Mansoura University
Faculty of Computers and Information Sciences
Department of Computer Science
First Semester- 2020-2021

[CS412P] Distributed Systems

Grade : Fourth grade

By : Zeinab Awad

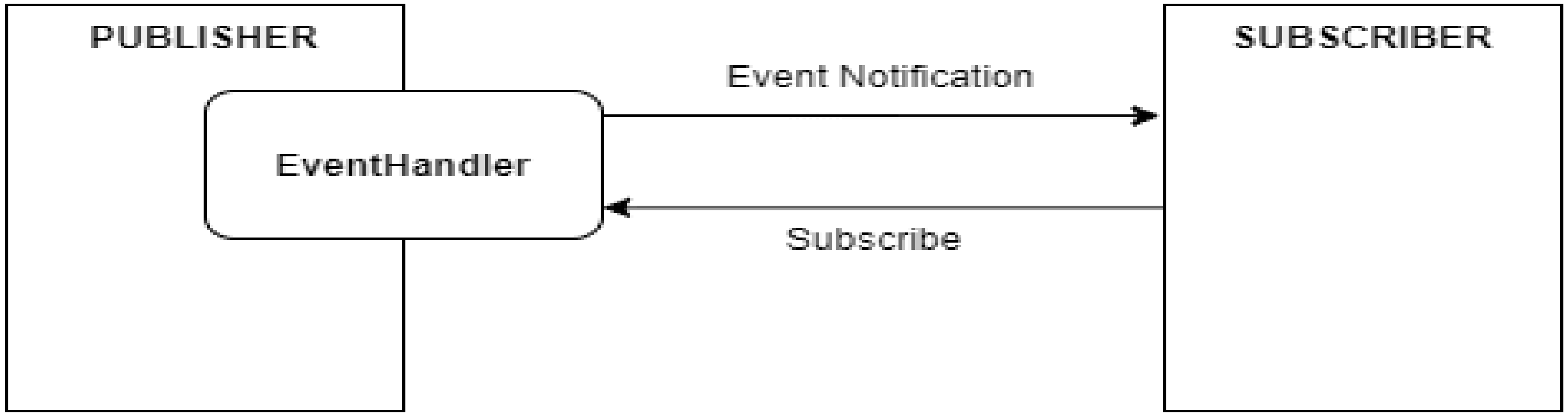
PUBLISH SUBSCRIBE MODEL

- The Publisher-Subscriber (pub-sub) pattern is an implementation of event-driven architecture. For implementing this pattern we will mainly write two classes **Publisher Class** and **Subscriber Class.**

IMPLEMENTATION OF PUBLISHER SUBSCRIBER PATTERN

- ***Subscriber class*** receives the event (like YouTube channel subscribers) and handles it as it's needed.
- ***Publisher class*** publishes an event (like YouTube channels video notification) for its' subscribers using an **EventHandler**.
- So there is an **EventHandler** involved in this process of **Publisher-Subscriber** pattern to get notifications from **YouTube Channel (Publisher)** and send it to **Channel subscribers**.
- ***Publisher class*** and ***Subscriber class*** doesn't have to know each other they both are connected to ***EventHandler***. ***Publisher*** will send the Notification Event to ***EventHandler*** and it will send the Notification Event to ***Subscribers***.

PUBLISHER SUBSCRIBER PATTERN



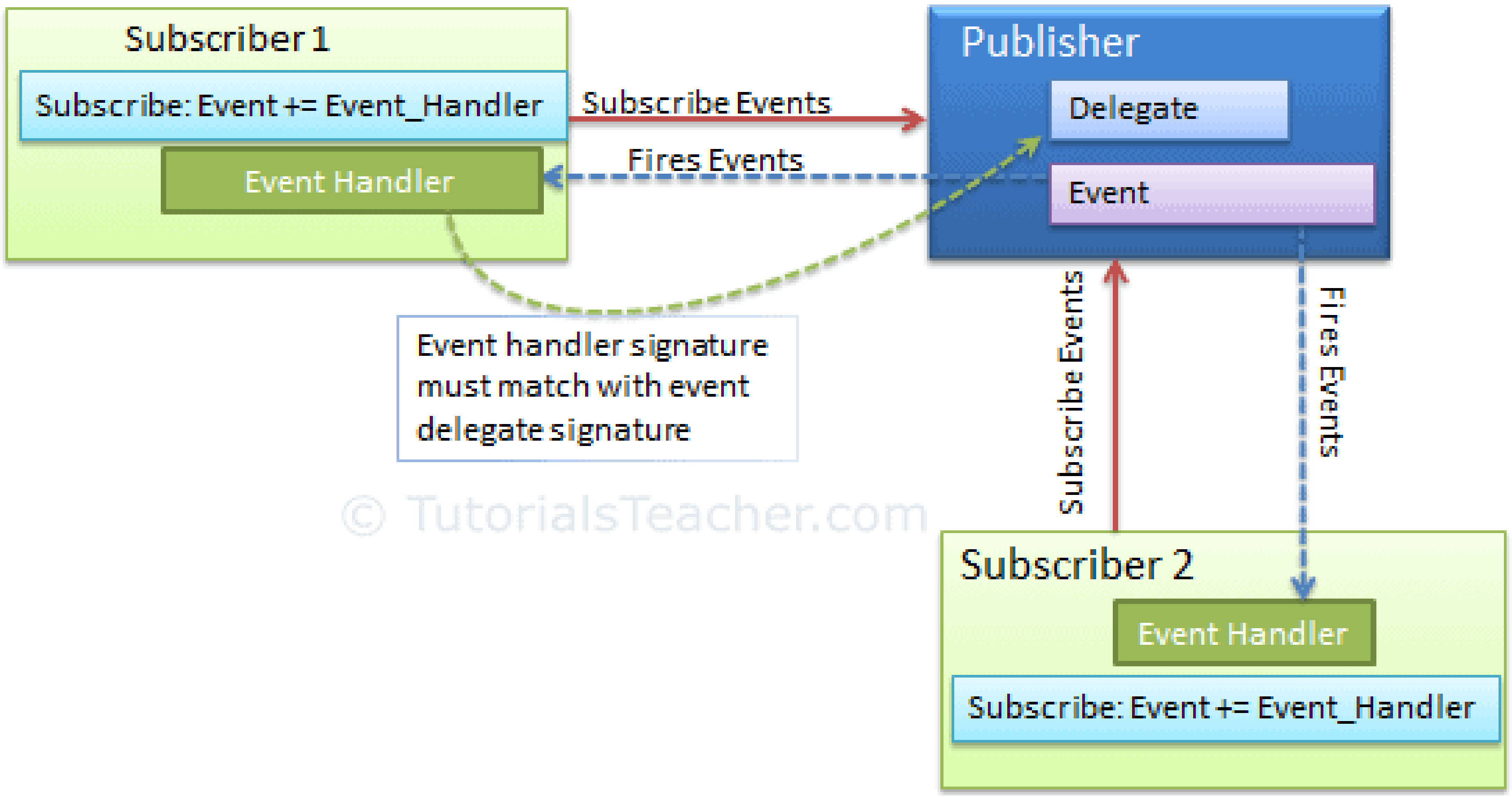
PUBLISH SUBSCRIBE MODEL

The class who raises events is called Publisher, and the class who receives the notification is called Subscriber. There can be multiple subscribers of a single event. Typically, a publisher raises an event when some action occurred. The subscribers, who are interested in getting a notification when an action occurred, should register with an event and handle it.

PUBLISH SUBSCRIBE MODEL

In C#, an event is an encapsulated delegate. It is dependent on the delegate. The delegate defines the signature for the event handler method of the subscriber class.

PUBLISH SUBSCRIBE MODEL



PUBLISHER CLASS WITH AN EVENT HANDLER

- the publisher class has two properties Publisher Name and Notification Interval.
  And an Event variable declared with a Delegate Function for Event handling.

PUBLISHER CLASS WITH AN EVENT HANDLER

```
using System;
using notification Interval;
using System. LINQ;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace Pub Sub
{
    class Publisher
    {
    //publishers name property
    public string Publisher Name { get; private set; }
    //publishers notification interval
    public int Notification Interval { get; private set; }
    // declare a delegate function named notify
    public delegate void Notify(Publisher p, Notification Event e);
    // declare an event variable onpublish based on the delegate method(event
handler) notify
    public event Notify On Publish;
    // class constructor
    public Publisher(string _publisher Name, int _notification Interval){
        Publisher Name = _publisher Name;
        Notification Interval = _notification Interval;
    }
```

PUBLISHER CLASS WITH AN EVENT HANDLER

```
//publish function publishes a Notification Event
    public void Publish(){

        while (true){

            // fire event after certain interval
            Thread.Sleep(NotificationInterval);

            if (OnPublish != null)
            {
NotificationEvent notificationObj = new NotificationEvent(DateTime.Now,
"New Notification Arrived from");
                OnPublish(this, notificationObj);
            }
            Thread.Yield();
        }
    }
}
    }
```

SUBSCRIBER CLASS

Subscriber class receives the event notification from subscribed publisher and prints events data with **OnNotificationReceived** function.

SUBSCRIBER CLASS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Pub_Sub
{
class Subscriber
    {
        public string SubscriberName { get; private set; }
    public Subscriber(string _subscriberName){
        SubscriberName = _subscriberName;
    }
    // This function subscribe to the events of the publisher
    public void Subscribe(Publisher p){
        // register OnNotificationReceived with publisher event
     p.OnPublish += OnNotificationReceived;  // multicast delegate
    }
```

SUBSCRIBER CLASS

```
 // This function unsubscribe from the events of the publisher
        public void Unsubscribe(Publisher p)
{
        // unregister OnNotificationReceived from publisher
        p.OnPublish -= OnNotificationReceived;  // multicast delegate
    }
    // It get executed when the event published by the Publisher( an event 
handler)
 protected virtual void OnNotificationReceived(Publisher p, NotificationEvent e)
{
     Console.WriteLine("Hey " + SubscriberName + ", " + e.NotificationMessage +"
- "+ p.PublisherName + " at " + e.NotificationDate);
    }
}
    }
```

Notification Event Class

- This **Notification Event** will be sent/published to the subscribers from the publisher

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Pub_Sub
{
    class NotificationEvent
    {
        public string NotificationMessage { get; private set; }
        public DateTime NotificationDate { get; private set; }
        public NotificationEvent(DateTime _dateTime, string _message)
        {
            NotificationDate = _dateTime;
            NotificationMessage = _message;
        }
    }
}
```

Main C# Class (Pub-Sub)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Pub_Sub
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creating Instance of Publishers
            Publisher youtube = new Publisher("Youtube.Com", 2000);
            Publisher facebook = new Publisher("Facebook.com", 1000);
            //Create Instances of Subscribers
            Subscriber sub1 = new Subscriber("zeinab");
            Subscriber sub2 = new Subscriber("mariam");
            Subscriber sub3 = new Subscriber("abdulrahman");
            //Pass the publisher obj to their Subscribe function
```

Main C# Class (Pub-Sub)

```
sub1.Subscribe(facebook); //sub1 subscribes to facebook publisher
sub3.Subscribe(facebook);
sub1.Subscribe(youtube);
sub2.Subscribe(youtube); //sub1.Unsubscribe(facebook);
            // Concurrently running multiple publishers thread for 
making continious notification update firing.
Task task1 = Task.Factory.StartNew(() => youtube.Publish());
Task task2 = Task.Factory.StartNew(() => facebook.Publish());
            Task.WaitAll(task1, task2);
        }
    }
}
```

PUBLISH SUBSCRIBE MODEL

Thanks