La saga **Grand Theft Auto**, desarrollada por **Rockstar Games**, se ha consolidado como una de las franquicias más influyentes y revolucionarias en la historia de los videojuegos. Desde su primera entrega en 1997, la serie se ha caracterizado por ofrecer mundos abiertos con muchos grados de libertad, historias cargadas de sátira social y una jugabilidad que combina acción, conducción y exploración en entornos urbanos inspirados en la realidad.

Tras el rotundo éxito de *GTA V* (2013), que se convirtió en uno de los juegos más vendidos de todos los tiempos, la expectativa por la siguiente entrega, *GTA VI*, solamente creció desde entonces.

12 años después, **Rockstar** aún no publica este juego. Su desarrollo estuvo caracterizado por el silencio y las demoras, a punto tal que se cuestiona su existencia. Algunos incluso hacen chistes del tipo "*We got **X** before **GTA VI***".

Por este motivo, los estudiantes de Algoritmos y Estructuras de Datos *decidieron tomar cartas en el asunto*, ayudando a **Rockstar** a completar el desarrollo del juego mediante la implementación de algunos sistemas.

En esta ocasión, se implementará el sistema de **Garaje**, inspirado en la versión de *GTA V*.

Funcionalidades a implementar

Se deberá implementar el TDA **Garaje**, en el que el jugador podrá guardar sus vehículos. Este TDA deberá permitir:

1. Agregar un vehículo, si hay espacio.
2. Mostrar todos los vehículos con su información.
3. Eliminar un vehículo, por nombre (si existe).
4. Mejorar **permanentemente** el garaje, agregando un espacio para vehículos.
5. Agregar **créditos** al garaje (para testear).
6. Mostrar el valor total del garaje.
7. Mostrar el costo total diario para el mantenimiento del garaje, que dependerá de los vehículos del jugador.
8. Exportar la información del garaje en un archivo de texto.
9. Cargar un garaje a partir de un archivo de texto.

Los vehículos tendrán un *nombre*, un *precio*, una *cantidad de ruedas*, y una *capacidad de gasolina*, en litros. Los tipos de vehículos que se pueden guardar, *por el momento*, son **autos** y **motos**. Todos los autos tienen *4* ruedas, y todas las motos tienen *2* ruedas.

El garaje, inicialmente, tendrá espacio para *5* vehículos. Se puede mejorar el garaje de forma **permanente** utilizando *50* créditos, agregando un espacio para vehículos. Estos créditos, por simplicidad, van a ser una moneda diferente al dinero del jugador, y solo se utilizará para mejoras del garaje.

El costo diario de mantenimiento del garaje será la suma del costo para mantener **cada** vehículo. El costo por vehículo es:

Costo por rueda * Cantidad de ruedas + Costo por litro * Capacidad de gasolina

Como los vehículos son de alta gama, el costo por rueda para autos es de *$50*, mientras que el costo por rueda para motos es de *$30*. El costo por litro de gasolina es *$1*, fijo para todos los vehículos.

El archivo de texto para guardar/cargar un garaje es de extensión **.csv**, y su formato es el siguiente:

CAPACIDAD_MÁXIMA, CRÉDITOS,

VEHICULO_1,

VEHICULO_2,

…

Y el formato para cada vehículo es:

NOMBRE, PRECIO, TIPO, CANTIDAD_RUEDAS, CAPACIDAD_GASOLINA

Algunas aclaraciones:

1. La implementación debe utilizar una de las dos implementaciones del **TDA Vector**, justificando la elección en base a algún criterio (temporal, espacial, escalabilidad, etc.)
2. **No se puede utilizar** ninguna de las estructuras de datos de la biblioteca *java.util* (en particular, no se puede usar *java.util.ArrayList*).

Estructuras de datos a implementar

Se deberá implementar el **TDA VectorEstatico** y el **TDA Vector** (dinámico). Dentro del repositorio base del TP encontrarán los métodos a implementar, junto con la documentación y el resultado esperado para cada una de las operaciones.

Algunas aclaraciones:

1. Se podrán agregar métodos y atributos a estas clases, **pero no se podrán modificar los métodos ya declarados**.
2. El **TDA Vector** (dinámico) **deberá implementar correctamente la redimensión** utilizando incrementos/decrementos de 1 o *factor-of-two* (multiplicar/dividir por 2). Implementar este TDA con un tamaño fijo arbitrario (constante) **DESAPRUEBA** automáticamente el TP.
3. Ambas implementaciones **deben mantener el orden** de los datos (ver los ejemplos de la documentación y los casos de prueba).

Programa principal

Para demostrar la funcionalidad de su código, se deberá implementar un programa principal sencillo que permita probar a mano el garaje. Se espera que, para cada una de las funcionalidades del garaje, se provea una opción para que el usuario elija, incluyendo la salida del programa.

También se espera que el programa, en todo momento, sea robusto (siga ejecutando aún si hay errores), y muestre de forma completa y clara toda la información que pueda ser relevante para el usuario, **incluyendo mensajes descriptivos en casos de error**.

La interfaz visual del programa es **libre**, mientras que sea interactiva, fácil de usar, y descriptiva.

Criterios de corrección

Para que el trabajo tenga buena recepción, se deberán cumplir las siguientes pautas:

1. Debe implementar las clases de manera totalmente original y propia. La detección de copias resultará en la **pérdida de la regularidad de los involucrados**.
2. El código **debe pasar** las pruebas automatizadas brindadas por la cátedra. Los trabajos que no ejecuten todas las pruebas correctamente **quedan desaprobados de forma automática**.
3. Las pruebas automatizadas **no pueden ser alteradas** de ninguna forma.
4. El código debe estar escrito en **camelCase**. Los nombres de las clases se escriben en **PascalCase** (todas las palabras con mayúscula). Los nombres de los archivos fuente deben tener el mismo nombre que la clase. Las constantes se escriben en **mayúsculas** y en **snake_case** (por ejemplo, UNA_CONSTANTE).
5. Es altamente recomendable utilizar un *formatter*. Pueden utilizar el estilo de código de Google para Java.

Los criterios de evaluación y corrección por parte de la cátedra son:

1. **Funcionalidad**:
    a. El programa debe pasar las pruebas automatizadas.
    b. El programa debe funcionar correctamente y sin errores durante la ejecución.
2. **Estilo de código**:
    a. **Uso del paradigma:** se debe aplicar **POO**, creando las clases que sean necesarias.
    b. **Eficiencia espacial**: el uso de memoria debe ser apropiado, tanto en términos de variables/atributos como de *tamaño en bytes* de los tipos de datos utilizados.
    c. **Eficiencia temporal**: la complejidad de sus métodos debe ser apropiada, considerando las estructuras a utilizar (**O(n)** como máximo, para este trabajo).
    d. **Modularización**: tanto de código en métodos como de responsabilidades en clases.
    e. **Documentación**: se deberá documentar claramente las pre/postcondiciones de sus métodos. Pueden usar *Javadoc*.
    f. **Uso de archivos**: la escritura/lectura de archivos debe ser apropiada, minimizando las aperturas y cerrando correctamente los archivos.
    g. **Buenas prácticas** de programación propuestas por la cátedra.

Formato de entrega

Para realizar la entrega, deben seguir los siguientes pasos:

1. Subir el código **fuente** de la entrega a la branch *main* del repositorio privado de GitHub. **No** se deben subir los archivos de configuración del IDE.
2. Generar un *release* sobre el commit que quieran entregar. En el *tag* del *release*, completar con el padrón. En el título, completar con “Entrega”.
3. Descargar el archivo .zip del *release*. Este archivo tendrá el formato **tpi-2c2025-USERNAME-PADRON.zip** si completaron correctamente el *tag*.
4. Subir el archivo .zip al campus. Solo se puede hacer **una entrega** en el campus.

Se corregirá el commit realizado sobre la branch *main*. El archivo .zip entregado en el campus solo será revisado ante casos excepcionales.

El plazo de entrega vence el día **LUNES 22 DE SEPTIEMBRE** a las **23:59 hrs**. No se aceptarán entregas fuera de término.

Repositorio base del TP

Tendrán a disposición un repositorio base preconfigurado para el desarrollo y entrega del TP, donde encontrarán las pruebas automatizadas que validan el funcionamiento de sus TDAs. Estas pruebas podrán ejecutarlas de forma local con **JUnit** antes de entregar para verificar que cumplen con los requisitos de entrega.

Para generar un repositorio, deben ingresar en el siguiente enlace:

Enlace al repositorio base del TP

Es recomendable, previo a empezar a trabajar, leer la siguiente guía para los TPs.