Delegates and Events

Callbacks

SoftUni Team

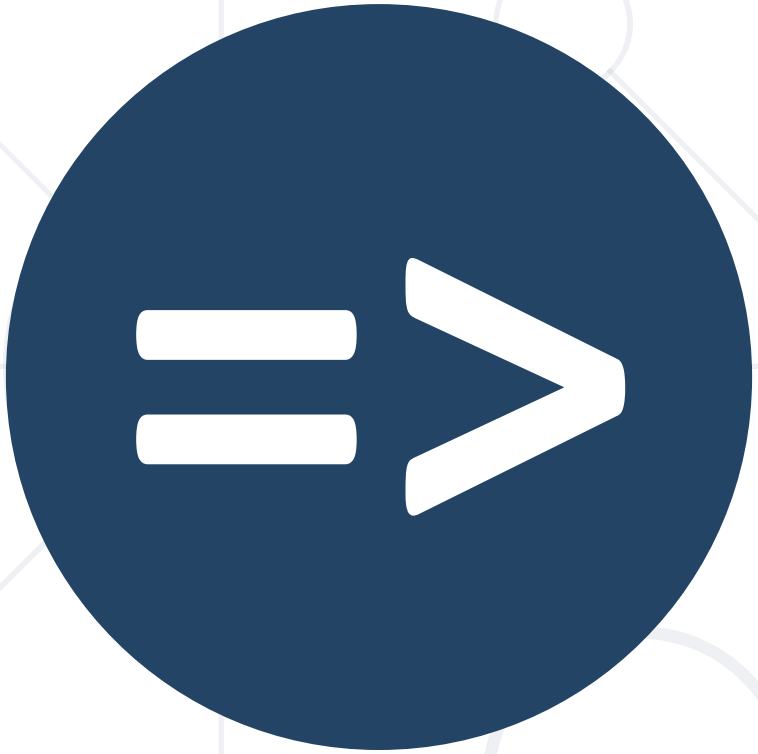
Technical Trainers

Software University

Software University

https://about.softuni.bg/

Table of Contents



Func<T, V>, Action<T>

Delegates

- A **delegate** is a type that represents references to methods with a particular parameter list and return type
- Used to pass **methods as arguments** to other methods
- Can be used to define callback methods

```
public delegate int Multiplier(int x, int y);
```

```
Multiplier calc = (x, y) => x * y;
```

Generic Delegates – Func<T, V>

- Initialization of a function

Lambda Expressions

Input type

Output type

```
Func<int, string> func = n => n.ToString();
```

Name

Input parameter

Return expression

- Input and output type can be **different types**
- Input and output type **must be from the declared type**
- **Func** generic delegate uses type parameters to define the number and types of input parameters and returns the type of the delegate

Generic Delegates – Action<T>

- In .NET **Action<T>** is a void method:

```
private void Print(string message)
{  Console.WriteLine(message);  }
```

- Instead of writing the method we can do:

```
Action<string> print =
  message => Console.WriteLine(message);
```

- Then we use it like that:

```
print("pesho");        // pesho
print(5.ToString()); // 5
```

Problem: Sum Numbers

- Read numbers from the console
- Use your own **function to parse** each element
- Print the **count** of numbers
- Print the **sum**

```
4, 2, 1, 3, 5, 7, 1, 4, 2, 12
```

```
10
41
```

```
85, 47, 91, 32, 83, 75, 81, 2
```

```
8
496
```

Solution: Sum Numbers

```
string input = Console.ReadLine();
Func<string, int> parser = n => int.Parse(n);
int[] numbers = input.Split(new string[] {", "},
    StringSplitOptions.RemoveEmptyEntries)
  .Select(parser).ToArray();
Console.WriteLine(numbers.Length);
Console.WriteLine(numbers.Sum());
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3168#1

Problem: Extract Uppercase Words

- Read a text from the console
- Filter only words, that **start** with a **capital** letter
- Use **`Func<string, bool>`** (predicate)
- Print each of the words on a new line

```
The following example shows
how to use Predicate
```

→

```
The
Predicate
```

```
Print count of words
```

→

```
Print
```

Solution: Extract Uppercase Words

```
Func<string, bool> checker =
  n => n[0] == n.ToUpper()[0];
var words = Console.ReadLine()
  .Split(new string[] {" "},
     StringSplitOptions.RemoveEmptyEntries)
  .Where(checker)
  .ToArray();
foreach (string word in words)
{
  Console.WriteLine(word);
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3168#2

Problem: Add VAT

- Read from the console **prices of items**
- Add **VAT** of 20% to all of them
- Use **`Func<double, double>`** (unary operator)

```
1, 3, 5, 7
```

```
1.20
3.60
6.00
8.40
```

```
10, 24, 12, 71
```

```
12.00
28.80
14.40
85.20
```

Solution: Add VAT

```
Func<double, double> addVat = p => p * 1.2;
double[] prices = Console.ReadLine()
    .Split(new string[] { ", " },
        StringSplitOptions.RemoveEmptyEntries)
    .Select(double.Parse)
    .Select(addVat)
    .ToArray();
foreach (var price in prices)
  Console.WriteLine($"{price:f2}");
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3168#3

Passing Functions to Method

- We can pass **Func<T>** to methods:

```
private int Operation(int number, Func<int, int> operation)
{
  return operation(number);
}
```

- We can use the method like that:

```
int a = 5;
int b = Operation(a, number => number * 5); // 25
int c = Operation(a, number => number - 3); // 2
int d = Operation(b, number => number % 2); // 1
```

Problem: Filter by Age

- Read from the console **n people** with their **age**
- Read a **condition** ("**older**" or "**younger**") and an age **filter**
- Read a **format type** for output and filter the people

```
4
Lucas, 20
Mia, 29
Noah, 31
Simo, 16
older
20
name age
```

```
Lucas - 20
Mia - 29
Noah - 31
```

```
4
Lucas, 20
Noah, 18
Mia, 29
Simo, 16
younger
20
name
```

```
Noah
Simo
```

Solution: Filter by Age (1)

```
// TODO: Read data from the console
Func<int, bool> tester = CreateTester(condition, age);
Action<KeyValuePair<string, int>> printer =
  CreatePrinter(format);
PrintFilteredStudent(people, tester, printer);
```

```
public static Func<int, bool> CreateTester
  (string condition, int age)
{
  switch (condition) {
    case "younger": return x => x < age;
    case "older": return x => x >= age;
    default: return null;
  }
}
```

Solution: Filter by Age (2)

```
public static Action<KeyValuePair<string, int>>
  CreatePrinter(string format)
{
  switch (format)
    case "name":
      return person => Console.WriteLine($"{person.Key}");
    // TODO: complete the other cases
    default: return null;
}
```

Check your solution here: https://judge.softuni.bg/Contests/Practice/Index/3168#4

Higher Order Functions – Examples

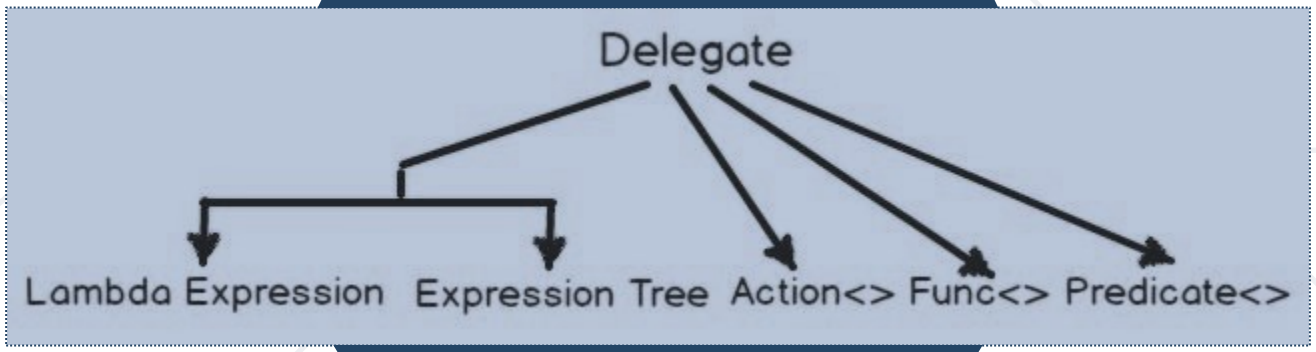
- **Higher-order functions** take other functions as arguments

```
int Aggregate(int start, int end, Func<int, int, int> func) {
  int result = start;
  for (int i = start + 1; i <= end; i++)
    result = func(result, i);
  return result;
}
```

```
Aggregate(1, 10, (a, b) => a + b)  // 55
```

```
Aggregate(1, 10, (a, b) => a * b)  // 3628800
```

```
Aggregate(1, 10, (a, b) => '' + a + b)  // "12345678910"
```



Predefined Boolean Delegates

Predicates

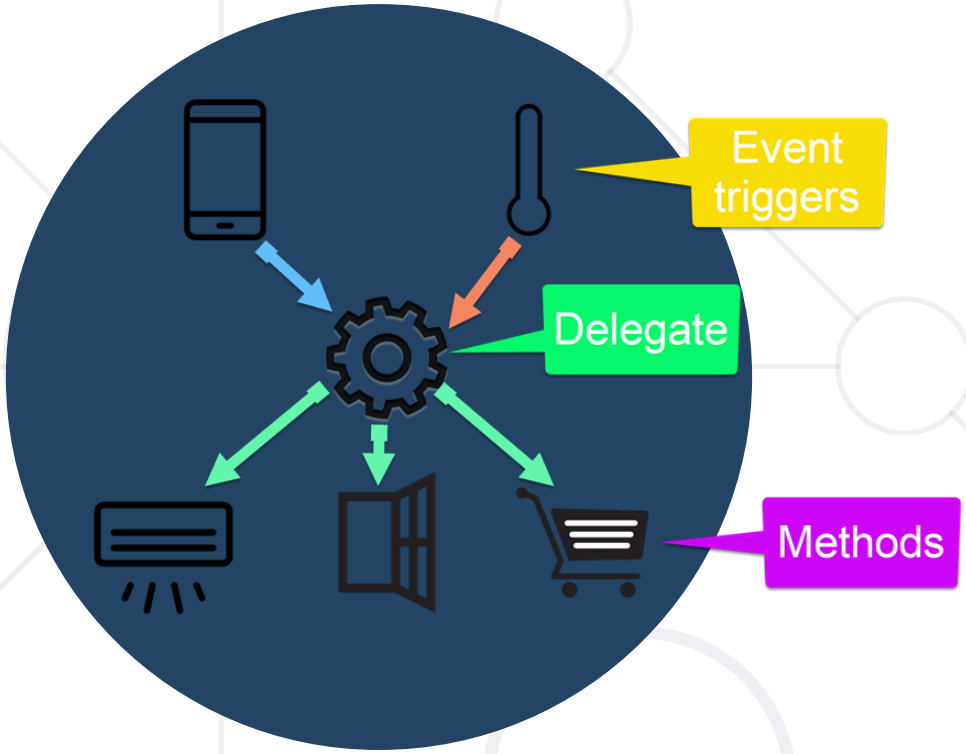
- Predicates are **predefined Boolean delegates** with the following signature

```
public delegate bool Predicate<T>(T obj)
```

- Define a way to check if an object meets some **Boolean** criteria
- Used by many methods in Array and List<T> to search for an element
- For example, **IList<T>.FindAll(Predicate<T>)** retrieves all elements meeting the criteria defined by the predicate

Predicates – Example

```
List<string> towns = new List<string>()
{
    "Sofia", "Burgas", "Plovdiv", "Varna",
    "Ruse", "Sopot", "Silistra"
};

List<string> townsWithS =
    towns.FindAll(delegate(string town)
    {
        return town.StartsWith("S");
    });

foreach (string town in townsWithS)
{
    Console.WriteLine(town);
}
```



Events and EventHandler

Events

- **Events** are **user actions** such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications
- **Events** are declared in a class and associated with the event **handlers** using **delegates**
- To receive an event, the event receivers should first "**subscribe to the event**"

Declaring Events (1)

```
public delegate void Notify();

class ProcessBusinessLogic {

  public event Notify ProcessCompleted; // event

  public void StartProcess() {
    Console.WriteLine("Process Started!");

    OnProcessCompleted();
  }

  protected virtual void OnProcessCompleted() {
    //if ProcessCompleted is not null then call delegate
    ProcessCompleted?.Invoke();
  }
}
```

First declare delegate type

Then, declare the event

Declaring Events (2)

```
static void Main()
{
  ProcessBusinessLogic bl = new ProcessBusinessLogic();
  bl.ProcessCompleted += bl_ProcessCompleted;
  bl.StartProcess();
}
public static void bl_ProcessCompleted()
{
  Console.WriteLine("Process Completed!");
}
```

register with an event

Microsoft Visual Studio Debug Console

```
Process Started!
Process Completed!
```

Subscribes and Unsubscribes

- The C# compiler automatically defines the **+=** and **-=** operators for events
  - **+= subscribes** for an event
  - **-= unsubscribes** for an event
- No other operations are allowed

The System.EventHandler Delegate (1)

- **System.EventHandler** defines a reference to a callback method, which handles events
- EventHandler represents a **method** with the **signature** of (object sender, EventArgs e) **returning void**
- No additional information is sent about the event, just a notification:

```
public delegate void EventHandler(object sender, EventArgs e);
```

- The **EventArgs** class is the **base class** with no information for the event

The System.EventHandler Delegate (2)

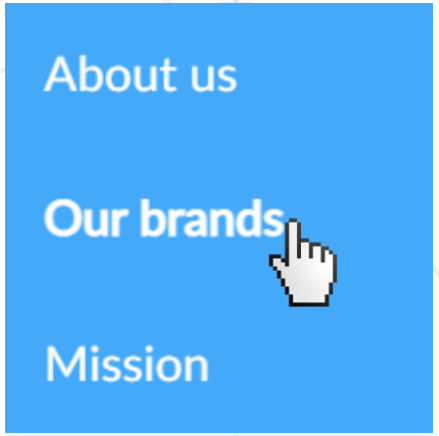
```
public class Button {
  public event EventHandler Click;
  public event EventHandler GotFocus; // And other types
}

public class ButtonExample {
  private static void OnButtonClick(object sender, EventArgs eArgs)
  {
    Console.WriteLine("OnButtonClick() event called.");
  }
```

```
public static void Main() {
  Button button = new Button();
  button.Click += new EventHandler(OnButtonClick);
}
```

Events in User Interfaces

- Events are **widely** used in Graphical User Interfaces (**GUIs**)
- Components such as buttons define a set of events (**OnClick**, **OnFocus**, **OnChange**, etc)
- External components can **subscribe** (listen) to a specific event **and react** to it

```
var button = GetButtonById("btn");
button.OnClick += (sender, args) =>
{
  // Code will be executed when button is clicked
};
```



UI Mouse Click Event Handler – Example

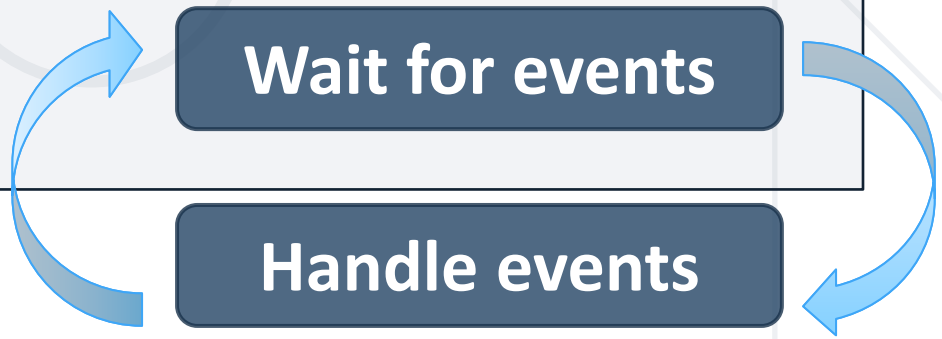
```
public partial class MainWindow : Window
{
  public MainWindow()
  {
    this.InitializeComponent();
    this.MouseDown += this.MainWindow_MouseClick;
  }

  private void MainWindow_MouseClick(
    object sender, MouseButtonEventArgs e)
  {
    MessageBox.Show(string.Format("Mouse clicked at ({0}, {1})",
    e.MouseDevice.GetPosition(this).X,
    e.MouseDevice.GetPosition(this).Y));
  }
}
```

Receives info about the click

Event Loop

- UI technologies usually have an **event loop** running
  - **Waits** for events from the underlying operating system **and notifies** the respective components

```
while (message != "quit")
{
  // Blocking operation – waits for an event from OS
  message = GetMessage();
  ProcessMessage(message);
}
```



Problem: Console Key Event

- Write program that, when you **press** the **[a]** or **[b]** keyboard key, **fires** an **event** that **writes** in color on the console the following message:
- You pressed the 'A' key.
- You pressed the 'B' key.
- No event handler for key {key}

```
a
You pressed the 'A' key.
b
You pressed the 'B' key.
h
No event handler for key h.
```

Solution: Console Key Event (1)

```
public delegate void PressKeyEvent();
public class Keyboard
{
  public event PressKeyEvent PressKeyA = null;
  public event PressKeyEvent PressKeyB = null;

  public void PressKeyAEvent()
  {
    if (PressKeyA != null) { PressKeyA.Invoke(); }
  }
  public void PressKeyBEvent()
  {
    if (PressKeyB != null) { PressKeyB.Invoke();
  }
}
```

Code example continues

Solution: Console Key Event (2)

```
  public void Start() {
    while (true) {
      string keyPressed = Console.ReadLine();
      switch (keyPressed) {
        case "a": PressKeyAEvent(); break;
        case "b": PressKeyBEvent(); break;
        default:
          Console.WriteLine("No event handler for key {0}.“
            , keyPressed); break;
      }
    }
  }
}
```

Solution: Console Key Event (3)

```
 static void Main()
{
  Keyboard keyboard = new Keyboard();

  keyboard.PressKeyA += new PressKeyEvent(PressKeyAWriter);
  keyboard.PressKeyB += PressKeyBWriter;

  while (true)
  {
    keyboard.Start();
  }
}
```

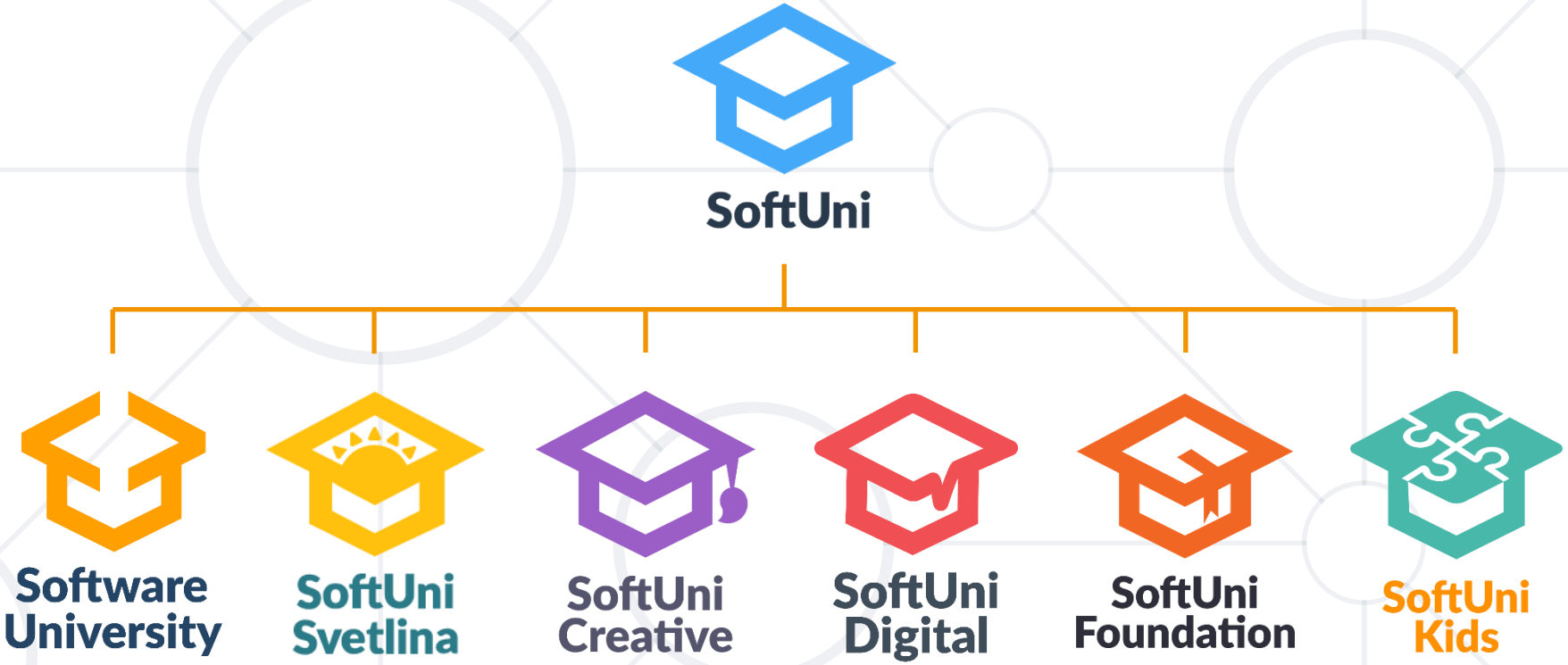
Solution: Console Key Event (4)

```
static private void PressKeyAWriter() {
  Console.ForegroundColor = ConsoleColor.Blue;
  Console.WriteLine("You pressed the 'A' key.");
  Console.ForegroundColor = ConsoleColor.Gray;
}
static private void PressKeyBWriter() {
  Console.ForegroundColor = ConsoleColor.Green;
  Console.WriteLine("You pressed the 'B' key.");
  Console.ForegroundColor = ConsoleColor.Gray;
}
```

Summary

- Delegates are **data types** that **hold methods** as their value
- **Some generic delegates** in C#:
  - Action<T>, Func<T, TResult> and Predicate<T>
- **Events** allow **subscribing for notifications** about something happening in an object
- When an **event "happens",** all subscribers are **notified**

Questions?



License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg