Usman Institute of Technology

Department of Computer Science Fall 2022

Name: Muhammad Waleed

Roll no: 20B-115-SE

Course: Software Design and Architecture (SE-308)

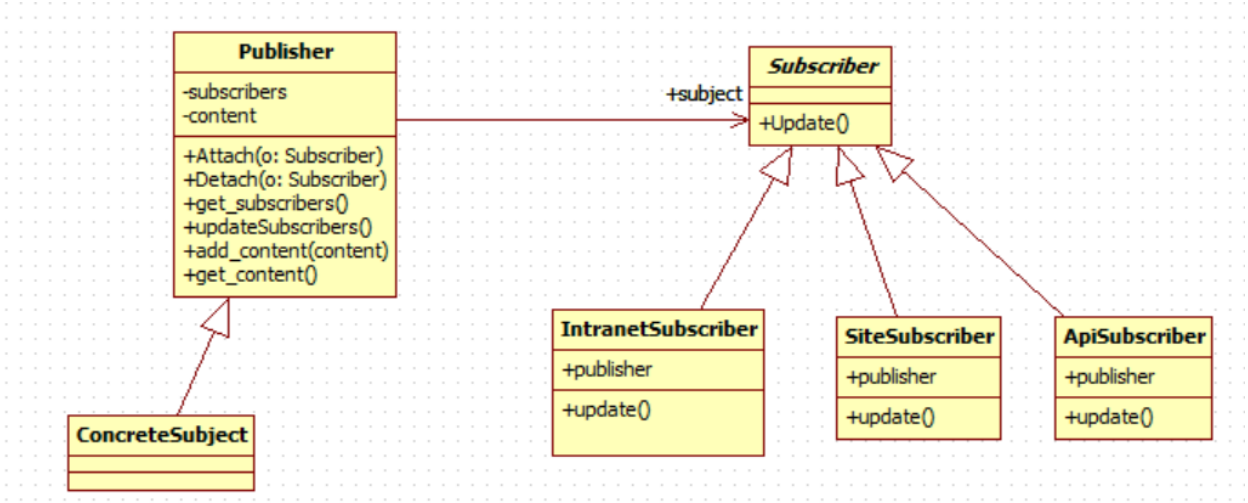
Course Instructor: Misbah ud Din

Date: 1-Dec-2022

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Lab Tasks:

ObserverDesignPattern:



Code:

```
from abc import ABC, abstractmethod


class Publisher:
    def __init__(self):
        self.__subscribers = []
        self.__content = None

    def attach(self, subscriber):
        self.__subscribers.append(subscriber)

    def detach(self):
        self.__subscribers.pop()

    def get_subscribers(self):
        return [type(x).__name__
        for x in self.__subscribers]

    def updateSubscribers(self):
        for sub in self.__subscribers:
            sub.update()

    def add_content(self, content):
        self.__content = content
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

```
    def get_content(self):
        return ("Content:" + self.__content)


class Subscriber(ABC):
    @abstractmethod
    def update(self):
        pass


class SiteSubscriber(Subscriber):
    def __init__(self, publisher):
        self.publisher = publisher
        self.publisher.attach(self)

    def update(self):
        print(type(self).__name__,
        self.publisher.get_content())
# -------------------- # Subscriber 2
class IntranetSubscriber(Subscriber):
    def __init__(self, publisher):
        self.publisher = publisher
        self.publisher.attach(self)
    def update(self):
        print(type(self).__name__,
        self.publisher.get_content())
# -------------------- # Subscriber 3
class ApiSubscriber(Subscriber):
    def __init__(self, publisher):
        self.publisher = publisher
        self.publisher.attach(self)
    def update(self):
        print(type(self).__name__,
        self.publisher.get_content())


publisher = Publisher()
for subs in [SiteSubscriber, IntranetSubscriber, ApiSubscriber]:
    subs(publisher)
print("All Subscriber: ", publisher.get_subscribers())
print("------------------------------------------------")

publisher.add_content('Update content on all subscribers.')
```

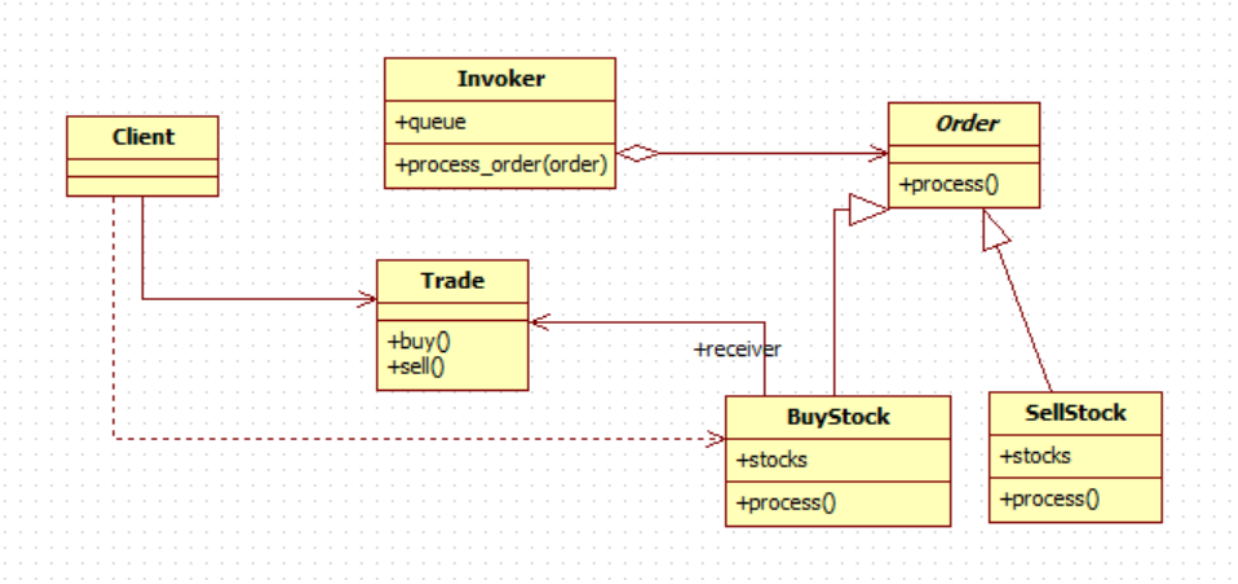
Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

```
publisher.updateSubscribers()

print("-----------------------------------------------")

publisher.detach()
print("Remaining Subscriber: ", publisher.get_subscribers())
print("-----------------------------------------------")

publisher.add_content('Updated content for remaining subscriber.')
publisher.updateSubscribers()
```

Ouput:

```
All Subscriber:  ['SiteSubscriber', 'IntranetSubscriber', 'ApiSubscriber']
-----------------------------------------------
SiteSubscriber Content:Update content on all subscribers.
IntranetSubscriber Content:Update content on all subscribers.
ApiSubscriber Content:Update content on all subscribers.
-----------------------------------------------
Remaining Subscriber:  ['SiteSubscriber', 'IntranetSubscriber']
-----------------------------------------------
SiteSubscriber Content:Updated content for remaining subscriber.
IntranetSubscriber Content:Updated content for remaining subscriber.
```

CommandDesignPattern:



Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Code:

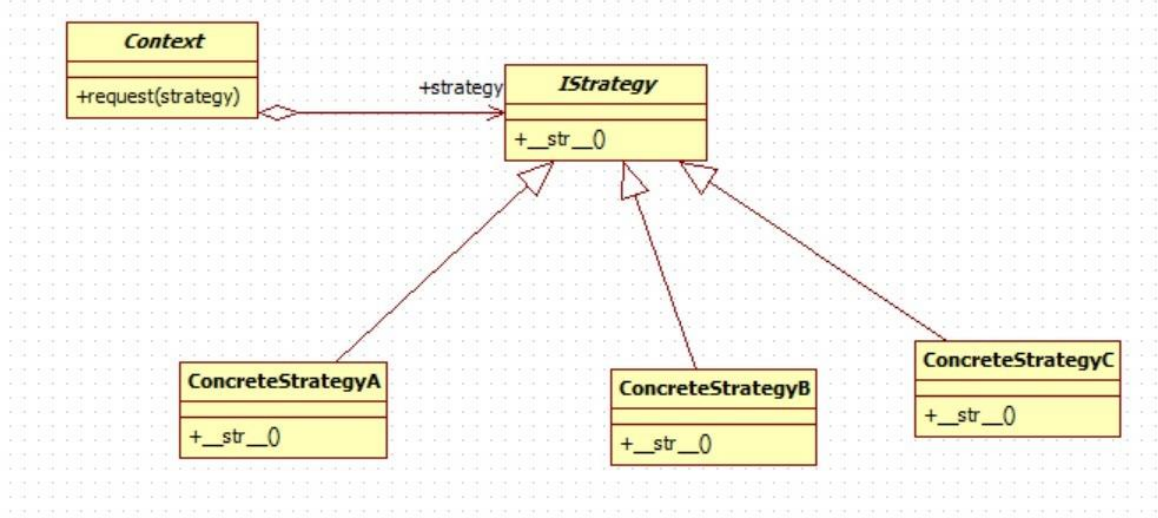
```python
from abc import ABC, abstractmethod
class Order(ABC):
    @abstractmethod
    def process(self):
        pass

class BuyStock(Order):
    def __init__(self, stock):
        self.stock = stock
    def process(self):
        self.stock.buy()

class SellStock(Order):
    def __init__(self, stock):
        self.stock = stock
    def process(self):
        self.stock.sell()

class Trade:
    def buy(self):
        print("Stock buy request placed.")
    def sell(self):
        print("Stock sell request placed.")
class Invoker:
    def __init__(self):
        self._queue = []
    def process_order(self, order):
        self._queue.append(order)
        order.process()

trade = Trade()
buy_stock = BuyStock(trade)
sell_stock = SellStock(trade)

invoker = Invoker()
invoker.process_order(buy_stock)
invoker.process_order(sell_stock)
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Output:

```
Stock buy request placed.
Stock sell request placed.
```

StrategyDesignPattern:



Code:

```python
from abc import ABCMeta, abstractmethod


class Context():
    # "This is the object whose behavior will change"
    @staticmethod
    def request(strategy):
    # "The request is handled by the class passed in"
        return strategy()


class IStrategy(metaclass=ABCMeta):
    "A strategy Interface"
    @staticmethod
    @abstractmethod
    def __str__():
        return "I am a Strategy"


class ConcreteStrategyA(IStrategy):
    "A Concrete Strategy Subclass"
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

```
    def __str__(self):
        return "I am ConcreteStrategyA"


class ConcreteStrategyB(IStrategy):
    "A Concrete Strategy Subclass"

    def __str__(self):
        return "I am ConcreteStrategyB"


class ConcreteStrategyC(IStrategy):
    "A Concrete Strategy Subclass"

    def __str__(self):
        return "I am ConcreteStrategyC"


# The Client
CONTEXT = Context()
print(CONTEXT.request(ConcreteStrategyA))
print(CONTEXT.request(ConcreteStrategyB))
print(CONTEXT.request(ConcreteStrategyC))
```

Output:

```
I am ConcreteStrategyA
I am ConcreteStrategyB
I am ConcreteStrategyC
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Home Tasks:

Sorting Strategy:

```python
#strategy design pattern example

class SortStrategy:
    def sort(self, data):
        raise NotImplementedError

class QuickSort(SortStrategy):
    def sort(self, data):
        print('Quick sort')

class MergeSort(SortStrategy):
    def sort(self, data):
        print('Merge sort')

class Sorter:
    def __init__(self, sort_strategy):
        self.sort_strategy = sort_strategy

    def sort(self, data):
        self.sort_strategy.sort(data)

if __name__ == '__main__':
    quick = QuickSort()
    merge = MergeSort()

    sorter = Sorter(quick)

    sorter.sort([1, 2, 3, 4, 5])

    sorter.sort_strategy = merge

    sorter.sort([1, 2, 3, 4, 5])
```

Output:

```
Quick sort
Merge sort
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Youtube Observer:

```
class YouTubeChannel:
    def __init__(self):
        self.subscribers = []
        self.video = None

    def attach(self, subscriber):
        self.subscribers.append(subscriber)

    def detach(self, subscriber):
        self.subscribers.remove(subscriber)

    def notify(self):
        for subscriber in self.subscribers:
            subscriber.update()

    def upload(self, video):
        self.video = video
        self.notify()

class Subscriber:
    def __init__(self, name):
        self.name = name

    def update(self):
        print(f'{self.name} has been notified of a new video')

yt = YouTubeChannel()

s1 = Subscriber('Waleed')
s2 = Subscriber('Bajwa')
s3 = Subscriber('Farhan')
s4 = Subscriber('Huzzu')

yt.attach(s1)
yt.attach(s2)
yt.attach(s3)
yt.attach(s4)

yt.upload('Python Tutorial')
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

Output:

```
Waleed has been notified of a new video
Bajwa has been notified of a new video
Farhan has been notified of a new video
Huzzu has been notified of a new video
```

FoodCommand:

```python
from abc import ABC, abstractmethod

class Food:
    @abstractmethod
    def cook(self):
        pass

class Pizza(Food):
    def cook(self):
        print('Pizza is cooking')

class Burger(Food):
    def cook(self):
        print('Burger is cooking')

class FoodCommand:
    def __init__(self, food):
        self.food = food

    def execute(self):
        self.food.cook()

class Waiter:
    def __init__(self):
        self.__commands = []

    def add_command(self, command):
        self.__commands.append(command)

    def remove_command(self, command):
        self.__commands.remove(command)

    def execute_commands(self):
        for command in self.__commands:
```

Muhammad Waleed
20b-115-se
Lab#08
Sir Misbah Ud Deen

```
            command.execute()

if __name__ == '__main__':
    pizza = Pizza()
    burger = Burger()

    pizza_command = FoodCommand(pizza)
    burger_command = FoodCommand(burger)

    waiter = Waiter()
    waiter.add_command(pizza_command)
    waiter.add_command(burger_command)

    waiter.execute_commands()
```

Output:

```
Pizza is cooking
Burger is cooking
```