Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2024-1

4 de Abril de 2024
Actividad

Actividad 1

Programación Orientada a Objetos Avanzado

Entrega

- **Lugar:** Repositorio personal de GitHub — Carpeta: `Actividades/AC1`
  El código debe estar en la rama (*branch*) por defecto del repositorio: `main`.
- **Fecha máxima de entrega:** 8 de Abril 20:00

Introducción

Desde el departamento de tránsito, liderado por el famoso Cristiano Ruznaldo, te han pedido empezar a hacer las clases que se utilizarán en una simulación que analiza los gastos energéticos y la contaminación asociada a 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 completar. Al finalizar la actividad, debes asegurar que este archivo estar subido en el lugar de entrega correspondiente.
- No modificar `test.py`: Archivo ejecutable que contiene los *tests*. Deberás utilizarlo para 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.

Habrán distintos tipos de vehículos de distintas marcas, y todos estos tendrán un rendimiento de kilómetros por cantidad de energía consumida. Además de esto, 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.

La energía de los autos de divide en dos tipos de consumos, estos pueden ser a bencina (L) o eléctricos (W). Por un lado, los autos a bencina poseen una bencina favorita, 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 kilómetros 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 un 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 calcular el gasto de este movimiento restando a la energía disponible `kilometros/self.rendimiento`. Finalmente el método retorna el texto `"Anduve por {numero}Km y gaste {L}L de bencina."` informando la cantidad de kilómetros que logró recorrer y las unidades de energía en litros L de bencina utilizados en `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 calcular el gasto de este movimiento restando a la energía disponible `kilometros/self.rendimiento`. Retorna el texto `"Anduve por {N}Km y gaste {K}W de energia electrica"` informando la cantidad de kilómetros que logró recorrer y las unidades de energía en W utilizados.

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

  - 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 Telsa`:
  Clase que representa un auto eléctrico. Debe heredar de la clase `AutoElectrico`, con un cambio en el siguiente método:

  - Modificar `def recorrer(self, kilometros) -> str`:
    Debe llamar al método de la clase padre para recorrer, pero debe modificar y retornar el texto retornado por la clase padre agregando el texto `"de forma inteligente"` al final, y considerando un espacio entre ellos.

- Modificar `class FaitHibrido`:
  Clase que representa un auto híbrido. 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, la primera mitad las recorre como auto a bencina, y la segunda mitad las recorre como auto eléctrico. Debe retornar la concatenación de ambos textos del método recorrer de las clases padre, considerando un espacio entre ellos.

Notas

- **No puedes hacer *import* de otras librerías externas a las entregadas en el archivo.**
- Recuerda que la ubicación de tu entrega es en **tu repositorio de Git**. En la rama (*branch*) por defecto del repositorio: `main`.
- Se recomienda completar la actividad en el orden del enunciado.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- 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*.

Ejecución de *tests*

En esta actividad se provee de `test.py` que contiene diferentes *tests* que ayudan a validar el desarrollo de la actividad. Para ejecutar este archivo, desde la terminal/consola debes escribir `python3 test.py` y se ejecutarán todos los *tests* de la actividad.

En cambio, si deseas ejecutar un subconjunto de *tests*, puedes ejecutar lo siguiente en la terminal/consola:

- `python3 -m unittest test.VerificarClaseVehiculo`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `Vehiculo`.
- `python3 -m unittest test.VerificarClaseAutoBencina`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `AutoBencina`.
- `python3 -m unittest test.VerificarClaseAutoElectrico`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `AutoElectrico`.
- `python3 -m unittest test.VerificarClaseCamioneta`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `Camioneta`.
- `python3 -m unittest test.VerificarClaseTelsa`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `Telsa`.
- `python3 -m unittest test.VerificarClaseFaitHibrido`: para ejecutar solo el subconjunto de *tests* relacionado a la clase `FaitHibrido`.

Importante: recuerda que si `python3` no funciona, probar con el comando específico de tu computador. Este puede ser `py`, `py3` o `python`.