Continuando con lo que realizamos en el laboratorio, creamos una derivación de la clase **AutoElectrico**. A esta clase le añadí variables adicionales, como cicloDeCarga y tiempoDeCarga. También reutilicé la lógica que implementé en la clase **Vehiculo** para verificar si el auto está apagado o encendido. El primer punto que realicé fue...

1. **Derivación de clases**: Crear subclases como AutoDeCombustion, Motocicleta y Camión basadas en una clase base que es Vehículo es una manera genial de utilizar la herencia y nos ayuda a reutilizar el código de forma más efectiva.

2. **Heredé el constructor y lo utilicé en todas las demás clases** Para facilitar la inicialización de las clases derivadas, utilicé el constructor de la clase base Vehiculo. Esto permitió que los atributos comunes, como Year, Color y Modelo, fueran configurados de manera consistente desde la clase base, evitando duplicar lógica en cada subclase.

3. **Modificador** protected: Usar protected para las variables de velocidad y encendido es una decisión muy buena, ya que garantiza que solo las clases que están en la jerarquía puedan acceder a ellas. Además, personalizar el método Frenar en la clase Motocicleta le aporta un toque personal, ideal para representar comportamientos específicos de cada vehículo.

4. **Encapsulación en AutoDeCombustion**: Hacer que variables como capacidadEntanque, cantidadCombustible y nivelAceite sean privadas es una práctica realmente acertada en programación orientada a objetos. Esto ayuda a proteger los datos, pudiendo acceder a ellos únicamente a través de métodos o propiedades apropiadas. También, mejorar el método Frenar para que esté relacionado con el consumo de gasolina es una gran manera de simular el comportamiento real de un vehículo.

5. **Lógica personalizada en Motocicleta**: Sobrescribir el método Acelerar para que la velocidad sea el doble que la de un auto muestra un diseño excelente para diferenciar entre vehículos. Además, reutilizar la lógica del método Frenar es una estrategia inteligente para mantener el código claro y eficiente.

6. **Encapsulación en Camión**: Hacer que la variable horasUsoMotor sea privada en esta clase tiene sentido, ya que es un atributo que pertenece específicamente a los camiones. Es evidente que estás aplicando buenas prácticas de encapsulación de manera constante.

Definición mas clara

SUBCLASE AUTOELECTRICO

Punto 1. En la subclase herede el constructor de la base vehículo.

Punto 2. Encapsule la propiedad protected int velocidad ; en la clase base para poder utilizarla en las demás clases.

Punto 3. Mande a llamar el método acelerar para poder sobrescribirlo para poder disminuir la carga dependiendo si acelera.

Punto 4. Llamé la lógica de MostrarEstadoEncendido() de la clase base Vehiculo para revisar si el auto está apagado o encendido.

punto 5: Llamé la lógica de apagar y encender de la clase base para integrarla en la lógica de la subclase.

Punto 6. De igual forma encapsulé 3 variables de tipo int en la clase correspondiente.

SUBCLASE AUTODECOMBUSTION

Paso 1. En esta subclase herede el constructor de la clase base Vehículo.

Punto 2. Cree 3 encapsulamientos de tipo int y le asigne su respectiva cantidad.

Punto 3. Llamé el método frenar para sobrescribirlo utilizando la lógica de la clase base y añadí que consuma combustible al frenar agregándole cantidadCombustible -= 1;

Punto 4. Método llamado NivelAceite() permite obtener el valor actual de la variable privada nivelAceite.

Paso 5. El método llamado CantidadCombustible() permite obtener el valor actual de la variable privada cantidadCombustible.

SUBCLASE MOTOCICLETA

Paso 1. Heredar el constructor de la clase base Vehículo.

Punto 2. Creé 4 encapsulamientos de tipo int y una de tipo string en la subclase Motocicleta.

Punto 3 Llamé la lógica de acelerar () para sobrescribirla y agregar que la motocicleta pueda acelerar al doble en velocidad.

Punto 4. Con el método Marca() devuelve el valor actual de la variable privada marca.

Punto 5. sobrescribí el método apagar(); utiliza la lógica de la clase base Vehiculo para mantener el comportamiento estándar de apagar la motocicleta.

Paso 6. Sobrescribir el método de frenar (); la lógica del de la clase base Vehículo para mantener el comportamiento estándar de frenar la motocicleta;

SUBCLASE CAMION

Punto 1. Heredar el constructor de la clase base Vehículo.

Punto 2. Crearon 3 encapsulamientos de tipo int de igual forma le asigne su respectiva cantidad.

Punto 3. Implementé el método registrarHorasMotor para actualizar las horas de uso del motor de manera controlada.

Punto 4. Implementé el método cargar para actualizar de manera controlada la cantidad de carga actual del vehículo en esto me refiero a que la cantidad que el camión esta transportando .

Punto 5. Implementé el método verificarFrenos para evaluar el estado de los frenos del vehículo.

Paso 6. En esta subclase también Sobrescribí el método frenar utilizando la lógica de la clase base y extendí su funcionalidad en la subclase.