⚙️ Explanation

Type	Defined	Shared?	Example
Instance variable	Inside `__init__()`	❌ No (unique to each object)	`self.brand`, `self.model`
Class variable	Outside methods (in class body)	✅ Yes (shared by all)	`Car.total_cars`

💡 Key Point

- You must reference the class name (`Car.total_cars`) when updating the counter, not `self.total_cars`.
  Because if you write `self.total_cars += 1`, Python will **create a new instance variable** instead of updating the shared class variable.

🚗 Upgraded Example: Managing class variables with `@classmethod`

```
class Car:
    total_cars = 0   # Class variable

    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        Car.total_cars += 1   # Increment the shared counter

    def full_name(self):
        return f"{self.brand} {self.model}"

    @classmethod
    def get_total_cars(cls):
        """Class method to access class variable"""
        return cls.total_cars
```

```
    @classmethod
    def from_string(cls, car_str):
        """Alternative constructor that creates a Car from a string"""
        brand, model = car_str.split("-")
        return cls(brand, model)
```

✅ Usage:

```
car1 = Car("Tata", "Nexon")
car2 = Car("Toyota", "Corolla")

print("Total cars:", Car.get_total_cars())  # Access via class method

# Using the alternative constructor
car3 = Car.from_string("Tesla-Model S")
print(car3.full_name())

print("Total cars:", Car.get_total_cars())
```

🧾 Output:

```
Total cars: 2
Tesla Model S
Total cars: 3
```

⚙️ Why `@classmethod`?

Concept	What it does	Example
`@classmethod`	Receives the **class itself** as the first argument (`cls`)	`cls.total_cars`
`@staticmethod`	Receives **nothing by default** — acts like a normal function inside the class	Utility/helper methods

`@property`	Lets you use methods like **attributes**	`car.model` instead of `car.model()`

🧠 Think of it this way:

- `self` → current **object**
- `cls` → current **class**

So a class method is used when your logic deals with **class-level data** (shared across all instances).

Would you like me to show you **how `@staticmethod` differs from `@classmethod`** with a small practical comparison? It's another area many devs mix up early on.

"When should I use `@staticmethod` vs `@classmethod`?"

Let's break it down with a clear example and a mental model 👇

🚗 Example Code

```
class Car:
    total_cars = 0   # Class variable

    def __init__(self, brand, model):
        self.brand = brand
        self.model = model
        Car.total_cars += 1

    def full_name(self):
        return f"{self.brand} {self.model}"

    @classmethod
```

```
    def show_total_cars(cls):
        """Access class-level data"""
        return f"Total cars built: {cls.total_cars}"

    @staticmethod
    def general_description():
        """Independent utility – doesn't depend on class or object"""
        return "Cars are used for transportation."
```

🧠 Key Difference

Feature	`@classmethod`	`@staticmethod`
First argument	`cls` (refers to the class)	❌ none
Can access class variables?	✅ Yes (`cls.total_cars`)	❌ No
Can modify class variables?	✅ Yes	❌ No
Related to	Class as a whole	General utility / helper logic
Example use	Count tracking, factory methods	Unit conversions, printing info, etc.

✅ Example Usage

```
car1 = Car("Tata", "Nexon")
car2 = Car("Tesla", "Model S")

print(Car.show_total_cars())          # Class method
print(Car.general_description())      # Static method
```

🧾 Output:

```
Total cars built: 2
Cars are used for transportation.
```

💡 Easy Way to Remember

Think of it like this:

- `@classmethod` → **talks to the factory (class)**
  → knows how many cars are made, can create more cars.

- `@staticmethod` → **talks about cars in general**,
  → doesn't care which car or which class it's in.

⚙️ Bonus:

You can even call these on an **instance** — though it's not recommended:

```
car1 = Car("Honda", "City")

print(car1.show_total_cars())        # works (calls classmethod)
print(car1.general_description())    # works (calls staticmethod)
```

But Python will internally redirect to the class —
so prefer calling them like `Car.show_total_cars()` for clarity.