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

**ALUMNO**: FERNANDO PÉREZ

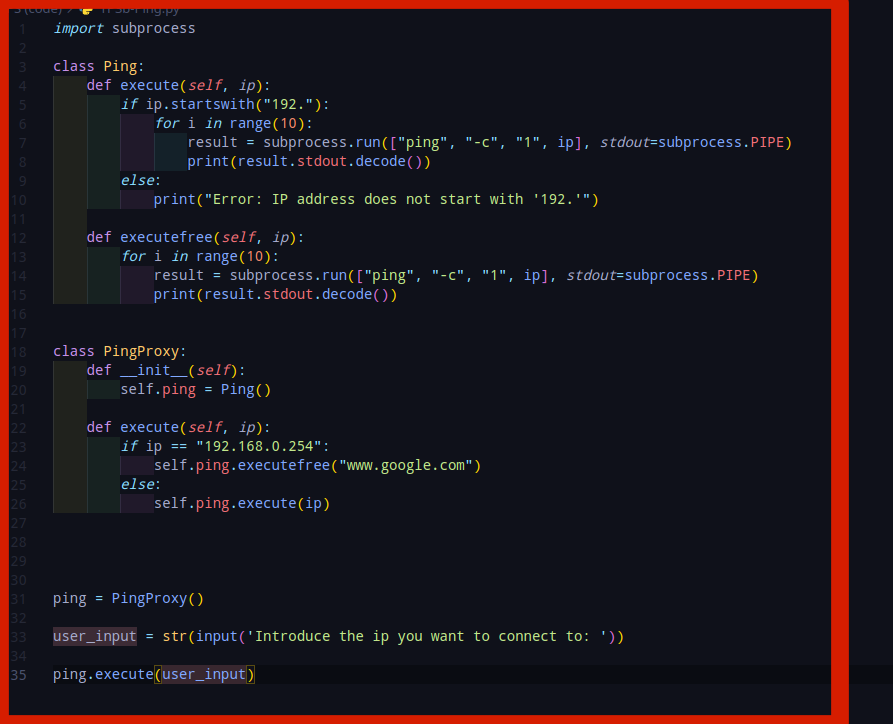
**DOCENTES**: Dr. Pedro E. Colla y Hernán Sanchez

**UADER-FCYT**

**1)** Se crea una clase Ping que tiene un método “execute” que si la dirección provista por el usuario comienza con “192. ....” realiza 10 intentos de ping a la dirección IP anteriormente proporcionada.

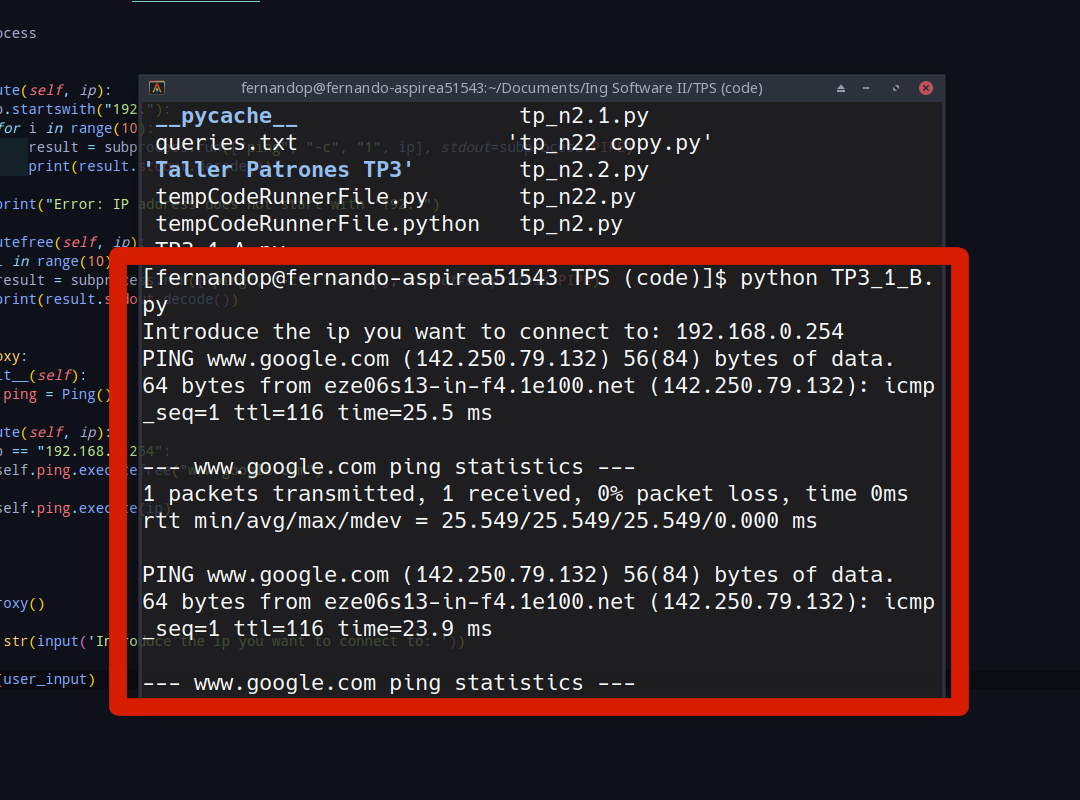
Luego, se agrega otro método llamado “executefree” que realiza lo mismo que el método anterior solo que no hace el control anterior de direcciones (si es “192. ...”).

Por último, se crea una clase PingProxy que posee un método “execute” como la clase Ping, con la diferencia que la clase PingProxy hace control de la dirección que le estamos pasando y ejecuta el método de la clase Ping que corresponda (execute o executefree), eso es posible gracias al patrón Proxy que nos permite controlar el acceso al objeto original mediante la clase PingProxy como se muestra a continuación:



El módulo subprocess lo que hace es ejecutar programas externos y recibir los resultados, lo cual el parámetro stdout lo que hace es recibir el mensaje del subproceso con el método subprocess.PIPE como se ve en la imágen, y el comando decode lo que hace es transformar los bytes en string y poder mostrar el mensaje.

A continuación, se muestra un ejemplo conectandose al ping de Google:

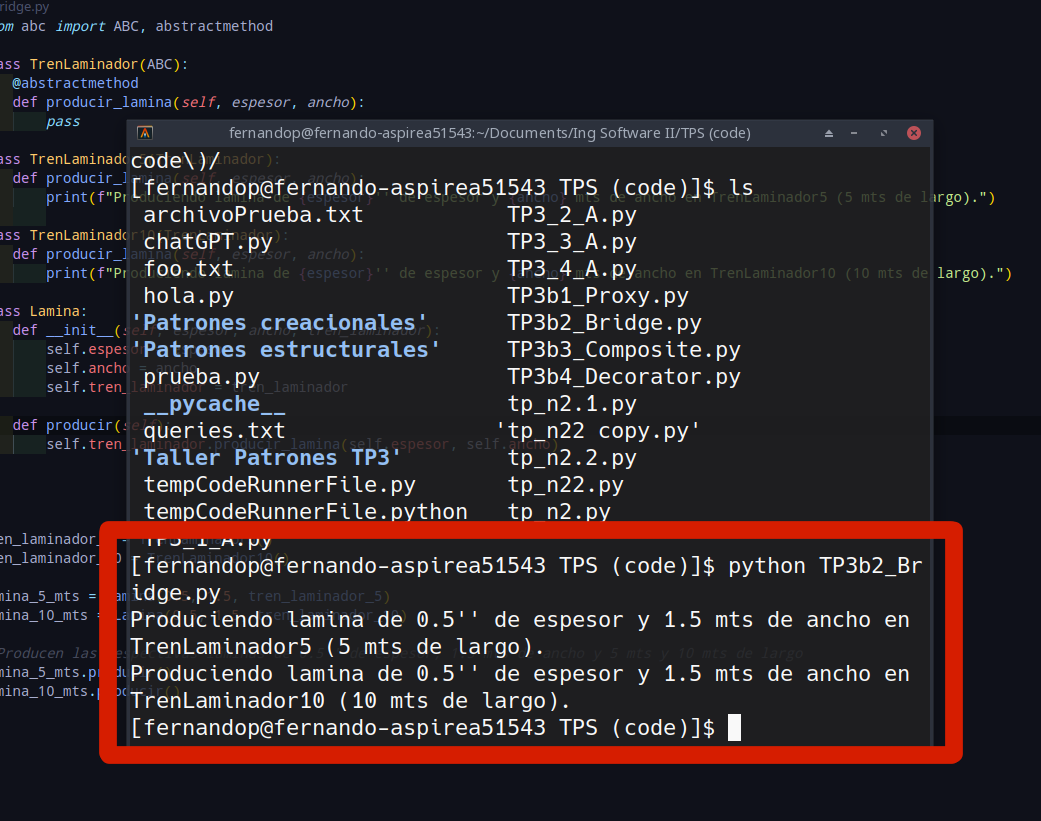


**2)** Se crea la clase Lámina que recibe como parámetro el espesor y el ancho y el largor que se va a querer, en éste caso para hacer ésto último, se necesita pasarle como parámetro el Tren Laminador encargado de fabricar las láminas de dicho largor. Para eso anteriormente se define una clase abstracta de Tren Laminador lo cuál tiene un método “producir” que da la orden de producir dichas láminas, pero al ser una clase abstracta ésta clase transmite el mensaje a la clase de Tren Laminador correspondiente, para eso se crean dos clases más de Tren Laminador con el largo que se desa producir. Una clase de Tren Laminador para construir láminas de 5 mts de largo y otra clase para construir 10 mts de largo.Y luego en la clase Lámina le pasamos el parámetro del tren laminador que deseamos utilizar y la clase abstracta se hace cargo de que eso suceda gracias al uso del patrón Bridge que nos permite dividir clases muy grandes en conjunto de clases relacionadas entre si, como se muestra a continuación el código:



El patrón bridge lo que hace es permitirnos separar la abstracción de su implementación como se puede ver, que la clase abstracta Lámina está separada de la implementación de los trenes laminador.

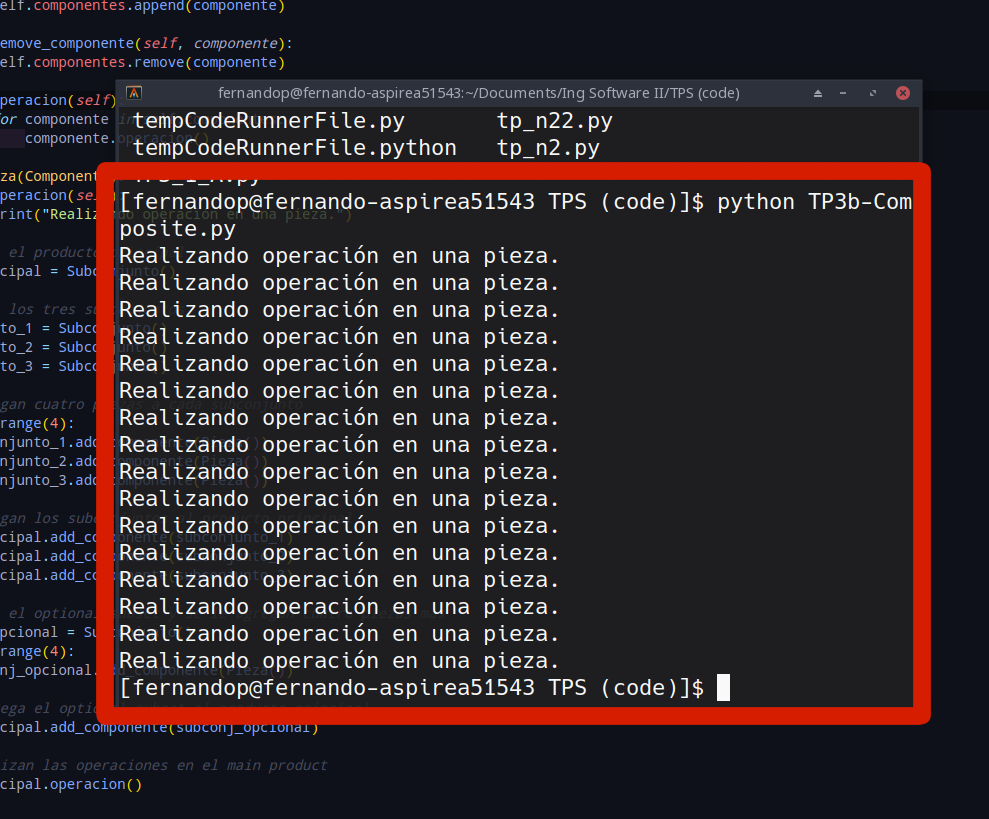
Acá un ejemplo de código de su funcionamiento:



**3)** Para resolver este ejercicio se hace uso del patrón Composite lo que nos permite componer distintos objetos en estructuras de árbol y utilizar esas estructuras como objetos individuales. Se crea la clase abstracta Componente que nos pemite pasar la operación que se desea realizar sobre la pieza, y luego se crean las clases Pieza y Subconjunto, la cual la clase pieza sería el componente en si mismo y la clase Subconjunto tiene una lista de componentes como se puede ver en la imagen a continuación que se crea el porducto principal como un subconjunto y se definen luego, otros subconjuntos y a esos se les agregan piezas y por último esos subconjuntos con piezas añadidas se agregan al producto principal y por último se define un subconjunto opcional el cual el proceso se repite y se agrega por último, nuevamente al producto principal para finalizar con su producción, se muestra una parte del código a continuación:

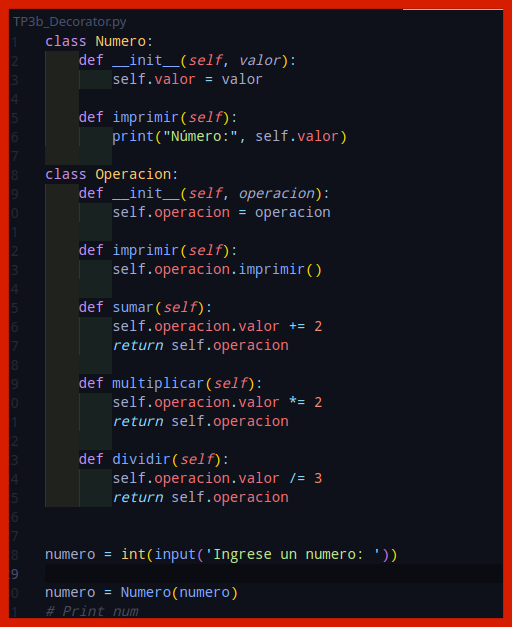


Mostramos como sería el programa en funcionamiento en la siguiente imagen:

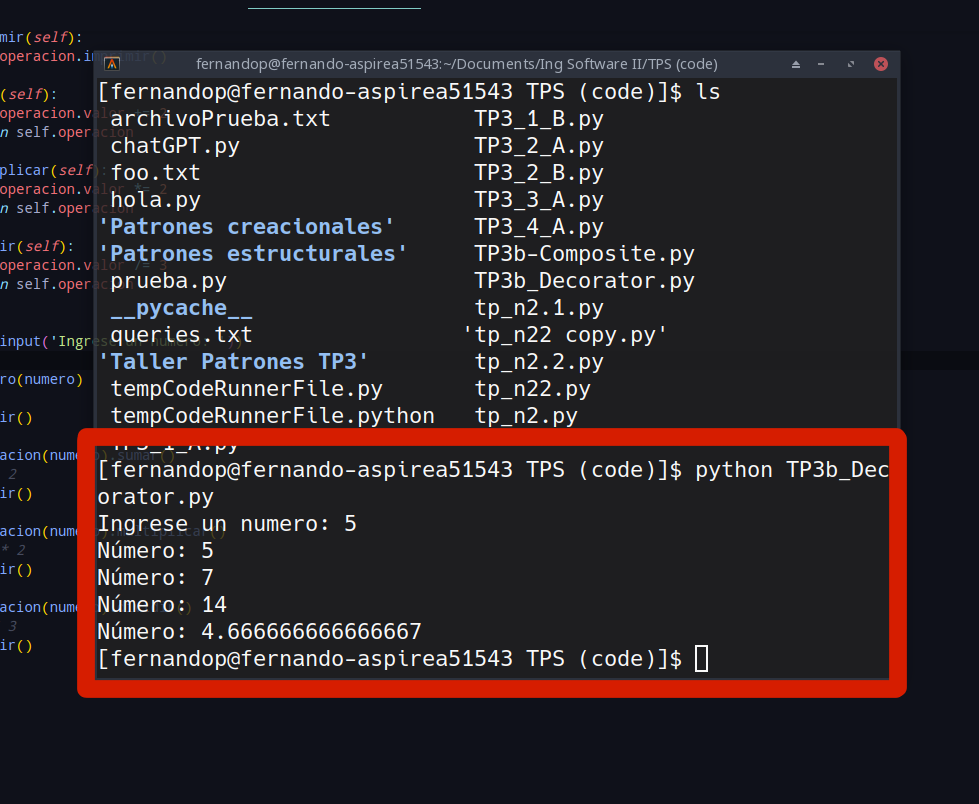


Como se dijo anteriormente, una vez que se crearon todos los subconjuntos y se añadieron las piezas correspondientes se realizan todas las operaciones correspondientes como se ve en la imagen anterior.

**4)** Para resolver este ejercicio se utiliza el patrón Decorator que nos permite adjuntar comportamientos nuevos a objetos ya creados, en el ejercicio se crea una clase Numero que representa el valor con el cual vamos a trabajar y luego se crea la clase Operacion que es la clase base para el uso del patrón Decorator, que decora la clase Numero y agregan modificaciones y funcionalidades adicionales como suma, multiplicación y división sin modificar su estructura, como se muestra en la siguiente imagen:



A continuación mostramos un ejemplo de funcionalidad haciendo los calculos anidados, primero suma (suma 2), después multiplicación (multiplica x 2) y por último división (divide por 3) para el numero con valor 5, así que cada resultado de la operación anterior se utiliza en la operación siguiente y se puede ver que los resultados son acertados:



**5)** El patrón Flyweight sirve para reducir los requerimientos de memoria debido a la información redundante que puede tener un objeto para mejorar el rendimiento, eso se logra mediante la inicialización o referencias entre multiples objetos y un ejemplo claro de la utilización del patrón podría ser cuando se desea, supongamos, desarrollar un juego en el cuál van existir muchos enemigos en pantalla, en vez de crear una nueva instancia para cada enemigo el patrón Flyweight nos permite compartir una única instancia de cada tipo de enemigo creado, lo cuál reduce significativamente el uso de memoria. Por ejemplo, se crea una clase Enemigo (Flyweight) que tenga solamente información sobre la imagen, la velocidad del enemigo, salud y demás. Luego se crea una clase de enemigo concreto como por ejemplo EnemigoGigante que tenga las coordenadas concretas de cada enemigo y que referencie a un objeto de tipo Enemigo (que es de estilo Flyweight), si ya existe un objeto con esas mismas características se utiliza ese, sin crear uno nuevo. Por lo tanto se reduce la cantidad de memoria necesaria para almacenar los objetos y mejora el rendimiento.

**GITHUB REPO:** <https://urlis.net/patr-estruct>