Python Composition

Table of contents

Composition is a way of building a class out of other objects: an object of one class holds an object of another class as one of its attributes. This forms a **has-a** relationship (e.g. a `Car` **has-a** `Engine`) — as opposed to inheritance's **is-a** relationship (e.g. a `SportsCar` **is-a** `Car`).

Basic pattern

```
class Car():
    def __init__(self, engine: Engine) -> None:
        self._engine = engine   # Car has-a engine

    def start(self):
        self._engine.start()
```

The outer object (`Car`) doesn't need to know *how* the inner object (`Engine`) works internally — it just calls the inner object's public methods. This keeps each class focused on a single responsibility.

Worked example: `Car`/`Engine`

```
class Engine:
    def __init__(self, horsepower, fuel_type="gasoline"):
        self.horsepower = horsepower
        self.fuel_type = fuel_type
        self.running = False
```

```
    def start(self):
        if not self.running:
            self.running = True
            print(f"{self.fuel_type.capitalize()} engine with {self.horsepower} HP started.")
        else:
            print("Engine is already running.")

    def stop(self):
        if self.running:
            self.running = False
            print("Engine stopped.")
        else:
            print("Engine is already off.")


class Car:
    def __init__(self, make, model, engine: Engine):
        self.make = make
        self.model = model
        self.engine = engine

    def start(self):
        print(f"Starting the {self.make} {self.model}...")
        self.engine.start()

    def stop(self):
        print(f"Stopping the {self.make} {self.model}...")
        self.engine.stop()

    def swap_engine(self, new_engine: Engine):
        if self.engine.running:
            print("Stopping current engine before swap...")
            self.engine.stop()
        print(f"Swapping engine in {self.make} {self.model}...")
        self.engine = new_engine
        print("New engine installed!")

>>> small_engine = Engine(150, "gasoline")
>>> car = Car("Toyota", "Corolla", small_engine)
>>> car.start()
Starting the Toyota Corolla...
```

```
Gasoline engine with 150 HP started.

>>> big_engine = Engine(300, "diesel")
>>> car.swap_engine(big_engine)   # safely stops the old engine first, if running
```

Composition vs. inheritance

- Use **composition** ("has-a") when an object is *made up of* other objects, or *uses* another object to do part of its job.
- Use **inheritance** ("is-a") when a new class is a more specific *version* of an existing class.

Composition tends to be more flexible: a composed attribute can be freely swapped for any object supporting the same interface (see *duck typing*), without needing any shared base class.