Universidad Autónoma de Ciudad Juárez



División Multidisciplinaria Ciudad Universitaria

Cuadro comparativo de lenguajes de Programación Orientada a Objetos

Programación II

Docente: Alan Ponce

Ulises Hernández Rivadeneyra 159965

Licenciatura en Ingeniería de Software

08 de marzo de 2019

Introducción

En el siguiente documento vamos a ver los diferentes puntos de la programación procedural y la orientada a objetos, ambos como ya sabemos son de los paradigmas más populares, puesto que muchos lenguajes potentes incluyen dichos paradigmas. Es importante conocer los paradigmas, puesto que en un proyecto tenemos que escoger el que se adapte a las necesidades de este, es importante mencionar que no existe un paradigma mejor que otro, solo existen las necesidades diferentes, por lo cual tratar de poner alguno en primer lugar es relativo.

**Lenguaje procedural**

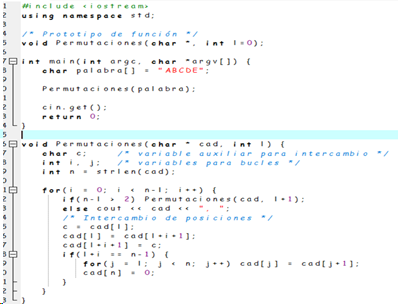
Empecemos definiendo el paradigma procedural. Este paradigma es aplicable tanto en lenguajes de programación de bajo nivel como en lenguajes de alto nivel, ¿Por qué?, una de las razones es la respuesta con la relación de los tamaños en los programas, también por la usabilidad que este permite a la hora de programar. El lenguaje de bajo nivel, no quiere decir menos potencia o peor que, solo significa que la programación está más estrechamente ligada con las características técnicas, es decir un nivel reducido de abstracción.

En el caso de que esta técnica se aplique en lenguajes de alto nivel, recibirá el nombre de programación funcional. Esta técnica consiste en basarse de un número muy bajo de expresiones repetidas, englobarlas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.

Aterrizando la idea, con este paradigma podemos ayudarnos a la hora de programar, por que el programa se compone en módulos, estos pueden ser procedimientos, funciones, subrutinas o métodos, dependiendo del lenguaje de programación.

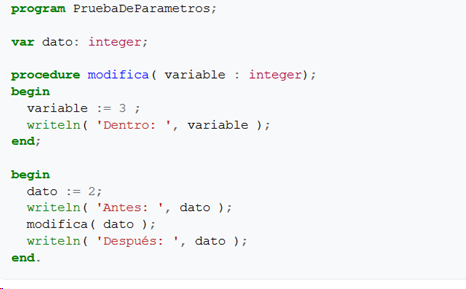
**Lenguajes de programación con paradigma procedural.**

**C++:** Es un lenguaje que se diseñó en 1979, es una extensión de C el cual permite la manipulación de objetos, es decir que también acepta la programación orientada a objetos, es multiparadigma.



Como podemos observar en este código, se agrupa en una función un pequeño algoritmo para hacer permutaciones a dicho elemento, es decir las diferentes combinaciones del elemento, en el main lo único que se utiliza es el llamado de la función con el parámetro de la palabra guardada, y en la función podemos ver la recursividad.

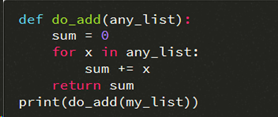
**Pascal:** Pascal es un lenguaje creado entre los años 1968 y 1969 y publicado en 1970. Su objetivo era crear un lenguaje que facilitara el aprendizaje de programación utilizando la programación estructurada. Este lenguaje se convirtió en una herramienta para la creación de aplicaciones de todo tipo. Pascal se caracteriza por ser un lenguaje de programación estructuradofuertemente tipado.



Este programa es interesante puesto que toca otro pequeño tema, el programa empieza definiendo el nombre de este, se declara la variable dato de tipo entero, se tiene el procedimiento modifica, el cual recibe una variable de tipo entero, le asigna el valor de 3 y lo escribe. Lógicamente siempre escribirá 3.

Después empieza el cuerpo del programa, en el cual a dato se le asigna el 2, se escribe el valor de dato el cual es 2, se llama al procedimiento modifica con dato, y se espera que se vea un 3, pero no es así puesto que las modificaciones que se le hacen al dato dentro de modifica solo son validas en el procedimiento, es decir pasa una variable por valor. Lo que se ve es un 2, se ve el paradigma procedural en la declaración del procedimiento para hacer alguna tarea.

**Python:** Python fue creado a finales de los ochenta, es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.



En este pequeño código podemos ver el uso de una función el cual recibe un arreglo, se define sum=0 y a este le va sumando todos los índices del arreglo, da este valor y al último utiliza la función y la imprime, podemos ver que Python es un lenguaje de muy alto nivel, y que el paradigma procedural se ve marcado en cómo se separan los procedimientos, incluso aunque l ejecución es sistemática, el código se entiende por qué se divide en partes.

**Paradigma orientado a objetos.**

¿Qué es la poo? La POO es un paradigma surgido en los años 1970, que utiliza objetos como elementos fundamentales en la construcción de la solución. Un objeto es una abstracción de algún hecho o ente del mundo real, con atributos que representan sus características o propiedades, y métodos que emulan su comportamiento o actividad.

Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial. Muchos de los objetos prediseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas. Está basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

**Lenguajes de programación con paradigma orientado a objetos**

**Scala:** Scala es un lenguaje de programación multi-paradigma diseñado para expresar patrones comunes de programación en forma concisa, elegante y con tipos seguros.

Integra sutilmente características de lenguajes funcionales y orientados a objetos. La implementación actual corre en la máquina virtual de Java y es compatible con las aplicaciones Java existentes. Su creación empezó en 2001 pero fue lanzado hasta 2004.

El nombre de scala se deriva de la palabra escalable, lo que significa que puede crecer con la demanda de los usuarios. A diferencia de cómo algunos piensan, scala no es una extensión de java, pero es completamente interoperable con ella.

Las versiones de java van desde la 2.0 hasta la 2.13.

Algunos hechos de la popularización de scala son:

Twitter utiliza scala en grandes porciones de su backend.

Apple Inc. usa Scala en ciertos equipos, junto con Java y el marco Play.

El New York Times reveló en 2014 que su sistema de gestión de contenido interno

Blackbeard está construido utilizando Scala, Akka y Play Framework.

Hay equipos dentro de Google que usan Scala, principalmente debido a adquisiciones como Firebase y Nest.

**Objective-C:** Es un lenguaje de programación creado como un superconjunto de C, pensando en los modelos de smalltalk. Se creó en 1980 por la corporación StepStone. Una de las inspiraciones para la creación de este subconjunto fue los problemas que las programación estructurada tenía años antes de los 80s,y al ver que smalltalk solucionaba algunos de esos problemas se empezó a adoptar la idea de que la programación orientada a objetos sería el futuro.

El objetivo que se marcó fue el tener un conjunto de bibliotecas utilizables, y que el código pudiera ser empaquetado en multiplataforma, lo que se convirtió en un compilador de bibliotecas potentes. El problema que surgió tiempo después fueron los problemas de reutilización aunque estos no dependen del lenguaje.

En el 98, next, la cual fue una empresa de informática creada por Steve Jobs licenció a objective C, lo que logró extender el soporte a objective C. Se dio a conocer por las herramientas que producía a tal grado, de dejar por completo el hardware y dedicarse directamente al software.

Lo que desató la popularización del desarrollo de apple y swift, puesto que se utilizaron herramientas de objective c en la interfaz, e incluso hasta la fecha la mayoría del Cocoa API de apple se basa en los objetos de interfaz de One step, y este es un entorno desarrollado de Objective C

**Ruby:** Ruby es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) [interpretado](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado), [reflexivo](https://es.wikipedia.org/wiki/Reflexi%C3%B3n_(inform%C3%A1tica)) y [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), creado por el programador [japonés](https://es.wikipedia.org/wiki/Jap%C3%B3n) [Yukihiro "Matz" Matsumoto](https://es.wikipedia.org/wiki/Yukihiro_Matsumoto), quien comenzó a trabajar en Ruby en [1993](https://es.wikipedia.org/wiki/1993), y lo presentó públicamente en [1995](https://es.wikipedia.org/wiki/1995). Combina una sintaxis inspirada en [Python](https://es.wikipedia.org/wiki/Python) y [Perl](https://es.wikipedia.org/wiki/Perl) con características de programación orientada a objetos similares a [Smalltalk](https://es.wikipedia.org/wiki/Smalltalk).

Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Entre sus características principales están los cuatro niveles de ámbito de variable, el manejo de excepciones, altamente portable,Amplia librería estándar,inyección de dependencias entre otras.

La popularidad de ruby creció por la creación de su framework RoR, aunque curiosamente este framework tiene la misma aceptación que el mismo odio. Muchos opinan que es muy débil en donde otras opciones como java o php son fuertes, y que por esta razón lo últimos son mejores. Es por eso que muchos opinan que ruby desaparecerá poco a poco, puesto que existen lenguajes que hacen lo mismo, de una mejor manera.

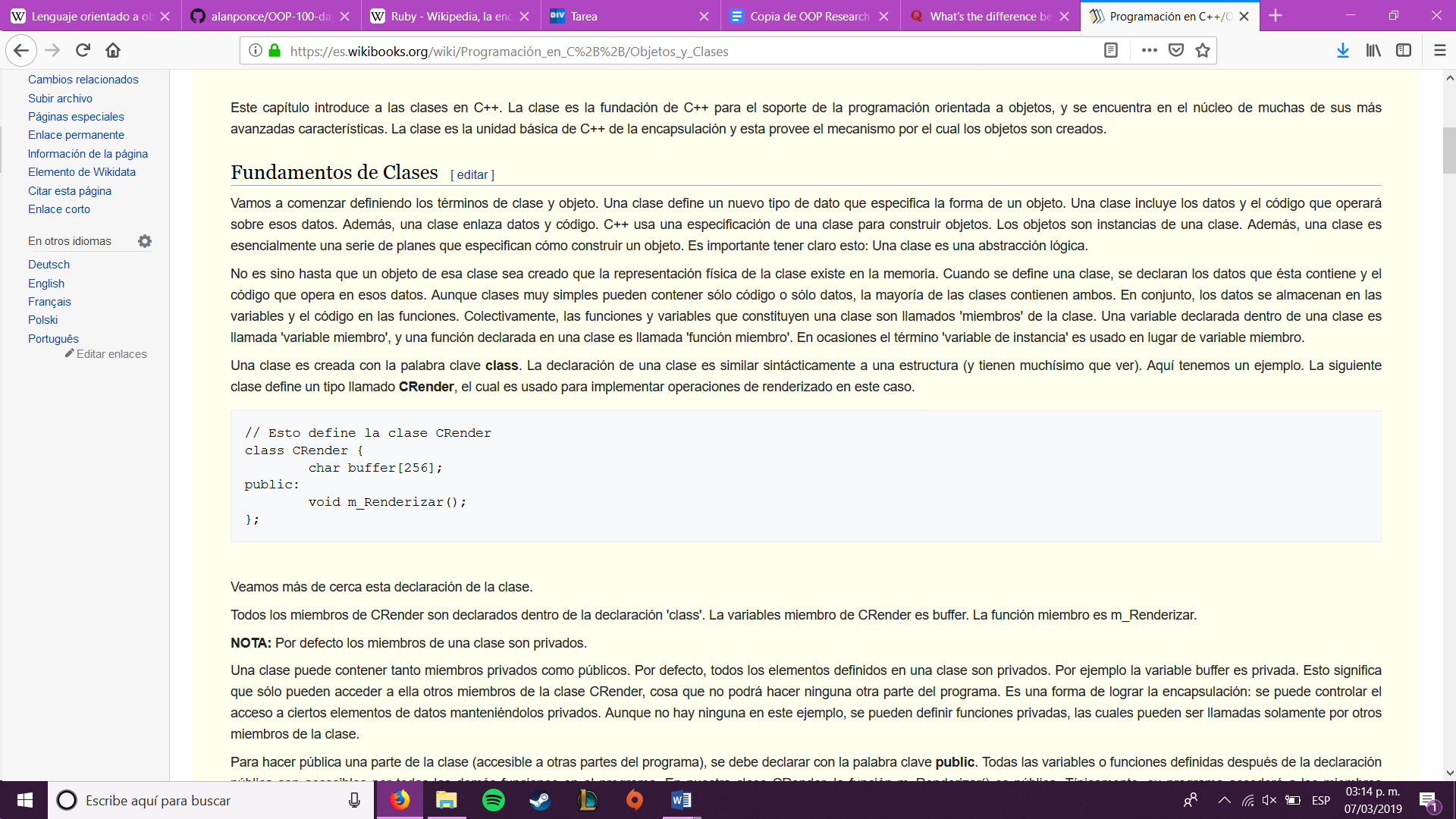
|  |  |
| --- | --- |
| Paradigma Procedural | Paradigma Orientada a Objetos |
| En proyectos chico es mas rapido. | En proyectos chicos es más lento. |
| El consumo de memoria es bajo. | El consumo de memoria es alto. |
| Su transparencia es baja. | Su transparencia es alta. |
| Es menos transportable. | Es más transportable. |
| Se desarrolla más rápido, puesto que no se piensa en muchas estructuras como las clases. | Se desarrolla lento, puesto que debes estructurar todo lo del proyecto. |
| Usa registros | Usa objetos |
| Utiliza llamada de procedimiento | Utiliza llamada de mensaje |

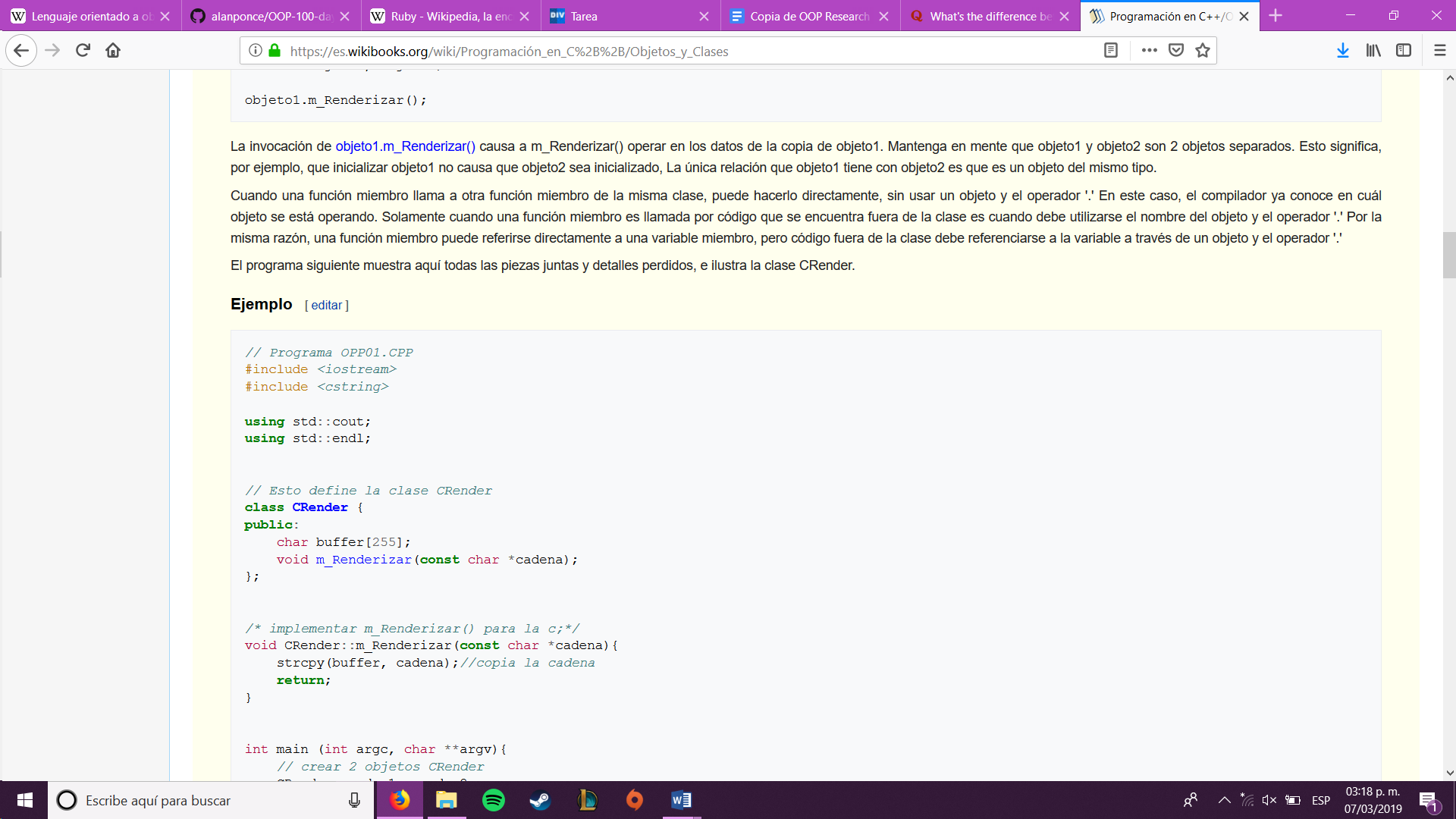
Lenguajes de Programación Orientada a Objetos

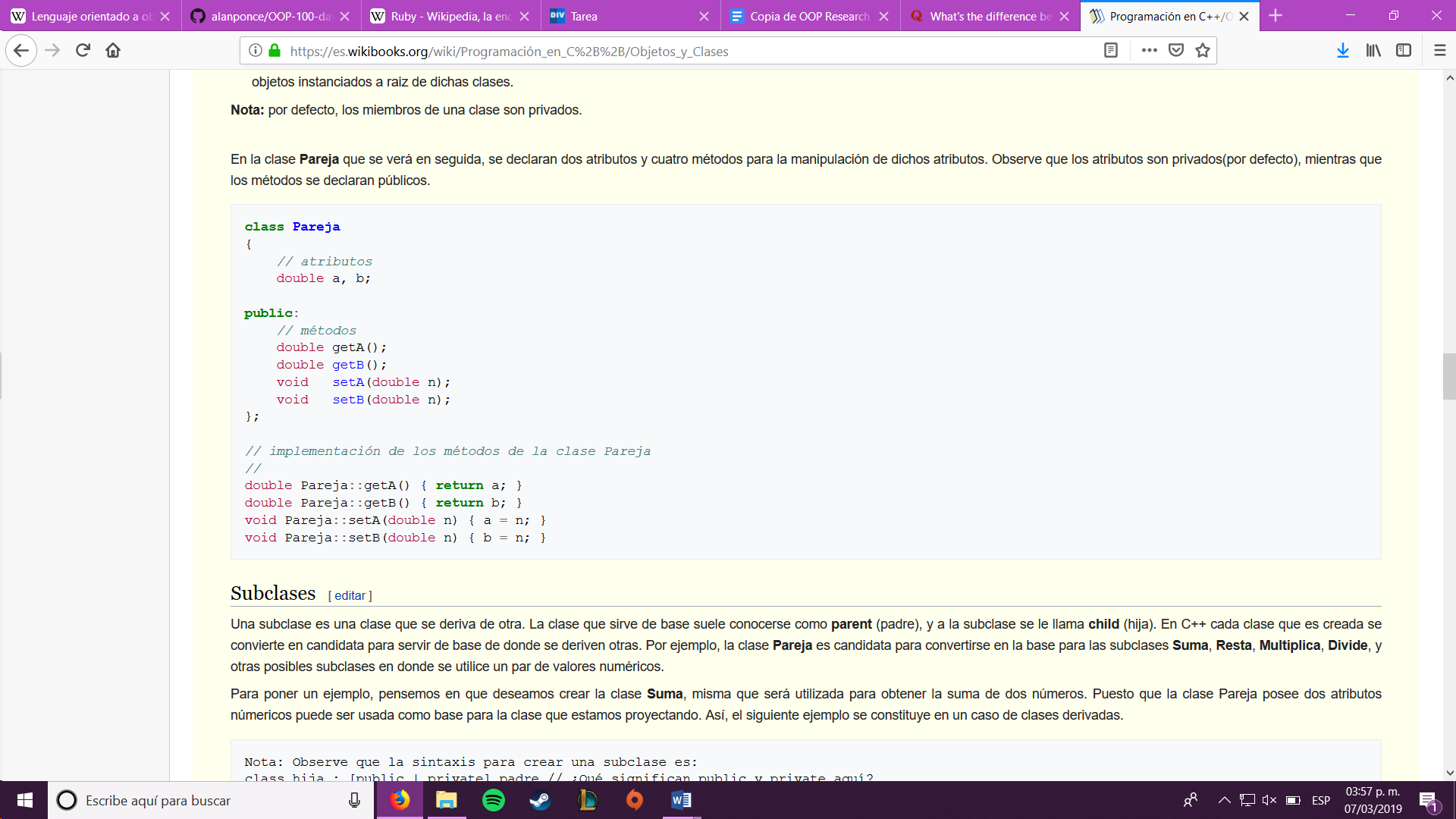
A continuación , vamos a ver algunos ejemplos de código de los elementos principales en la programación orientada a objetos (Herencia, Polimorfismo, Encapsulamiento, Clases) en diferentes lenguajes de programación.

C++

**Una clase** C++ usa una especificación de una clase para construir objetos. En c++una clase se crea con la palabra clave class.

Por defecto los miembros de la clase son privados, si se requiere lo contrario se define con la palabra public como en el siguiente código.





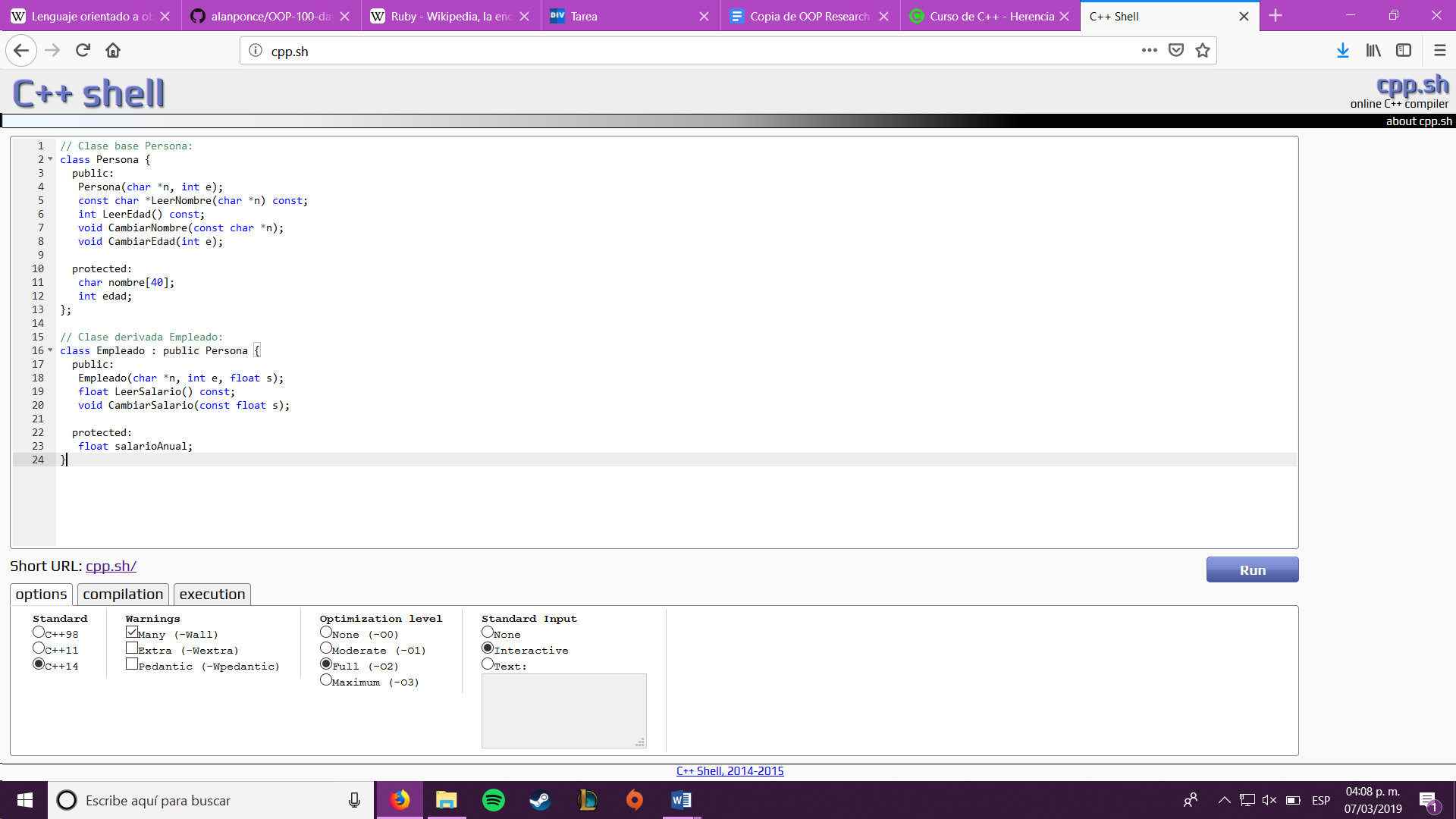
Aquí mismo podemos ver como son los métodos y atributos en c++;

**Herencia** En c++ La forma general de declarar clases derivadas es la siguiente:



**public**: los miembros heredados de la clase base conservan el tipo de acceso con que fueron declarados en ella.

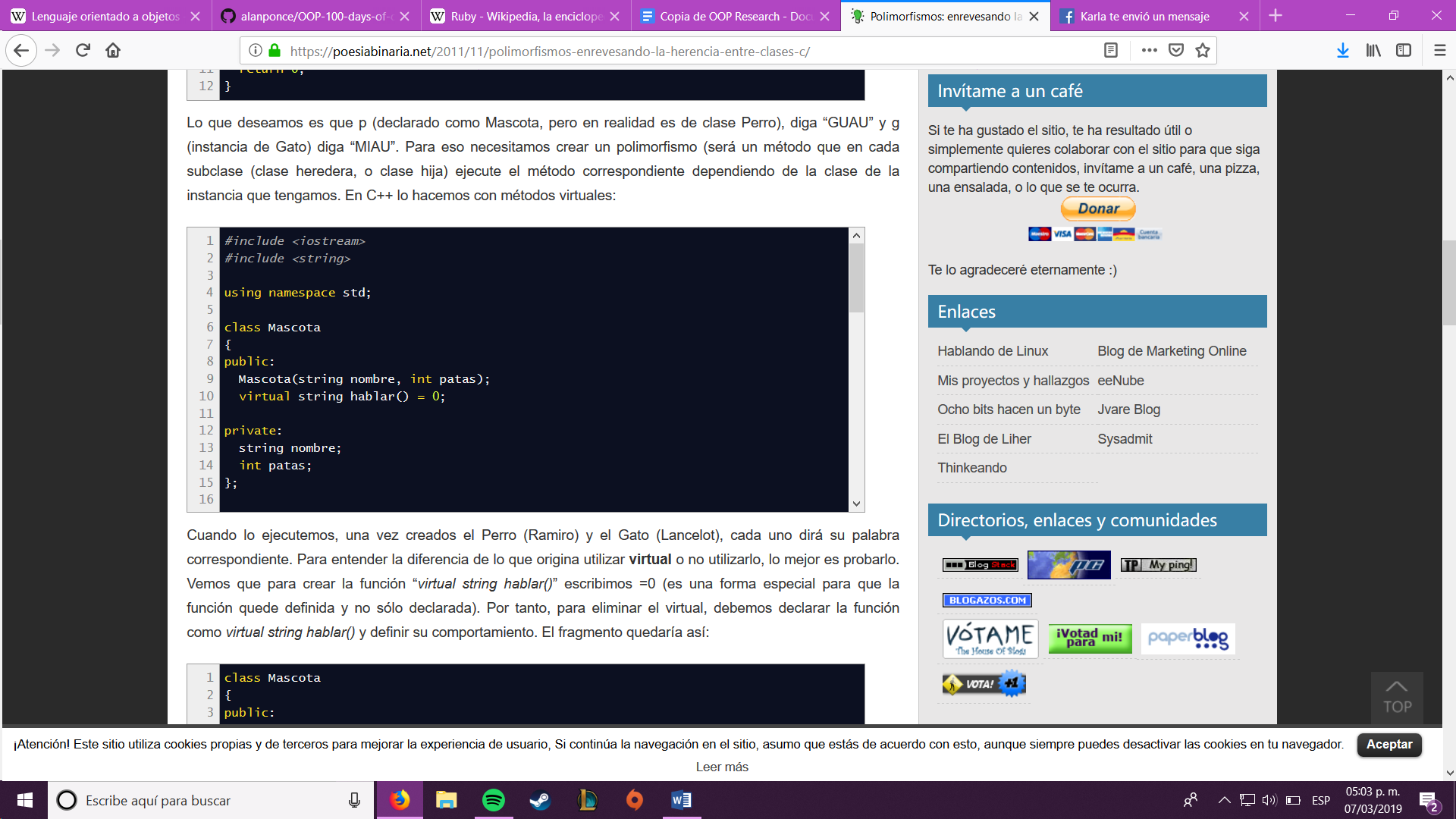
**private**: todos los miembros heredados de la clase base pasan a ser miembros privados en la clase derivada.



La declaración de los datos miembros como **protected** es para que las clases derivadas tengan acceso a ellos, aunque en general es recomendable declarar siempre los datos de nuestras clases como privados, de ese modo no son accesibles desde el exterior de la clase y además, las posibles modificaciones de esos datos, en cuanto a tipo o tamaño, sólo requieren ajustes de los métodos de la propia clase.

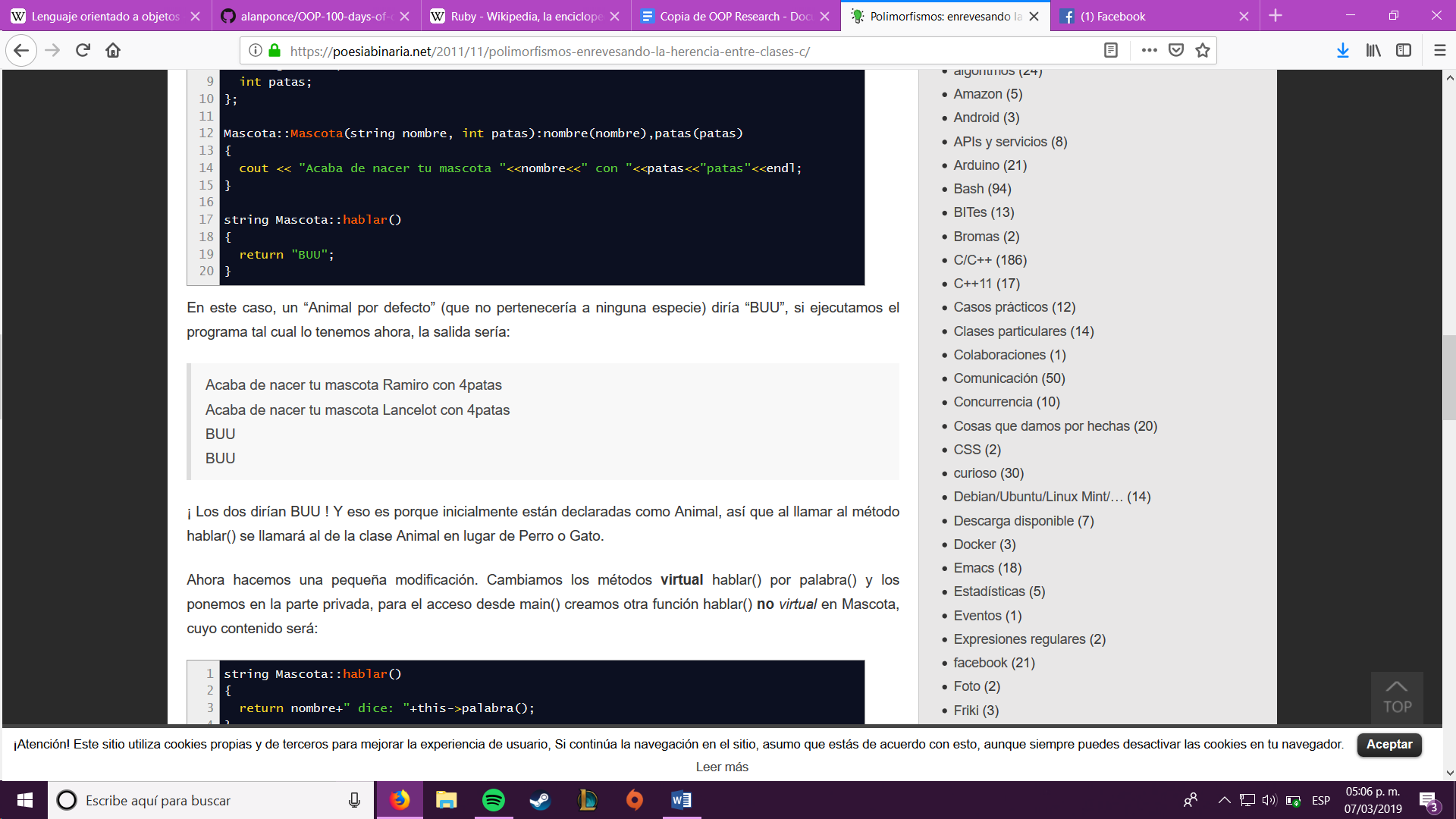
Vemos que la clase empleado deriva de la clase persona, la clase empleado puede acceder a los métodos de la clase persona.

**Polimorfismo** C++ permite el polimorfismo, que es la habilidad de los objetos de diferentes clases que están relacionados mediante la herencia para responder en forma diferente al mismo mensaje (es decir, a la llamada de función miembro). El mismo mensaje que se envía a muchos tipos de objetos diferentes toma "muchas formas", y de ahí viene el término polimorfismo.

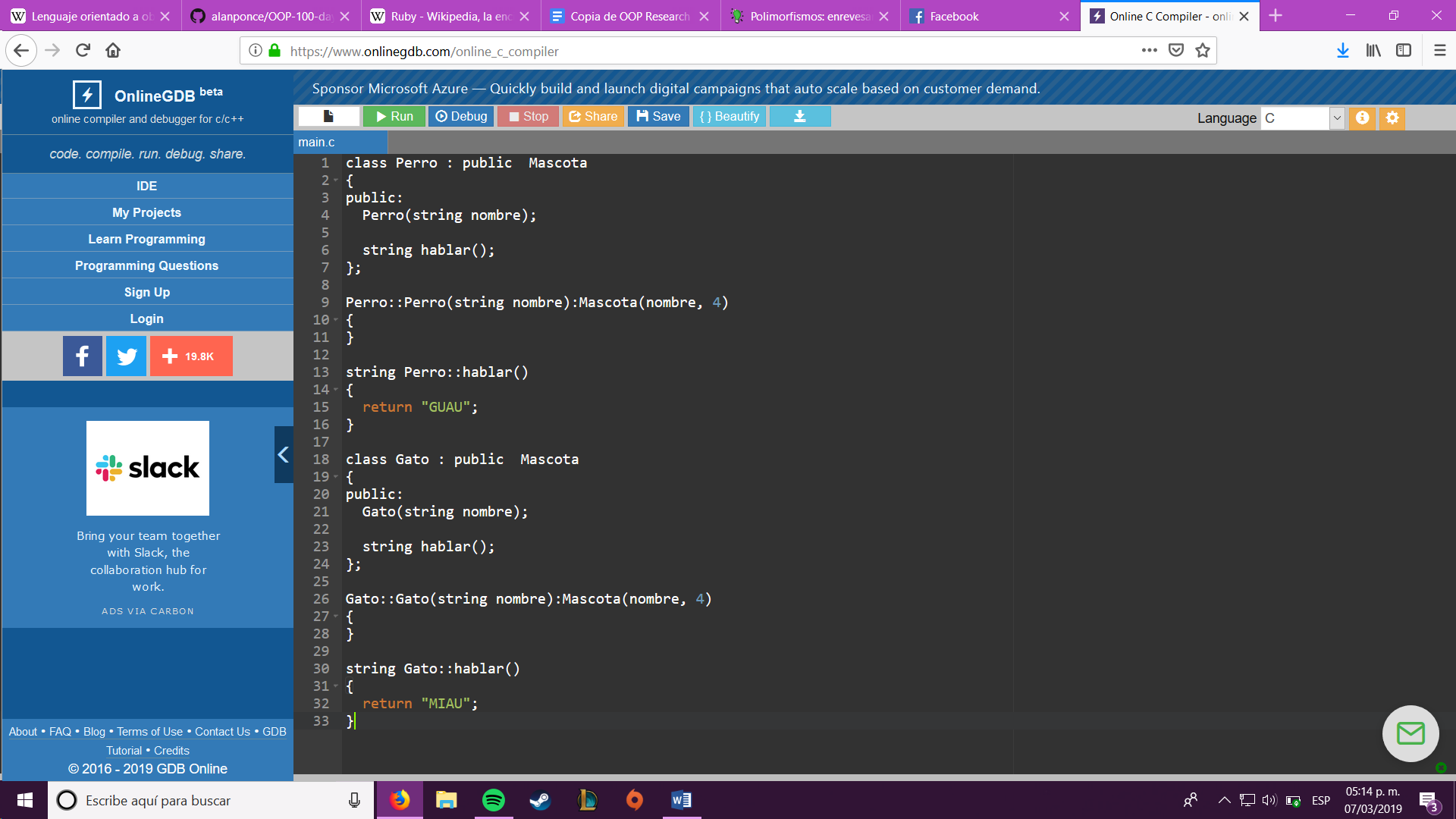


En este código tenemos la clase mascota, con el método hablar, vemos que en privado tenemos el atributo nombre, para tener diferentes tipos de mascotas.

Encapsulamiento

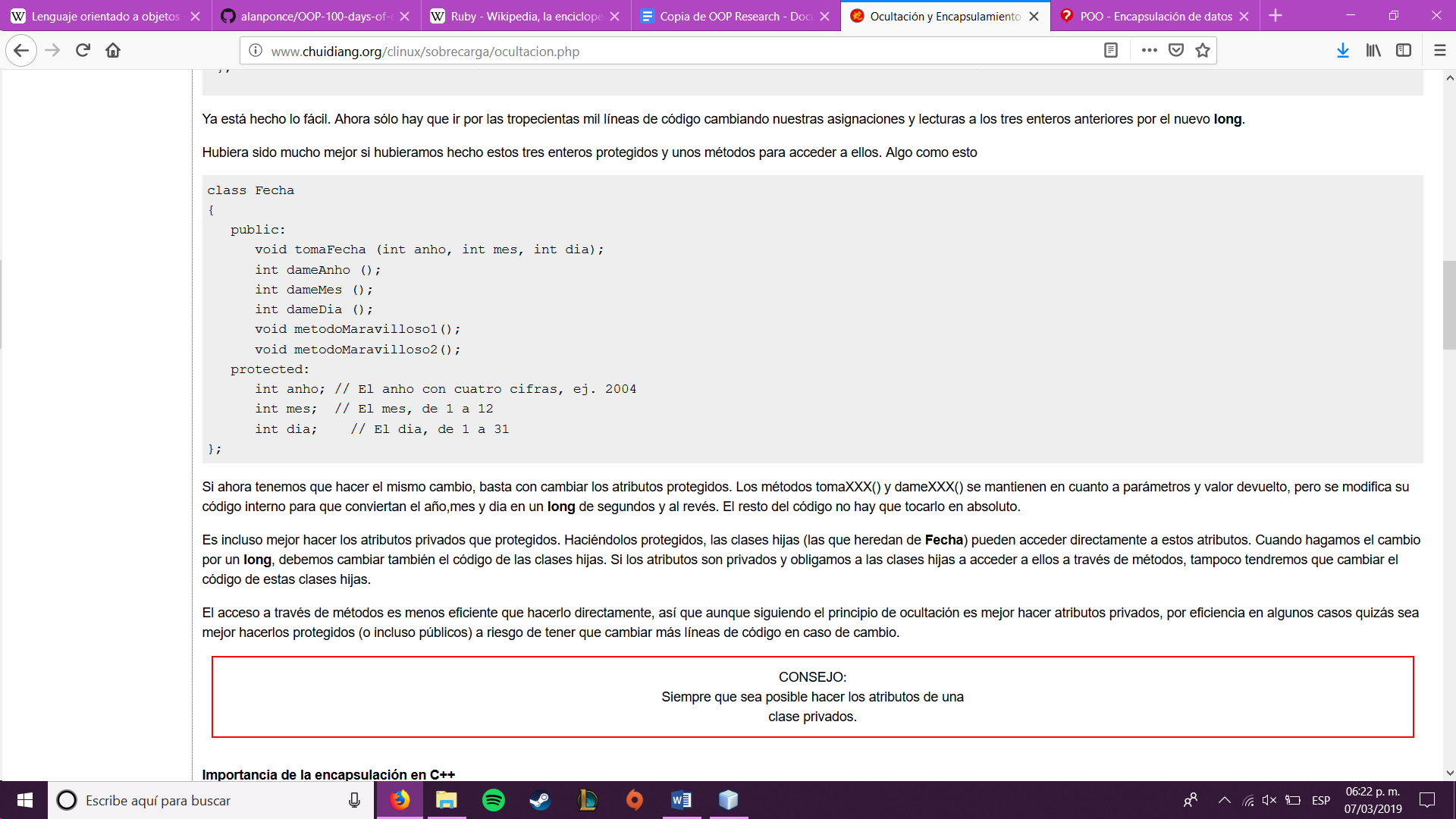


Aquí vemos que se declara mascota, para poner el nombre, las patas, y dependiendo del siguiente código se supone que esperamos la impresión de los mismos.

Con este código podemos esperar que cada vez que se cree un objeto del tipo gato, o perro estos impriman el respectivo hablar de cada uno. Aunque el método sea el mismo, dentro de cada clase tenemos métodos con diferente resultado, por eso es que si se ejecuta el programa, cada animal “hablara” diferente.

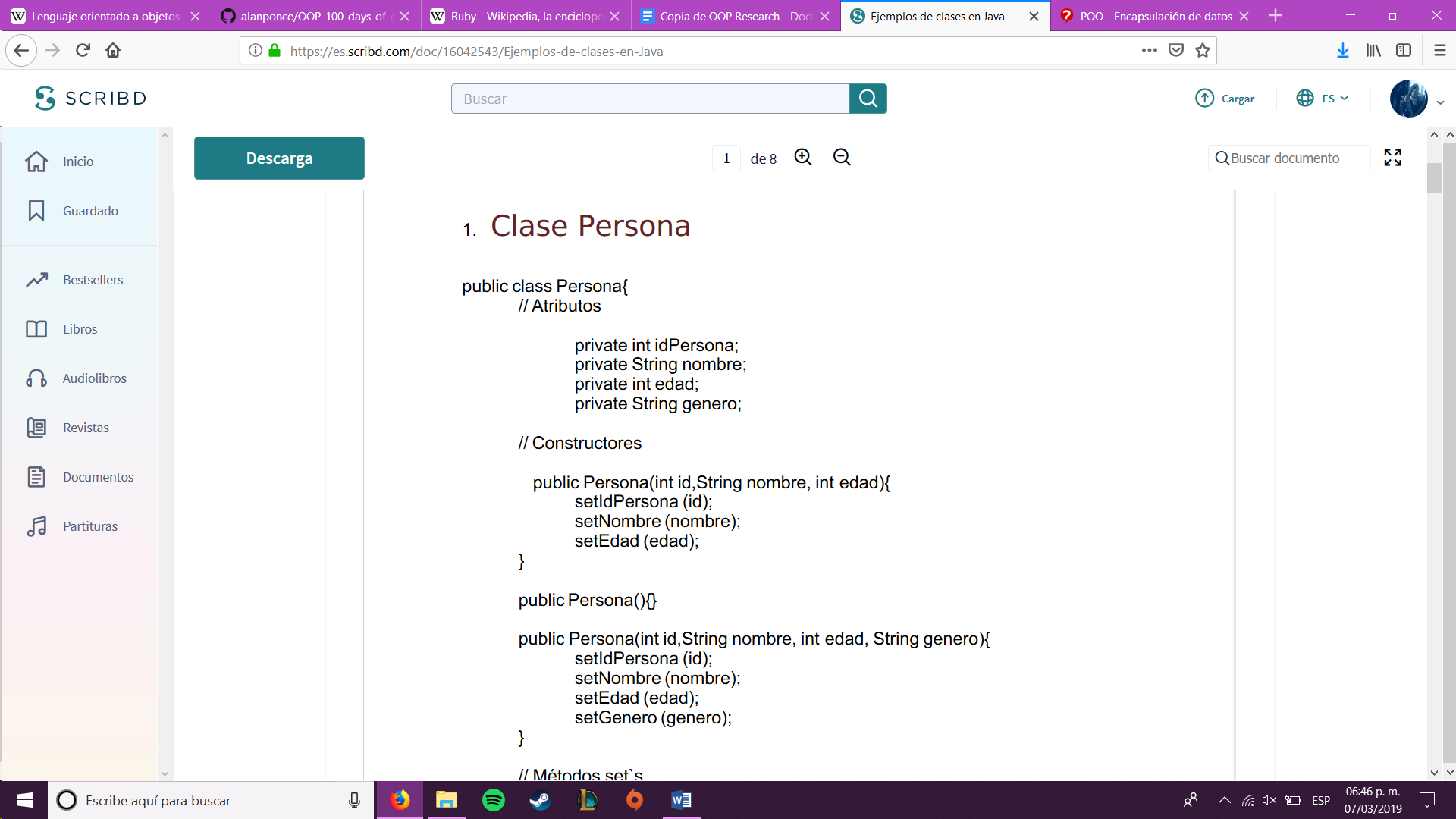
**Encapsulamiento**

En el siguiente código podemos ver el trabajo del encapsulamiento, es un pequeño programa pensado para tomar la fecha del momento de la captura, como podemos ver unos datos están en protected, para que solo te tenga acceso a ellos mediante un método, esto con el fin de proteger la información.

****

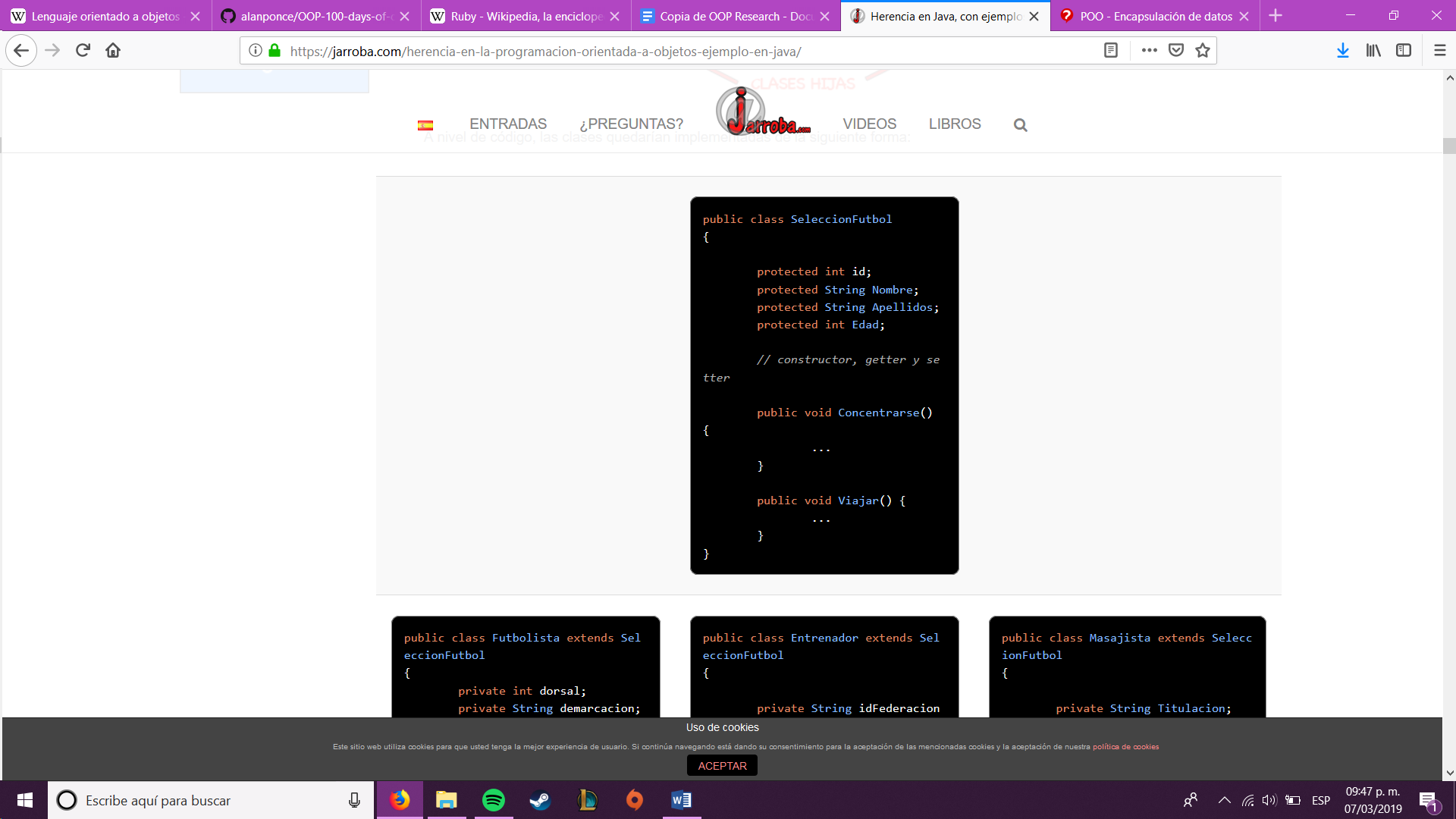
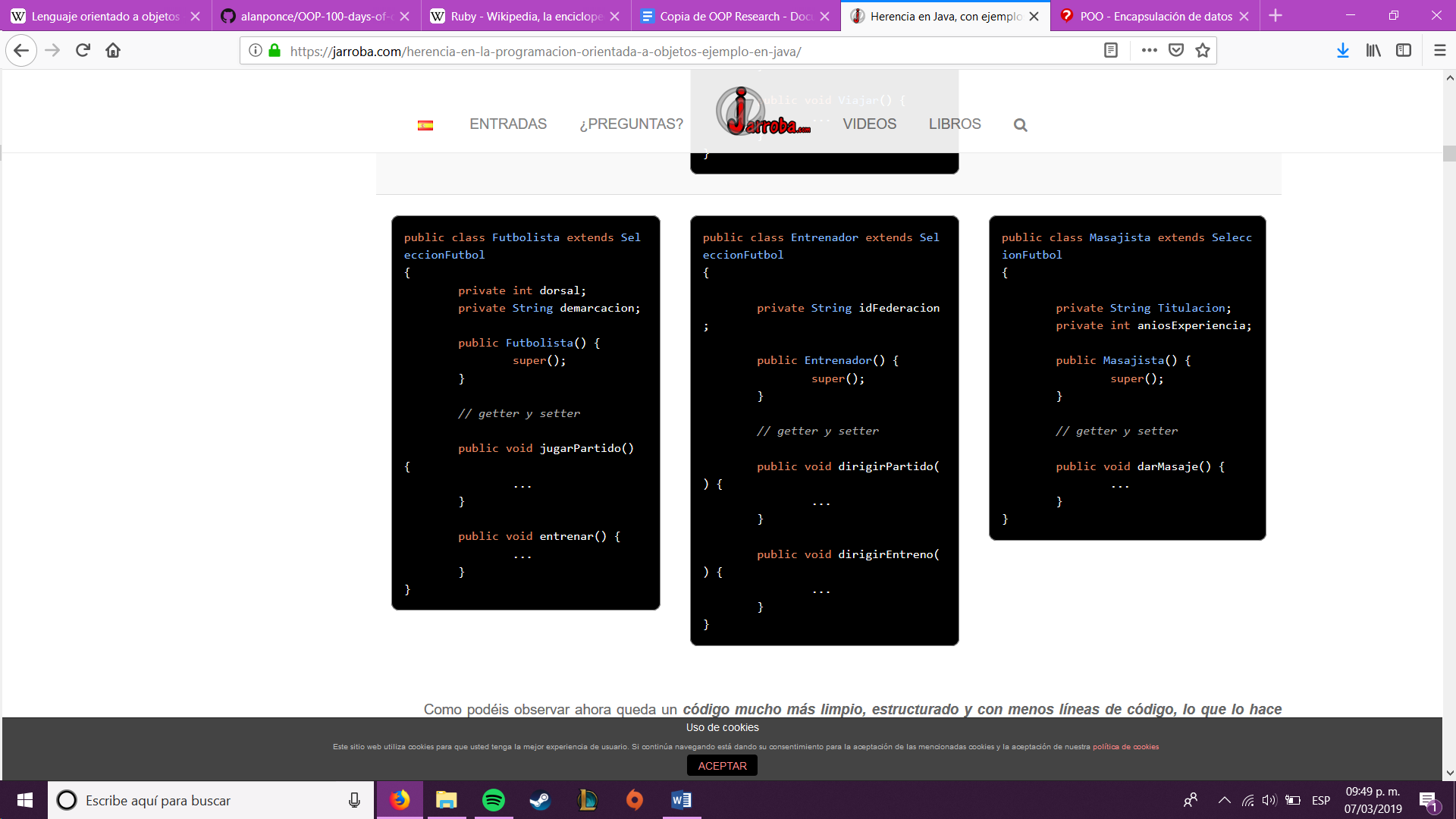
**Java**

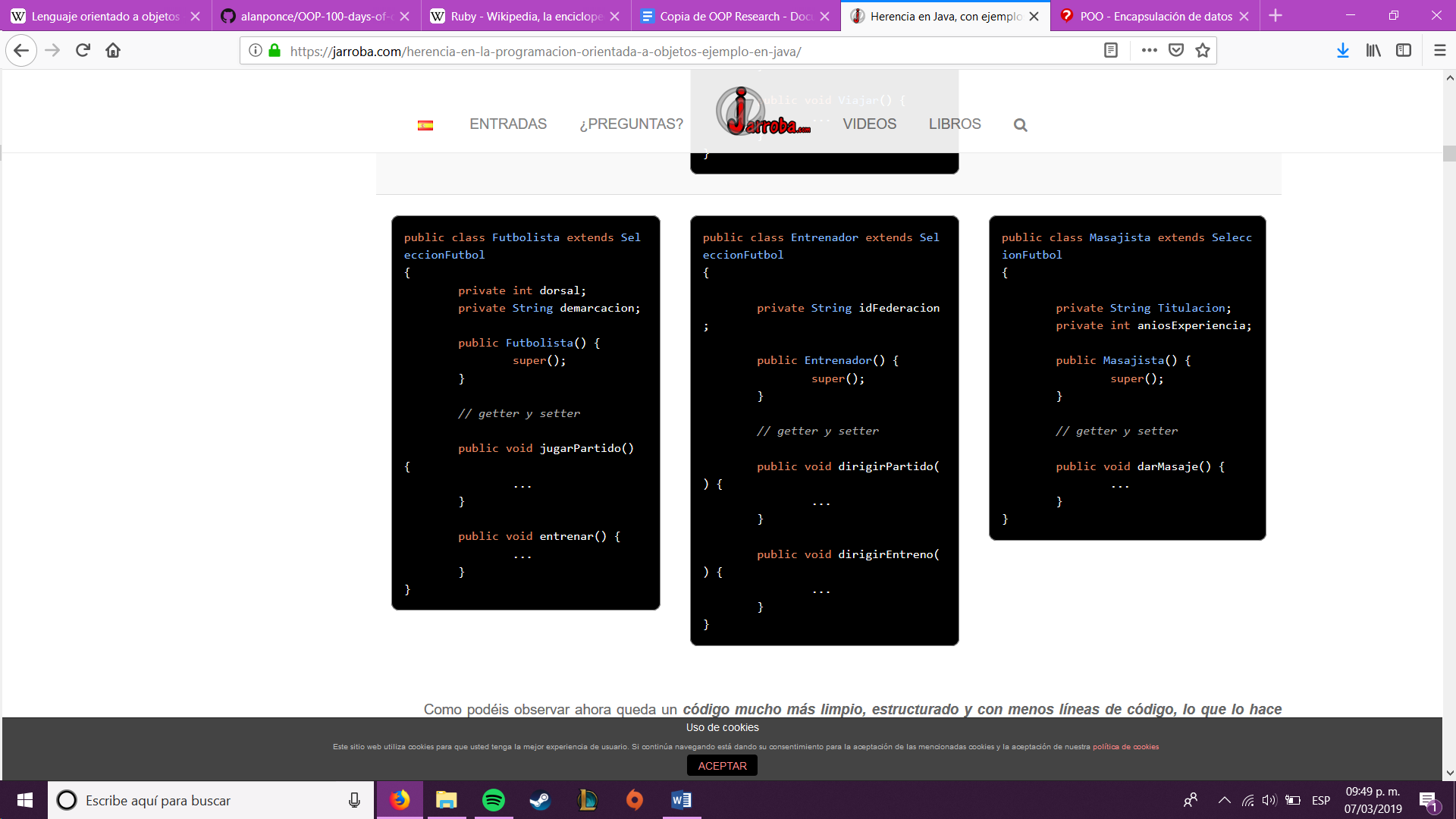
**Una clase**

Podemos ver que java es prácticamente igual que c++, lo que puede variar en este código son funciones, pero estas no varían por las clases si no por el mismo lenguaje que tiene diferentes funciones, aunque la mayoría de las palabras reservadas son iguales.

**Herencia**

En cuanto a la herencia, en java si tenemos unos cambios, tal vez no tan notorios pero si cambian en cuanto a c++. Vamos a ver los codigos y despues los explicamos paso a paso.

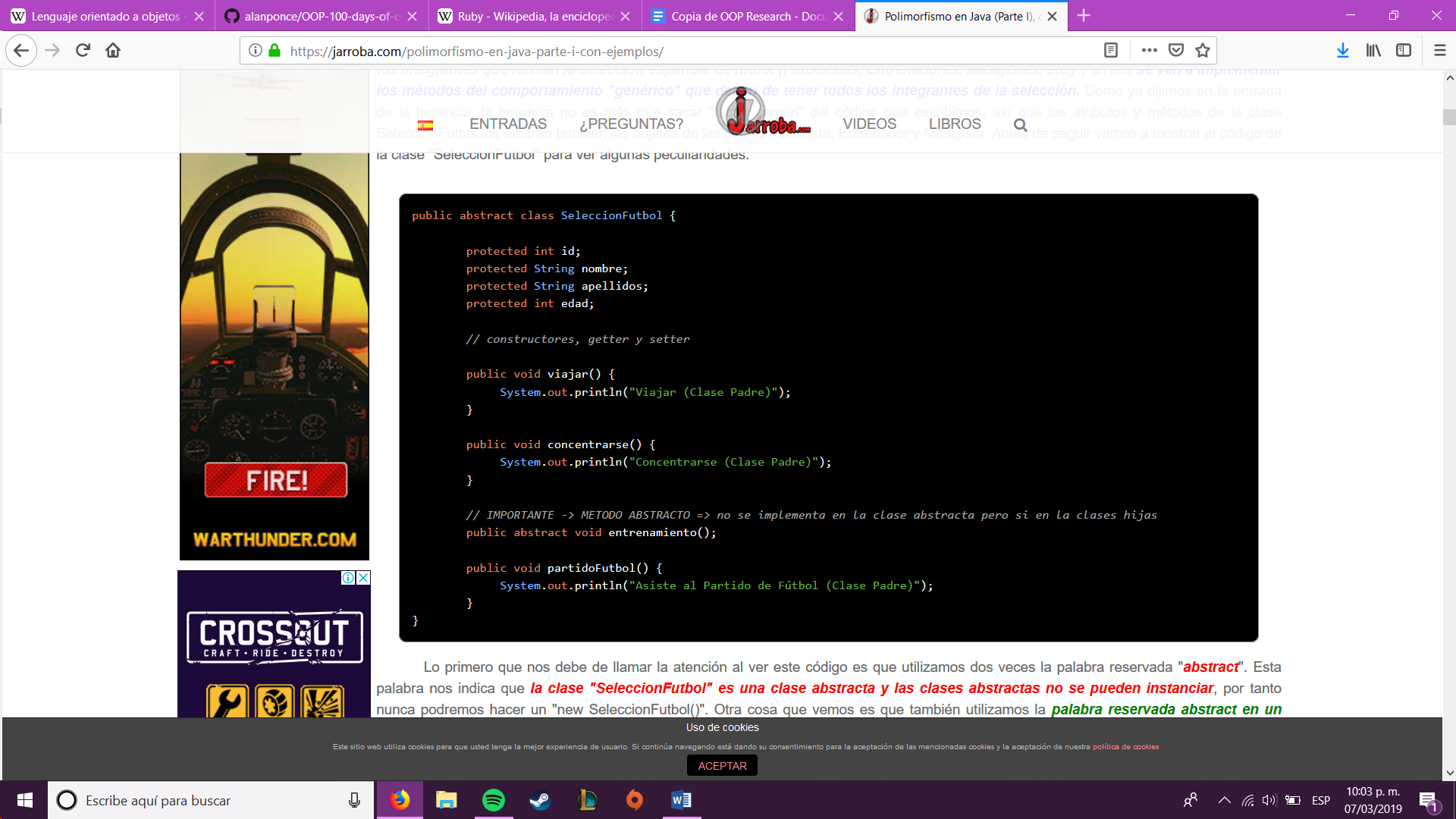


****

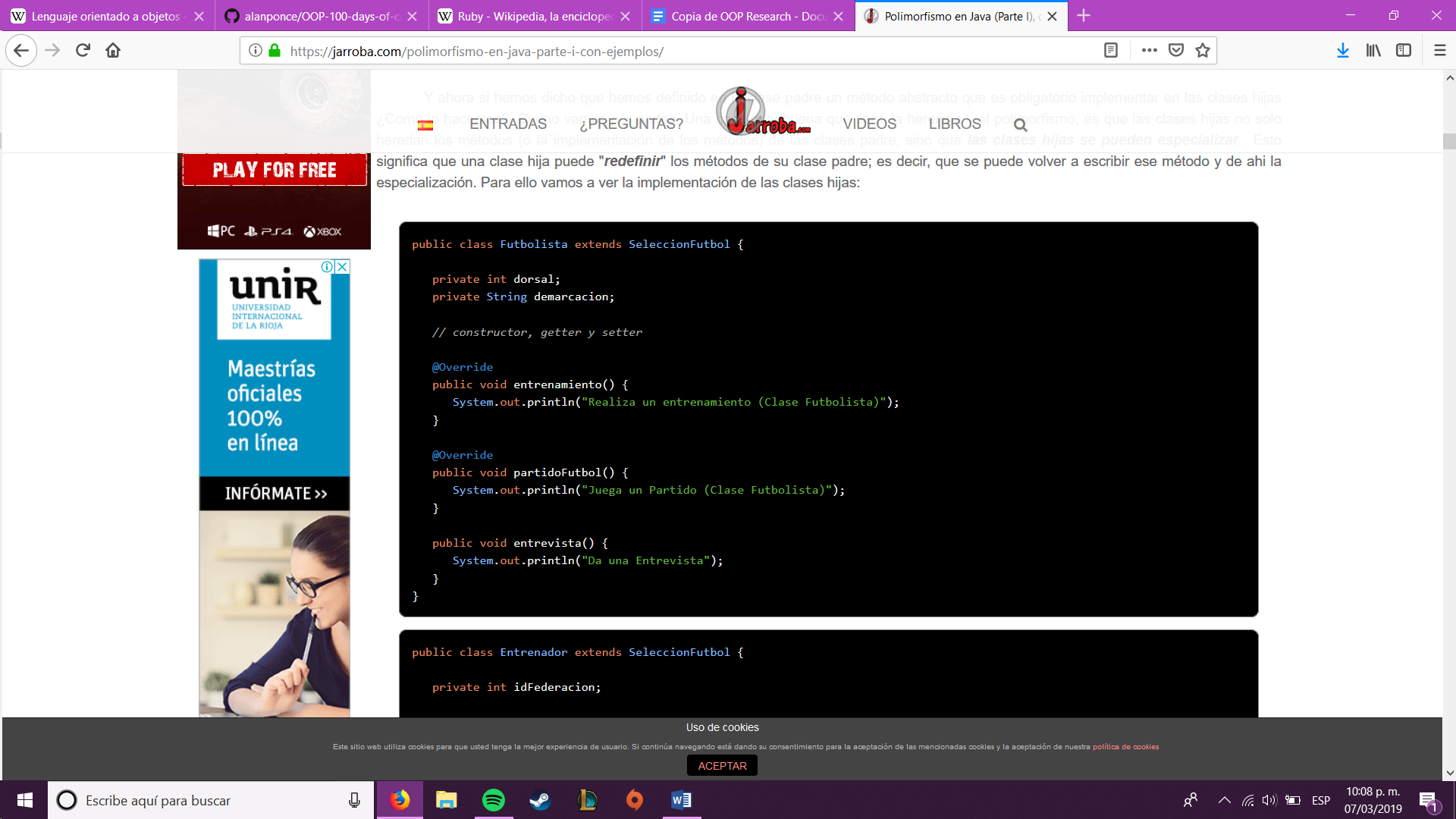
En estos códigos vemos cómo se estructura de manera ordenada lo que parece ser un equipo de fútbol, podemos ver la herencia primero que nada en el principio de las clases cuando después del nombre se pone un extends, este sirve para decirle a la clase hija, cual es la clase padre, también podemos ver la palabra super(), la cual sirve para llamar al constructor padre de esta manera se implementa la herencia en java.

**Polimorfismo**

Volvamos a ver este tema con el ejemplo de los futbolistas. Tenemos el siguiente codigo.

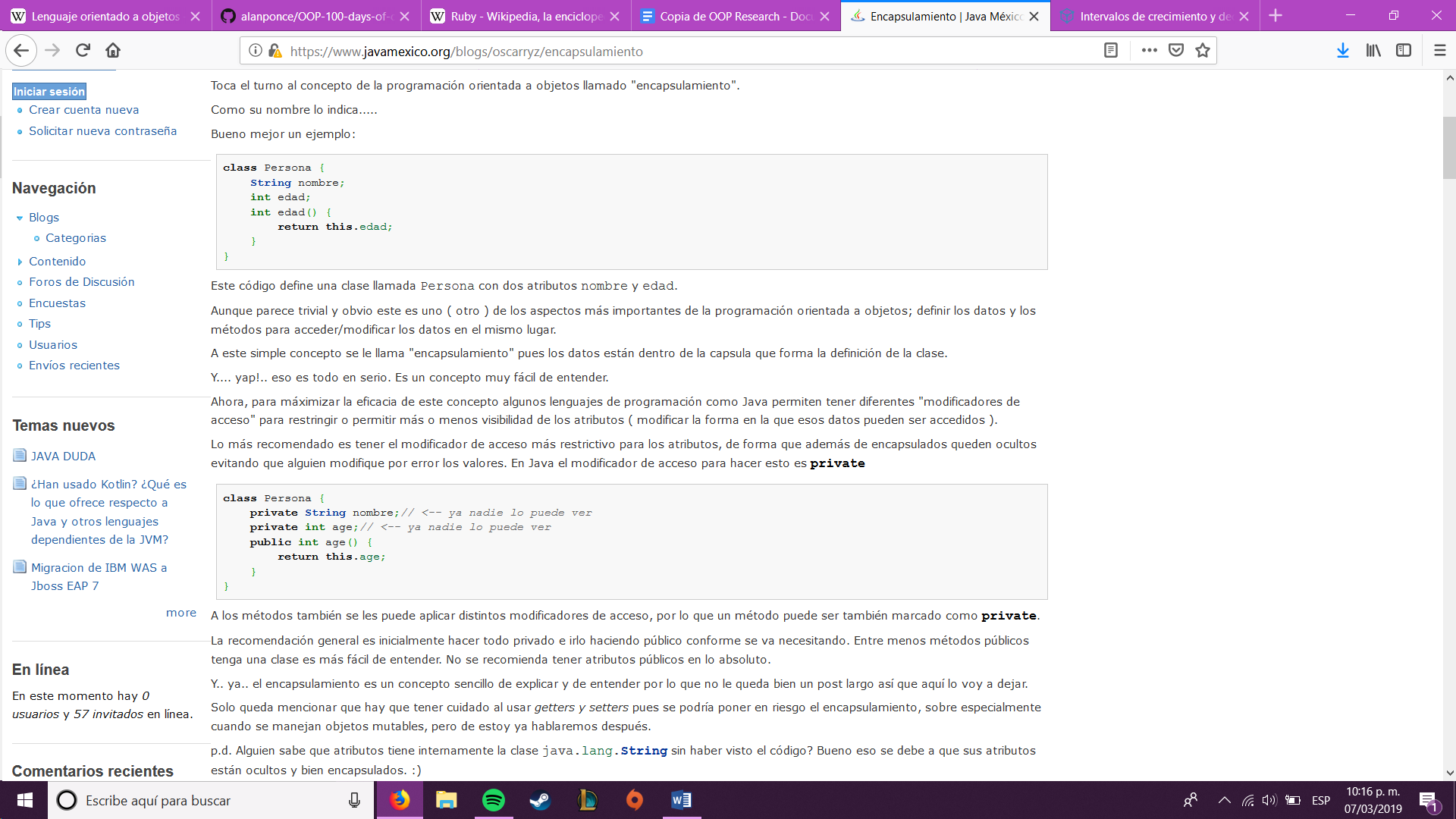


Lo primero que llama la atencion aqui es la palabra abstract en el cuerpo de la declaración de la clase, esto nos sirve para indicar que es una clase abstracta y que no se puede instanciar, es decir no podemos crear objetos de este tipo. También tenemos la misma palabra en un método, esto sirve para indicar que las clases derivadas deben tener el mismo método obligatoriamente.



Después en este código, vemos el método entrenamiento, el cual imprime “Realiza entrenamiento, clase futbolista”, si se ejecuta, ya que si se implementó un nuevo método, imprime lo que está justo en el método entrenamiento. Si la clase futbolista no tuviera un método entrenamiento con una redefinición, se ejecutaría el de la clase padre. SI definimos otra clase con el mismo método, esa clase imprimirá lo que exista en su método.

**Encapsulamiento**

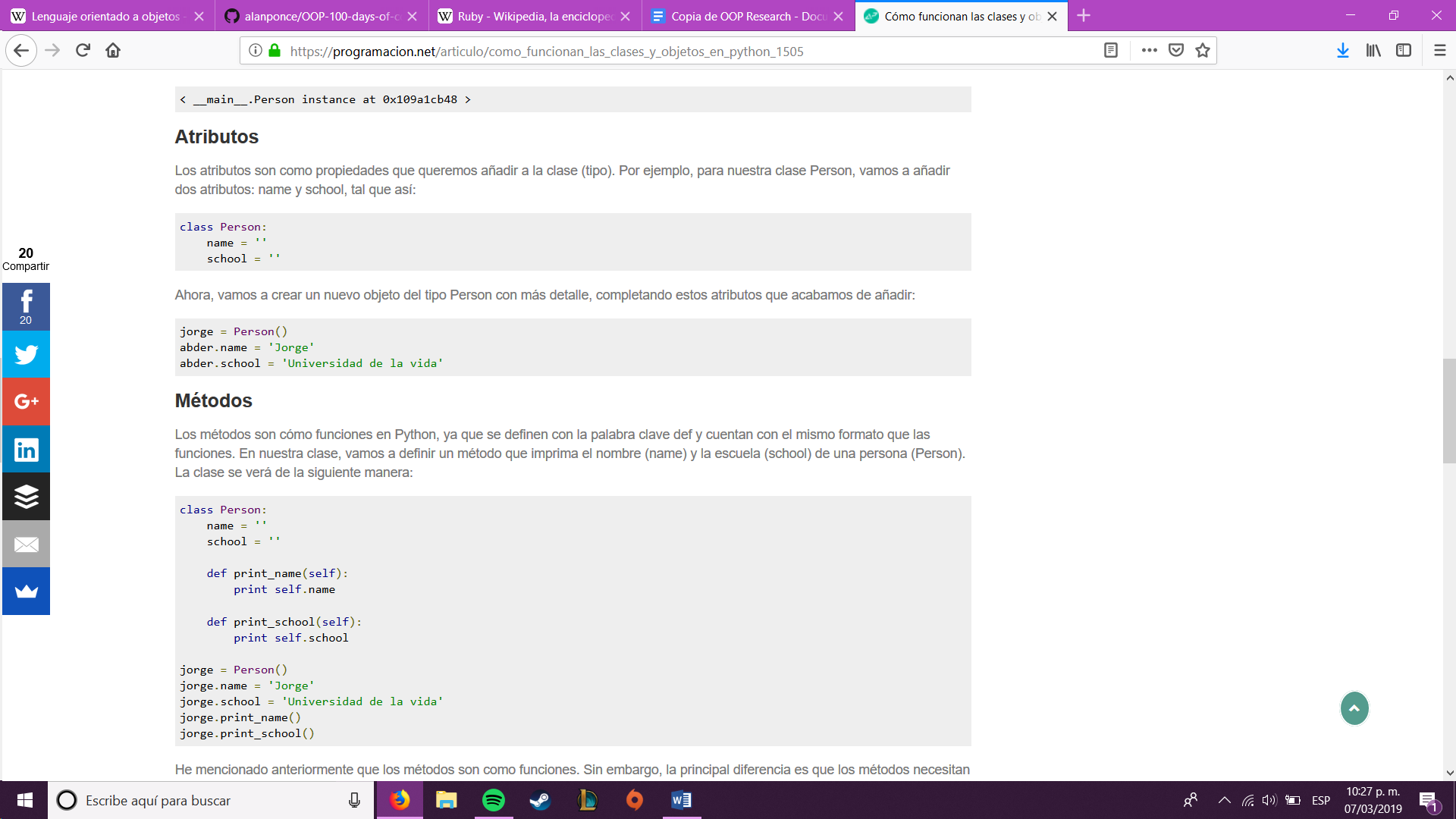
****

Este código es muy sencillo, las variables privadas no se ven, por lo que no se modifican por error.

**Python**

**Una clase**

Las clases en python son muy sencillas.



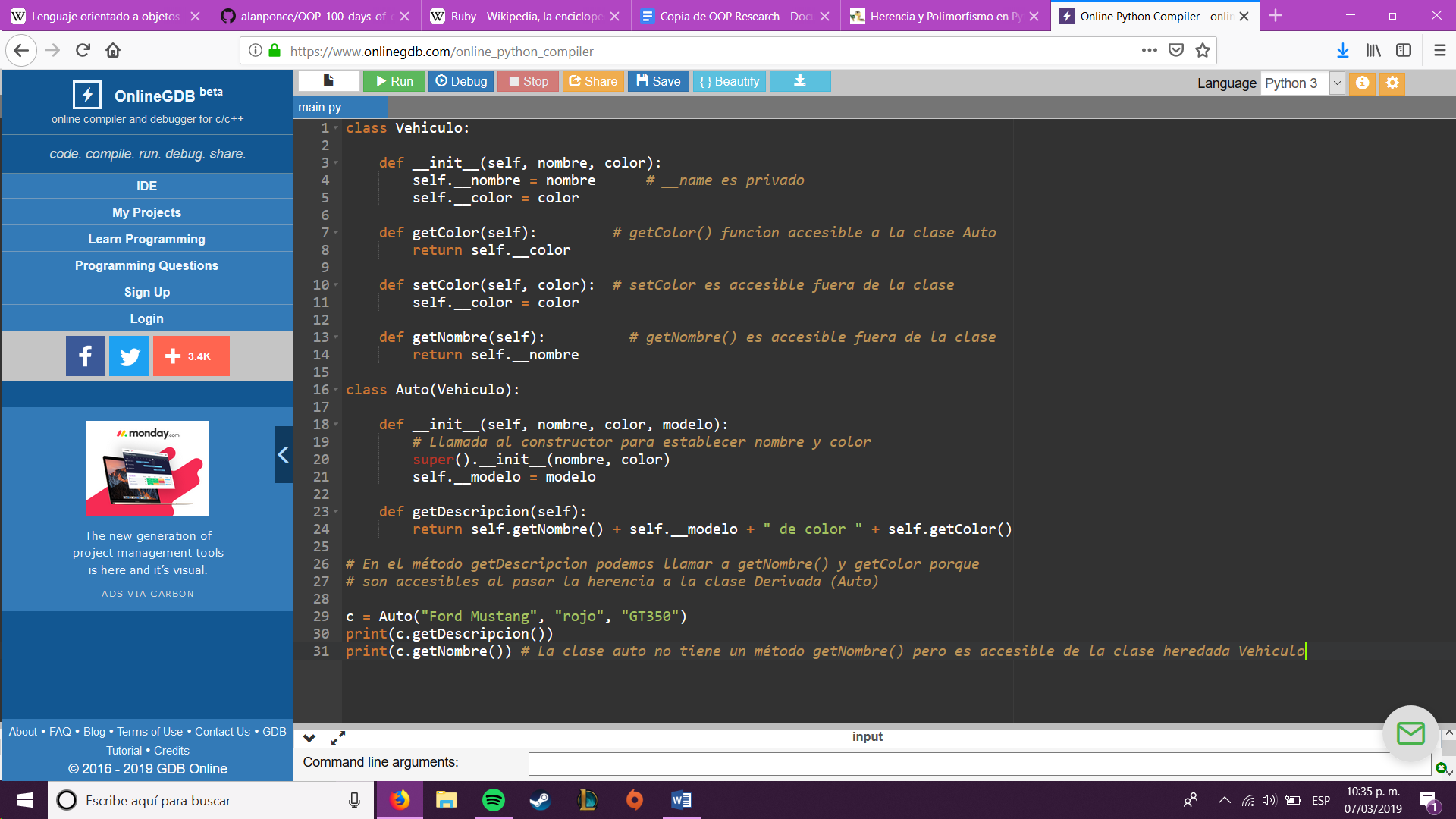
Como podemos ver, la clase inicia con class, el nombre de la clase y 2 puntos, después vemos los atributos y los métodos. Los métodos inician con un def, la principal diferencia es que los métodos necesitan tener un argumento convenientemente llamado self, que se refiere al objeto del método que está siendo llamado.

**Herencia**

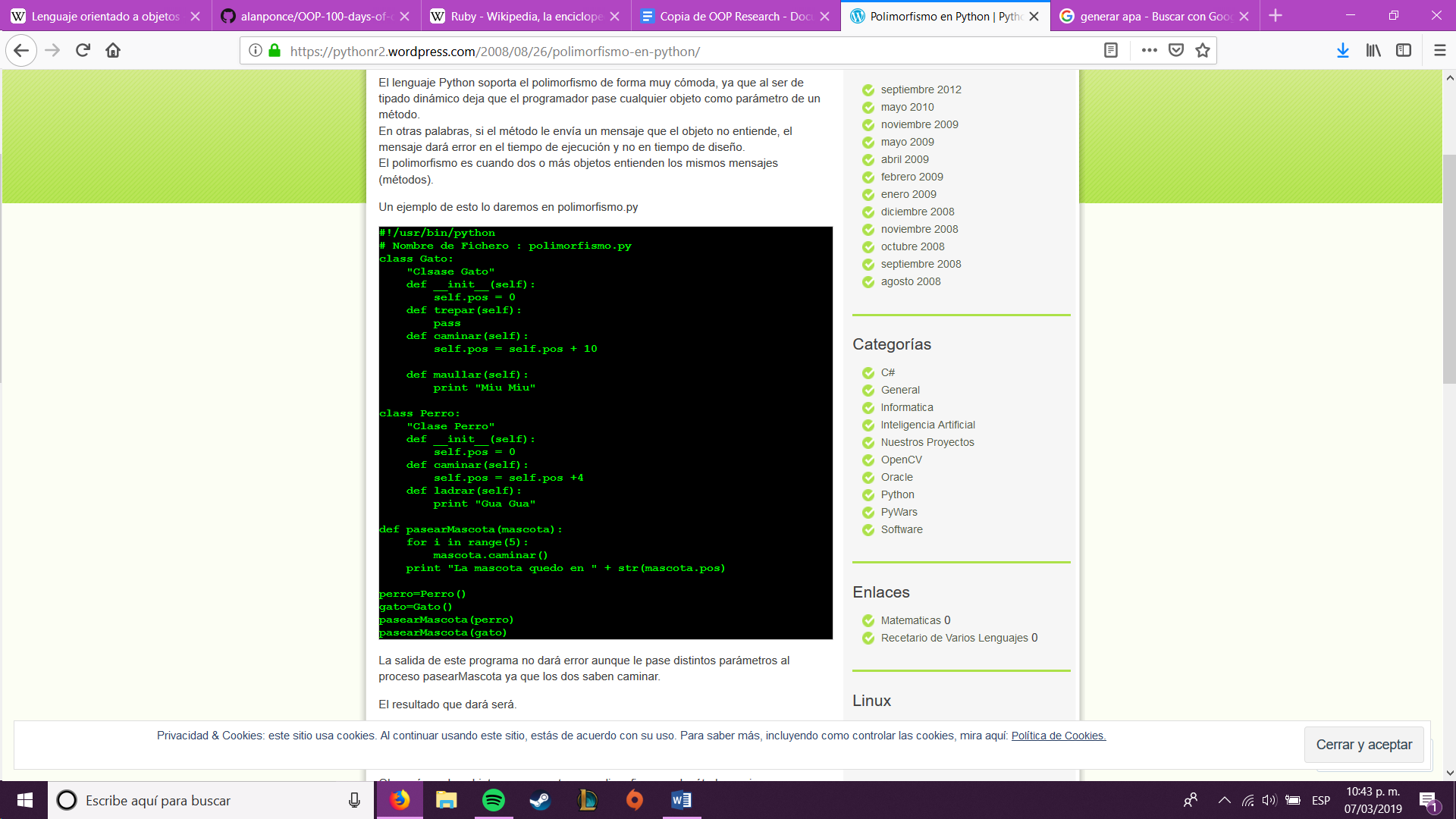
En el siguiente código, vemos la clase base vehículo, y la clase derivada auto. Aunque no se definió un método getnombre en la clase auto podemos acceder a ella ya que en la clase auto si existe, y se herede de ahí. En el código anterior, el método **super**() se utiliza para llamar al método de la clase base.

Así es **como trabaja el método super()**: Supongamos que necesitamos llamar al método **get\_informacion**() que está en la Clase Base desde la Clase Derivada, esto podemos hacerlo utilizando el siguiente código:

*super().get\_informacion()*

****

**Polimorfismo**

****

La salida de este programa no dará error aunque le pase distintos parámetros al proceso pasearMascota ya que los dos saben caminar. Aunque el resultado será diferente para ambos.

**Encapsulamiento**

Veamos un codigo comentado para entender el encapsulamiento en python.

# Encapsulacion en python

class Automotor(object) :

"""Clase de la cual heredan las demas y que es del nuevo estilo en python"""

def \_\_init\_\_(self, ensambladora, nombre) :

self.setEnsambladora(ensambladora)

self.setNombre(nombre)

print "Automotor", nombre, "Creado!!!"

def setEnsambladora(self, ensambladora) :

"""Determina la ensambladora"""

self.\_\_ensambladora = ensambladora

def getEnsambladora(self) :

"""Retorna la ensambladora"""

return self.\_\_ensambladora

def setNombre(self, nombre) :

"""Determina el nombre del automotor"""

self.\_\_nombre = nombre

def getNombre(self) :

"""Retorna el nombre del automotor"""

return self.\_\_nombre

#Determinamos que el unico acceso a las variables es por los metodos

ensambladora = property(getEnsambladora, setEnsambladora)

nombre = property(getNombre, setNombre)

class Carro(Automotor) :

"""Esta clase hereda de Automotor y sus metodos"""

pass

if \_\_name\_\_ == '\_\_main\_\_':

carro = Carro("Toyota", "Celica")

moto = Motocicleta("Auteco", "Pulsar")

print "\n\n"

print "El carro es", carro.getNombre()

print "La moto es", moto.getNombre()

**Conclusiones**

En la programación es muy importante ver cuáles son nuestras necesidades para poder elegir de la manera más sabia las herramientas que necesitaremos. En cuanto a la programación procedural, es muy básica, funcional, fácil y rápida, nos puede servir para muchos factores y aunque se tiene una opinión de que no es servible, es todo dependiendo de que necesitamos, es decir puede que para un proyecto nos sea más funcional este paradigma. La programación orientada a objetos es más aceptada por el factor de la popularización, puesto que es uno de los paradigmas más empleados en proyectos, gracias a la versatilidad que esta ofrece y a las diferencias herrramientas, si se tuviera que elegir no pondría a ninguna en primer lugar, puesto que recalco, lo importante es lo que se va a desarrollar y apartir de ahi elegir las mejores herramientas.

**Bibliografía**

Java. (2011). *Javapoint*. Obtenido de https://www.javatpoint.com/history-of-scala

Knowlton, & Jim. (2009). *Python.* Anaya Multimedia-Anaya Interactiva.

Marcos. (2014). *Marcos Web*. Obtenido de http://www.marcocantu.com/epascal/English/ch01hist.htm

Mueller, J. P. (21 de Junio de 2015). *New Relic*. Obtenido de https://blog.newrelic.com/engineering/python-programming-styles/

Pozo, S. (Septiembre de 2000). *C++ con clase*. Obtenido de http://c.conclase.net/curso/index.php?cap=024

Scala. (s.f.). *Scala*. Obtenido de https://www.scala-lang.org/

Stroustrup, B. (2000). *El lenguaje de programación C++.* Madrid: Addison-Wesley Pub Co.

http://simson.net/ref/NeXT/

Apple. (26 de Junio de 2018). *Wayback Machine*. Obtenido de https://web.archive.org/web/20041011143829/http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/index.html

Carrero. (s.f.). *Programacion.net*. Obtenido de https://programacion.net/noticia/ruby\_on\_rails\_esta\_muerto\_1516

Bitacoras.com. (26 de Noviembre de 2011). *Poesiabinaria*. Obtenido de https://poesiabinaria.net/2011/11/polimorfismos-enrevesando-la-herencia-entre-clases-c /

http://www.chuidiang.org/clinux/sobrecarga/ocultacion.php

Prieto. (2 de Junio de 2009). *Scribd*. Obtenido de https://es.scribd.com/doc/16042543/Ejemplos-de-clases-en-Java

Moya, R. (8 de Abril de 2014). *Jarroba.com*. Obtenido de https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/

Reyes, O. (19 de septiembre de 2011). *Java Mexico*. Obtenido de https://www.javamexico.org/blogs/oscarryz/encapsulamiento

Caraballo, D. (s.f.). *Mi diario Python*. Obtenido de http://www.pythondiario.com/2016/10/herencia-y-polimorfismo-en-python.html

Anteon, E. (26 de Agosto de 2008). *Python R2*. Obtenido de https://pythonr2.wordpress.com/2008/08/26/polimorfismo-en-python/