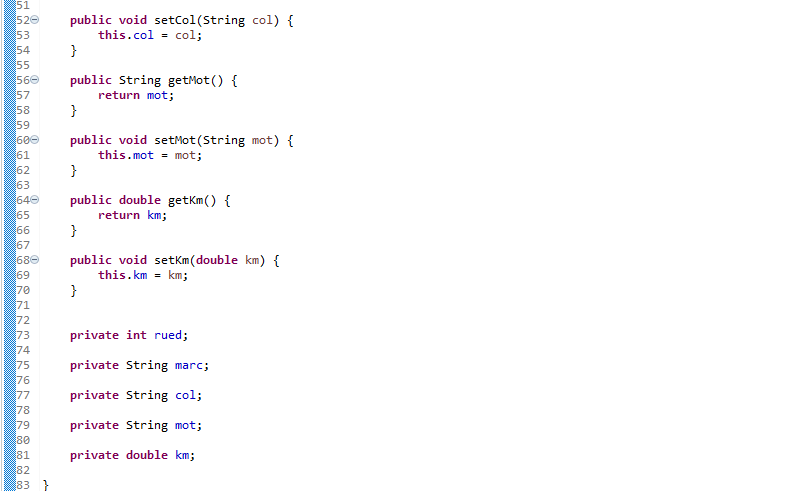
Utilizando el IDE Eclipse o Netbeans, proponer y realizar un programa en JAVA y posteriormente modificarlo aplicando la refactorización.

Indicar qué herramienta de refactorización has utilizado.

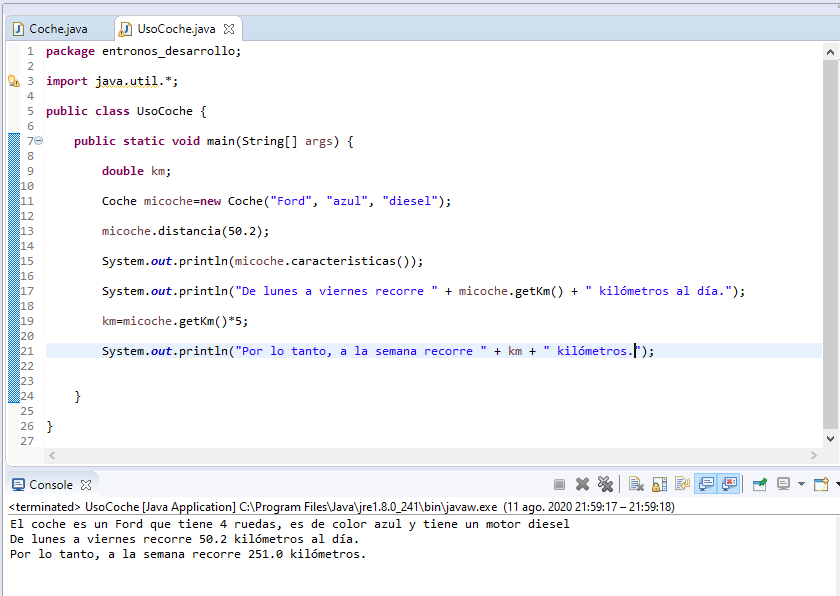
Para demostrar el antes y después de la refactorización, documentar cada paso, te puedes ayudar con capturas de pantalla o cualquier otro material en el que sea visible la refactorización.

A continuación, mostraré un sencillo programa hecho con Java desde Eclipse donde se visualiza la clase “coche” con sus correspondientes atributos y métodos.



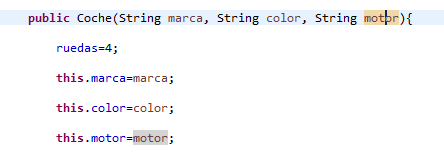


Seguidamente muestro la clase “UsoCoche” que es la que lleva el método “main” y se encarga de probar la clase “Coche” mostrada anteriormente.

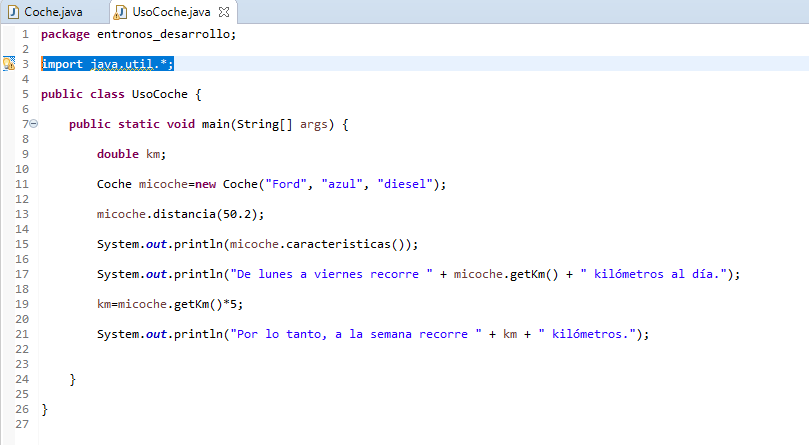


Ahora procederé a aplicarle una refactorización a algunos elementos del programa:

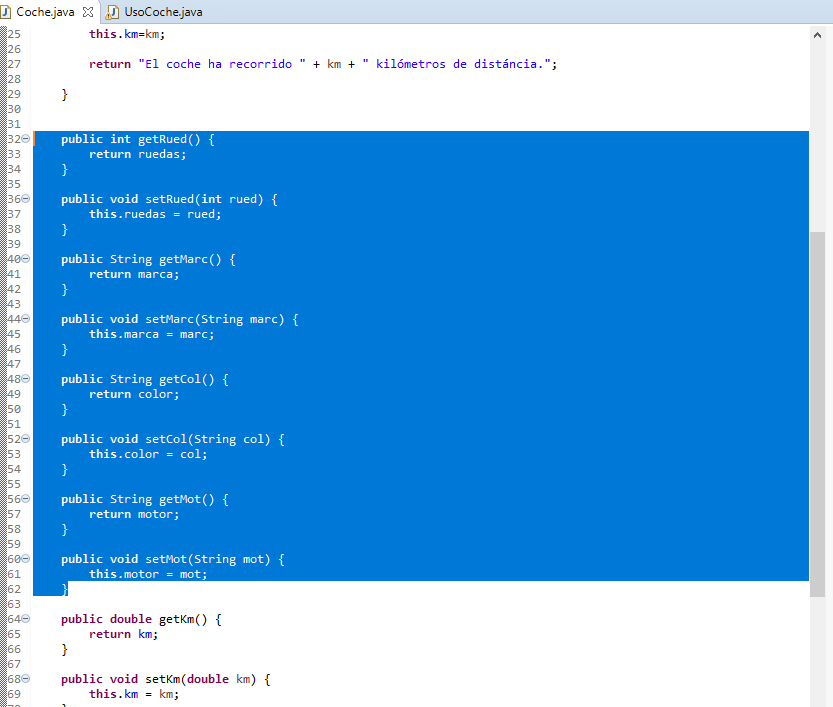
* Modifico los nombres por otros más representativos con la herramienta “Rename”, que se encuentra en la pestaña “Refactor”. Cualquier nombre que se debiera modificar se haría con esta herramienta, provoca que el nombre se cambie en todas las referencias hechas al elemento en el programa sin necesidad de hacerlo uno por uno.



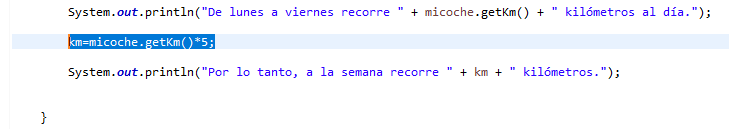
* Procedo, también, al borrado del código que invoca una librería que finalmente no se usa.

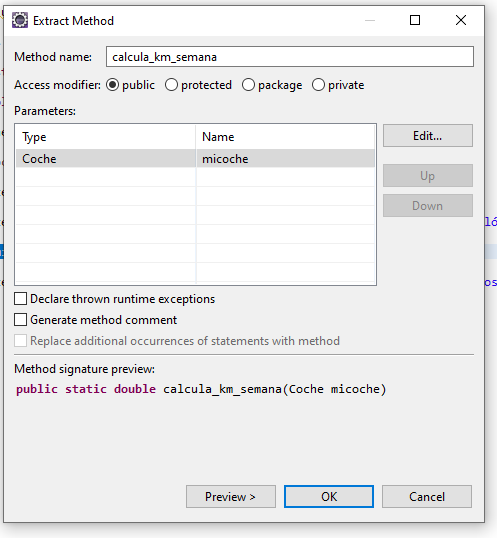


* Procedo al borrado de código muerto inutilizado.

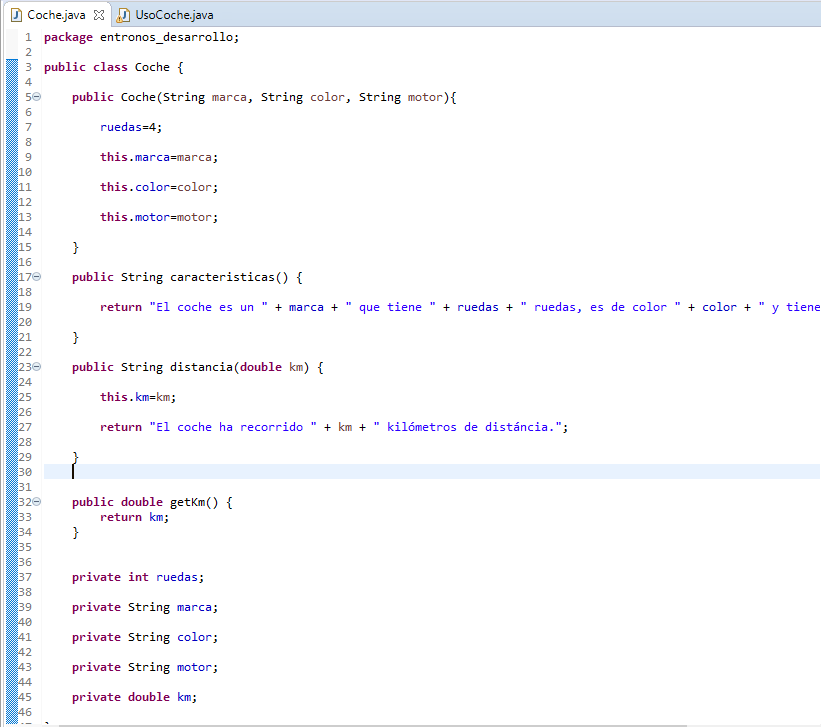


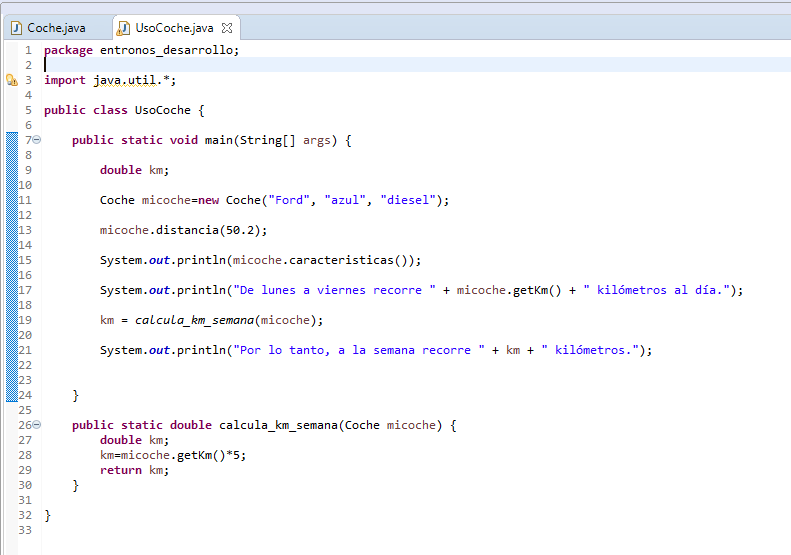
* Genero un método con la herramienta “Extract method” (se encuentra en la pestaña “Refactor”) partiendo de una línea de código de la clase “UsoCoche”.



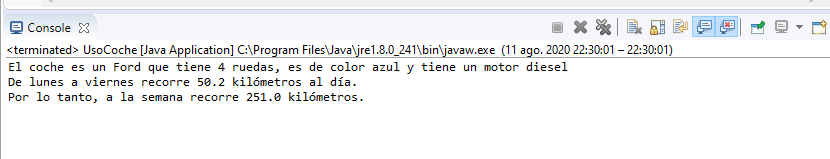


**Finalmente, el código queda de la siguiente manera:**





Por último, vemos que su resultado en consola es el mismo.



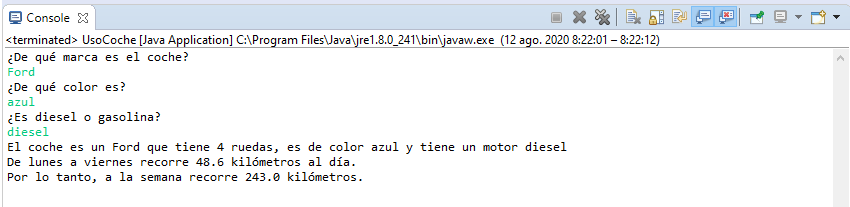
Análisis de código (estático/dinámico).

Al programa generado del caso práctico 1, analizarlo con uno de los analizadores de código, ya sea estático o dinámico, que están disponibles en los IDE de Netbeans o Eclipse.

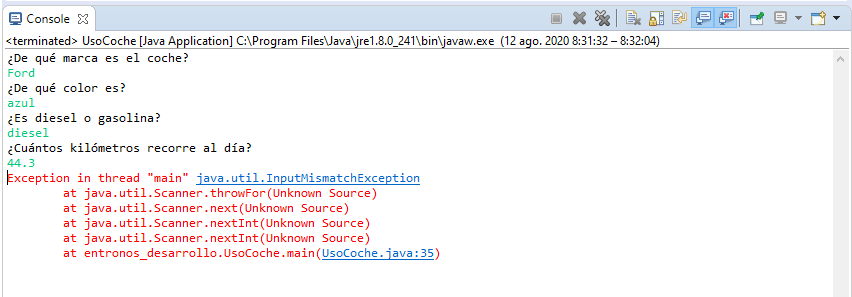
Indicar qué analizador de código has utilizado.

Documentar cada paso, te puedes ayudar con capturas de pantalla. Se deberá de visualizar el uso del analizador de código.

Con el objetivo de darle un poco más de participación al usuario, he introducido algún cambio al programa. De esta manera, se le piden los datos al usuario y estos quedan registrados para la creación del objeto “Coche”. Muestro, a continuación, la salida en consola:



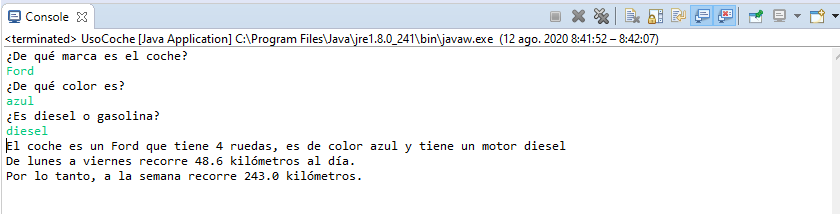
Observamos que el programa interactúa perfectamente con el usuario y trascurre sin ningún problema. Cuando ejecutamos el programa, se está aplicando un análisis de tipo dinámico. Si introdujésemos algún error que imposibilite la ejecución normal del programa, la salida en consola sería algo parecido a esto:



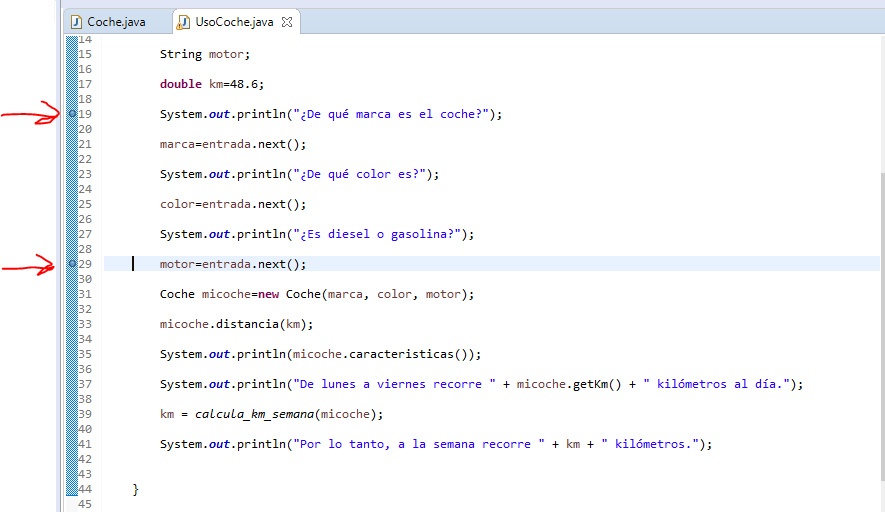
Podemos observar que nos lanza una excepción y nos indica el lugar donde se produce el error.

Dentro del IDE Eclipse (y en otros IDE’s también existe) tenemos la herramienta “Debug”, que nos aplica un análisis dinámico a nuestro programa, cuyo objetivo es probar y depurar los errores que pudieran existir dentro de este. Esta herramienta de análisis cuenta con la particularidad de permitirnos establecer puntos donde queramos que el programa detenga temporalmente su ejecución para el análisis de datos concretos, llamado “Toggle Breakpoint”.

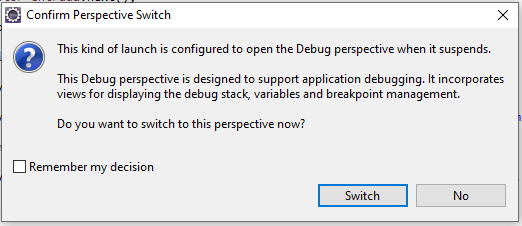
Procedemos a su uso, en el IDE Eclipse encontramos la herramienta “Debug” dentro de la pestaña “Run” del menú de navegación o pulsando la tecla F11. Vemos que la salida en consola es completamente correcta:



Aplicamos, ahora, los “Toggle Breakpoint”, también se encuentra dentro de la pestaña “Run”.

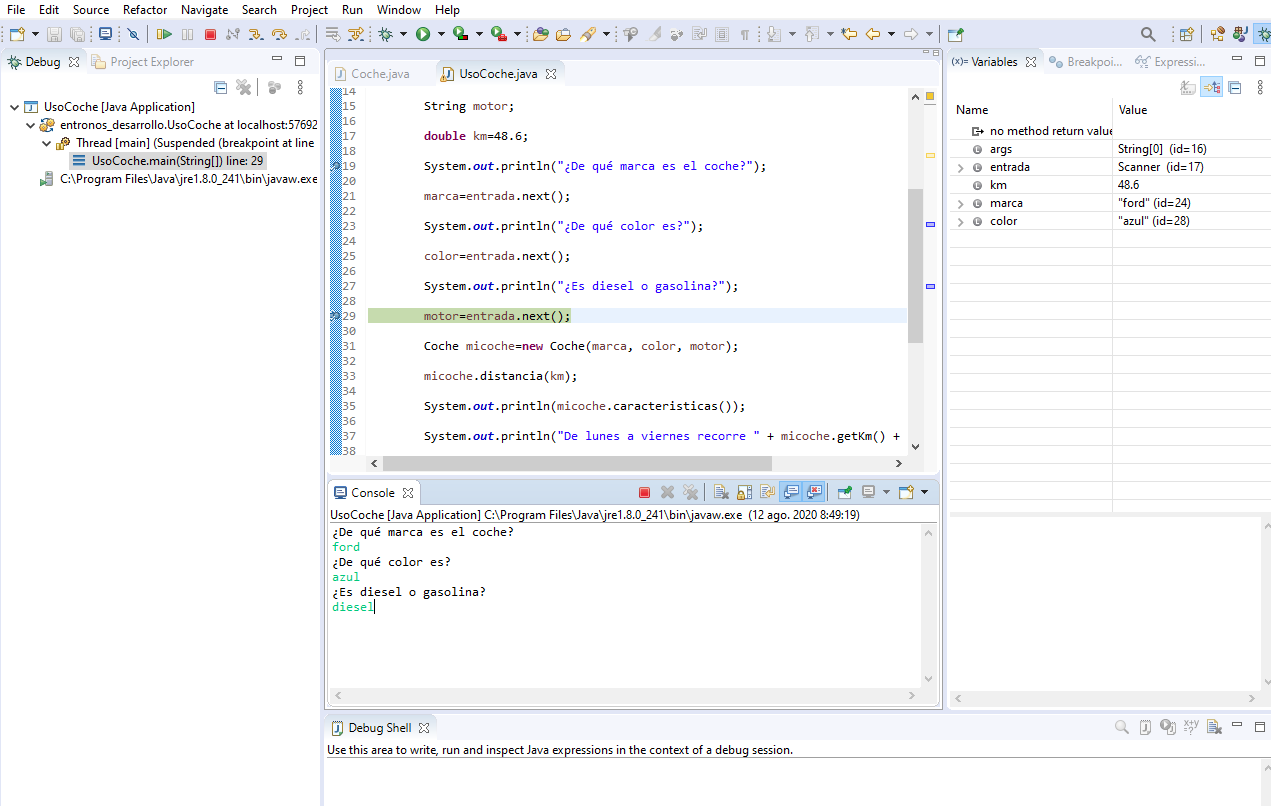


Y procedemos al “Debug”:

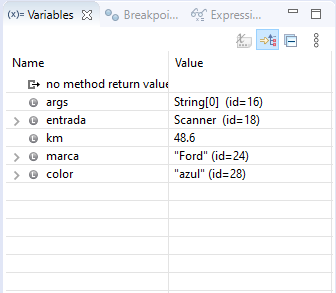


Durante el análisis deberemos usar las siguientes instrucciones según nuestra conveniencia:

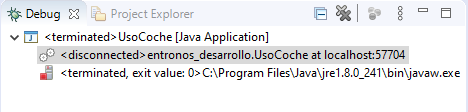
* RESUME (F8). Para continuar con la ejecución.
* SUSPEND. Para detener la ejecución en el momento que queramos.
* STOP. Para detener la depuración.
* STEP INTO (F5). Se detiene en la primera línea de código del método que se está ejecutando. De no existir, haría lo mismo que Step Over.
* STEP OVER (F6). Para pasar a la siguiente línea de código.
* STEP RETURN (F7). Para volver a la línea siguiente del método que hizo la llamada al método que se está depurando en ese momento.



Observamos que podemos visualizar el valor de las variables justo en el momento en el que estamos:



El análisis finaliza exitosamente:

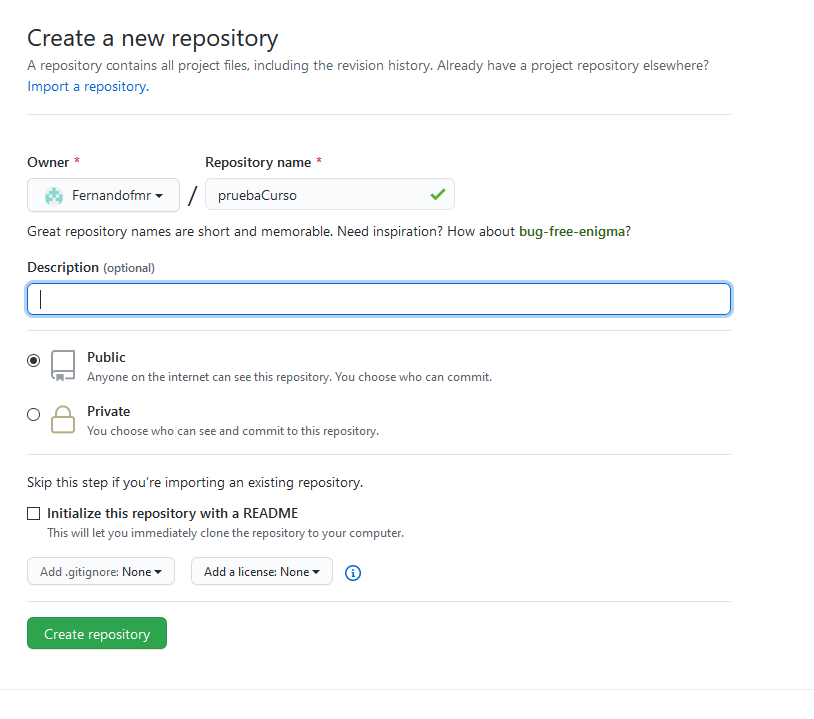


Genera un control de versiones al programa del caso práctico 1, al menos que aparezcan dos versiones. Utilizar uno de los clientes de control de versiones que viene integrado dentro del IDE Netbeans o Eclipse. Indicar qué cliente de control de versiones has utilizado.

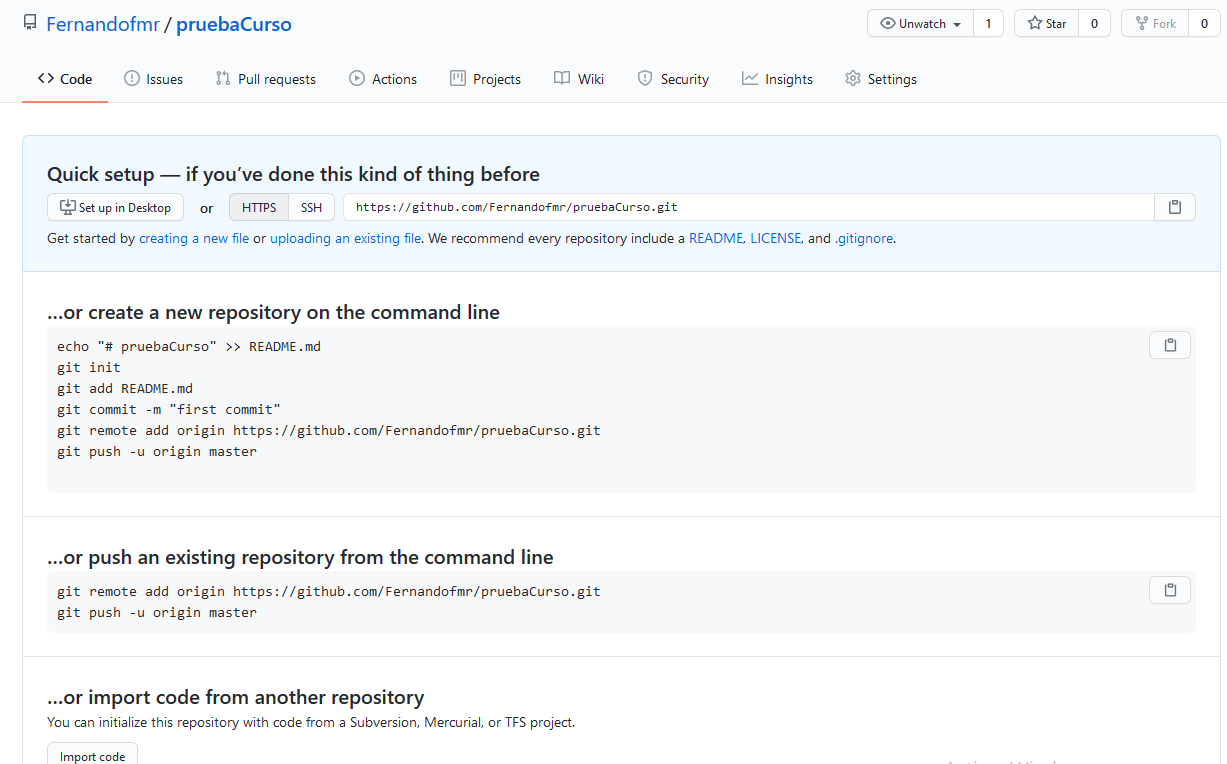
Documentar cada paso, te puedes ayudar con capturas de pantalla. Se deberá de visualizar el uso del control de versiones.

En mi caso he utilizado el programa Github para generar un control de versiones.

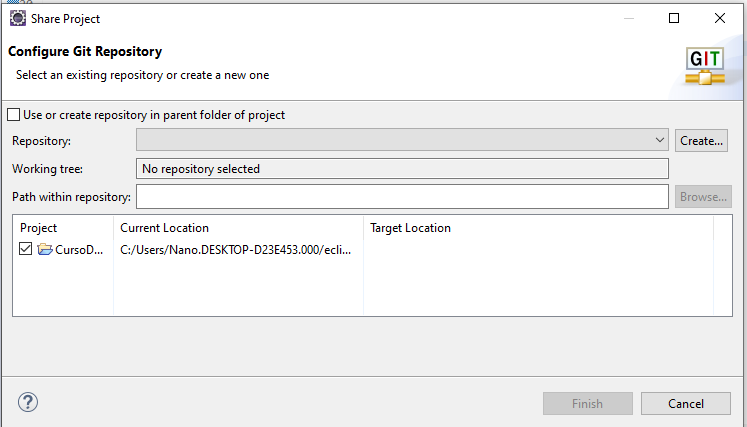
El primer paso ha sido crear una cuenta en Github y, una vez hecha la cuenta, he creado un repositorio.



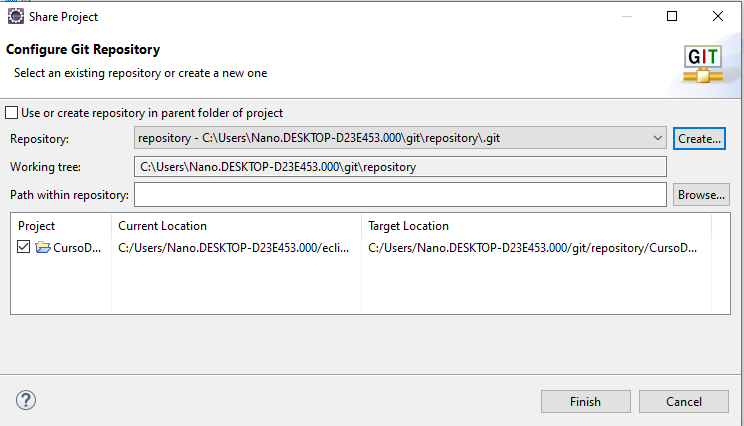
De esta forma, obtendremos la dirección del repositorio



Una vez creado, vamos a Eclipse y le damos click con el botón derecho al proyecto, damos click en “Team” y después “Share Project” y seguimos el proceso que marco a continuación:



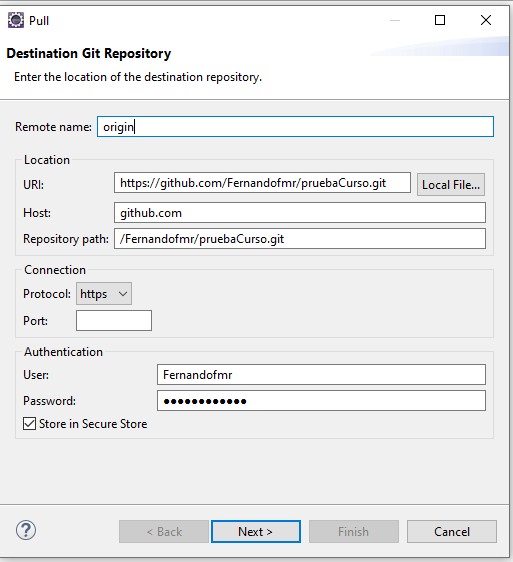
Introducimos la ruta del repositorio

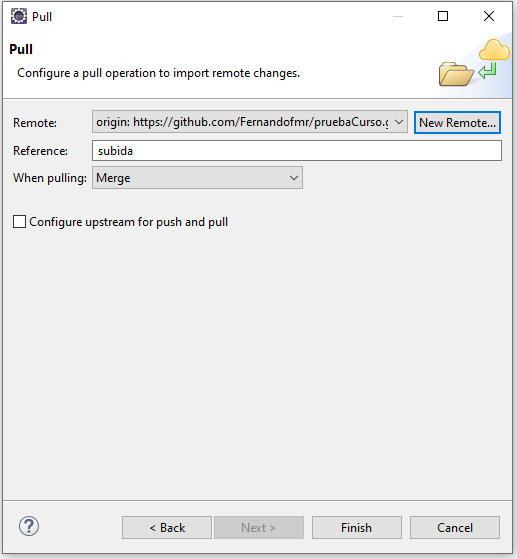


Se crea la conexión con Github

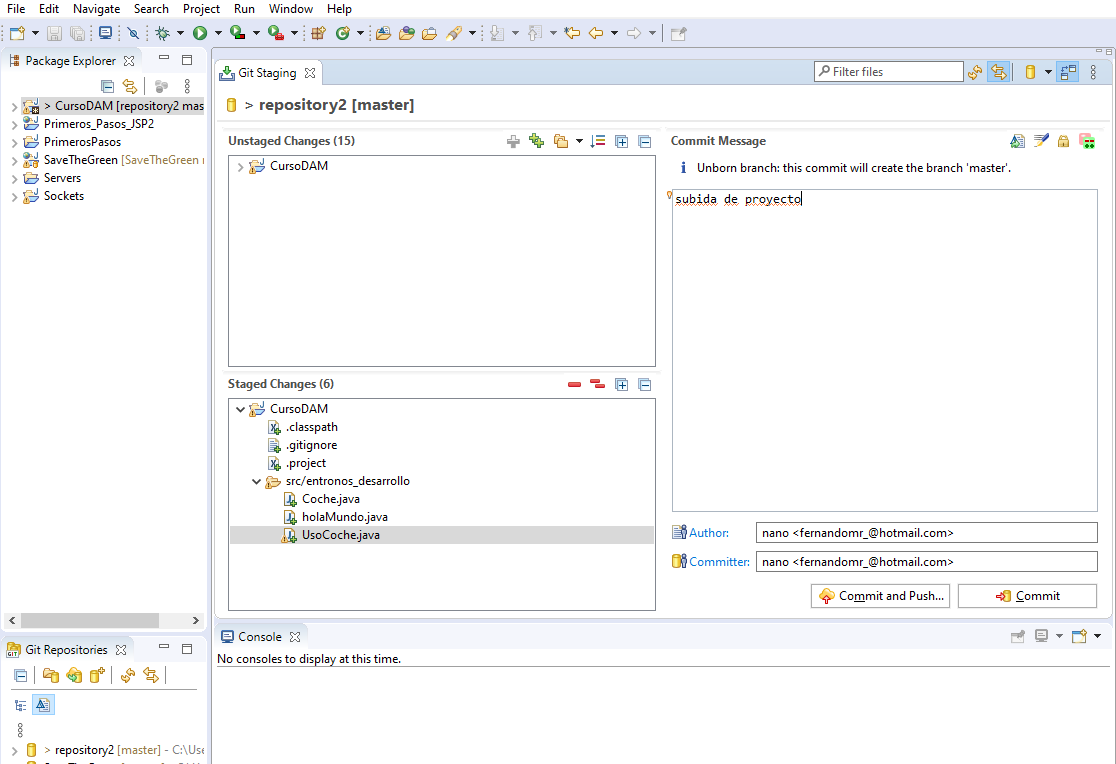


Subimos el proyecto al repositorio

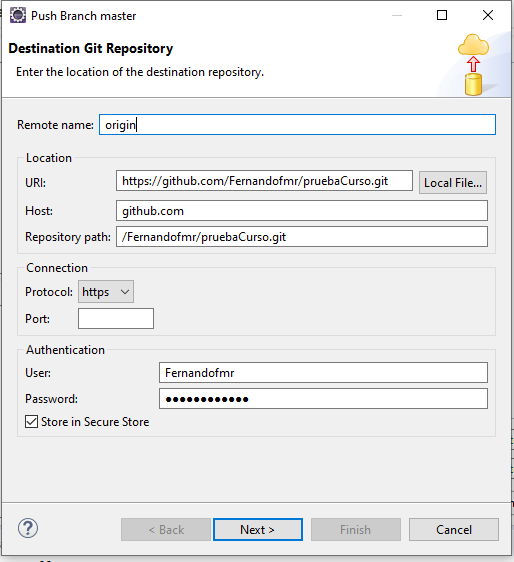


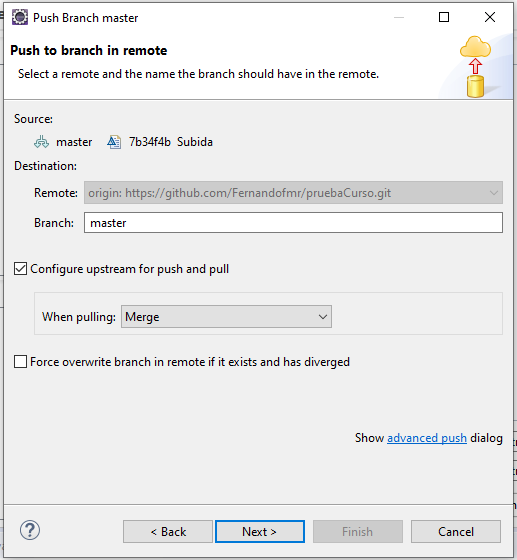


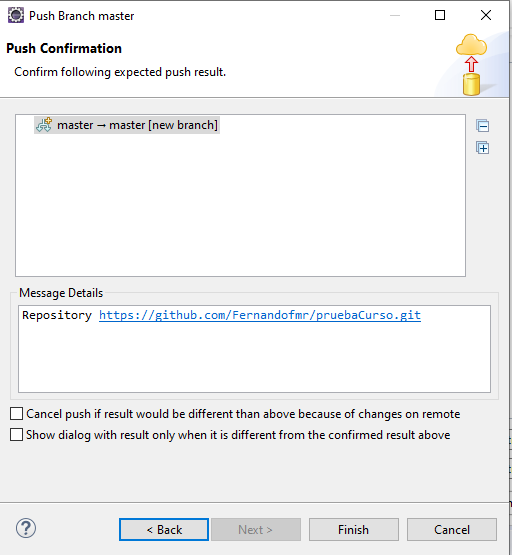
Hacemos un “commit” dándole click derecho al proyecto, indicándole lo que deseamos subir al repositorio y le ponemos un mensaje que nos servirá para localizar la versión

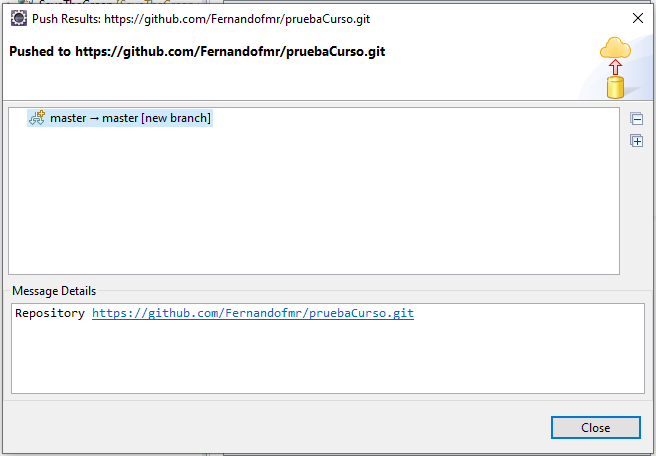


Seguimos los siguientes pasos









Por último, vamos a nuestra cuenta de Github y vemos que en el repositorio indicado se ha subido el proyecto satisfactoriamente:

