

Publisher and Subscriber design pattern

On Magazine

Name:Chethan Kumar K M

SRN:PES1PG23CA329

Publisher: In the magazine analogy, the publisher is like the magazine company itself. It creates and publishes content, such as articles, images, and advertisements, and makes them available to readers.

Subscriber: Subscribers are like the readers who have subscribed to the magazine. They are interested in receiving the content and updates from the magazine regularly. In a software system, subscribers are entities that are interested in certain types of events or messages.

Content: Content in this analogy represents the articles, images, and other materials that make up the magazine. In a software system, content can be any type of data or information that the publisher distributes to subscribers.

Subscription: Readers subscribe to the magazine to receive regular issues. Similarly, in software, subscribers subscribe to specific types of events or messages they are interested in.

Broadcasting: Instead of sending individual copies of the magazine to each subscriber, the magazine company broadcasts the latest issue to all subscribers simultaneously. Similarly, in software, publishers broadcast messages or events to all subscribers who have expressed interest in receiving them.

Decoupling: One of the key benefits of the publisher-subscriber pattern is decoupling. The publisher and subscribers are independent of each other.

Publishers do not need to know the identities of subscribers, and subscribers do not need to know the identities of publishers. This allows for more flexibility and scalability in the system.

Event-driven: The publisher-subscriber pattern is inherently event-driven. Publishers generate events (e.g., publishing a new article), and subscribers react to those events (e.g., displaying the new article to the user).

CODE:

```
class MagazinePublisher:
```

```
    def __init__(self):
```

```
        self.subscribers = []
```

```
    def subscribe(self, subscriber):
```

```
        self.subscribers.append(subscriber)
```

```
    def unsubscribe(self, subscriber):
```

```
        self.subscribers.remove(subscriber)
```

```
    def notify_subscribers(self, magazine_name):
```

```
        for subscriber in self.subscribers:
```

```
            subscriber.receive_magazine(magazine_name)
```

```
class MagazineSubscriber:
```

```
def __init__(self, name):
```

```
    self.name = name
```

```
def receive_magazine(self, magazine_name):
```

```
    print(f"{self.name} received the latest issue of {magazine_name}.")
```

```
if __name__ == "__main__":
```

```
    magazine_publisher = MagazinePublisher()
```

```
    num_subscribers = int(input("Enter the number of subscribers: "))
```

```
    for i in range(num_subscribers):
```

```
        subscriber_name = input(f"Enter the name of subscriber {i+1}: ")
```

```
        subscriber = MagazineSubscriber(subscriber_name)
```

```
        magazine_publisher.subscribe(subscriber)
```

```
    magazine_name = input("Enter the name of the magazine: ")
```

```
    magazine_publisher.notify_subscribers(magazine_name)
```

```
    unsubscribe_name = input("Enter the name of the subscriber to unsubscribe: ")
```

```
    for subscriber in magazine_publisher.subscribers:
```

```
if subscriber.name == unsubscribe_name:  
    magazine_publisher.unsubscribe(subscriber)  
    break
```

```
magazine_publisher.notify_subscribers(magazine_name)
```

OUTPUT:

Enter the number of subscribers: 3

Enter the name of subscriber 1: chethu

Enter the name of subscriber 2: malli

Enter the name of subscriber 3: ram

Enter the name of the magazine: sudha

chethu received the latest issue of sudha.

malli received the latest issue of sudha.

ram received the latest issue of sudha.

Enter the name of the subscriber to unsubscribe: chethu

malli received the latest issue of sudha.

ram received the latest issue of sudha.