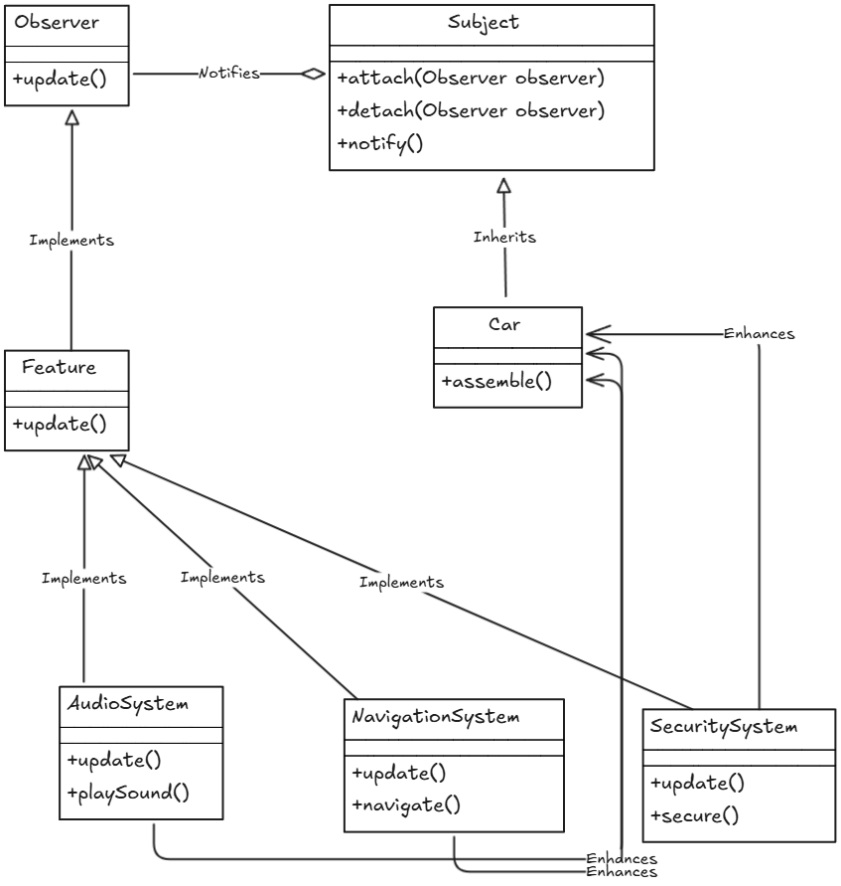
Python -pytest

Prompt

Observer

Use the Observer 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 Observer 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.

Observer



Gemini 1.5 Flash

Observer

รอบที่ 3

Code.py

```
class Subject:
    """Represents the car, which is the subject of observation."""

    def __init__(self):
        self._observers = []

    def attach(self, observer):
        """Attach an observer to the car."""
        self._observers.append(observer)

    def detach(self, observer):
        """Detach an observer from the car."""
        self._observers.remove(observer)

    def notify(self, event):
        """Notify all observers about the event."""
        for observer in self._observers:
            observer.update(event)



class Observer:
    """Abstract base class for observers."""

    def update(self, event):
        """Receive notification about an event."""
        raise NotImplementedError
```

```
class AudioSystem(Observer):
    """Represents an audio system as an observer."""

    def update(self, event):
        """Respond to car events related to audio."""
        if event == "start":
            print("Audio system: Playing welcome tune.")
        elif event == "stop":
            print("Audio system: Turning off.")
        elif event == "volume_up":
            print("Audio system: Increasing volume.")
        elif event == "volume_down":
            print("Audio system: Decreasing volume.")


class NavigationSystem(Observer):
    """Represents a navigation system as an observer."""

    def update(self, event):
        """Respond to car events related to navigation."""
        if event == "start":
            print("Navigation system: Starting navigation.")
        elif event == "destination":
            print("Navigation system: Setting new destination.")
        elif event == "route_guidance":
            print("Navigation system: Providing route guidance.")


class SecuritySystem(Observer):
    """Represents a security system as an observer."""

    def update(self, event):
        """Respond to car events related to security."""
        if event == "start":
```

```
            print("Security system: Disarming.")
        elif event == "stop":
            print("Security system: Arming.")
        elif event == "door_open":
            print("Security system: Door open alert.")
        elif event == "door_close":
            print("Security system: Door closed.")



# Example usage:
if __name__ == "__main__":
    car = Subject()

    audio_system = AudioSystem()
    navigation_system = NavigationSystem()
    security_system = SecuritySystem()

    car.attach(audio_system)
    car.attach(navigation_system)
    car.attach(security_system)

    print("Starting the car...")
    car.notify("start")

    print("Setting a destination...")
    car.notify("destination")

    print("Starting route guidance...")
    car.notify("route_guidance")

    print("Increasing volume...")
    car.notify("volume_up")

    print("Opening a door...")
```

```
    car.notify("door_open")

    print("Closing the door...")
    car.notify("door_close")

    print("Stopping the car...")
    car.notify("stop")
```

Test.py

```
import pytest
from code import Subject, AudioSystem, NavigationSystem, SecuritySystem



class MockObserver:
    """Mock observer for testing."""

    def __init__(self):
        self.received_events = []

    def update(self, event):
        self.received_events.append(event)



def test_attach_detach():
    """Test attaching and detaching observers."""
    car = Subject()
    observer = MockObserver()

    car.attach(observer)
    assert observer in car._observers

    car.detach(observer)
    assert observer not in car._observers
```

```
def test_notify():
    """Test notifying observers."""
    car = Subject()
    observer = MockObserver()
    car.attach(observer)

    car.notify("event_one")
    car.notify("event_two")

    assert observer.received_events == ["event_one", "event_two"]



def test_audio_system():
    """Test the audio system observer."""
    car = Subject()
    audio_system = AudioSystem()
    car.attach(audio_system)

    car.notify("start")  # Expect "Audio system: Playing welcome tune."
    car.notify("stop")  # Expect "Audio system: Turning off."
    car.notify("volume_up")  # Expect "Audio system: Increasing volume."
    car.notify("volume_down")  # Expect "Audio system: Decreasing volume."



def test_navigation_system():
    """Test the navigation system observer."""
    car = Subject()
    navigation_system = NavigationSystem()
    car.attach(navigation_system)

    car.notify("start")  # Expect "Navigation system: Starting navigation."
    car.notify("destination")  # Expect "Navigation system: Setting new destination."
```

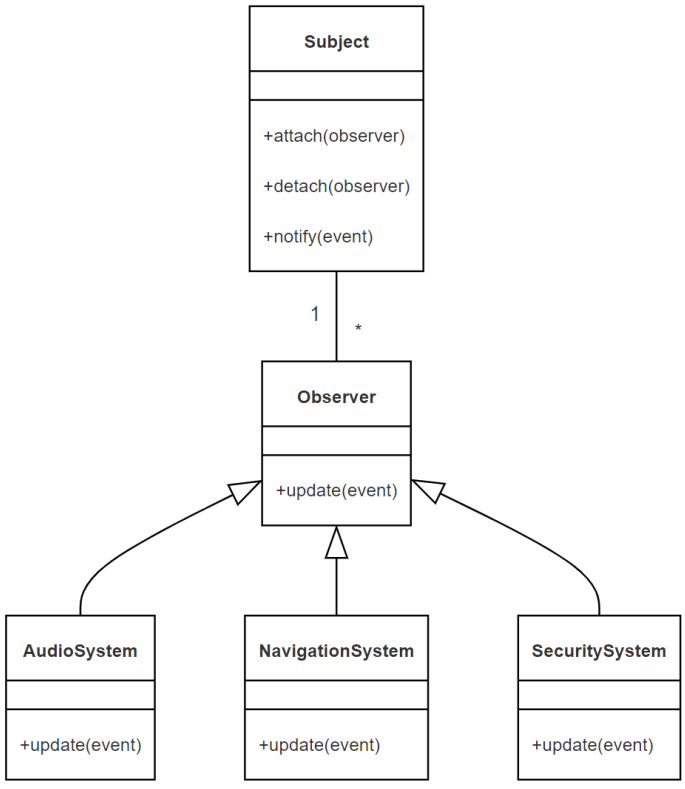
```
    car.notify("route_guidance")  # Expect "Navigation system: Providing route guidance."



def test_security_system():
    """Test the security system observer."""
    car = Subject()
    security_system = SecuritySystem()
    car.attach(security_system)

    car.notify("start")  # Expect "Security system: Disarming."
    car.notify("stop")  # Expect "Security system: Arming."
    car.notify("door_open")  # Expect "Security system: Door open alert."
    car.notify("door_close")  # Expect "Security system: Door closed."
```

Diagram From Code:



Result:

spilt class in code.py

TestObserver.py(Edit)

```
from Subject import Subject
from Observer import AudioSystem, NavigationSystem, SecuritySystem
```

5 Pass