

***ACTIVIDAD #1***

***PROGRAMACION ORIENTADA A OBJETOS***



Juan Pablo Rivera Arenas

Mtro. Carlos Iván Sepúlveda Castillo

*INTRODUCCION*

Como forma de iniciar este semestre, ahora dedicado a el área de la Programación Orientada a Objetos, área en donde se trabajara y por tanto, se aprenderá el uso mas extensivo no solo de programas como Visual Studio, sino que también a un particular lenguaje de programación que es ya bastante conocido desde hace ya unos años, tanto como para el usuario promedio, como para el programador principiante o profesional, no importa mucho el quien ni el cómo, pero todos en algún momento hemos oído hablar de Java.

Como se menciono anteriormente, Java es un lenguaje de programación lanzado en 1995, basándose en mayor medida entre C y C++, pero con cambios suficientes para diferenciarlo de ambos. Y si le suena de algo a alguien, probablemente sea por su popularidad los años 2000 y en actualidad.

*Pero volvemos al tema a mano.*

En esta ocasión, a modo de iniciar el aprendizaje con Java, se nos pidió la realización de un programa sencillo donde su función sea la de introducir, almacenar y mostrar datos, eso a grandes rasgos, y para dar más detalle a lo que acabo de decir, es necesario pasar al siguiente punto, donde se profundizara mas al respecto y directo en las entrañas del código y todas sus funciones.

***DESARROLLO***

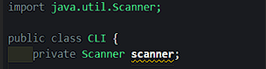
* ***CLI (COMMAND LINE INTERFACE)***

**Un CLI es una forma de tener interacción con un sistema operativo o software usando la introducción de comandos de texto para ello, esto dentro de una consola o terminal.**

****

**Lo que aquí se ve es el CLI realizado para ser ejecutado con la Actividad #1, que es justo en la que se entrara en detalle, su funcionamiento es de la siguiente forma:**

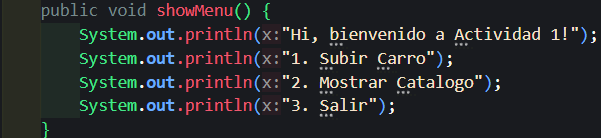
* **El uso del comando *Scanner* de la biblioteca estándar de Java le permite reconocer la entrada de un usuario desde la consola, para después definir una clase *CLI*, que contiene un atributo privado scanner de tipo *Scanner*. Este objeto se utilizará para leer la entrada del usuario:**

****

* **Ahora, nuestro siguiente paso es el constructor de la clase, cuya función es la de inicializar el objeto *scanner*para que lea la entrada desde la consola *(System.in),* cuya función es leer la entrada de datos desde la consola.**

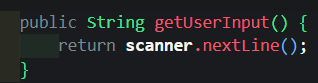
****

* **Para la impresión del menú de opciones de nuestra actividad, debemos de:**

****

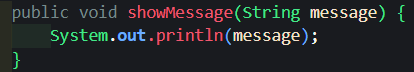
**Al realizar este método, se imprime un menú en la consola con tres opciones para el usuario: "Subir Carro", "Mostrar Catálogo" y "Salir". Es la forma en que se presenta la interfaz al usuario y donde puede interactuar con la misma, utilizando *“public void “*para darle acceso a las otras clases que veremos mas adelante, y para permitir su interacción, mientras que *PrintIn* cumple la función de encargarse de mostrar el menú del texto.**

* **Para obtener la entrada del usuario, haremos uso de lo siguiente:**

****

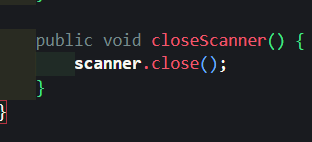
**Este método espera a que el usuario ingrese una línea de texto para después devolver esa entrada a modo de un *String*. Utiliza el método *nextLine()* del objeto scanner.**

**Para mostrar un mensaje, siguiendo la misma línea, funciona de la siguiente manera:**

****

**Este método permite mostrar un mensaje personalizado en la consola,toma un *String*como argumento y lo imprime.**

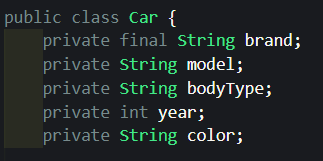
**Al final, cerraremos en Scanner:**

****

**Con todos estos pasos, es que a grandes rasgos explicamos nuestro archivo CLI y el cómo se compone.**

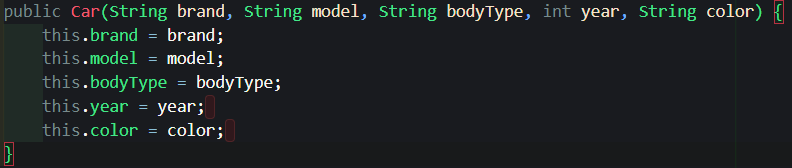
* ***Archivo Car.java***

**Este archivo es una parte fundamental de nuestro código, puesto que esta clase es la encargada de encapsular la información de un coche y detalles los cuales necesitaremos para esta actividad.**

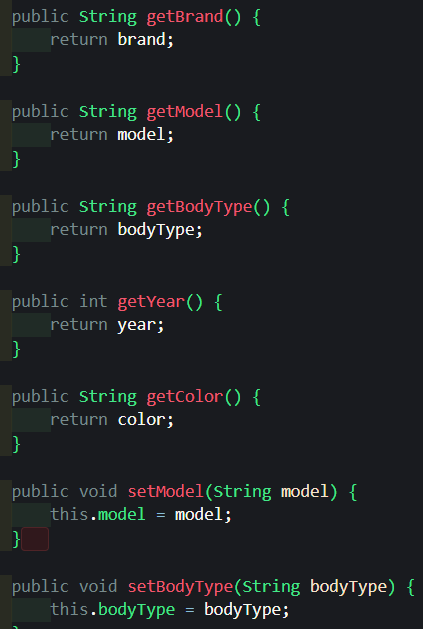
****

**Aquí tenemos que entender que valor tiene cada una de estas líneas, siendo tales valores:**

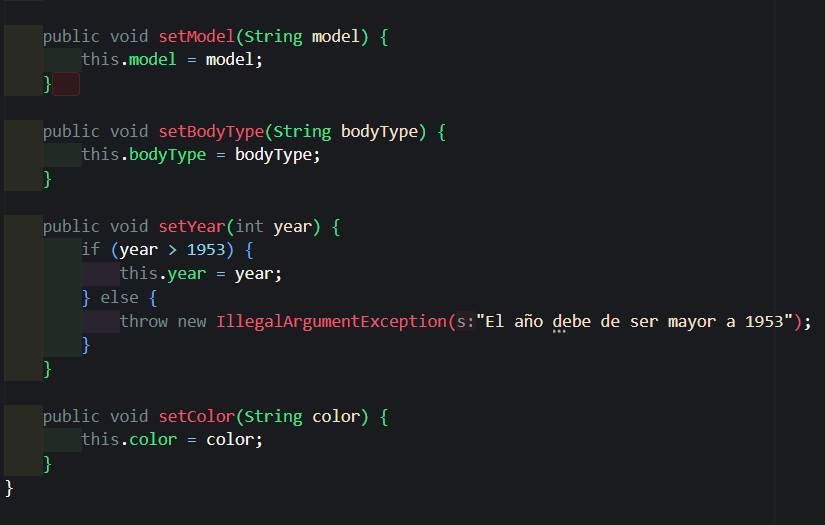
* ***brand:* Una cadena (String) que representa la marca del coche. Es final, lo que significa que su valor no puede cambiar una vez es establecido.**
* ***model:* Una cadena que representa el modelo del coche. Este atributo puede SI puede ser modificado.**
* ***bodyType:* Una cadena ayuda a definir el tipo de carrocería del coche *(Sedan, SUV, Coupe, Convertible, Hatchback, Pick Up, Wagon fueron las carrocerías seleccionadas)* También puede ser modificado.**
* ***year:* Un entero que representa el año de fabricación del coche. Este atributo puede ser modificado, pero de forma limitada.**
* ***color:* Representa el color del coche. Este atributo también puede ser modificado.**

****

**Esto es el constructor de la clase, encargado de inicializar al proporcionar los valores que ahí se piden y se explicaron en el punto anterior.**

****

**Aquí se hace uso del método llamado *“getters”,* cuya función permite a los valores de los atributos privados de la clase, así devolviendo cada valor correspondiente a cada atributo.**

****

**Aquí el método utilizado es conocido como "setters" y permite modificar los valores de los atributos de la clase, tales como:**

* ***setModel y setBodyType:* Permiten cambiar el modelo y el tipo de carrocería del coche, respectivamente.**
* ***setYear:* Permite cambiar el año del coche, pero incluye una validación que lanza una excepción *(IllegalArgumentException)* si el año es menor o igual a 1953. Esto asegura que solo se acepten años válidos.**
* ***setColor:* Permite cambiar el color del coche.**

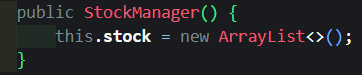
**Eso seria todo respecto a como esta estructurado nuestro archivo Car.java, pasando al siguiente archivo.**

* ***StockManager.java***

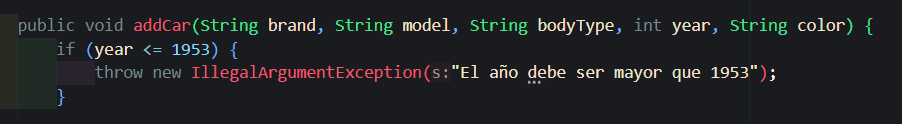
**La función principal de este archivo es encargarse del sistema de gestión del inventario de automóviles al que le daremos vida:**

****

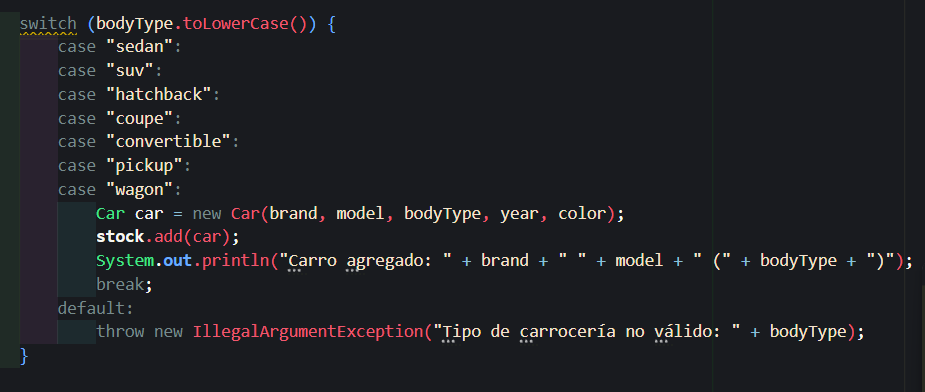
**Aquí la clase *“StockManager “*es la clase que gestiona el inventario de automóviles mientras el atributo *“stock”* declara un “*ArrayList”* de objetos Car, que almacenará los automóviles en el inventario.**

****

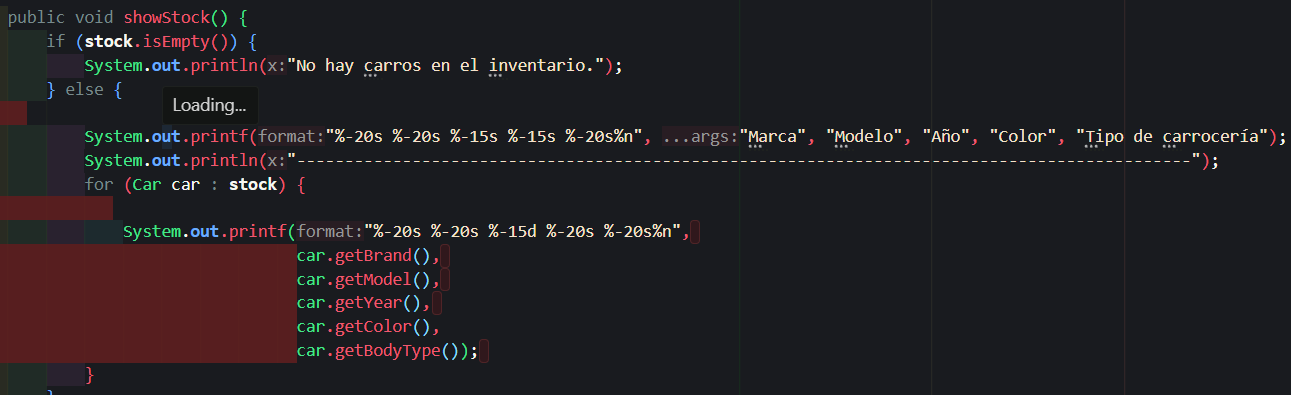
**Aquí, el constructor se encarga de inicializar *“ArrayList” stock* cuando se crea una instancia de *“StockManager”.***

****

**El método *“addCar”* permite agregar un nuevo automóvil al inventario, incluida una validación de Año que se verifica con el año del automóvil y que este sea mayor que 1953. Si no lo es, se lanza una excepción *“IllegalArgumentException”.***

****

**Para la validación de tipo de carrocería se utiliza un *switch*para verificar que el tipo de carrocería sea uno de los tipos válidos *(sedan, SUV, hatchback, etc.).* En caso de si ser válido, se crea un nuevo objeto *Car* y se agrega a *stock.* Si no es válido, se lanza una excepción.**

****

**En esta parte del archivo, el método “showStock“, que muestra todos los automóviles en el inventario. Fuera el caso donde:**

* ***Verificación de Inventario:* Si stock está vacío, se imprime un mensaje indicando que no hay carros en el inventario.**
* ***Impresión de Encabezados:* Si hay carros, se imprimen los encabezados de las columnas.**
* ***Iteración y Formateo:* Se itera sobre cada automóvil en stock y se imprimen sus atributos (marca, modelo, año, color y tipo de carrocería) en un formato tabular.**

**Nota: Toda esta parte de las columnas fue un añadido adicional de mi parte, las columnas se gestionaron mediante el uso de *“System.out.printf”*, que permite formatear la salida de texto en la consola. En cada especificador de formato *(%-20s, %-15s, etc.)* indica cómo se debe alinear y espaciar el texto:**

* + ***%-20s*: Alinea a la izquierda y reserva 20 caracteres para la cadena (por ejemplo, "Marca").**
  + ***%-15s:* Alinea a la izquierda y reserva 15 caracteres (por ejemplo, "Año").**
  + ***Encabezados:* Se imprimen los nombres de las columnas: "Marca", "Modelo", "Año", "Color" y "Tipo de carrocería".**

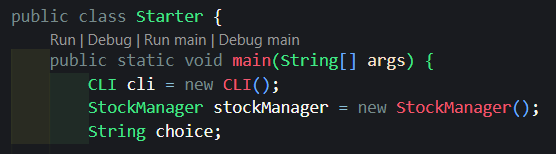
**Al final, se imprime una línea de guiones para separar los encabezados de los datos de los automóviles, mejorando la forma en como se puede leer la información.**

***Starter.java***

**La clase Starter, que actúa como punto de entrada para la aplicación de gestión de inventario de automóviles. A continuación, se explica cada parte del código:**

******

**Se importa la clase CLI, cuya función es manejar la interfaz de línea de comandos (CLI) para interactuar con el usuario.**

****

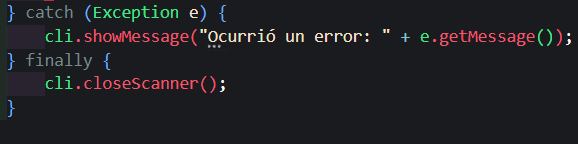
**Clase *Starter*es la clase principal que contiene el método *main*, el punto de entrada de la aplicación donde:**

* ***Instancias:* Se crean instancias de CLI y StockManager. CLI se utiliza para interactuar con el usuario, y StockManager gestiona el inventario de automóviles.**
* ***Variable choice*: Se declara una variable para almacenar la opción seleccionada por el usuario.**

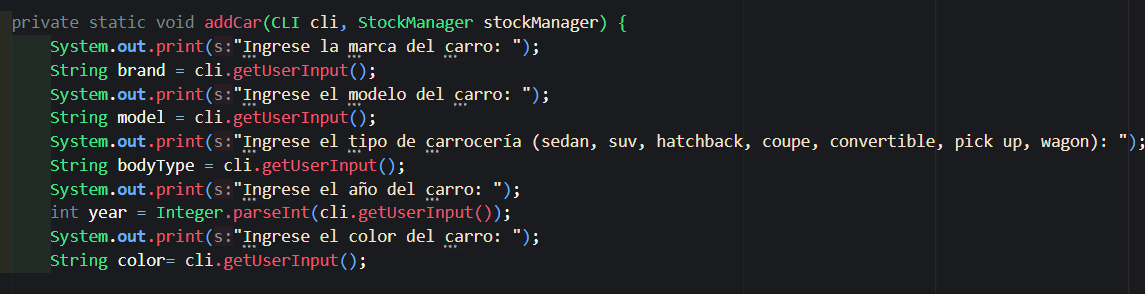
****

**Utilizaremos un bloque try, utilizado para manejar excepciones que puedan ocurrir durante la ejecución, en conjunto con:**

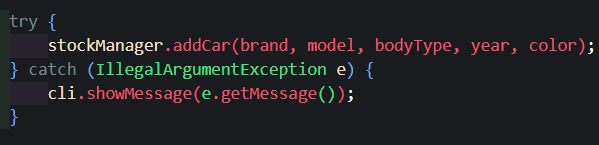
* ***Bucle do-while:* Este bucle se ejecuta hasta que el usuario elige salir (opción "3").**
  + **Mostrar Menú: Se llama a *cli.showMenu()* para mostrar las opciones disponibles al usuario.**
  + ***Obtener Entrada del Usuario:* Se llama a *cli.getUser* Input() para capturar la opción seleccionada.**
  + ***Estructura switch:* Dependiendo de la opción elegida:**
    1. ***Caso "1"*: Llama al método *addCar* para agregar un nuevo automóvil.**
    2. ***Caso "2":* Muestra el inventario de automóviles llamando a *stockManager.showStock().***
    3. ***Caso "3":* Muestra el mensaje de despedida.**
    4. ***Caso default:* Si la opción no es válida, se muestra un mensaje de error.**

****

**Con el uso de un bloque *catch,* se captura cualquier excepción que ocurra durante la ejecución del bloque try y muestra un mensaje de error al usuario y al usar un bloque *finally* se ejecuta al final, independientemente de si ocurrió una excepción o no. Aquí se cierra el escáner de entrada (presumiblemente para liberar recursos).**

****

**El método *addCar* se encarga de recopilar la información necesaria para agregar un nuevo automóvil al inventario, al solicitar al usuario que ingrese la marca, modelo, tipo de carrocería, año y color del automóvil. Cada entrada se obtiene a través de *cli.getUser Input().***

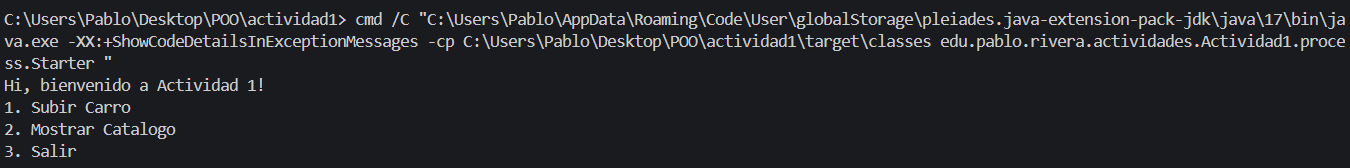
****

**Como último paso de este archivo, en la opción Agregar Automóvil, se llama a *stockManager.addCar*para agregar el automóvil al inventario utilizando la información recopilada.**

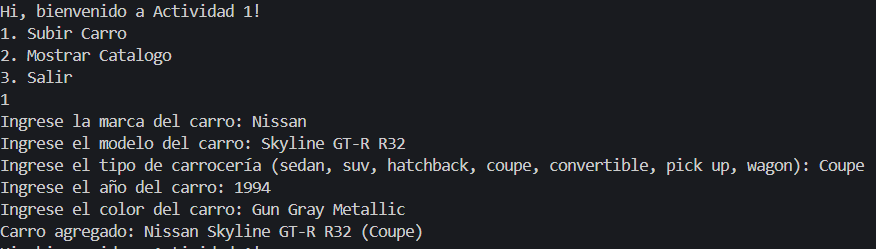
**Para manejo de excepciones: Si se lanza excepción, un *IllegalArgumentException* (por ejemplo, si los datos ingresados son inválidos), se captura y se muestra el mensaje de error correspondiente al usuario.**

**Ahora, tras explicar a detalle la función de cada paso, es momento de ejecutarlo y mostrar su funcionamiento.**

* **Código al ser ejecutado.**

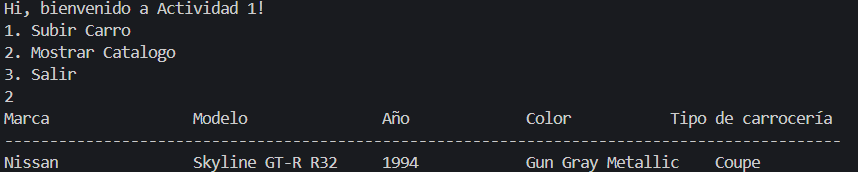
****

* **Al elegir la primera opción**

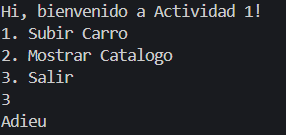
****

**(Apareceran los apartados a llenar por el usuario)**

* **Al seleccionar la segunda opción, si se selecciono anteriormente la primera opción y se llenan los datos.**

****

* **Al seleccionar la tercera opción independiente de cualquier circunstancia**

****

***CONCLUSION***

**Quiero empezar con una opinión pero que considero importante, de primeras es un tema complejo, complicado de entender de una sola vez y hacerse de la sintaxis y conceptos de este de una es complejo y no sencillo en lo absoluto de tragar.**

**Sin embargo, al realizar la actividad, es cuando la situación y la complejidad del aprendizaje dan un rumbo distinto y se convierte en algo mas comprensible y hasta hace más sentido que todo lo escrito, claramente siendo practico, la diferencia de aprendizaje es mas completa y da mas margen a diseccionar el código y entender parte por parte, línea por línea y paso por paso.**

**A nivel de la práctica, puedo ver el por que Java es un lenguaje muy utilizado, puesto que, a pesar de esto ser un ejercicio muy sencillo, las posibilidades que brinda el lenguaje para la creación de diferentes proyectos es bastante extensa y con un lenguaje que aunque complejo de principio, no es tan diferente de los ya conocidos, haciendo su uso, una vez entendido, más practico y dinámico con una mayor posibilidad en ámbitos más allá de solo proyectos estudiantiles o empresariales, pudiendo trasladarse a asuntos mas complejos en donde dará la talla mas que de forma sobresaliente.**

**Solo me queda imaginar los usos que se le podría dar a esto mas adelante, solo quedara esperar y dar pie a entenderlo mejor para explotar todas sus posibilidades.**