# **Observer**

Позволяет объектам подписываться на события других объектов и получать уведомления о них.

from abc import ABC, abstractmethod

class Subject(ABC):

def \_\_init\_\_(self):

self.observers = []

def attach(self, observer):

self.observers.append(observer)

def detach(self, observer):

self.observers.remove(observer)

def notify(self, message):

for observer in self.observers:

observer.update(message)

class Observer(ABC):

@abstractmethod

def update(self, message):

pass

class ConcreteSubject(Subject):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.state = None

def getState(self):

return self.state

def setState(self, state):

self.state = state

self.notify(state)

class ConcreteObserver(Observer):

def \_\_init\_\_(self, subject):

subject.attach(self)

def update(self, message):

print(f"Observer got message: {message}")

subject = ConcreteSubject()

observer = ConcreteObserver(subject)

subject.setState("New state")

# Observer got message: New state

# **Strategy**

Определяет семейство алгоритмов и позволяет выбирать конкретный алгоритм в RUNTIME.

from abc import ABC, abstractmethod

class Strategy(ABC):

@abstractmethod

def execute(self, data):

pass

class ConcreteStrategyA(Strategy):

def execute(self, data):

print("Strategy A executing")

class ConcreteStrategyB(Strategy):

def execute(self, data):

print("Strategy B executing")

class Context:

def \_\_init\_\_(self, strategy):

self.strategy = strategy

def setStrategy(self, strategy):

self.strategy = strategy

def runStrategy(self, data):

self.strategy.execute(data)

strategy = ConcreteStrategyA()

context = Context(strategy)

context.runStrategy(123)

# Strategy A executing

context.setStrategy(ConcreteStrategyB())

context.runStrategy(123)

# Strategy B executing

# **State**

Позволяет объекту менять поведение в зависимости от состояния.

from abc import ABC, abstractmethod

class State(ABC):

@abstractmethod

def handle(self):

pass

class ConcreteStateA(State):

def handle(self):

print("State A handle")

class ConcreteStateB(State):

def handle(self):

print("State B handle")

class Context():

def \_\_init\_\_(self):

self.state = None

def setState(self, state):

self.state = state

def request(self):

self.state.handle()

context = Context()

stateA = ConcreteStateA()

stateB = ConcreteStateB()

context.setState(stateA)

context.request()

# State A handle

context.setState(stateB)

context.request()

# State B handle