Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-2

24 de Agosto de 2023
Actividad

Actividad 1

Programación Orientada a Objetos II

Entrega

- **Lugar:** En Canvas - **Actividad 1**
- **Fecha máxima de entrega:** 28 de agosto 20:00

Introducción

Desde el departamento de tránsito, te han pedido empezar a hacer las clases que se utilizarán en una simulación para analizar los gastos energéticos y la contaminación asociada a los distintos tipos de vehículos.

Archivos

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- Entregar Modificar `clases.py`: Aquí encontrarás la definición básica de las clases que debes implementar y completas. Finalmente, debes subir este archivo a Canvas.
- No modificar `test.py`: Este archivo contiene los tests que podrás utilizar pasa ir viendo si lo desarrollado hasta el momento cumple con lo pedido por el departamento de tránsito.

Modelo de datos

Para reflejar y hacer la simulación, te han entregado una serie de especificaciones técnicas que deberás implementar en tu modelo de datos. Lo que te entregan es lo siguiente:

Habrán distintos tipos de vehículos, y todos estos tendrán un rendimiento de kilómetros por cantidad de energía consumida. Además de esto, pueden ser de distintas marcas, todos poseen un identificador único que debe ser calculado en tiempo de ejecución de forma automática y una energía disponible, que por razones físicas no puede ser menor a cero.

Los autos tendrán un tipo de consumo, estos pueden ser a bencina o eléctricos. Por un lado, los autos a bencina poseen una bencina favorita (puede ser 93, 95 o 97), y tienen una forma particular de recorrer kilómetros.

Por otro lado, los autos eléctricos poseen una vida útil de la batería, y también poseen su propia forma de recorrer.

Actualmente solo se evaluarán 3 tipos de autos distintos: Una camioneta genérica, un Telsa y un FaitHibrido. Las camionetas además de ser autos a bencina, poseen una capacidad de maleta. Por otro lado, el Telsa es un auto eléctrico, pero que al momento de recorrer lo hace de forma inteligente, pues tiene conducción automática. Finalmente, el FaitHibrido es un auto que tiene un comportamiento especial, pues es un auto a bencina y eléctrico a la vez, por lo que posee las características de ambos autos, y tiene una forma de recorrer especial.

Llevando modelo al código

- Modificar `class Vehiculo`:
  Clase abstracta que representa un vehículo. Posee la variable de clase `identificador`, y además incluye los siguientes métodos:

  - Modificar `def __init__(self, rendimiento, marca, energia, *args, **kwargs) -> None`:
    Este es el inicializador de la clase, y debe asignar los siguientes atributos:

`self.rendimiento`	Un `int` que representa los km que puede andar un vehículo por unidad de energía utilizada (ya sea bencina, electricidad o híbrido).
`self.marca`	Un `str` que representa la marca del vehículo.
`self._energia`	Un `int` que representa la cantidad de kilómetros que podrá recorrer el vehículo antes de tener que volver a rellenar. Es una atributo privado que tiene un valor de 120 por defecto, y cuyo valor no puede bajar de 0.
`self.identificador`	Un `int` que sirve para identificar el vehículo. Para setearlo, debe fijarlo como el valor actual de la variable de clase `Vehiculo.identificador`, y luego se le debe sumar 1 a la misma variable. De esta forma, se logra que todos los vehículos tengan un identificador diferente.

  - Modificar `def recorrer(self, kilometros) -> None`:
    Método abstracto, que obliga a las clases hijas a que este método sea implementado.

  - Modificar `def autonomia(self) -> float`:
    Property encargada de informar la autonomía. Retorna la cantidad de kilómetros que puede andar el vehículo, resultante de calcular la energía disponible multiplicada por el rendimiento.

  - Modificar `def energia(self) -> int`:
    Property encargada de manejar la energía disponible. El *getter* debe retornar el valor del atributo privado de energía, y el *setter* debe encargarse de que el atributo privado no pueda tener un valor menor a 0, es decir, en caso de recibir uno menor a 0, este debe ser acotado a 0.

- Modificar `class AutoBencina`:
  Clase utilizada para representar vehículos que utilizan bencina como combustible. Hereda de `Vehiculo`, y posee los siguientes métodos:

  - Modificar `def __init__(self, bencina_favorita, *args, **kwargs) -> None`:
    Este es el inicializador de la clase. Además de llamar al método de la clase padre, debe asignar los siguientes atributo:

`self.bencina_favorita`	Un `int` que representa el octanaje de la bencina de preferencia.

  - Modificar `def recorrer(self, kilometros) -> str`:
    Método encargado de simular el desplazamiento del vehículo. Si el vehículo puede andar el total de los kilómetros pedidos, entonces recorre dicha cantidad de kilómetros. En caso de no poder, recorre solo la autonomía. Además, debe restar a la energia disponible `kilometros/self.rendimiento` a la energía actual del vehículo y retorna un texto diciendo `"Anduve {numero}Km y gasté {L}L de bencina."` informando la cantidad de kilómetros que logró recorrer y los litros de bencina utilizados, siendo L un `int`.

- Modificar `class AutoElectrico`:
  Clase utilizada para representar autos eléctricos. Hereda de `Vehiculo`, y posee los siguientes métodos:

  - Modificar `def __init__(self, vida_util_bateria, *args, **kwargs) -> None`:
    Inicializador de la clase. Además de llamar al método de la clase padre, debe setear el siguiente atributo.

`self.vida_util_bateria`	Un `int` que representa los años que le quedan de vida util a la batería.

  - Modificar `def recorrer(self, kilometros) -> str`:
    Método encargado de simular el desplazamiento del vehículo. Si el vehículo puede andar el total de los kilómetros pedidos, entonces recorre dicha cantidad de kilómetros. En caso de no poder, recorre solo la autonomía. Además, debe restar a la energía disponible `kilometros/self.rendimiento` a la energía actual del vehículo. retorna el texto `"Anduve por {N} kilometros y gasté {K} W..."` informando la cantidad de kilómetros que logró recorrer y las unidades de energía en W que consumió utilizados.

- Modificar `class Camioneta`:
  Clase que representa una camioneta. Debe heredar de la clase `AutoBencinca`:

  - Modificar `def __init__(self, capacidad_maleta, *args, **kwargs) -> None`:
    Inicializador de la clase. Además de llamar al método de la clase padre, debe setear el siguiente atributo.

`self.capacidad_maleta`	Un `int` que representa la capacidad de la maleta.

- Modificar `class FaitHibrido`:
  Clase que representa un auto eléctrico. Debe heredar de las clases `AutoBencina` y `AutoElectrico`:

  - Modificar `def __init__(self, *args, **kwargs) -> None`:
    Debe llamar al método de la clase padre. Además de esto, todos los FaitHibrido poseen una vida útil de la batería de 5 años.

  - Modificar `def recorrer(self, kilometros) -> str`:
    Método que simula el recorrido del auto. De los kilometros recibidos, una mitad las recorre como auto a bencina, y otra mitad las recorre como auto eléctrico. Debe retornar la concatenación de ambos textos de recorrer de las clases padre.

- Modificar `class Telsa`:
  Clase que representa un auto eléctrico. Debe heredar de la clase `AutoElectrico` y posee los siguientes métodos

  - Modificar `def recorrer(self, kilometros) -> str`:
    Debe llamar al método de la clase padre para recorrer, pero debe retornar el texto retornado por la clase padre agregándo el texto `"de forma inteligente"` al final.

Notas

- Recuerda que la ubicación de tu entrega es en **Canvas**. Puedes utilizar *git* para tener un respaldo de tu trabajo, pero finalmente se revisará lo subido al buzón de Canvas.
- Se recomienda completar la actividad en el orden del enunciado.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.

Objetivos de la actividad

- Implementar clases siguiendo especificaciones técnicas.
- Hacer uso de clases y métodos abstractos para una correcta modelación.
- Manejar concepto de herencia y multiherencia correctamente.
- Hacer correcto uso de *properties*.
- Utilizar métodos de clases padres correctamente.
- Resolver bien el problema del diamante.
- Probar código mediante la ejecución de *test*.