UNIVERSIDAD TECNOLOGICA DE HUEJOTZINGO

ACTIVIDAD 7

POLIMORFISMO

ABELGARCIA JUAREZ

3514110714

7 “A”

Contenido

[**Polimorfismo en c#** 4](#_Toc464592162)

[**Polimorfismo por Herencia:** 4](#_Toc464592163)

[**Polimorfismo por Interface:** 10](#_Toc464592164)

[GitHub 12](#_Toc464592165)

[Instalar GitHub 12](#_Toc464592166)

[CONCLUSION 16](#_Toc464592167)

[REFERENCIAS 16](#_Toc464592168)

Introducción

En este trabajo se nos en cargo todo sobre polimorfismo es un poco complicado, pero al entenderle es más sencillo, en este ejercicio que se nos dejó nos quedó más clero de lo que es polimorfismo y sus usos en la programación gracias a eso lo podremos aplicar en nuestros futuros proyectos.

Y así tener unos programas más eficientes en las siguientes actividades podremos ver más afondo de lo que es y de que trata el polimorfismo sin más pre ángulo comencemos.

# **Polimorfismo en c#**

Polimorfismo es uno de los pilares de la programación orientada a objetos que mas se utiliza, lo único que es dejado de lado por los programadores en cuanto a su uso en las aplicaciones de negocios.

En este post trataré de explicar su utilización en la programación. Primero tratemos de definir la palabra polimorfismo según una búsqueda por internet seria, Poli = muchos y Morfo = formas; si lo decimos literalmente seria “muchas formas”.

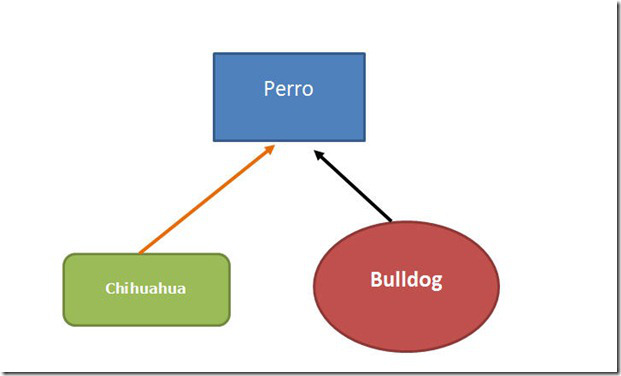
En programación seria la capacidad que tiene una clase en convertirse en un nuevo objeto sin cambiar su esencia y luego volver al objeto origina de donde salió.

Hay tres tipos de polimorfismo definamos cada uno:

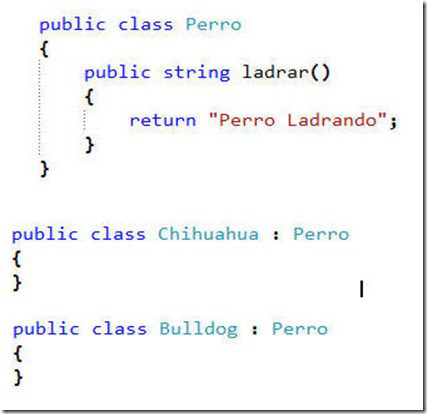
* 1. polimorfismo por herencia: cuando se hereda de una clase normal y puedo convertirme en ella.
  2. polimorfismos por abstracción: cuando puedo heredar de una clase abstracta y puedo convertirme en ella.
  3. polimorfismo por interface: es la posibilidad que tenemos de implementar una interface y puedo convertirme en ella.

# **Polimorfismo por Herencia:**

Este tipo de polimorfismo es el más común que existe, y tiene la facultad de heredar de una clase padre y reemplazarla.

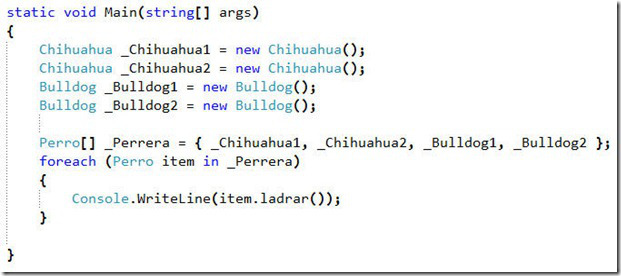
[](https://codigofuentenet.files.wordpress.com/2012/07/snap511.jpg)

En un proyecto de consola tendremos que crear estas tres clases:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap58.jpg)

En la clase perro tenemos un método que solo nos devuelve un string. por otra parte, las clases chihuahua y bulldog serán clases que heredarán el comportamiento de la clase perro.

Ahora supongamos que estos dos perros son callejeros y viene el control de animales y los meta a la perrera. Esta perrera será un arreglo o Array en nuestro program, de la siguiente manera:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap59.jpg)

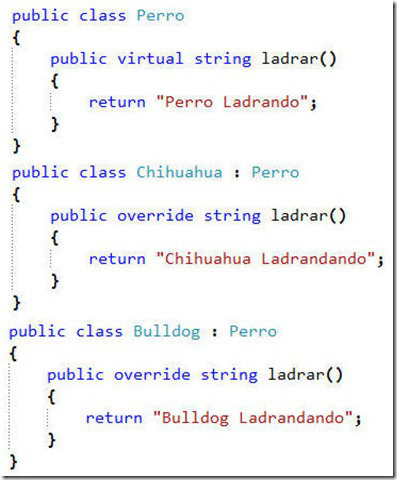
En el arreglo perrera he metido cada uno de los perros que instanciamos, si lo corremos nos daría esto:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap60.jpg)

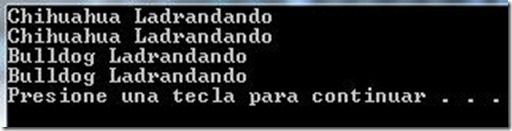
El problema en esta ejecución es que, aunque metimos distintos perros dentro de nuestra perrera como todos heredan de perros siempre dará el mismo mensaje. otra cosa que podemos observar es que en el foreach cuando ejecutamos el Console.WriteLine no nos da error esto se debe que al foreach le hemos dicho que todo está basado en la clase perro, de esta forma el runtime sabe que para estar en la perrera hay que heredar de la clase perro en pocas palabras ser un perro.

Cambiemos el comportamiento de los hijos; veamos supongamos que no todos los perros ladran de la misma forma el chihuahua ladra distinto que el bulldog.

Para lograr esto la clase perro tiene que dejar o permitir que los hijos modifiquen el comportamiento; miremos como quedan las clases ya modificadas:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap65.jpg)

Primero el método Ladrar es de tipo virtual esto me permitirá sobrescribir este método en los hijos, en las clases hijos usaremos la palabra clave override para sobrescribir el método virtual del padre. Si corremos la aplicación de consola nos dará esto:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap66.jpg)

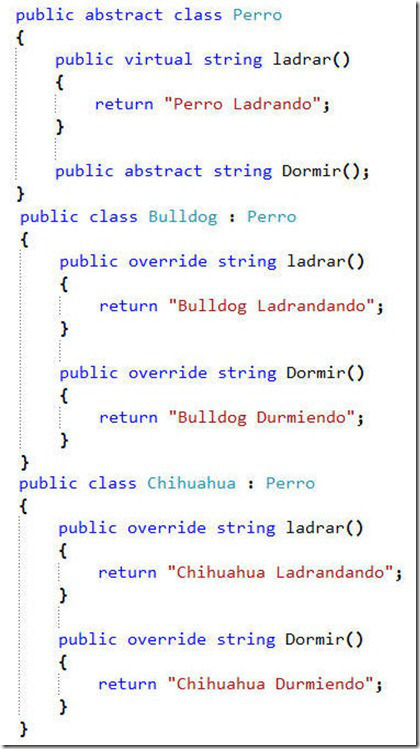
Cuando corrimos nuestro programa no se ejecutó el método del padre sino los cambios que hicimos en los hijos.

Polimorfismo por Abstracción:

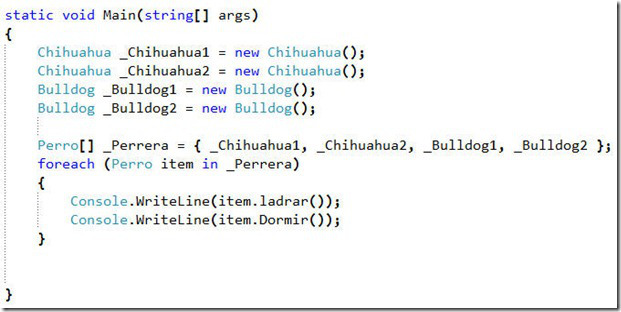
Este tipo de polimorfismo se da con el uso de las clases abstractas. Pero que es una clase abstracta es aquella que además de lo normal que contiene una clase tiene comportamientos que, si están definidos, pero no implementados.

Vemos lo siguiente que pasaría si en el ejemplo anterior nos vemos en la necesidad de implementar un comportamiento en los perros que sea dormir. El problema que no todos los perros duermen de la misma forma cada uno lo hace a su manera.

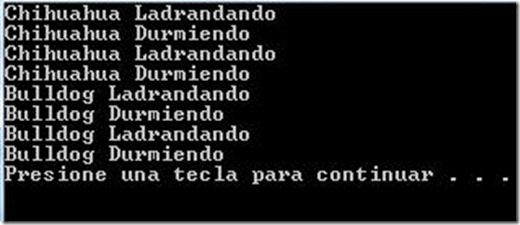
Primero para lograr esto debemos de declarar la clase perro como abstracta y luego declarar el método dormir para luego implementarlo en las clases hijos, de esta forma:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap73.jpg)

Como pueden ver hemos declarado la clase perro como abstracta y el método dormir también. Así que el Main del Program tendríamos que implementar el método abstracto de esta forma:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap74.jpg)

si lo corremos nos daría este resultado:

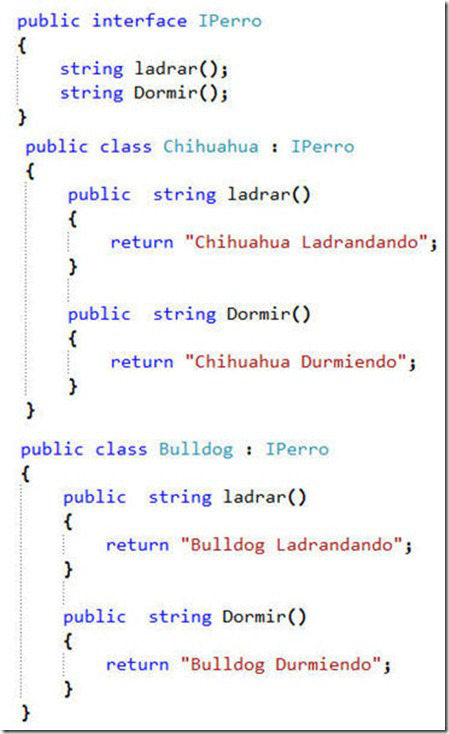
[](https://codigofuentenet.files.wordpress.com/2012/07/snap75.jpg)

Hay que recordar que las clases abstractas no se pueden instanciar.

# **Polimorfismo por Interface:**

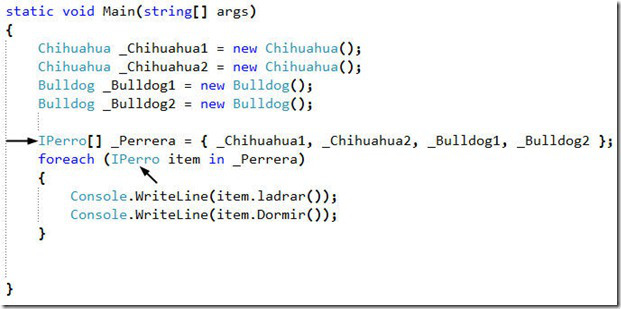
Es uno de los polimorfismos más importantes porque está basado por contratos, que son los encargados de decirme que puedo hacer o no y como debo de hacerlo.

En nuestro ejemplo vamos a crear un contrato que se llamara IPerro y definiremos los comportamientos que queremos en nuestra interfaz para luego implementarlo en nuestras clases hijos, de esta forma:

[](https://codigofuentenet.files.wordpress.com/2012/07/snap87.jpg)

Como pueden ver al convertir nuestra clase perro en una interfaz, tengo que especificar el comportamiento que quiero por parte de mi interfaz; en este caso Ladrar y Dormir.

En el Main del Program tendríamos que cambiar esto.

[](https://codigofuentenet.files.wordpress.com/2012/07/snap83.jpg)

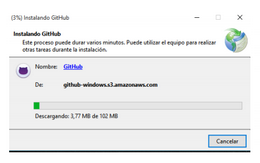
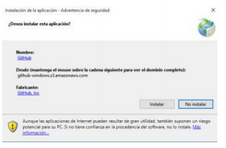
Una de las ventajas de hacer este tipo de polimorfismo es que oculta la implementación del comportamiento. Donde podemos ver este tipo de esquema es el WCF, por que el cliente solo puede ver el servicio y no la implementación del método.

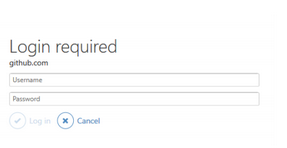
# GitHub

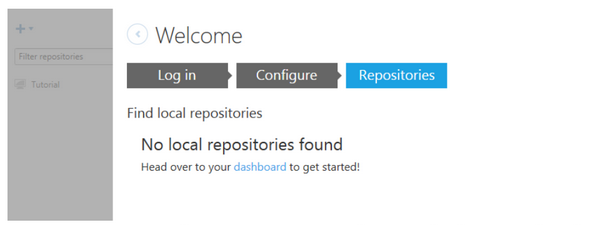
GitHub para Escritorios es una forma sencilla para contribuir a desarrollos de proyectos de forma colaborativa, además de llevar un control de versiones agilizando la implementación de aplicaciones.

## Instalar GitHub

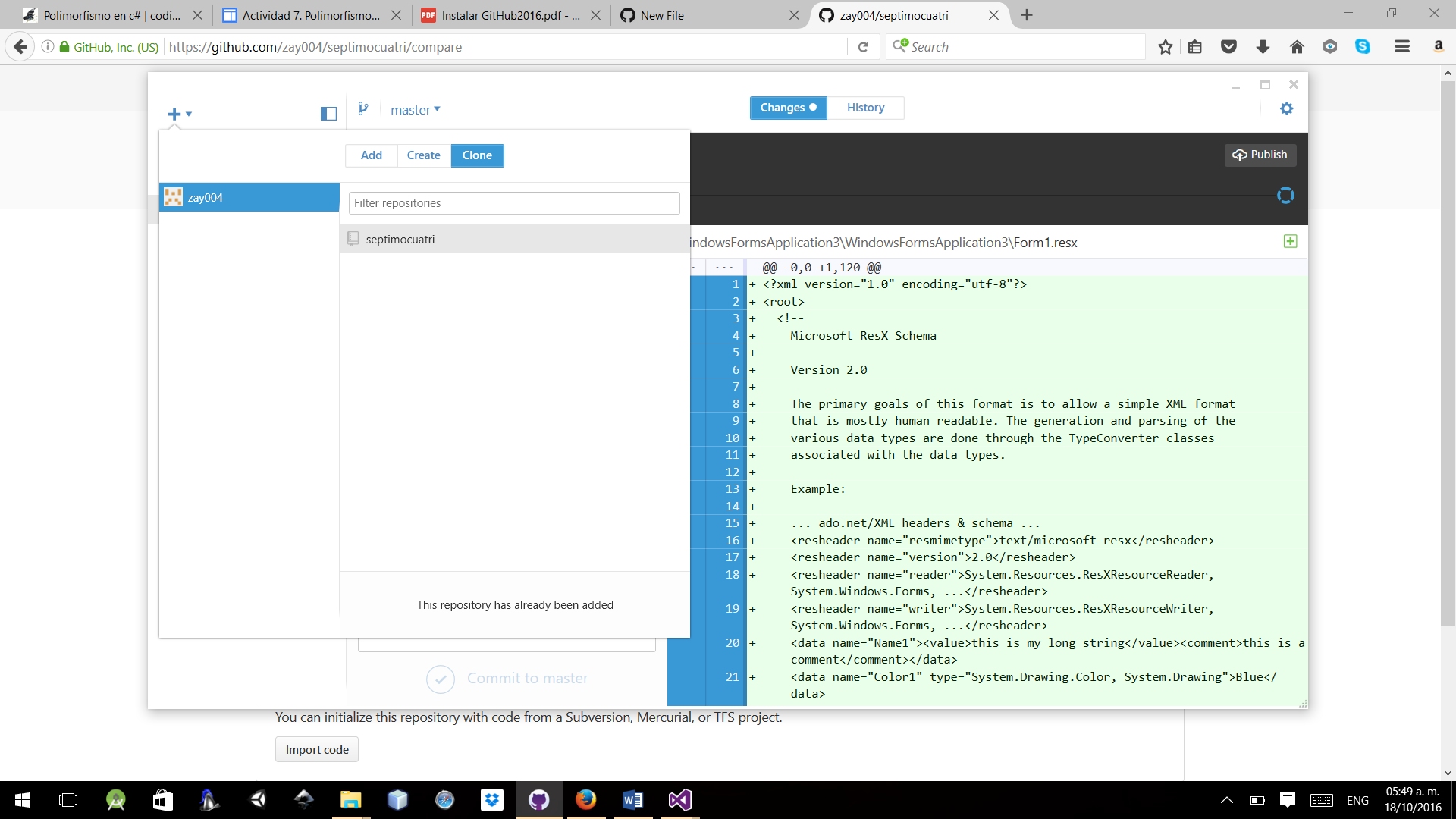
Lo primero es descargar GitHub e instalarlo



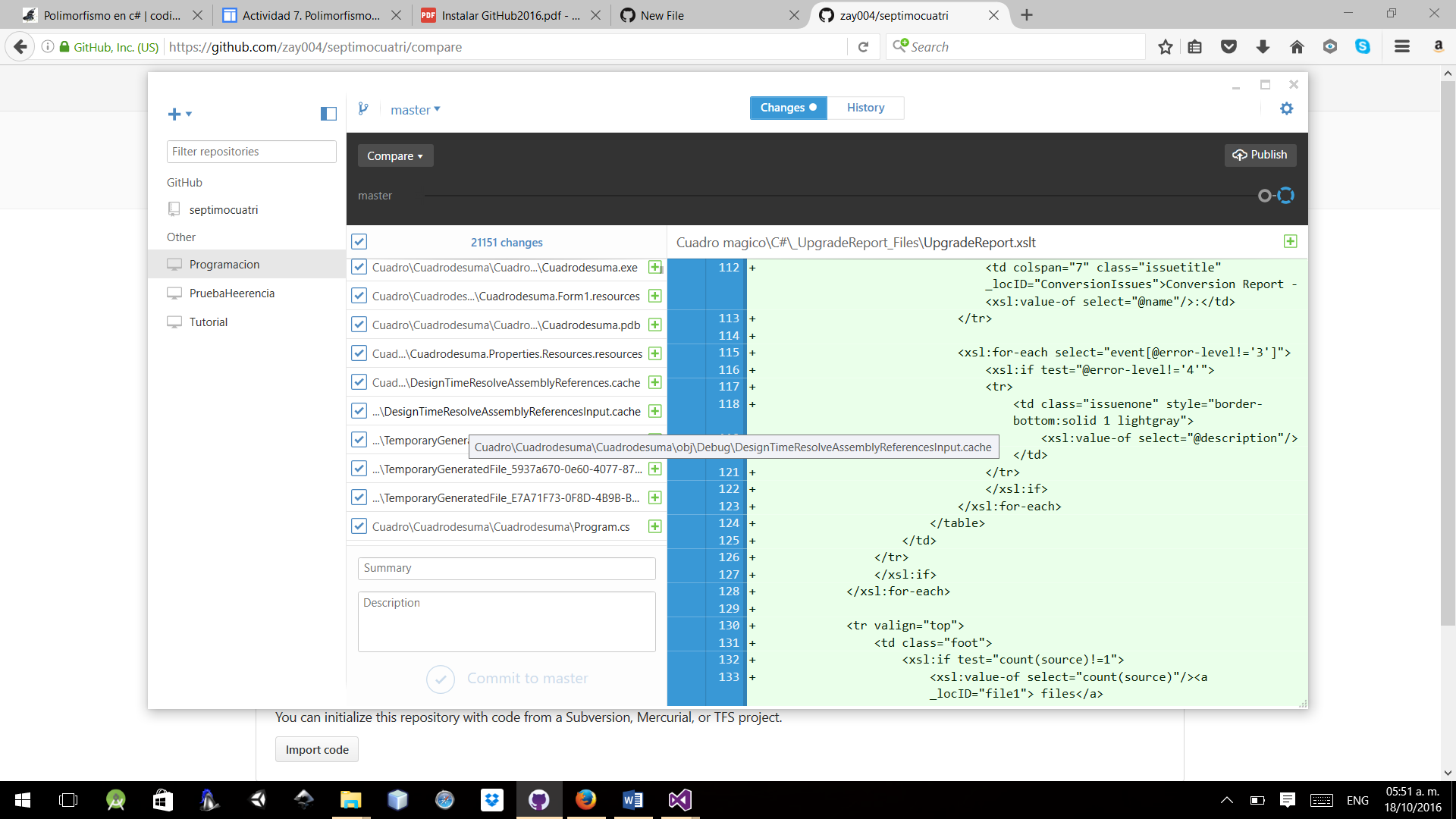
Iniciamos sesión con una cuenta previamente creada

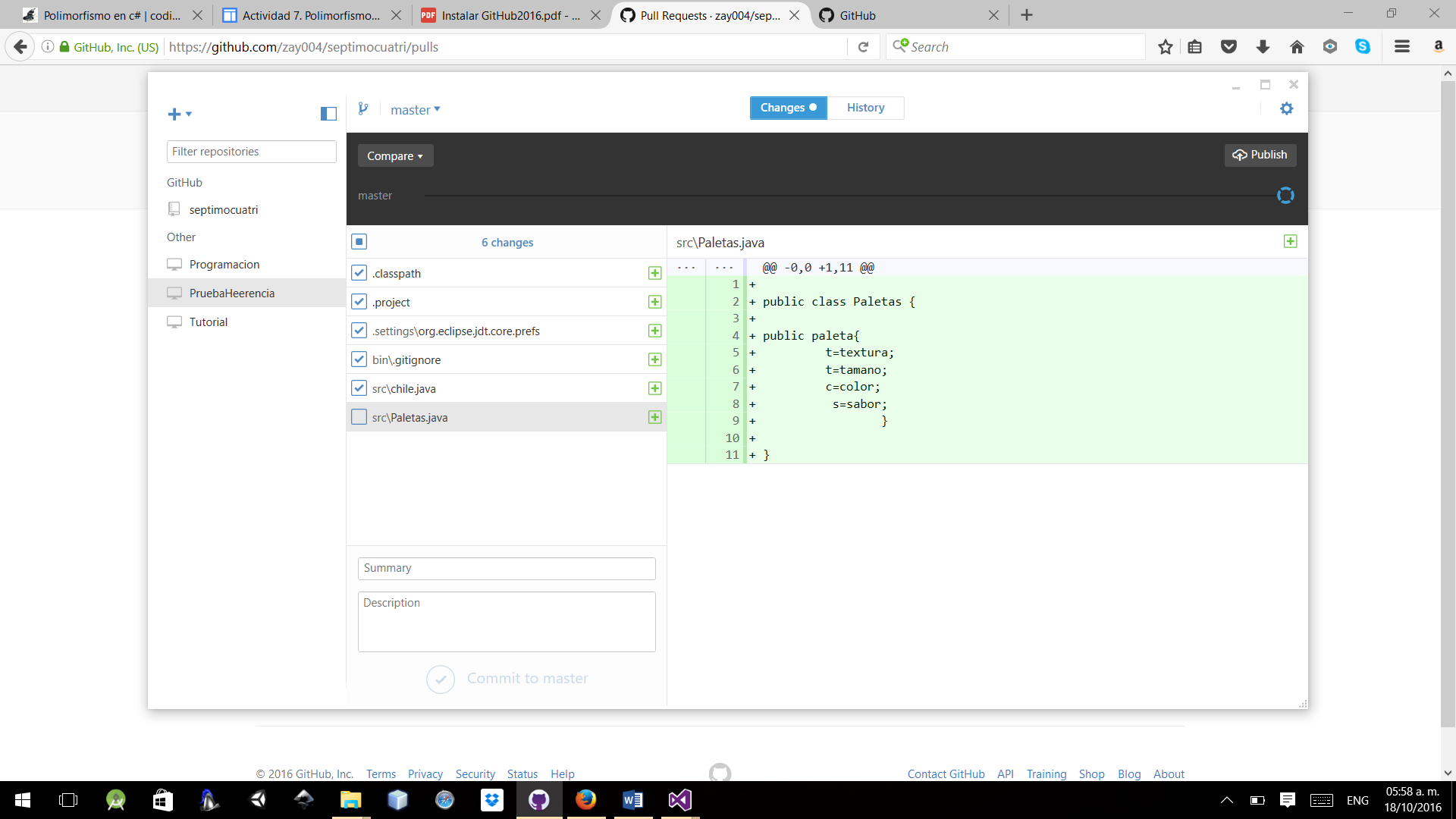
La bienvenida de GitHub

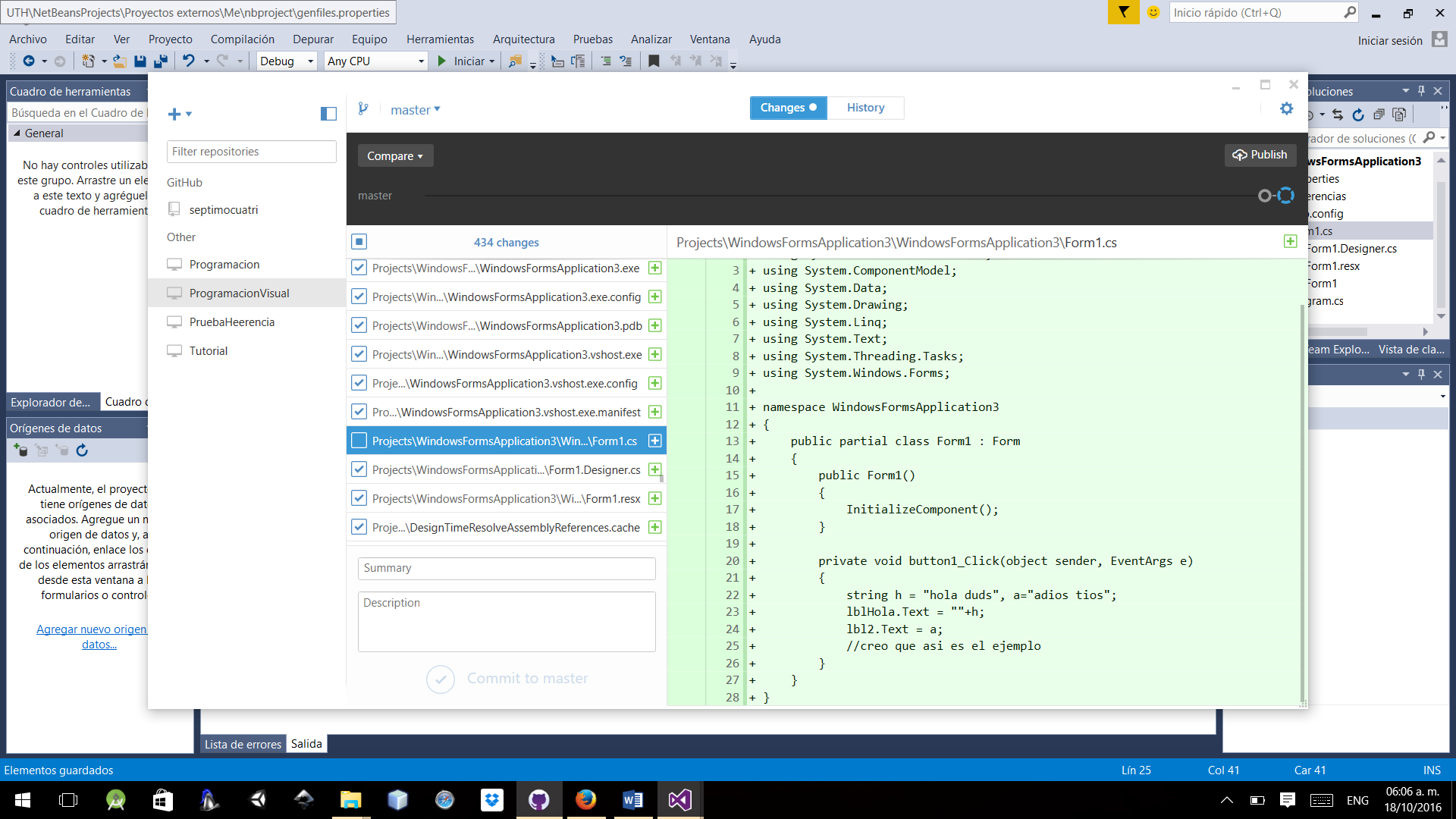
En esta parte sincronizamos nuestros proyectos con la aplicación GinHub



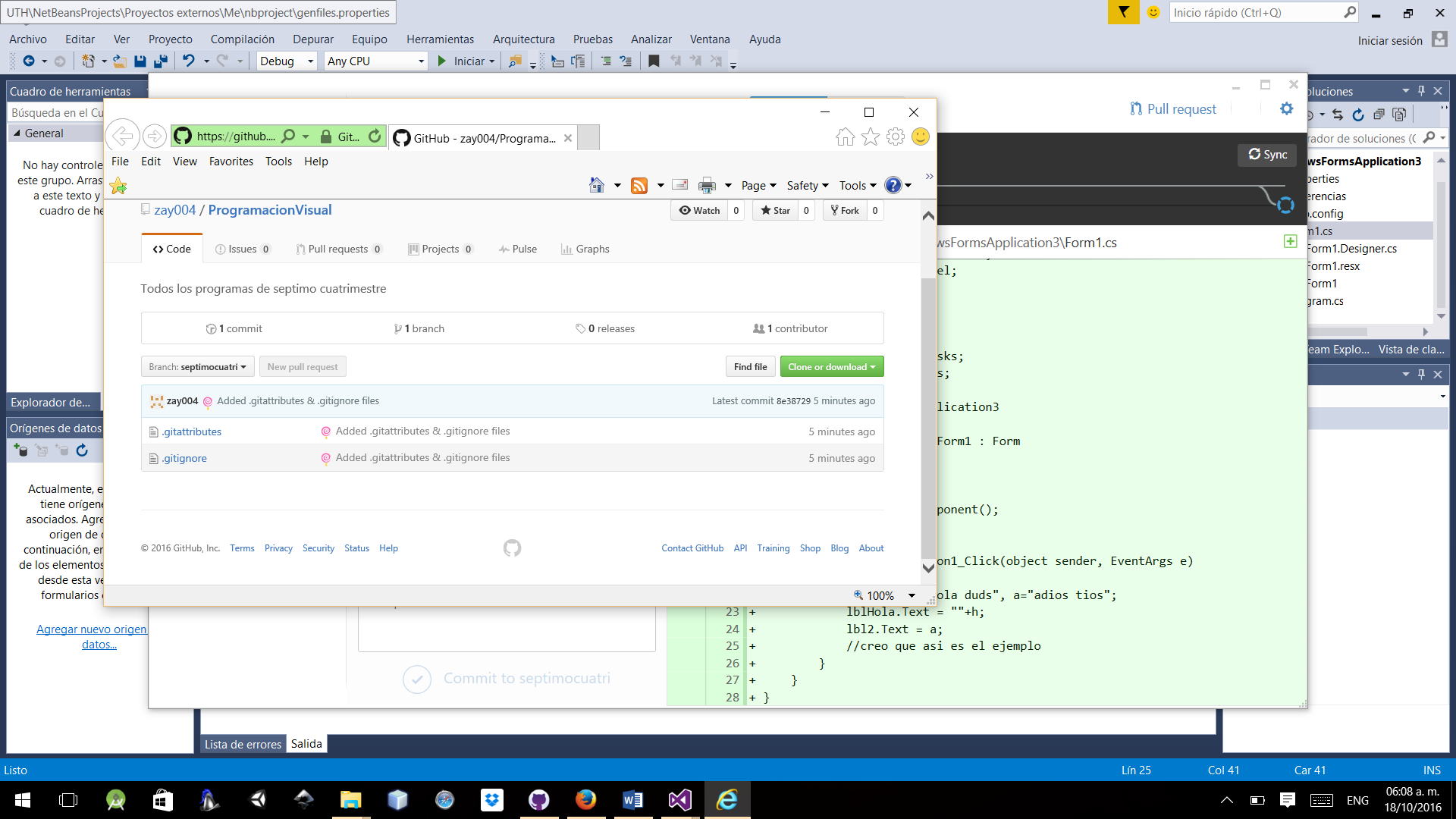
Visualizaremos nuestros archivos sincronizados o clonados



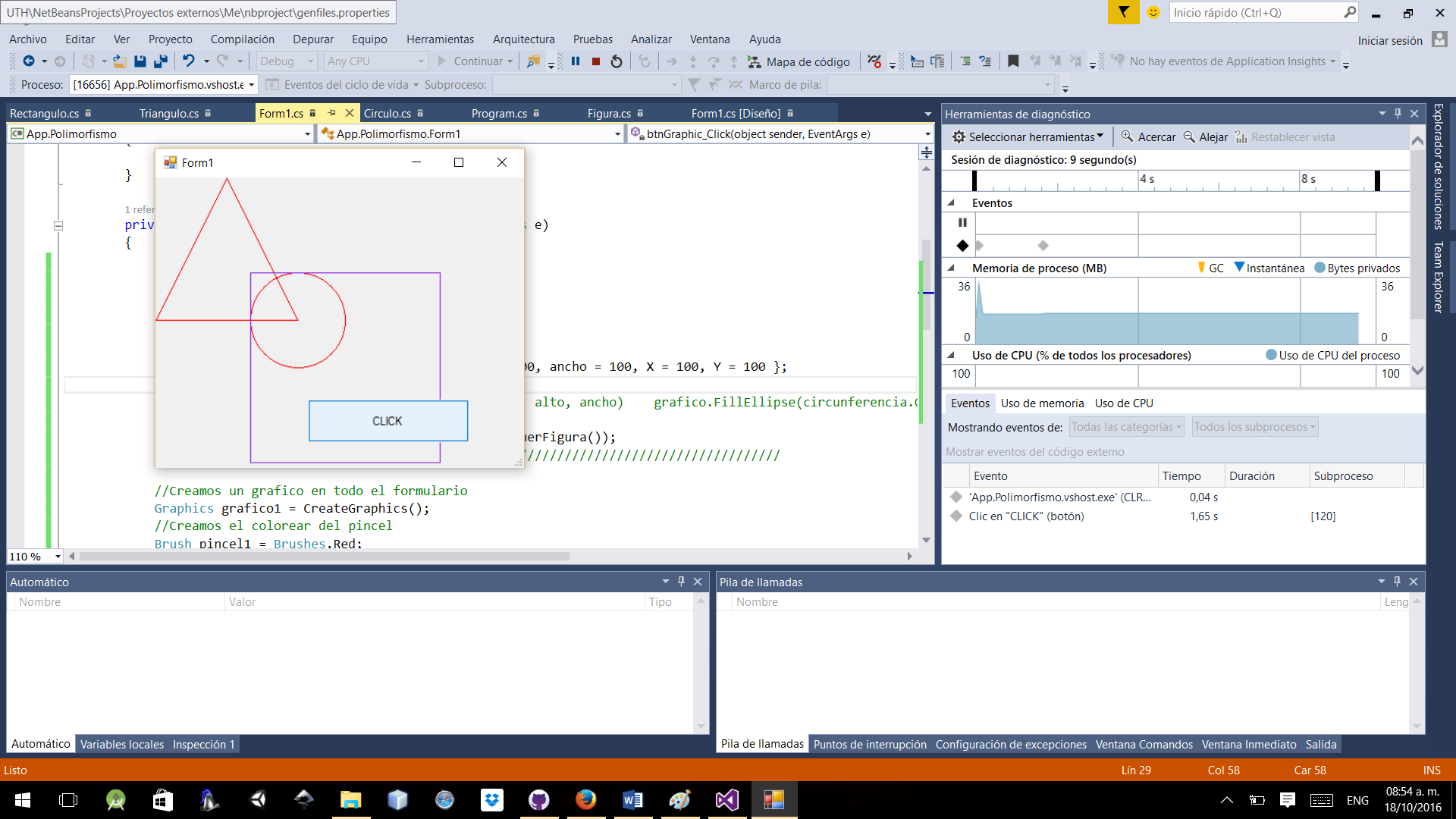
Podremos ver el código de una case especifica

En esta podemos ver como se modificó se le agrego un texto comentado se actualizara automáticamente gracias a la aplicación

En esta parte veremos en internet la notificación



En el siguiente ejercicio realizamos los ejercicios uno por uno, pero los imprimí todos juntos



# CONCLUSION

Al repasar lo que es polimorfismo pude comprender un poco más de programación y descubrí algunas cosas que no me acordaba gracias a esta práctica.

# REFERENCIAS

* <http://csharp.net-tutorials.com/classes/abstract-classes/>
* <https://www.tutorialspoint.com/csharp/csharp_generics.htm>
* Crear cuenta: https://github.com/join
* Visual Studio: https://visualstudio.github.com/