Python -pytest

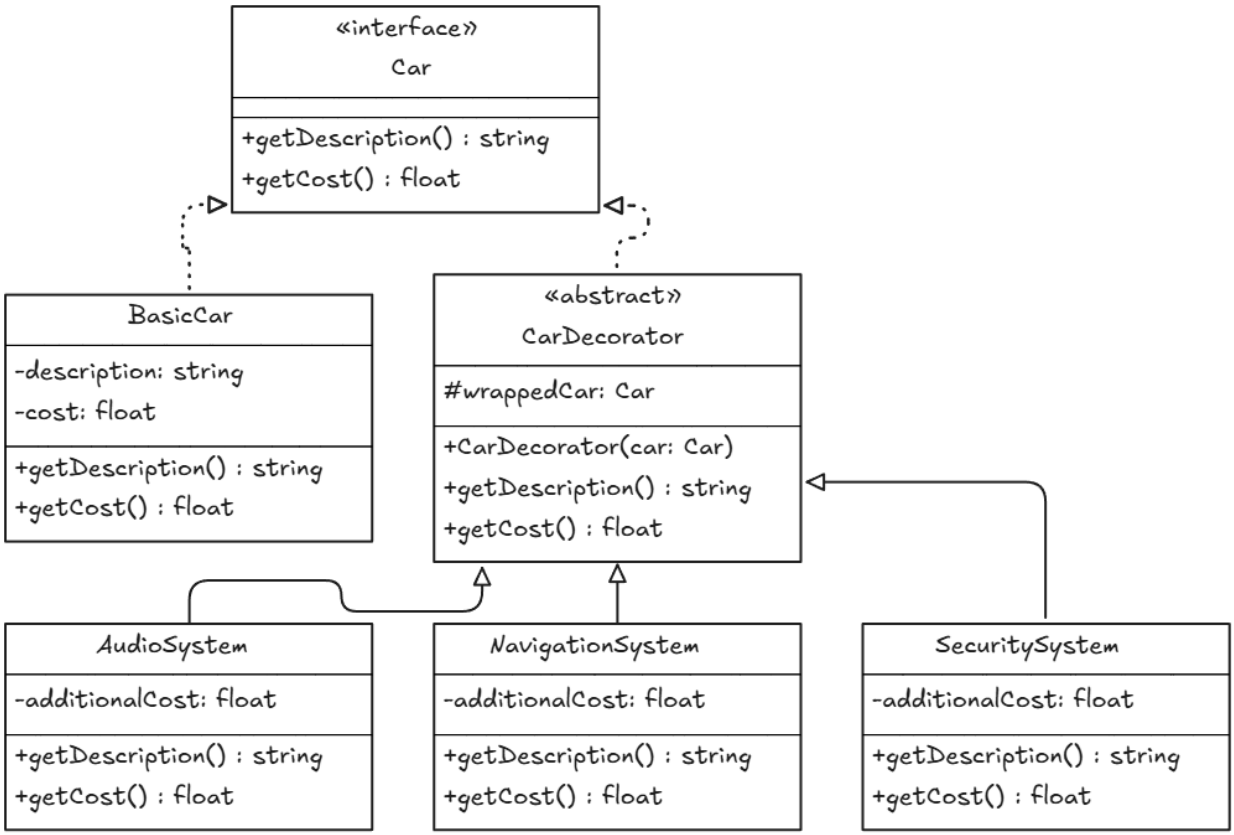
Prompt

-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 3

Code.py

```
# car.py

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

    def __init__(self, model, price):
        self.model = model
        self.price = price
        self.features = []  # Store added features

    def description(self):
        base_description = f"This is a {self.model} car."
        if self.features:
            base_description += " It has: " + ", ".join(self.features) + "."
        return base_description



# decorators.py
def add_feature(feature_name, price):
    """Decorator to add a feature to a car."""

    def decorator(car_function):
        def wrapper(self, *args, **kwargs):
            self.price += price
            self.features.append(feature_name)
            return car_function(self, *args, **kwargs)

        return wrapper
```

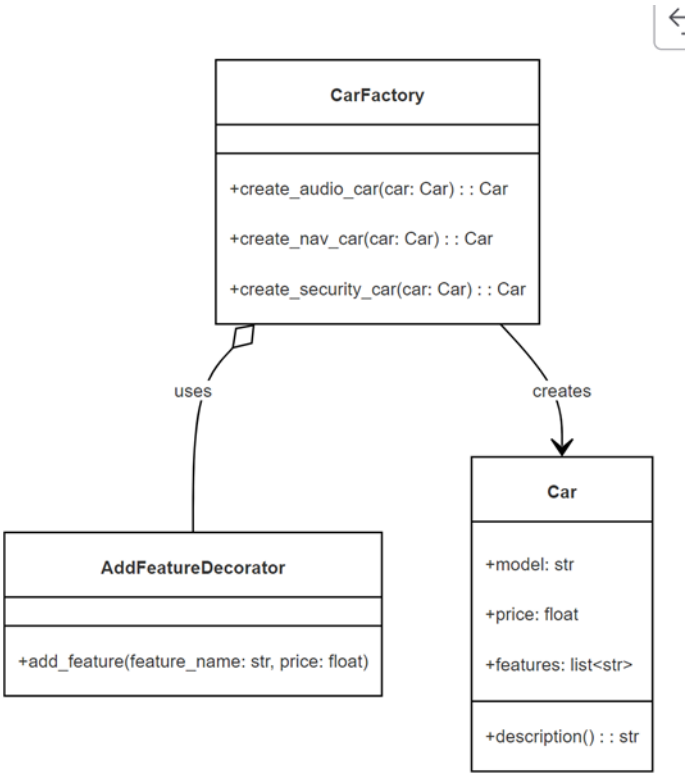
```
    return decorator


class CarFactory:
    """Factory class to create cars with features."""

    @add_feature("audio system", 1000)
    def create_audio_car(self, car):
        return car

    @add_feature("navigation system", 1500)
    def create_nav_car(self, car):
        return car

    @add_feature("security system", 2000)
    def create_security_car(self, car):
        return car
```

Test.py

```
# test_car.py

from code import Car, CarFactory



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_car():
    factory = CarFactory()
```

```
    audio_car = factory.create_audio_car(Car("Audio Model", 10000))
    assert audio_car.description() == "This is a Audio Model car. It has: audio system."
    assert audio_car.price == 11000



def test_nav_car():
    factory = CarFactory()
    nav_car = factory.create_nav_car(Car("Nav Model", 10000))
    assert nav_car.description() == "This is a Nav Model car. It has: navigation system."
    assert nav_car.price == 11500



def test_security_car():
    factory = CarFactory()
    security_car = factory.create_security_car(Car("Security Model", 10000))
    assert (
        security_car.description() == "This is a Security Model car. It has: security system."
    )
    assert security_car.price == 12000



def test_combined_features():
    factory = CarFactory()
    fully_loaded = factory.create_security_car(
        factory.create_nav_car(factory.create_audio_car(Car("Fully Loaded", 10000)))
    )
    assert (
        fully_loaded.description()
        == "This is a Fully Loaded car. It has: audio system, navigation system, security system."
    )
    assert fully_loaded.price == 14500
```

Diagram From Code:



Result:

split class in code.py

TestDecorator.py(Edit)

from Car import Car

from Decorator import CarFactory

1 Pass, 4 Fail