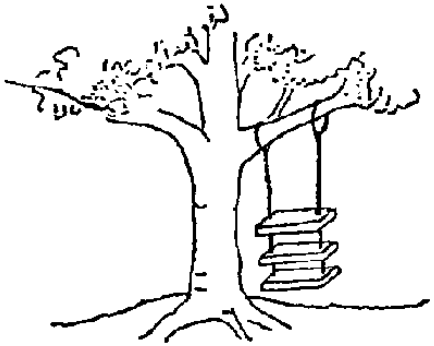
Delegate & Event Examples

Software Entwicklung



Overview

- Delegate NumberChanger
- Multicast Delegate NumberChanger
- Multicast Delegate Employee
- Delegate with Bookstore

- Happy Birthday – Congrats Event
- Metronome – Tick Event
- Elevator – Warning Event
- School - Fire Alarm Event

Exercise with Delegates

Delegate NumberChanger

Delegate Calc

Delegate PrintString

Example NumberChanger

- Delegate:
  - take an integer parameter and
  - returns an integer value

- Write a Program that can Add, Mul, Div and Sub Numbers which all share the NumberChanger Syntax

```
delegate int NumberChanger(int n);
NumberChanger nc1 = new NumberChanger(AddNum);
NumberChanger nc2 = new
NumberChanger(MultNum);
```

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl
{
   class TestDelegate
   {
      static int num = 10;
      public static int AddNum(int p)
      {
         num += p;
         return num;
      }

      public static int MultNum(int q)
      {
         num *= q;
         return num;
      }
      public static int getNum()
      {
         return num;
      }
```

Number Changer

NumberChanger - Main

```
static void Main(string[] args)
{
    //create delegate instances
    NumberChanger nc1 = new NumberChanger(AddNum);
    NumberChanger nc2 = new NumberChanger(MultNum);

    //calling the methods using the delegate objects
    nc1(25);
    Console.WriteLine("Value of Num: {0}", getNum());
    nc2(5);
    Console.WriteLine("Value of Num: {0}", getNum());
    Console.ReadKey();
}
```

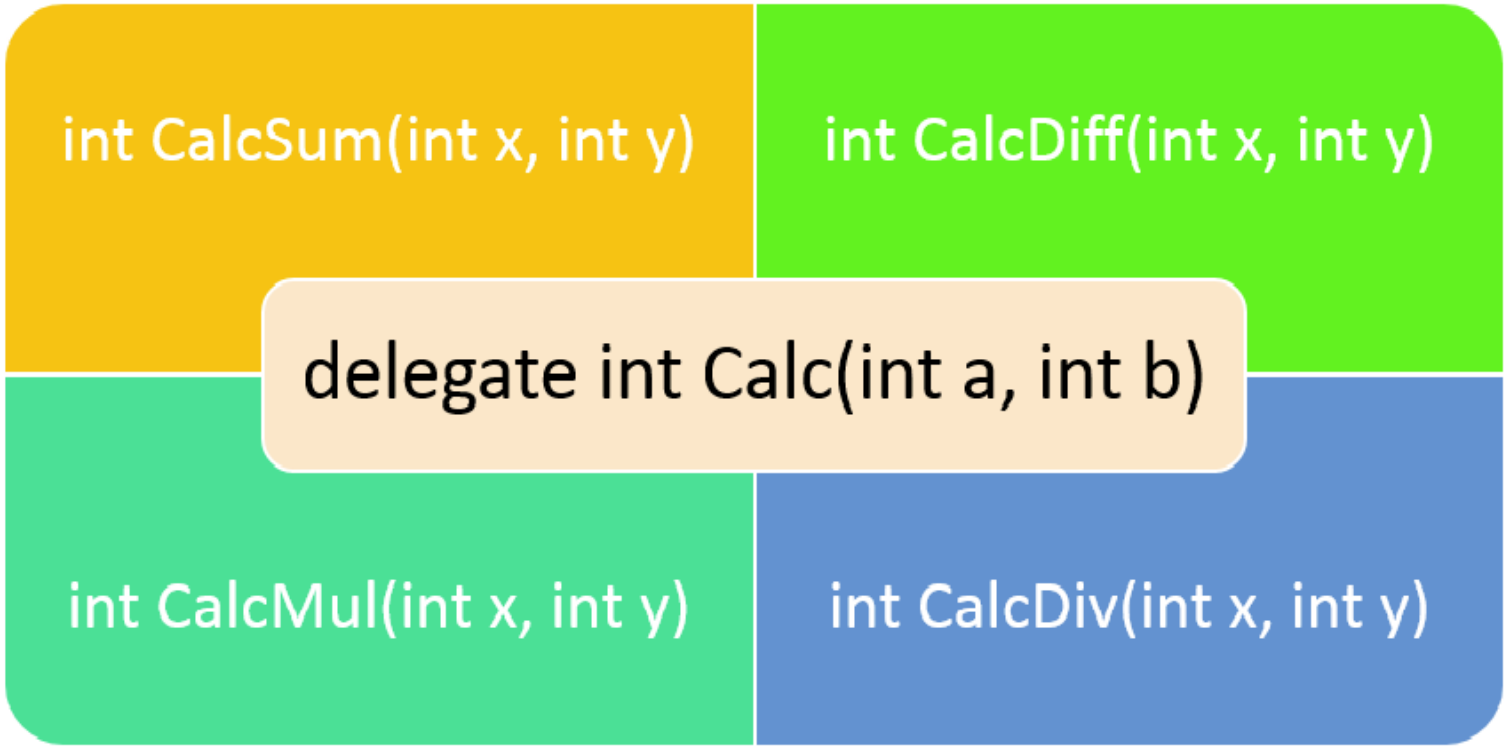
Multicasting of a Delegate

```
static void Main(string[] args)
{
    //create delegate instances
    NumberChanger nc;
    NumberChanger nc1 = new NumberChanger(AddNum);
    NumberChanger nc2 = new NumberChanger(MultNum);
    nc = nc1;
    nc += nc2;

    //calling multicast
    nc(5);
    Console.WriteLine("Value of Num: {0}", getNum());
    Console.ReadKey();
}
```

Ausgabe:
Value of Num: 75

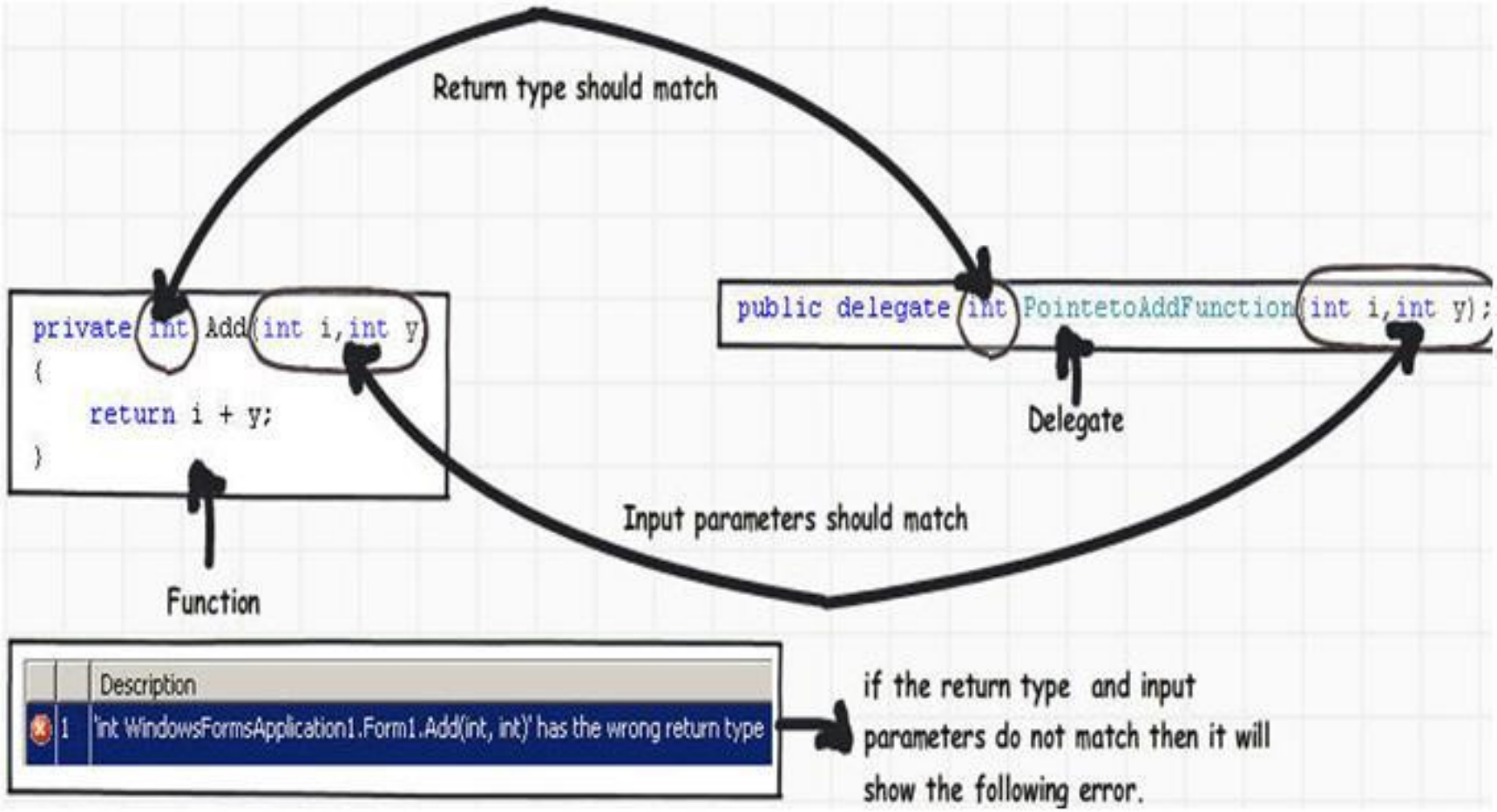
Example: Calc

- Write a Program that can Add, Mul, Div and Sub Numbers which all share the Calc Syntax:



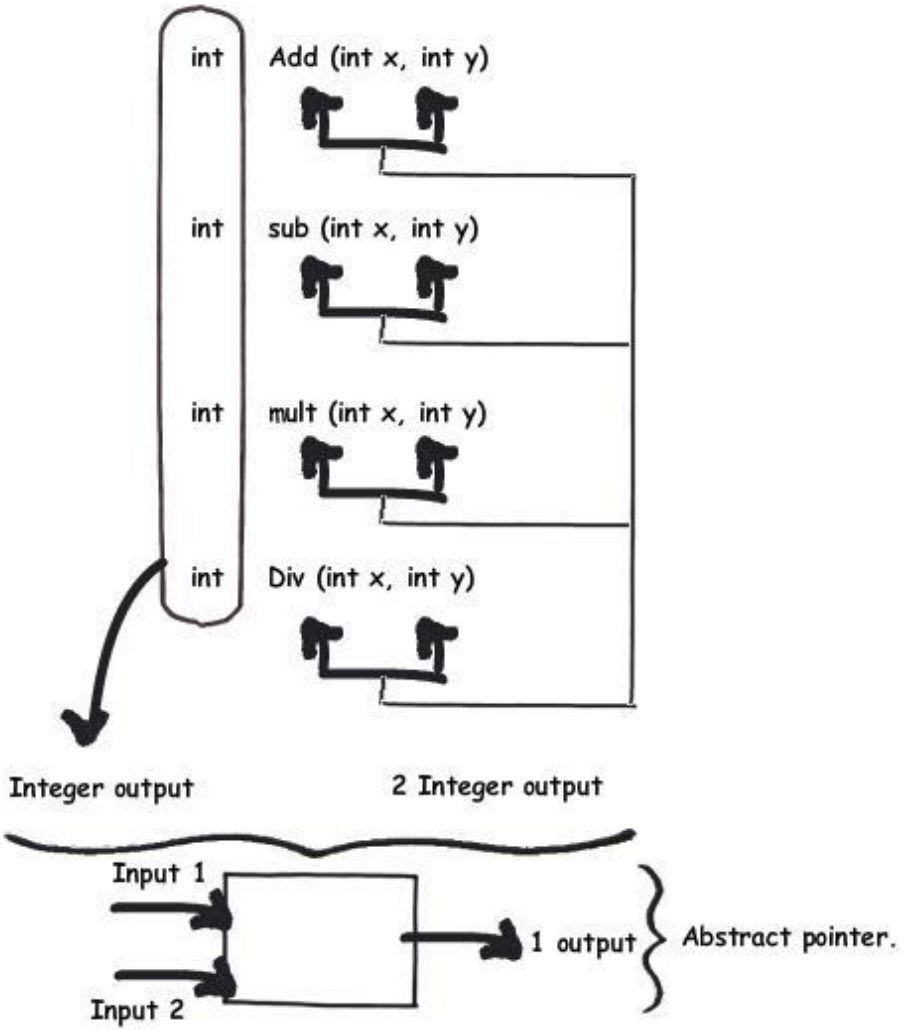
https://www.codeproject.com/Articles/85296/6-important-uses-of-Delegates-and-Events

Example: Calc

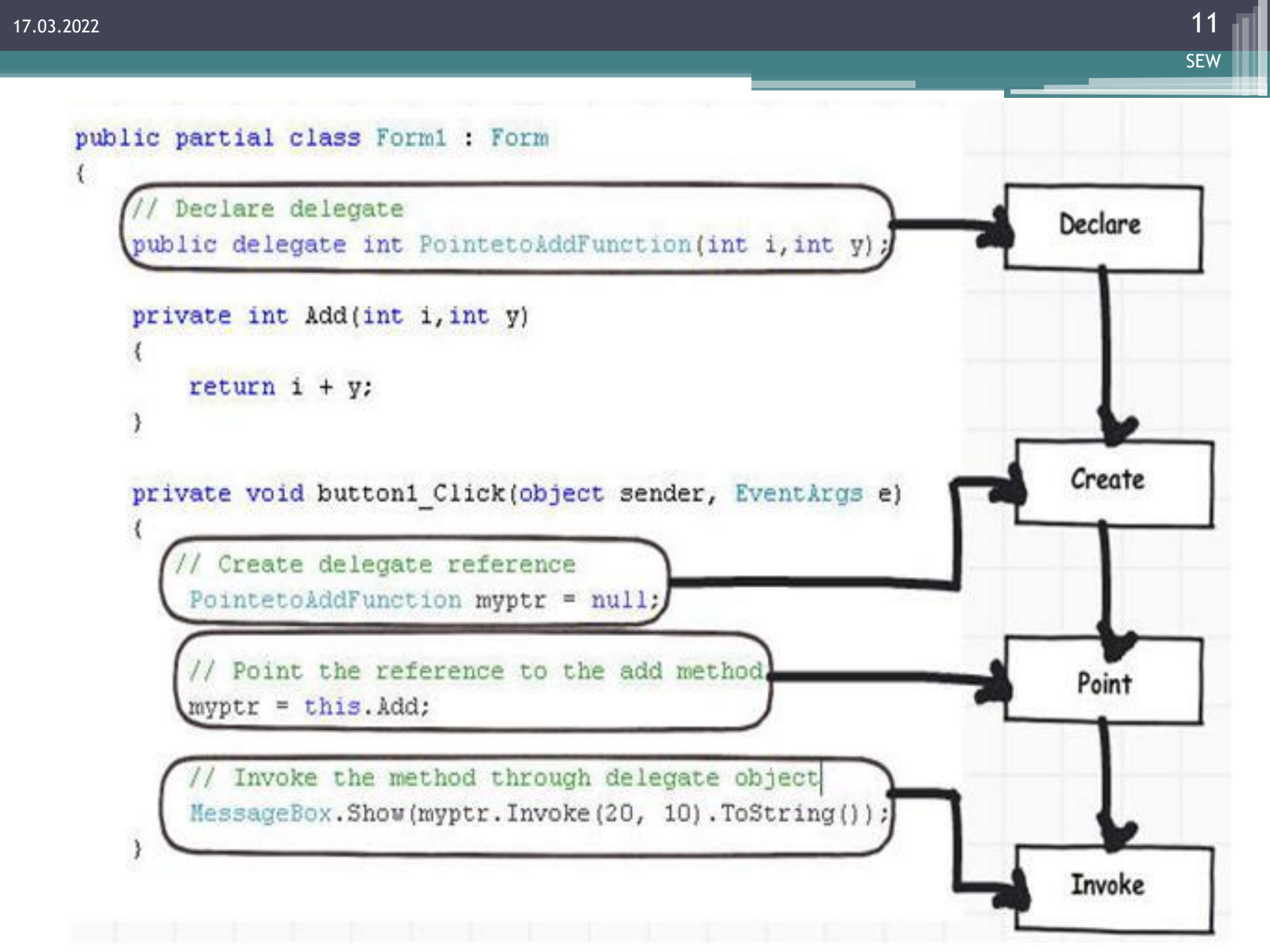


Example: Calc

```
private int Add(int i, int y) {
    return i + y;
}

// Declare delegate
public delegate int
PointerToFunction(int i, int y);
```



```
public partial class Form1 : Form
{
    // Declare delegate
    public delegate int PointetoAddFunction(int i,int y);

    private int Add(int i,int y)
    {
        return i + y;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        // Create delegate reference
        PointetoAddFunction myptr = null;

        // Point the reference to the add method
        myptr = this.Add;

        // Invoke the method through delegate object
        MessageBox.Show(myptr.Invoke(20, 10).ToString());
    }
```



Example: PrintString

- Create a Class PrintString with the following delegate declaration:

```
public delegate void printString(string s);
```

- WriteToScreen schreibt in die Console
- WriteToFile schreibt in eine Textdatei

```
static void Main(string[] args)
{
    printString ps1 = new printString(WriteToScreen);
    printString ps2 = new printString(WriteToFile);
    sendString(ps1);
    sendString(ps2);
    Console.ReadKey();
}
```

```
// this method takes the delegate as parameter and uses it to
// call the methods as required
public static void sendString(printString ps)
{
    ps("Hello World");
}
```

```
class PrintString
{
   static FileStream fs;
   static StreamWriter sw;

   // delegate declaration
   public delegate void printString(string s);

   // this method prints to the console
   public static void WriteToScreen(string str)
   {
      Console.WriteLine("The String is: {0}", str);
   }

   //this method prints to a file
   public static void WriteToFile(string s)
   {
      fs = new FileStream("c:\\message.txt",
      FileMode.Append, FileAccess.Write);
      sw = new StreamWriter(fs);
      sw.WriteLine(s);
      sw.Flush();
      sw.Close();
      fs.Close();
   }
```

Printer Event Example

PrintHelper:

Print Decimal, Money, Temperature, Hexadecimal

http://www.tutorialsteacher.com/codeeditor?cid=cs-AhemdS

Publisher: PrintHelper

- Write a PrintHelper class
  - that prints integers in different formats
  - like number, money, decimal, temperature and hexadecimal
  - include a beforePrintEvent
    to notify the subscriber of the BeforePrint event
    before it going to print the number
- PrintHelper is a publisher class
  that publishes the beforePrint event

Print Helper

```
public class PrintHelper    {
    // declare delegate
    public delegate void BeforePrint();
    //declare event of type delegate
    public event BeforePrint beforePrintEvent;
    public PrintHelper() {  }
    public void PrintNumber(int num)
    {
        //call delegate method before going to print
        if (beforePrintEvent != null) beforePrintEvent();
        Console.WriteLine("Number: {0,-12:N0}", num);
    }
    public void PrintDecimal(int dec)...
    public void PrintMoney(int money)...
    public void PrintTemperature(int num)...
    public void PrintHexadecimal(int dec)...
}
```

Notice that in each print method, it first checks to see if beforePrintEvent is not null and then it calls *beforePrintEvent()*.

beforePrintEvent is an object of type BeforPrint delegate, so it would be null if no class is subscribed to the event and that is why it is necessary to check for null before calling a delegate.

http://www.tutorialsteacher.com/codeeditor?cid=cs-AhemdS

Create a Subscriber

```
class Number
{
    private PrintHelper _printHelper;
    public Number(int val)
    {
        _value = val;
        _printHelper = new PrintHelper();
        //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent +=
            printHelper_beforePrintEvent;

    }
    //beforePrintevent handler
    void printHelper_beforePrintEvent() {
        Console.WriteLine("BeforPrintEventHandler: " +
            "PrintHelper is going to print a value");
    }

    private int _value;
    public int Value...
    public void PrintMoney()...
    public void PrintNumber()...
}
```

All the subscribers must provided a handler function, which is going to be called when a publisher raises an event.

Number class creates an instance of PrintHelper and subscribes to the beforePrintEvent with the "+=" operator and gives the name of the function which will handle the event (it will be get called when publish fires an event).

printHelper_beforePrintEvent is the event handler that has the same signature as the BeforePrint delegate in the PrintHelper class.

Call Print Methods

- Create a Numbers Class and call the PrintMethods

```
class Program
{
    static void Main(string[] args)
    {
        Number myNumber = new Number(100000);
        myNumber.PrintMoney();
        myNumber.PrintNumber();
    }
}
```

BeforePrintEventHandler: PrintHelper is going to print value
Money: $ 1,00,000.00
BeforePrintEventHandler: PrintHelper is going to print value
Number: 1,00,000

Event Arguments

- can also pass data as an argument to their subscribed handler

- event passes arguments to the handler as per the delegate signature

```
public delegate void BeforePrint(string message);
public event BeforePrint beforePrintEvent;
```

NumberArgs

NumberArgs class has a printHelper_beforePrintEvent function with string parameter

```
public class NumberArgs
{
    private PrintHelperArgs _printHelper;
    public NumberArgs(int val) {
        _value = val;
        _printHelper = new PrintHelperArgs();
        //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }
    //beforePrintevent handler
    void printHelper_beforePrintEvent(string message) {
        Console.WriteLine("BeforePrintEvent fires from {0}", message);
    }
    private int _value;
    public int Value...

    public void PrintMoney() {
        _printHelper.PrintMoney(_value);
    }
    public void PrintNumber() {
        _printHelper.PrintNumber(_value);
    }
}
```

```
BeforePrintEvent fires from PrintMoney.
Money: $ 1,00,000.00
BeforePrintEvent fires from PrintNumber.
Number: 1,00,000
```

PrintHelperArgs

NumberArgs class has a printHelper_beforePrintEvent function with string parameter

```
public class PrintHelperArgs
{
    public delegate void BeforePrint(string message);
    public event BeforePrint beforePrintEvent;
    public PrintHelperArgs() { }
    public void PrintNumber(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent.Invoke("PrintNumber");
        Console.WriteLine("Number: {0,-12:N0}", num);
    }
    public void PrintDecimal(int dec) ...
    public void PrintMoney(int money)...
    public void PrintTemperature(int num)...
    public void PrintHexadecimal(int dec)...
}
```

Points to Remember

- Use event keyword with delegate type to declare an event
- Check event is null or not before raising an event
- Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.
- Function that handles the event
  - is called event handler
  - Event handler must have same signature as declared by event delegate
- Events can have arguments which will be passed to handler function

Points to Remember

- Events can also be declared static, virtual, sealed and abstract
- An Interface can include event as a member
- Events will not be raised if there is no subscriber
- Event handlers are invoked synchronously if there are multiple subscribers
- The .NET framework
  - uses an EventHandler delegate and
  - an EventArgs base class

Multicast Delegate Employee

```
Company c = new Company();
c.PromoteEmployees();
```

```
class Company {

    public List<Employee> Employees { get; }

    public delegate void CustomDel(Employee employee);
    CustomDel SalaryRaiseDel, PositionDateDel, EmployeePromoteMulticastDelegate;
```

Company

- Create a CustomDelegate
  - with an Employee as Parameter
  - void as return type

```
Company c = new Company();
c.PromoteEmployees();
```

  - Create 3 Delecates:
    - Sal(aryRaise)Del -> EmployeeSalaryRaise Method
    - Pos(itionDate)Del -> PositionDateUpdate Method
    - EmployeePromoteMulticastDelegate = SalDel + PosDel

Promote Employee

```
class Employee    {
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime PositionChangeDate { get; set; }
    public decimal Salary { get; set; }
}
```

```
class Company {
    public delegate void CustomDel(Employee employee);
    CustomDel SalaryRaiseDel, PositionDateDel, EmployeePromoteMulticastDelegate;

    public List<Employee> Employees { get; }
    public Company() ...
    public void PromoteEmployees()
    {
        foreach (Employee emp in Employees)
        {
            EmployeePromoteMulticastDelegate(emp);

            Console.WriteLine($"{emp.Id} - {emp.Name}"  );
            Console.Write (" LastUpdate " + emp.PositionChangeDate.ToShortDateString());
            Console.WriteLine(" Salary = " + emp.Salary);
        }
    }
```

Employee - Multicast Delegate

```
class Company {

    public List<Employee> Employees { get; }

    public delegate void CustomDel(Employee employee);
    CustomDel SalaryRaiseDel, PositionDateDel, EmployeePromoteMulticastDelegate;


    public Company() {
        Employees = new List<Employee>() {
         new Employee(){Id = 1, Name = "Max", Salary = 10},
         new Employee(){Id = 2, Name = "John", Salary = 20},
         new Employee(){Id = 3, Name = "Kurt", Salary = 15},
         new Employee(){Id = 4, Name = "Nik", Salary = 25},};

        SalaryRaiseDel = new CustomDel(EmployeeSalaryRaise);
        PositionDateDel = new CustomDel(PostionDateUpdate);
        //multiple delegates assigned
        EmployeePromoteMulticastDelegate = PositionDateDel
                                         + SalaryRaiseDel;
    }
```

Update & Raise Methods…

- Static Methods in the Company-Class

```
//set position change date to current time
private void PostionDateUpdate(Employee employee)
{
    employee.PositionChangeDate = DateTime.Now;
}

//raise employee salary by 10%
private void EmployeeSalaryRaise(Employee employee)
{
    employee.Salary += employee.Salary + employee.Salary
        * (decimal)0.1;
}
```

Delegate with Bookstore

The BookDB class encapsulates a bookstore database that maintains a database of books. It exposes a method, ProcessPaperbackBooks, which finds all paperback books in the database and calls a delegate for each one.

The delegate type that is used is named ProcessBookDelegate.

The Test class uses this class to print the titles and average price of the paperback books.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/how-to-declare-instantiate-and-use-a-delegate

Book & ProcessBookDelegate

```
// Describes a book in the book list:
public struct Book
{
    public string Title;        // Title of the book.
    public string Author;       // Author of the book.
    public decimal Price;       // Price of the book.
    public bool Paperback;      // Is it paperback?

    public Book(string title, string author, decimal price, bool paperBack)
    {
        Title = title;
        Author = author;
        Price = price;
        Paperback = paperBack;
    }
}

// Declare a delegate type for processing a book:
public delegate void ProcessBookDelegate(Book book);
```

BookDB

```
// Declare a delegate type for processing a book:
public delegate void ProcessBookDelegate(Book book);

// Maintains a book database.
public class BookDB
{
    // List of all books in the database:
    ArrayList list = new ArrayList();

    // Add a book to the database:
    public void AddBook(string title, string author, decimal price, bool paperBack)
    {
        list.Add(new Book(title, author, price, paperBack));
    }

    // Call a passed-in delegate on each paperback book to process it:
    public void ProcessPaperbackBooks(ProcessBookDelegate processBook)
    {
        foreach (Book b in list)
        {
            if (b.Paperback)
                // Calling the delegate:
                processBook(b);
        }
    }
}
```

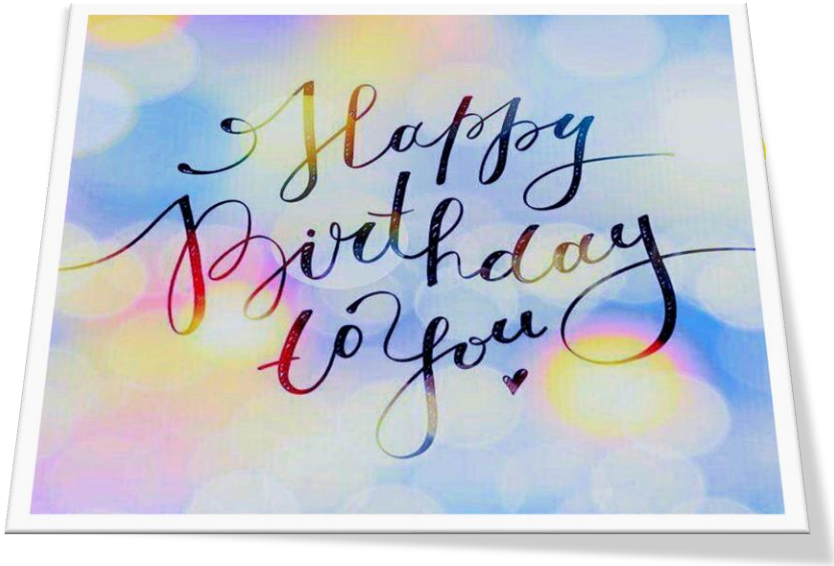
PriceTotaller

```
// Class to total and average prices of books:
class PriceTotaller
{
    int countBooks = 0;
    decimal priceBooks = 0.0m;

    internal void AddBookToTotal(Book book)
    {
        countBooks += 1;
        priceBooks += book.Price;
    }

    internal decimal AveragePrice()
    {
        return priceBooks / countBooks;
    }
}
```

Test mit Main

```
// Class to test the book database:
class Test
{
    // Print the title of the book.
    static void PrintTitle(Book b)
    {
        Console.WriteLine($"   {b.Title}");
    }

    // Execution starts here.
    static void Main()
    {
        BookDB bookDB = new BookDB();

        // Initialize the database with some books:
        AddBooks(bookDB);

        // Print all the titles of paperbacks:
        Console.WriteLine("Paperback Book Titles:");

        // Create a new delegate object associated with the static
        // method Test.PrintTitle:
        bookDB.ProcessPaperbackBooks(PrintTitle);

        // Get the average price of a paperback by using
        // a PriceTotaller object:
        PriceTotaller totaller = new PriceTotaller();

        // Create a new delegate object associated with the nonstatic
        // method AddBookToTotal on the object totaller:
        bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

        Console.WriteLine("Average Paperback Book Price: ${0:#.##}",
                totaller.AveragePrice());
    }

    // Initialize the book database with some test books:
    static void AddBooks(BookDB bookDB)
    {
        bookDB.AddBook("The C Programming Language", "Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);
        bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, true);
        bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);
        bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true);
    }
}
```

```
/* Output:
Paperback Book Titles:
   The C Programming Language
   The Unicode Standard 2.0
   Dogbert's Clues for the Clueless
Average Paperback Book Price: $23.97
*/
```



Happy Birthday Events

Write a Happy Birthday Event

Use a Delegate, that gets a name and returns a „Happy Birthday {name}“

Use a SendWishes Event

Write a Congratulate Method

Declare Delegate and Event

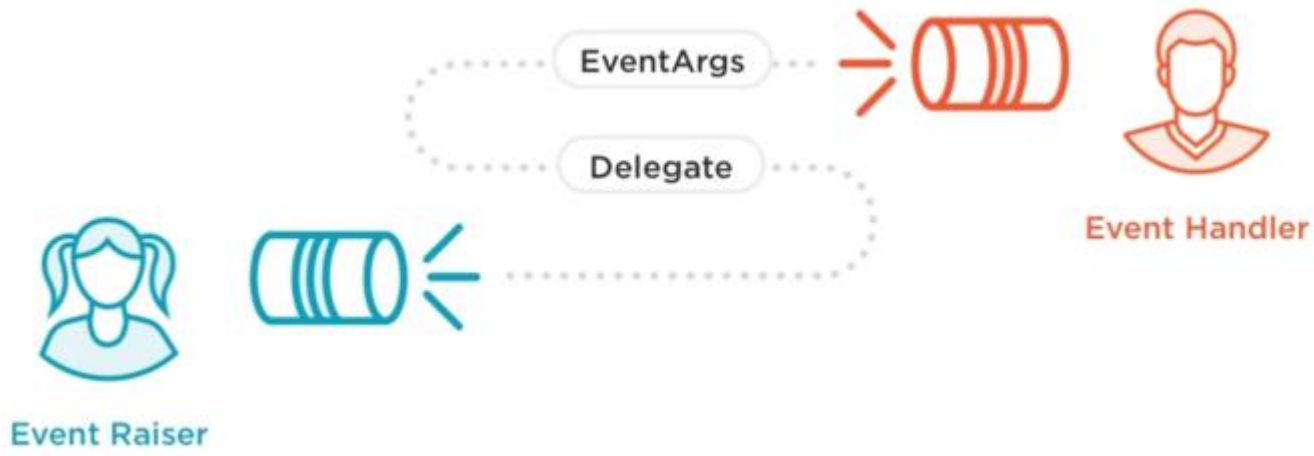
- declare a delegate type for the event
  - `public delegate string MyDel(string str);`

- declare the event itself,
  using the event keyword like:
  - `event MyDel MyEvent;`

Happy Birthday Wishes with Events

```
public delegate string BirthdayDelegate(string str);

4 Verweise
class Person
{
    event BirthdayDelegate SendWishes;

    1-Verweis
    public Person()
    {
        this.SendWishes += new BirthdayDelegate(this.Concratulate);
    }
    1-Verweis
    public string Concratulate(string nickname)
    {
        return "Happy Birthday " + nickname;
    }
    1-Verweis
    public static void Test()
    {
        Person p = new Person();
        string result = p.SendWishes("Sweety");
        Console.WriteLine(result);
    }
}
```

Add „Have a Wonderful day, in a second method – call it when the Event was raised…



Example Metronome

Subscribe(Metronome)
HeardIt(Metronome, EventArgs)

Example - Ticker

- Create a Class Metronome with an event Tick
- And a delegate TickHandler
- Use the Metronome and the EventArgs as Parameter

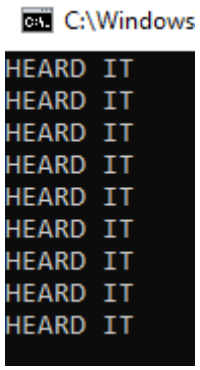
```
public delegate void TickHandler(Metronome m, EventArgs e);
```

- *Info:* **EventArgs**
  - Represents the base class for classes that contain event data, and provides a value to use for events that do not include event data.

Example - Ticker

- Write a Class Listener with the Methods
  - Subscribe(Metronome)
  - HeardIt(Metronome, EventArgs)
  - Test the Publish-Subscriber

```
public class Metronome...
2 Verweise
public class Listener...
0 Verweise
class Test {
    0 Verweise
    static void Main() {
        Metronome m = new Metronome();
        Listener l = new Listener();
        l.Subscribe(m);
        m.Start();
    }

}
```



Class Metronome

```
public class Metronome
{
    public event TickHandler Tick;
    public EventArgs e = null;
    public delegate void TickHandler(Metronome m, EventArgs e);
    public void Start()
    {
        while (true)
        {
            System.Threading.Thread.Sleep(3000);
            if (Tick != null)
            {
                Tick(this, e);
            }
        }
    }
}
```

Class Listener

```
public delegate void TickHandler(Metronome m, EventArgs e);


public class Listener
{
    public void Subscribe(Metronome m)
    {
        m.Tick += new Metronome.TickHandler(HeardIt);
    }
    private void HeardIt(Metronome m, EventArgs e)
    {
        System.Console.WriteLine("HEARD IT");
    }

}
```

Expand… using DateTime

```
public delegate void TickHandler(Metronome m, EventArgs e);
```

```
public class TimeOfTick : EventArgs
{
    private DateTime TimeNow;
    public DateTime Time
    {
        set
        {
            TimeNow = value;
        }
        get
        {
            return this.TimeNow;
        }
    }
}
```

```
HEARD IT AT 26.01.2019 14:28:54
HEARD IT AT 26.01.2019 14:28:57
HEARD IT AT 26.01.2019 14:29:00
HEARD IT AT 26.01.2019 14:29:03
HEARD IT AT 26.01.2019 14:29:06
```

Class Metronome

```
HEARD IT AT 26.01.2019 14:28:54
HEARD IT AT 26.01.2019 14:28:57
HEARD IT AT 26.01.2019 14:29:00
HEARD IT AT 26.01.2019 14:29:03
HEARD IT AT 26.01.2019 14:29:06
```

```
public class Metronome
{
    public event TickHandler Tick;
    public delegate void TickHandler(Metronome m, TimeOfTick e);
    public void Start()
    {
        while (true)
        {
            System.Threading.Thread.Sleep(3000);
            if (Tick != null)
            {
                TimeOfTick TOT = new TimeOfTick();
                TOT.Time = DateTime.Now;
                Tick(this, TOT);
            }
        }
    }
}
```

Class Listener

```
HEARD IT AT 26.01.2019 14:28:54
HEARD IT AT 26.01.2019 14:28:57
HEARD IT AT 26.01.2019 14:29:00
HEARD IT AT 26.01.2019 14:29:03
HEARD IT AT 26.01.2019 14:29:06
```

```
public class Listener
{
    public void Subscribe(Metronome m)
    {
        m.Tick += new Metronome.TickHandler(HeardIt);
    }
    private void HeardIt(Metronome m, TimeOfTick e)
    {
        System.Console.WriteLine("HEARD IT AT {0}", e.Time);
    }

}
```

```
public delegate void TickHandler(Metronome m, EventArgs e);
```

HeardIt like Button1_Click

When you add a button to a form in C#
and double click on the button in the form designer:

- you are taken to a method equivalent to "Heardit“
- but it will be named something like Button1_Click

```
protected void Button1_Click(object sender, EventArgs e)
{

}
```

```
private void HeardIt(Metronome m, TimeOfTick e)
{
    System.Console.WriteLine("HEARD IT AT {0}", e.Time);
}
```

Elevator

„Warining Overload!!!“

"Here we go again, you see it as half empty—I see it as half full."

Angabe - Personenlift

Ein Aufzug (Personenlift) besitzt eine maximale Beförderungskapazität in kg. Zur Wartung der Liftanlage besitzt jede Anlage eine eindeutige Bezeichnung zB: 2009/12345. Der Lift funktioniert nur ordnungsgemäß, wenn die maximale Beförderungskapazität nicht überschritten wird. Steigen also laufend Personen ein, wird das Gewicht kontrolliert. Im Falle der Überschreitung des zulässigen Gesamtgewichtes gibt die Anlage eine Meldung (zB Warnton) ab.

Schreiben Sie eine Klasse „Lift" mit den nötigen Eigenschaften, Methoden und Events. Die Methode „Zusteigen" erwartet einen Parameter (Instanz der Klasse Person) aus der das hinzugekommene Gewicht ausgelesen werden kann. Das Überschreiten der Kapazität soll das Auslösen eines Events zur Folge haben.

Aufgabe 1:
Testen Sie die Funktion der beiden Klasse in dem Sie in der Main-Methode eine Instanz der Klasse Lift erzeugen und kontinuierlich Personen zusteigen lassen bis das Event ausgelöst wurde. Geben Sie auf der Konsole den Namen der Person aus, welche den Lift als letztes betreten hat und wieder aussteigen sollte (= Eventhandler).

Aufgabe 2:
Sollte der Lift überfüllt werden, so soll beim Auslösen des Events nicht die letzte Person genannt werden, sondern jene, die eine optimalere Auslastung der Kapazität gewährleistet.

Elevator

```
public class Elevator
{
    Attributes
    Events
    1-Verweis
    public Elevator(String label, int maxWeight)...
    1-Verweis
    public void Boarding(Passenger p)...
    1-Verweis
    private void CheckWeight()...
    1-Verweis
    public void AlarmTone(Passenger p)...
    0 Verweise
    public void Unboarding(Passenger p)...

    1-Verweis
    public static void TestElevator()
    {
        Elevator ele = new Elevator("2009/12345", 500);
        for (int i = 0; i < 7; i++)
        {
            int weight = 50 + i * 10;
            String name = "Max" + i;
            Passenger p = new Passenger(name, weight);
            ele.Boarding(p);
        }
    }
}
```

```
public class Passenger
{
    2 Verweise
    public String Name { get; }
    2 Verweise
    public int Weight { get; }
    1-Verweis
    public Passenger(String name, int weight)
    {
        this.Name = name;
        this.Weight = weight;
    }
}
```

```
Person Pessenger6 verlässt den Lift 2009/12345.
Event too much Weight: 560
```

Fire Alarm

Ausgabe:

```
Probealarm!! Schüler: Ferdinand begibt sich in den Hof 1
Probealarm!! Schüler: Franz begibt sich in den Hof 1
Probealarm!! Schüler: Maria begibt sich in den Hof 1
Probealarm!! Schüler: Fritz begibt sich in den Hof 2
Probealarm!! Schüler: Martha begibt sich in den Hof 2
Drücken Sie eine beliebige Taste . . .
```

Feueralarm

- Bei einem Feueralarm muss die Schule unverzüglichst geräumt werden. Auf Grund der großen Schüleranzahl müssen sich die Klassen aber an unterschiedlichen Plätzen einfinden.
- Der Feueralarm (Event der Klasse Schule) wird von einer Methode der Klasse Schule ausgelöst und muss bei einem jeden Schüler „registriert“ werden – dazu muss jedem Schüler im Konstruktor die Instanz der Schule mit übergeben werden, um auf das Event zugreifen zu können (nähere Details siehe weiter unten).

Fire

- Test with this samplecode

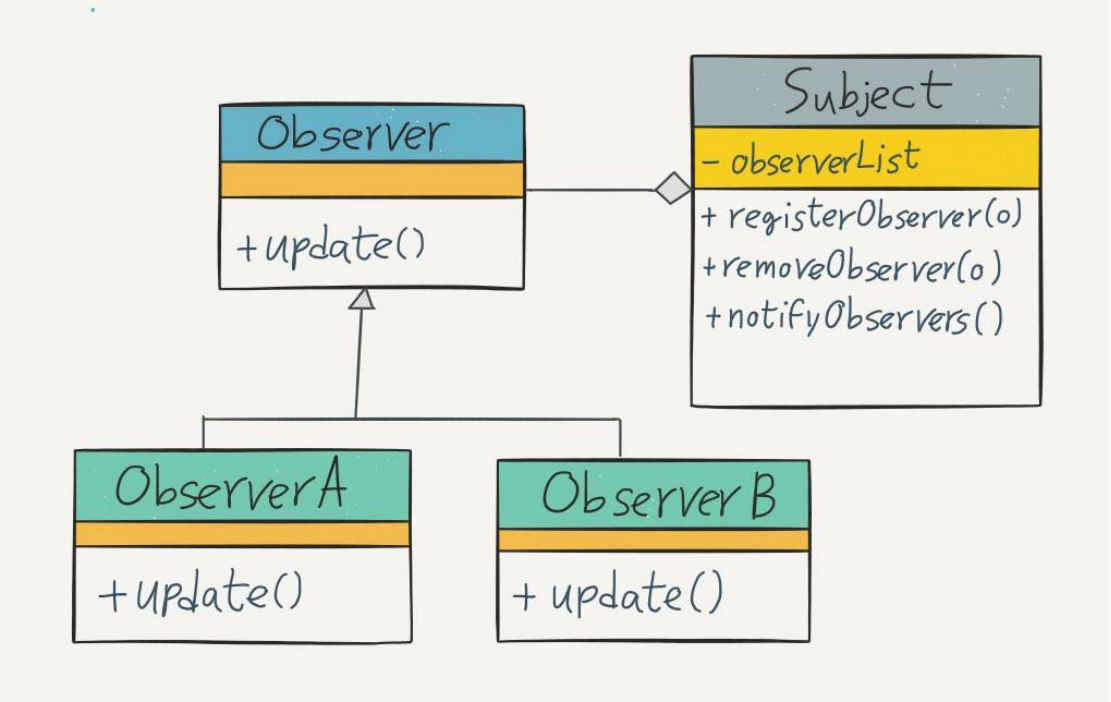
```
public class School
{
    public delegate void FireDelegate(String type);
    public event FireDelegate FireAlarm;
    public School() { }

    //Each Class gets a diffrent Number
    //write a switch case for some classes
    public int GetPlaceNumber(Clazz c)...

    //Raise the FireAlarm Event
    public void StartFireAlarm(String type)...
}

public class Clazz {
    public String Name { get; set; }
    public Clazz(String name) {  this.Name = name;  }
}
public class Pupil
{
    private String name;
    private Clazz clazz;
    private School school;

    //Subscribe the EscapeRoute to the FireAlarm Event in the Constructor
    public Pupil(String name, School school, Clazz clazz)...

    //Print: ALARM: Schüler {Name} begibt sich auf den Hof {Nr}
    //Use the Method GetPlaceNumber from School
    private void EscapeRoute(String type)...
}
```

```
public static void TestFireAlarm()
{
    School school = new School();
    Clazz clazz = new Clazz("3BHITT");
    Pupil s1 = new Pupil("Ferdinand", school, clazz);
    Pupil s2 = new Pupil("Franz", school, clazz);
    Pupil s3 = new Pupil("Maria", school, clazz);
    clazz = new Clazz("2BHITT");
    s1 = new Pupil("Fritz", school, clazz);
    s2 = new Pupil("Martha", school, clazz);
    school.StartFireAlarm("Probealarm");
}
```

Solution: School, Clazz, Pupil

```
public class School
{
    public delegate void FireDelegate(String type);
    public event FireDelegate FireAlarm;
    public School() { }

    //Each Class gets a diffrent Number
    //write a switch case for some classes
    public int GetPlaceNumber(Clazz c)
    {
        switch (c.Name)
        {
            case "3BHITT":
                return 1;
            case "2BHITT":
                return 2;
            default:
                return 5;
        }
    }

    //Raise the FireAlarm Event
    public void StartFireAlarm(String type)
    {
        FireAlarm(type);
    }
}
```

```
public class Clazz {
    public String Name { get; set; }
    public Clazz(String name) {  this.Name = name;  }
}
public class Pupil
{
    private String name;
    private Clazz clazz;
    private School school;

    //Subscribe the EscapeRoute to the FireAlarm Event in the Constructor
    public Pupil(String name, School school, Clazz clazz)
    {
        this.name = name;
        this.clazz = clazz;
        this.school = school;
        school.FireAlarm += EscapeRoute;
    }

    //Print: ALARM: Schüler {Name} begibt sich auf den Hof {Nr}
    //Use the Method GetPlaceNumber from School
    private void EscapeRoute(String type)
    {
        Console.WriteLine("{0}!! Schüler: {1} begibt sich in den Hof {2} ",
            type , name, school.GetPlaceNumber(clazz));
    }
}
```



Observer Pattern

https://www.philipphauer.de/study/se/design-pattern/observer.php

Löse Aufgabenstellung Zeitungsverlag