Python -pytest

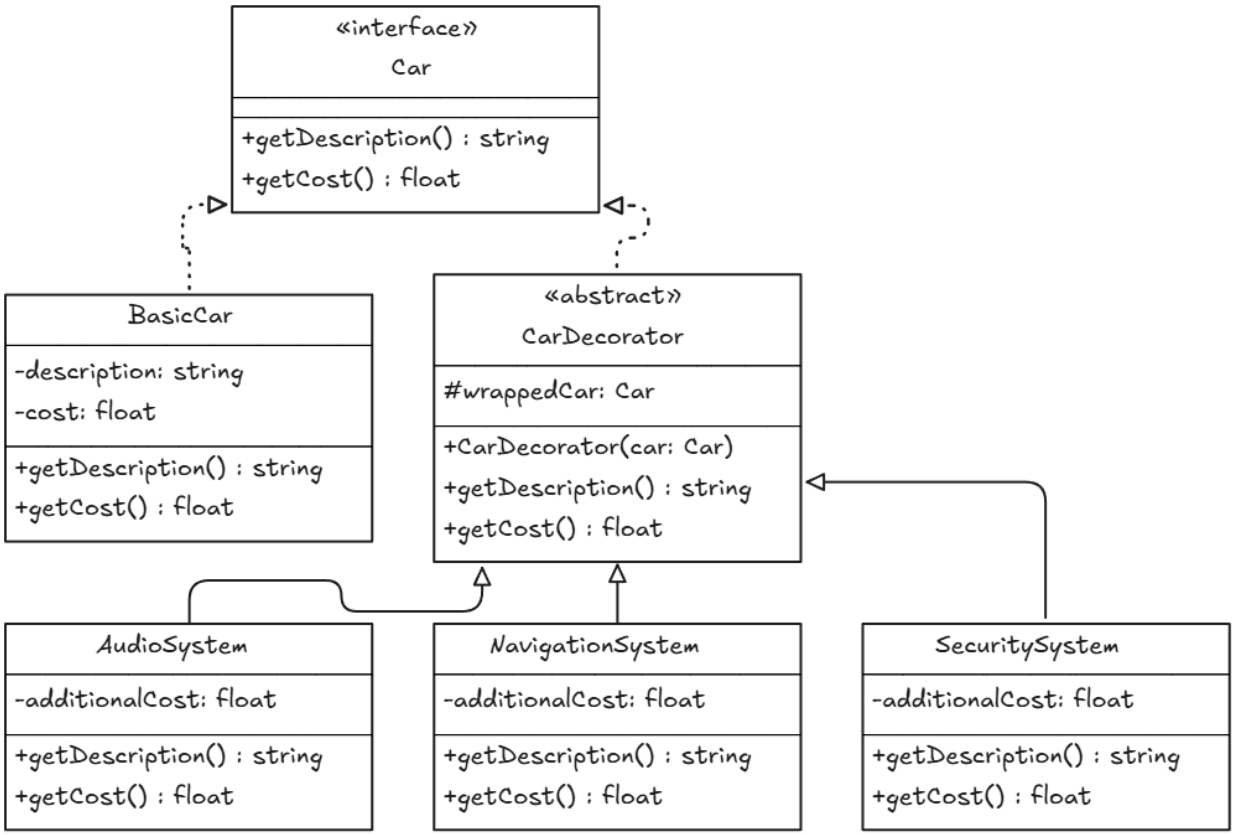
Prompt

-Decorator

Use the Decorator pattern to write code in Python programming language and write pytest to test the given code that have 100% statement coverage from the requirements below. Use Decorator pattern to design automotive system development by adding features to cars, such as audio system, navigation system or security systems without changing the main structure of the car.

Diagram

Decorator



Gemini 1.5 Pro

Decorator

รอบที่ 1

Code.py

```
# car.py

class Car:
    """Represents a basic car."""

    def __init__(self, model, price):
        self.model = model
        self.price = price

    def description(self):
        return f"This is a {self.model} car."



# decorators.py
def audio_system(car):
    """Decorator to add an audio system to a car."""

    class AudioEquippedCar:
        def __init__(self):
            self.car = car
            self.audio_price = 1000

        def description(self):
            return self.car.description() + " It has an audio system."

        @property
        def price(self):
            return self.car.price + self.audio_price
```

```
    return AudioEquippedCar()


def navigation_system(car):
    """Decorator to add a navigation system to a car."""

    class NavEquippedCar:
        def __init__(self):
            self.car = car
            self.nav_price = 1500

        def description(self):
            return self.car.description() + " It has a navigation system."

        @property
        def price(self):
            return self.car.price + self.nav_price

    return NavEquippedCar()


def security_system(car):
    """Decorator to add a security system to a car."""

    class SecurityEquippedCar:
        def __init__(self):
            self.car = car
            self.security_price = 2000

        def description(self):
            return self.car.description() + " It has a security system."

        @property
```

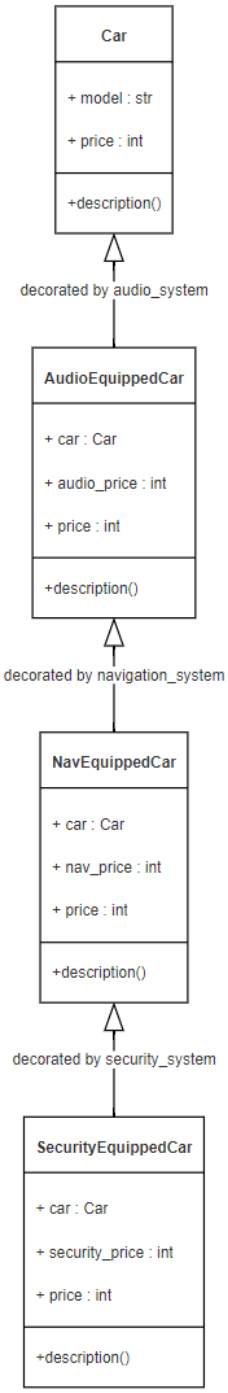
```
        def price(self):
            return self.car.price + self.security_price


    return SecurityEquippedCar()
```

Test.py

```
# test_car.py

from code import Car
from code import audio_system, navigation_system, security_system



def test_base_car():
    basic_car = Car("Basic Model", 10000)
    assert basic_car.description() == "This is a Basic Model car."
    assert basic_car.price == 10000



def test_audio_decorator():
    car_with_audio = audio_system(Car("Audio Model", 10000))
    assert car_with_audio.description() == "This is a Audio Model car. It has an audio system."
    assert car_with_audio.price == 11000



def test_navigation_decorator():
    car_with_nav = navigation_system(Car("Nav Model", 10000))
    assert car_with_nav.description() == "This is a Nav Model car. It has a navigation system."
    assert car_with_nav.price == 11500



def test_security_decorator():
    car_with_security = security_system(Car("Security Model", 10000))
    assert (
```

```
        car_with_security.description() == "This is a Security Model car. It has a security system."
    )
    assert car_with_security.price == 12000



def test_combined_decorators():
    fully_loaded_car = security_system(
        navigation_system(audio_system(Car("Fully Loaded", 10000)))
    )
    assert (
        fully_loaded_car.description()
        == "This is a Fully Loaded car. It has an audio system. It has a navigation system. It has a security
system."
    )
    assert fully_loaded_car.price == 14500
```

Diagram From Code:



Result:

split class in code.py

TestDecorator.py(Edit)

```
from Car import Car
from Decorator import audio_system, navigation_system, security_system
```

5 Pass