**TRABAJO PRÁCTICO N°3 - ING. SOFTWARE II**

**ALUMNO**: FERNANDO PÉREZ

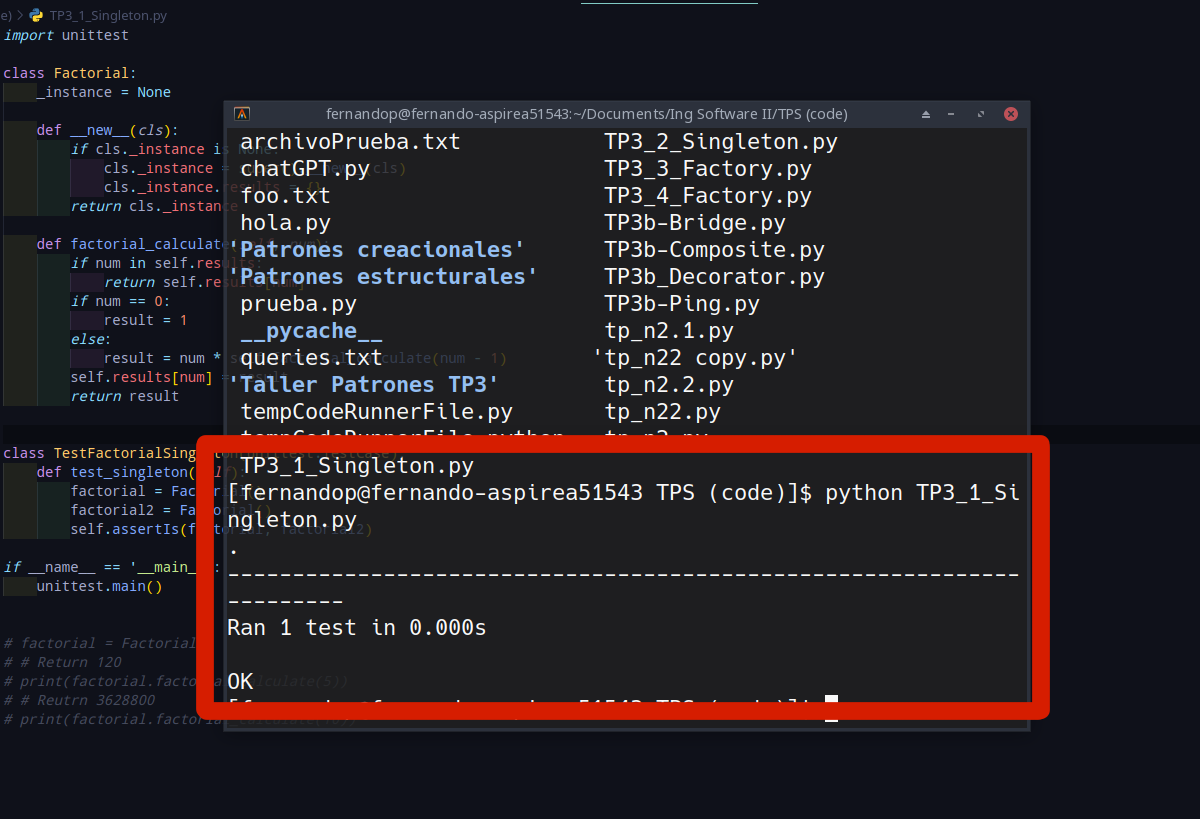
**DOCENTES**: Dr. Pedro E. Colla y Hernán Sanchez

**UADER-FCYT**

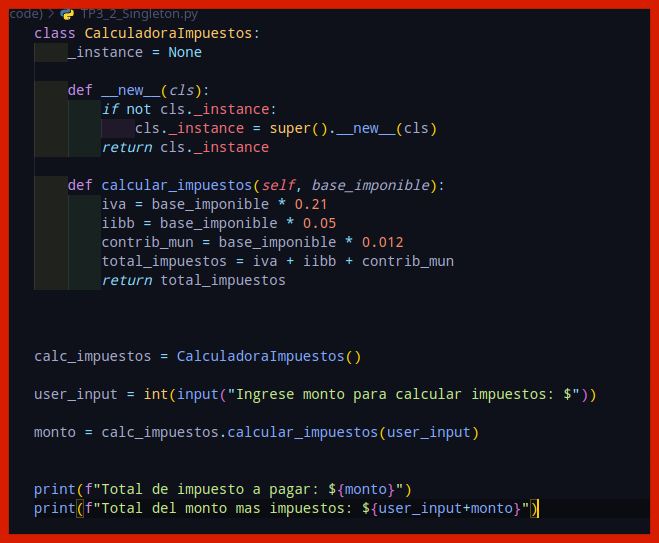
**1)** Para este punto se hace uso del patrón Singleton ya que se requiere utilizar la misma instancia de clases y éste patrón nos permite aseguraron que sea así ya que crea una instancia del objeto que se desea solo si no hay ninguna creada, por lo tanto al crear una instancia de clases no se volverá a crear otra y nos asegura de que siempre se trabaje con la misma instancia, como se muestra a continuación:



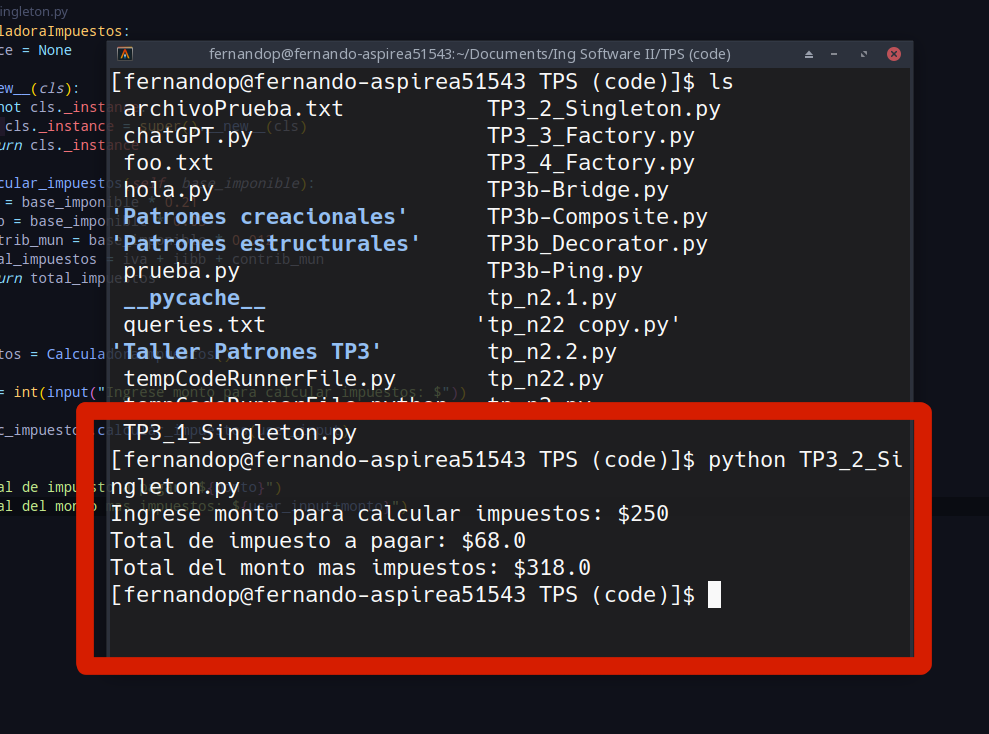
Para probar que se cumple la utilización de una sola instancia de clase, se importa el módulo unittest (como se ve en la imagen). Y al hacer la prueba unitaria TestFactorialSingleton crea dos instancias de clase de la clase Factorial y verifica que ambas sean la misma instancia utilizando el método assertIs, si las dos instancias son la misma, la prueba pasa como se ve en la prueba que nos retorna un “OK” mostrada en la imagen siguiente:



**2)** Este ejercicio decidí hacerlo también bajo la modalidad del patrón Singleton ya que va a ser una sola instancia de objeto la que calcule los valores de impuestos, por lo tanto iniciamos la clase como cualquier clase elaborada con el patrón Singleton que verifique si un objeto de tipo “CalculadoraImpuestos” fue creada y sino lo fue cree el objeto ya listo para instanciar, la clase “CalculadoraImpuestos” tiene también un método para justamente calcular los impuestos como se ve en la siguiente imagen:



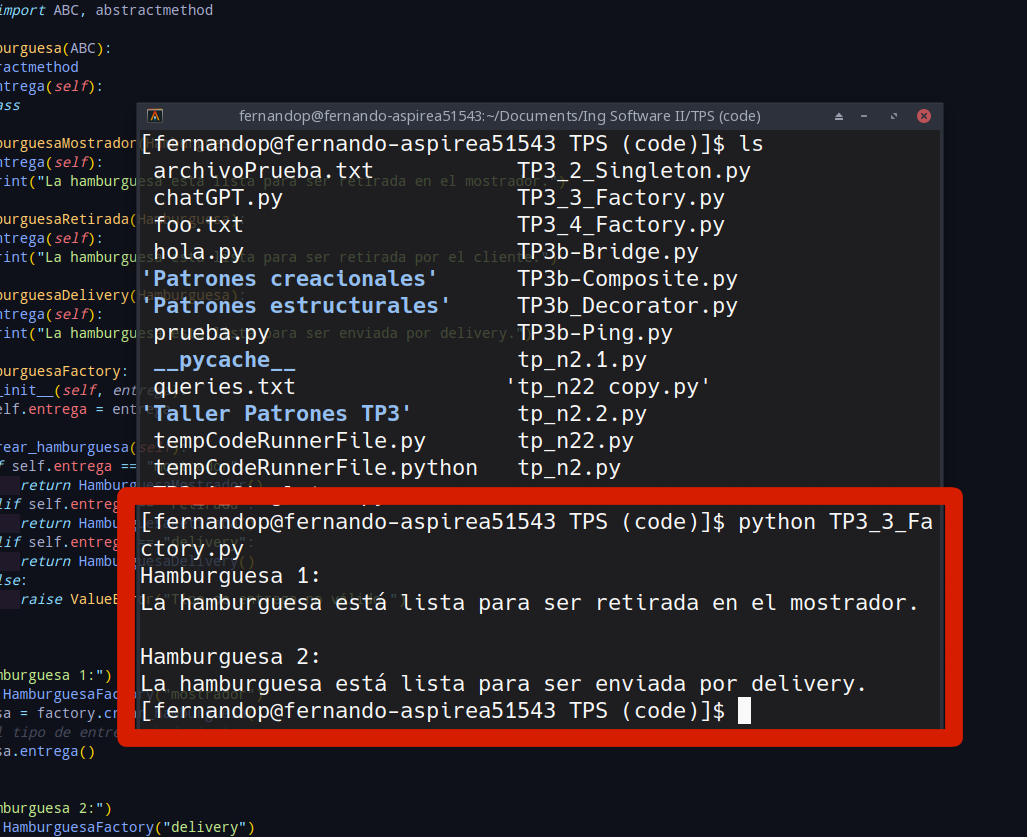
Para verificar que funciona se hace el calculo de un valor ingresado como se muestra a continuación:



**3)** En el ejercicio 3 se utilizó el patrón Factory ya que nos proporciona una interfaz para crear objetos en una superclase permitiendonos con subclases alterar el tipo de objeto que se cree, por lo tanto se utilizó este patrón para crear una clase HamburguesaFactory que se encargue de recibir el parámetro del tipo de objeto que se desea crear (ya sea el tipo de pedido que se desea, recibir por cliente, en mostrador o delivery) y así alterar el tipo de objeto (Hamburguesa) y crearlo del tipo que se desee (delivery, mostrador o para cliente) como se muestra en la imagen a continuación:



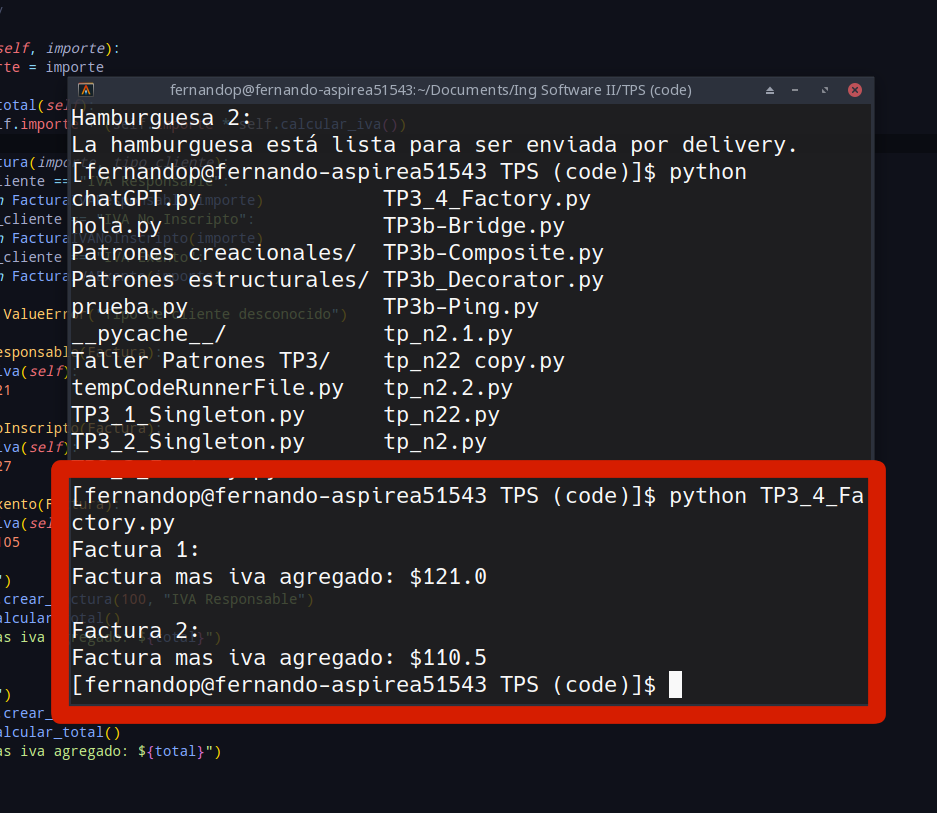
Y en la siguiente imagen se muestra un ejemplo de uso para esta implementación en la cual se crean dos hamburguesas, una para retirar en mostrador y otra para llevar por delivery:



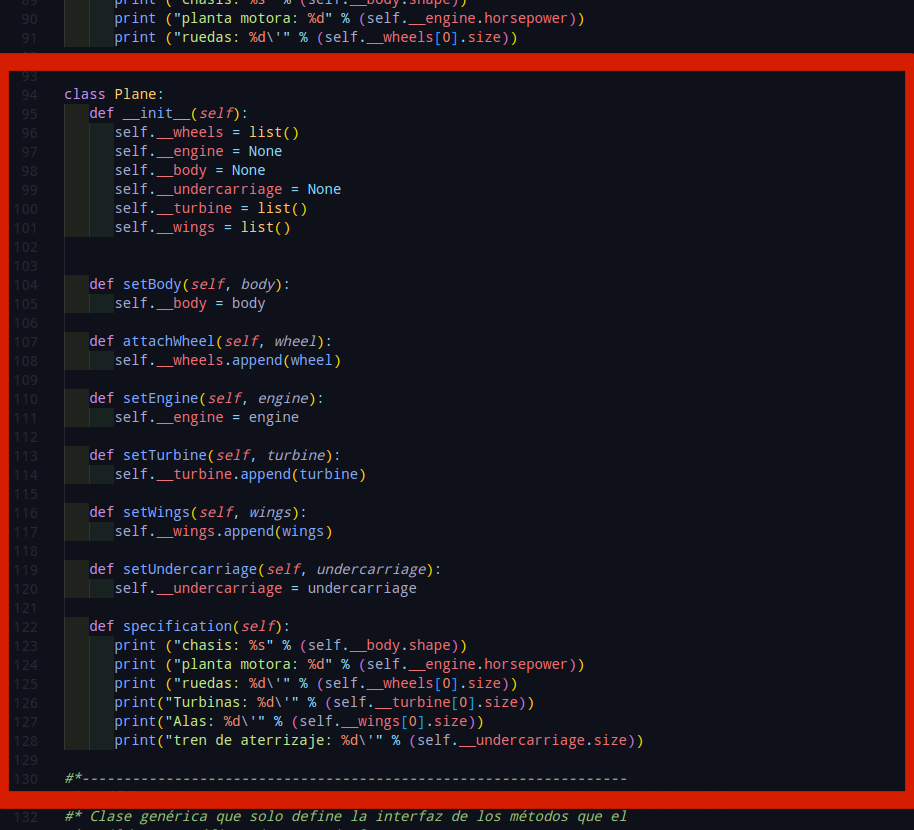
**4)** En este ejercicio se utilizó nuevamente el patrón Factory ya que como se dijo anteriormente permite a las subclases modificar el tipo de objeto, en este caso como se necesitan crear Facturas de determinados tipos (IVA Responsable, Excento o IVA no Inscripto) nos sirve para crear los diferentes objetos, ejemplo del código en la siguiente imagen:



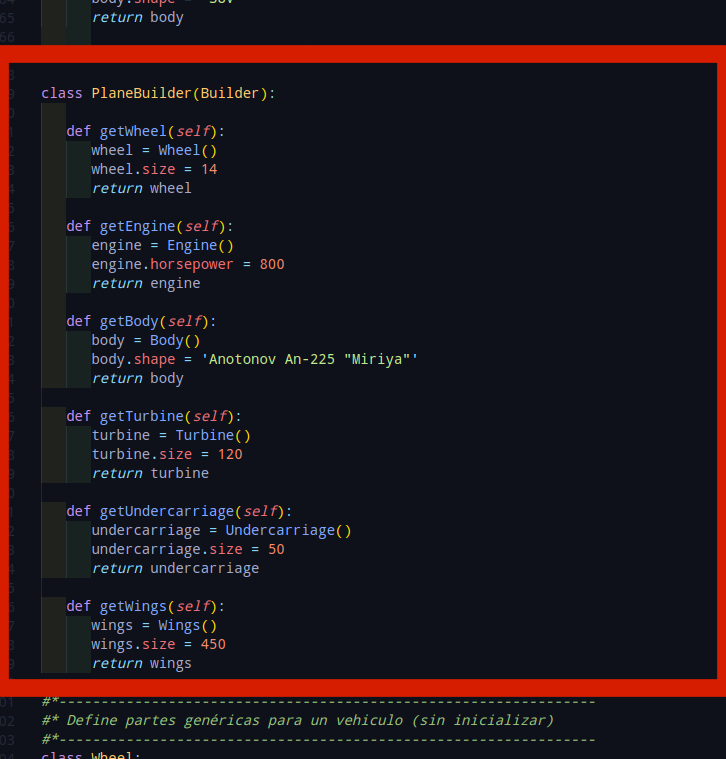
En la siguiente imagen se ve un ejemplo de prueba en la que se le pide dos facturas, una con el calculo de IVA Responsable y otra con IVA Exento como se ve a continuación:



**5)** En el ejercicio 5 se extiende el ejemplo de Taller para que ahora no solo se puedan construir autos sino también aviones, pasando el parámetro de lo que se desea crear, se muetra la clase en la siguiente imagen:



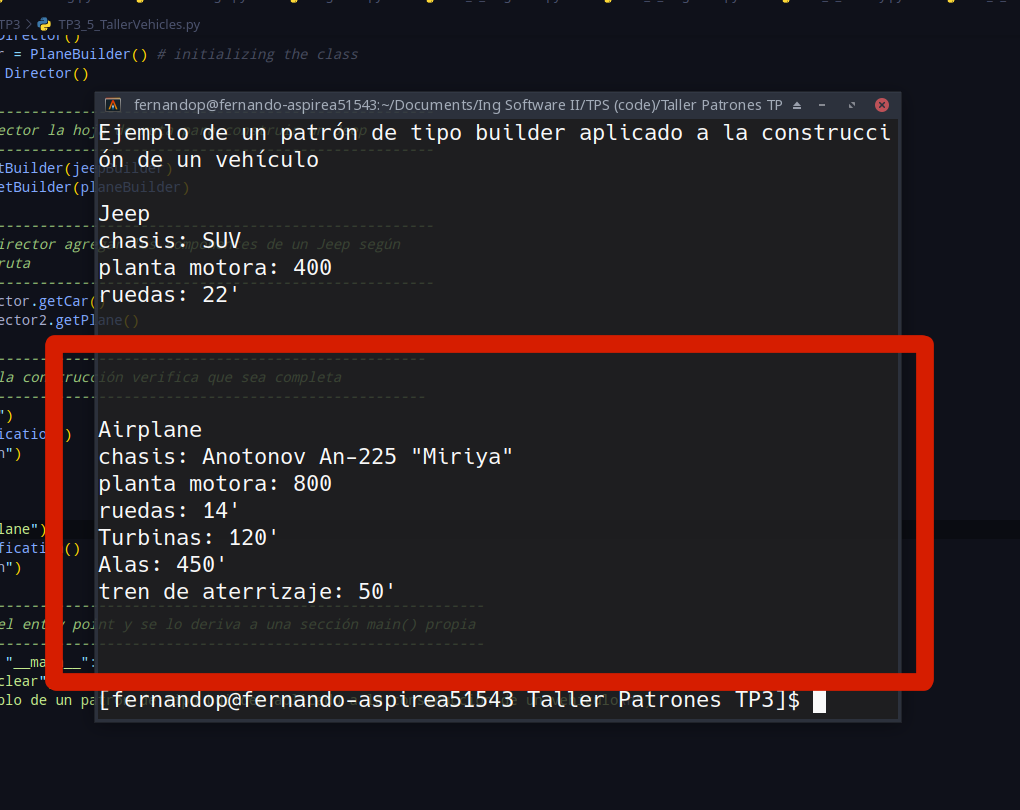
Como se ve se crea la clase Plane con los métodos para establecer las ruedas, turbinas, tren de aterrzaje, el motor y el chasis, y por último el método de especificaciones para poder visualizar todos los componentes que posee y sus detalles, en la próxima imagen se muestran los métodos que hacen a la creación del avión:



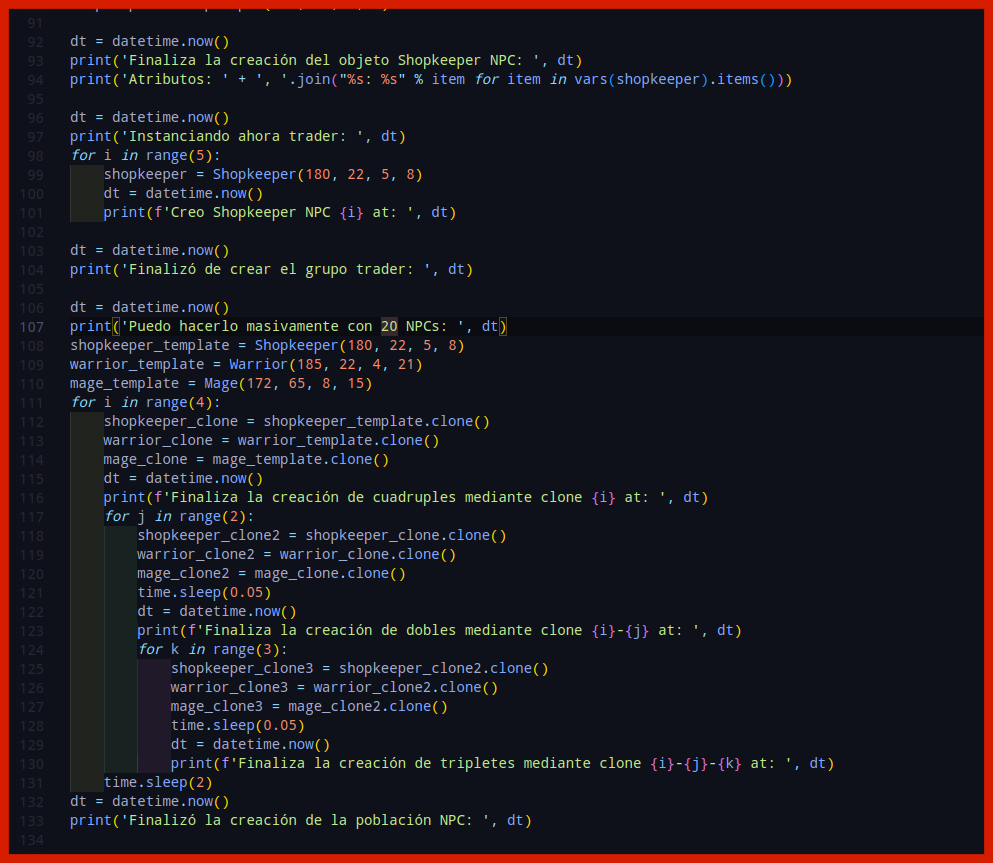
Como se ve en la imagen se encuentran todos los métodos para obtener las turbinas, las ruedas, el tren de aterrizaje, las alas, el chasis y por último el motor.



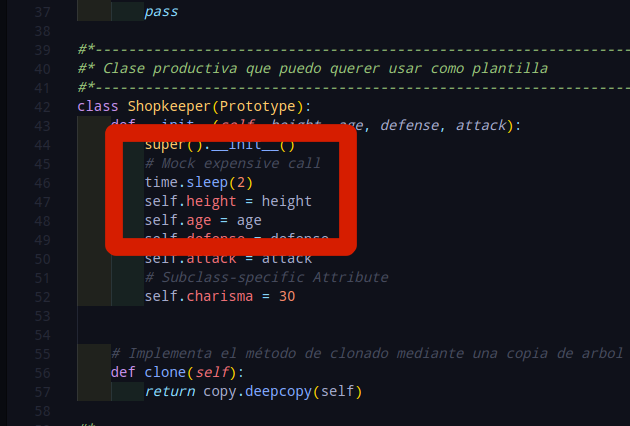
En la imagen anterior se ve el método que crea el avión la cual obtiene el chasis, el motor, las ruedas, turbinas, alas, tren de aterrizaje y luego devuelve el avión completo como se puede ver a continuación:



**6)** En este ejercicio como se utiliza el patrón Prototype, y cada clase creada puede ser clonada y así generar diferentes instancias, al haber ya 10 instancias en el ejemplo dado, para producir 20 instancias lo que se hace es anidar ciclos for para clonar los npcs creados y así alcanzar la meta de 20 como se muestra a continuación:



Y para reducir el tiempo de la carga de procesamiento a 2 segundos, lo que se hace es disminuir el valor del método time.sleep() a 2 seg dentro de la clase propia del npcs con lo cuál para cada npcs creado el tiempo de creación será de 2 segundos, como se muestra en la imagen a continuación:



**7)** El patrón “abstracy factory” puede ser utilizado en la creación de una imobiliaria, ya que proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta.

Por lo tanto en una inmobiliar vamos a tener mucho muebles con la misma base de objeto por ejemplo, si tenemos un sillón para living, el sillón como tal siempre será el mismo lo único que va a cambiar es su estilo por ejemplo, victoriano, moderno o vintage entre otros, por lo tanto podemos tener un fabrica abstracta llamada “SillonFactory” (por nombrarla de alguna manera) y lo que hace es que cuando le pasemos el parámetro del estilo que deseamos esa clase abstracta llama a la fábrica concreta que crea ese estilo concreto de sillón de living por lo tanto si se desa crear un sillón con estilo moderno, el estilo requerido sería pasado como parámetro a la fábrica abstracta y esta última llamaría a la fábrica concreta por ejemplo “SillonFactoryModerna” que sería la encargada de crear ese objeto.

Con esto lo que se logra es que el cliente no tenga que conocer todas las fábricas encargadas de la creación de ciertos estilos en específico ni su implementación, el cliente se refiere siempre a la misma fábrica y utiliza siempre el mismo método y parámetros para cambiar de estilos.

**GITHUB REPOSITORY:** <https://urlis.net/patr-creac>