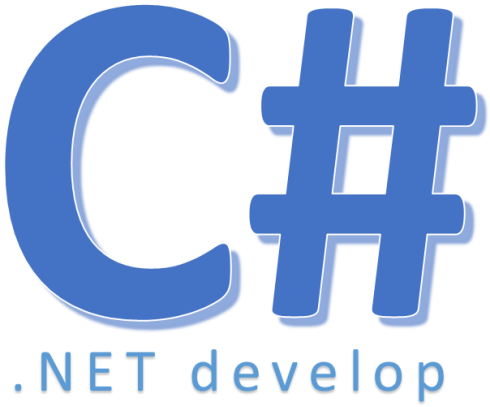
Fachhochschule Nordwestschweiz
Hochschule für Technik

DELEGATES & EVENTS



Martin Kropp, Yves Senn
University of Applied Sciences Northwestern Switzerland

Learning Targets

- You
  - can explain the concepts Delegates & Events, Predicates
  - can compare Delegates, Events, and Predicates with the corresponding concepts in the Java language
  - can explain the differences between Action, Function and Predicate Delegates
  - can apply the concepts efficiently for software development

Content

- Delegates
- Event Handling in C# using Delegates
- Predefined Delegates

Need for delegates

```
interface ISaySomething
{
    void SaySomething() { Console.WriteLine(“Something"); }
}

class SayHello : ISaySomething
{
    public void SaySomething() { Console.WriteLine("Hello"); }
}


// call whatever ISaySomething implementation is provided
static void DoSomething(ISaySomething saySomething)
{
    saySomething.SaySomething();
}
```

Contract

Contract Fulfillment

Implementation

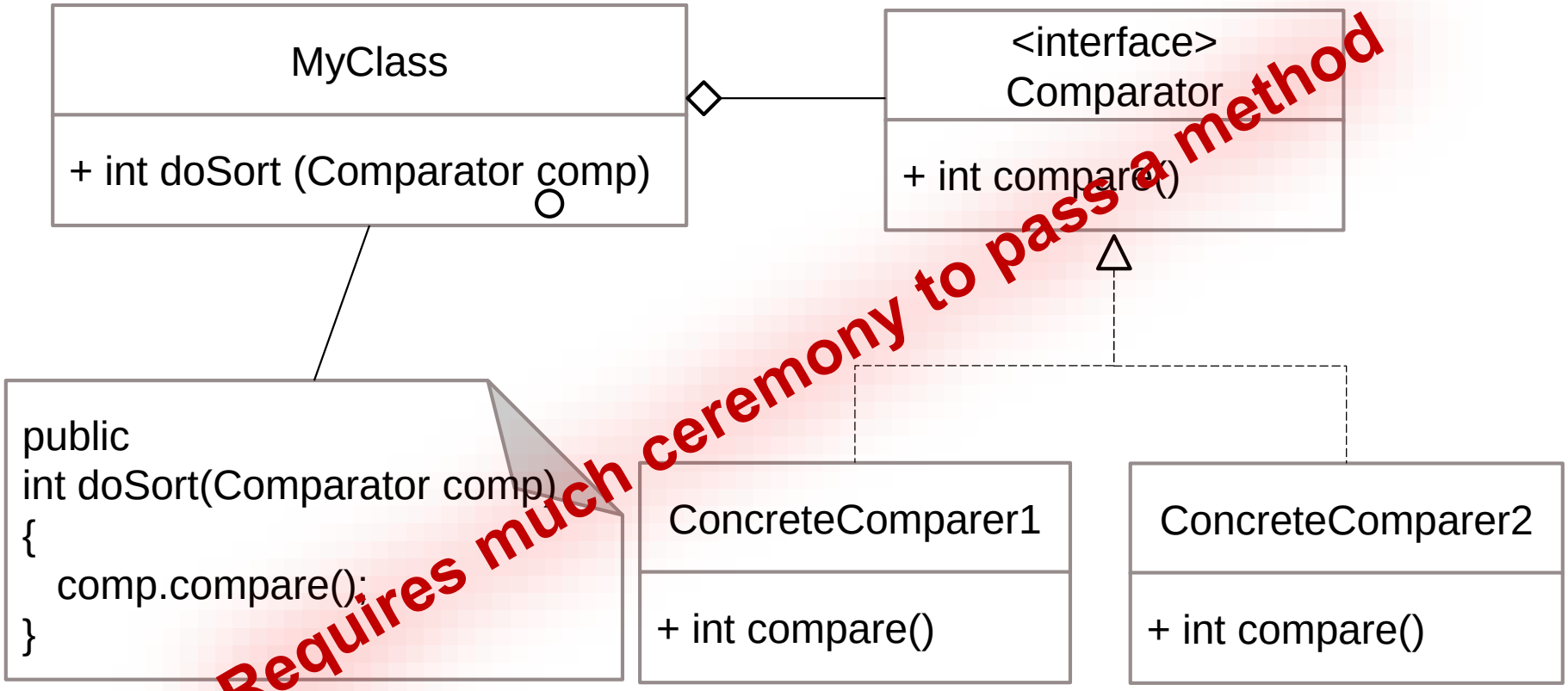
Usage

A Case Study

"A program needs to sort data by different comparison criteria, using different compare methods."

This leads to surprising complexity if implemented the traditional object oriented-way.

The traditional Object Oriented-way



→ It'd be much simpler to just pass the method as parameter

The traditional Object Oriented-way

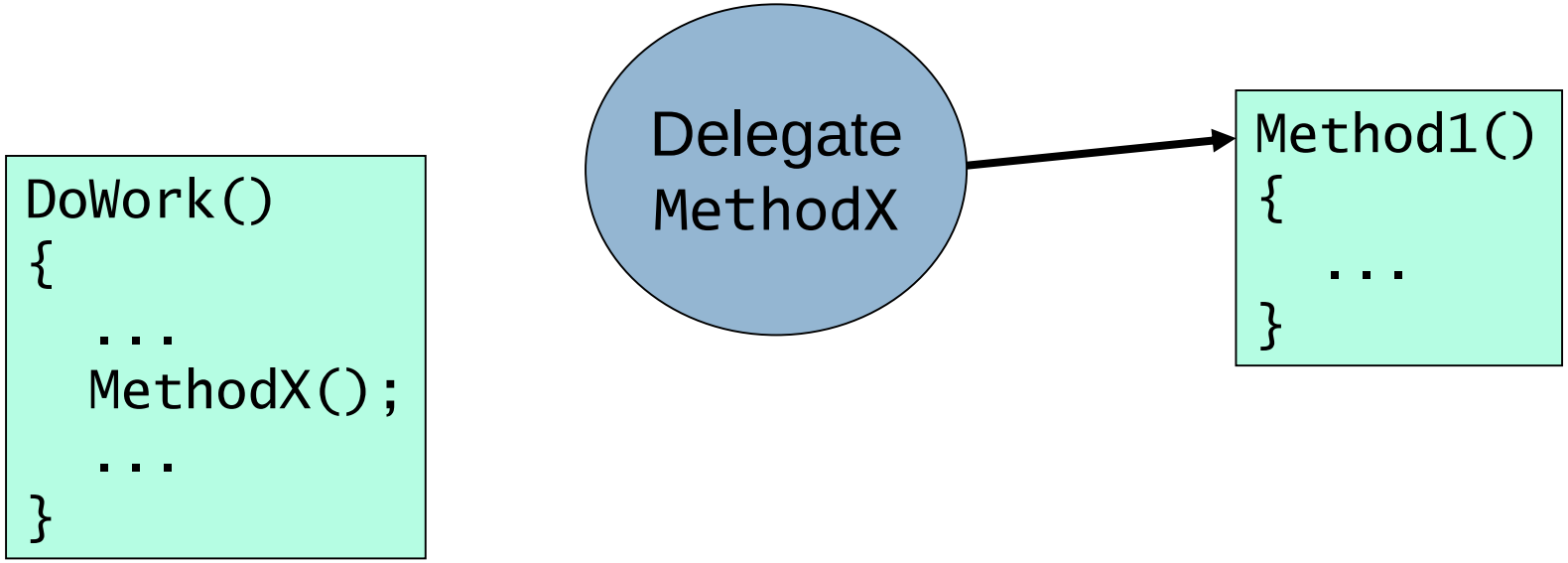
You have to provide a full implementation of the Comparator interface, even if you only care about the compare-method:

```
class MyComparator implements Comparator<Integer>
{
    @Override
    public int compare(Integer o1, Integer o2)
    {
       return (o1>o2 ? -1 : (o1==o2 ? 0 : 1));
    }
}
```

```
Collections.sort(list, new MyComparator() );
```

The C# Way

A delegate is a **reference to a method**:



Use of Delegates

No need to encapsulate the method within a class!

```
public delegate void SaySomething();

static void SayHello()
{
   Console.WriteLine("Hello");
}

// the delegate is passed as an argument
static void DoSomething(SaySomething saySomethi
{
    saySomething();
}

DoSomething(SayHello);
```

Contract

Contract Fulfillment

Implementation

Usage

How to use Delegates

1. Declare a delegate *type*

```
delegate void AMethodWithAStringParameter(string param1);
```

2. Declare a delegate *variable*

```
AMethodWithAStringParameter a;
```

3. Implement a method with the above method signature

```
void SayHello(string name)
{
    Console.WriteLine($"Hello {name}");
}
```

4. Use the delegate variable

```
a = SayHello;
a("Kurt");                          // Hello Kurt
```

Sorting - The C#-way

- The Sort-Method signature

```
// public delegate int Comparison<in T>(T x, T y);

List<T>.Sort(Comparison<T> comparison);
```

→ You can pass any method with the delegate signature

- Use a method (a “delegate”)

```
static int MyComparer(int i1, int i2)
{
    return i1.CompareTo(i2);
};

list.Sort(MyComparer);
```

→ Lambdas make this even simpler, see upcoming weeks ☺

Delegates can be generic

1. Declare a **generic** delegate *type*

```
delegate void AMethodWithGenericParameter<T>(T param1);
```

2. Declare a delegate *variable*, ***that requires a string***

```
AMethodWithGenericParameter<string> a;
```

3. Implement a method with the above method signature

```
void SayHello(string name) { ... }
{
    Console.WriteLine($"Hello {name}");
}
```

4. Use the delegate variable

```
a = SayHello;
a("Generic Kurt!");                    // Hello Generic Kurt!
```

When to use delegates

Use delegates when you would use a single-method class otherwise:

- Comparers
- Event-handlers
- Mappings
- Callbacks
- ...

Characteristics of delegates

- Delegate are object-oriented, type-safe function pointers
- Methods may be static or non-static
- Delegate implementations must match the delegate type (*fulfil the contract*)
  - same parameters
  - same return type
- Delegates may be passed as parameters
- Foundation of .NET event handling
- A delegate stores methods, but not parameter values!
- Delegates are immutable
- Delegates are reference types

Worksheet – Part 1

Delegates are multicast-able

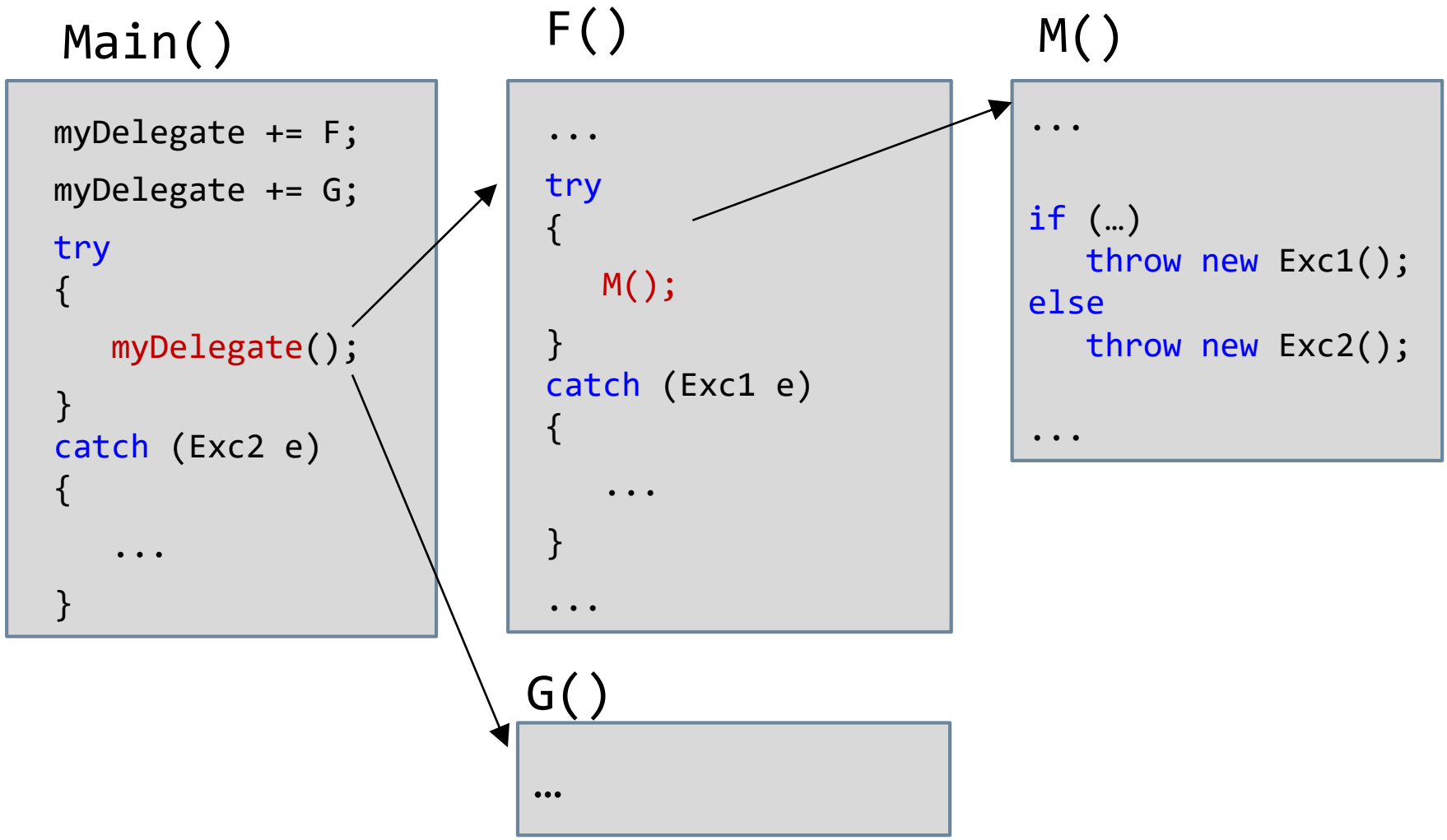
```
static void SayHello(string name)
{
    Console.WriteLine($"Hello {name}");
}

static void SayBye(string name)
{
    Console.WriteLine($"Bye {name}");
}


…
{
    AMethodWithAStringParameter a = SayHello;
    a += SayHello;
    a += SayBye;
    a -= SayHello;
    a("Kurt");
```

The base for event handling

Exception Handling with Delegates



Avoid return values

- If the multicast delegate returns a value, the value of the last call is returned
- If the multicast delegate has an `out` parameter, the parameter of the last call is returned
- `ref` parameters are passed through all invocations

→ Unmaintainable

→ Don't use return values!

Events

- Events and delegates are (almost) the same thing
- The event keyword is an ***access limiting modifier*** for delegates:

```
delegate void SaySomething(); // delegate type definition


SaySomething saySomethingDelegate; // instance of a delegate
```

vs.

```
// instance of an event with delegate type
event SaySomething saySomethingEvent;
```

Events

The `event` keyword limits the access of a delegate from outside:

- Only the declaring class can invoke it
- Other classes may append/remove event handlers, but not replace the whole list

Accessibility of Events

```
public class Model
{
    public delegate void SaySomething(string words);
    public event SaySomething SaySomethingEvent;
}

class Program
{
    static void SayHello(string name) { Console.WriteLine($"Hello {name}"); }
    static void Main(string[] args)
    {
        var m = new Model();

        // raise event
        m.SaySomethingEvent("Hello"); // won't compile

        // register event
        m.SaySomethingEvent += SayHello; // compiles
        m.SaySomethingEvent = SayHello; // won't compile
    }
}
```

Firing events

```
public class Model
{
    public delegate void SaySomething(string words);
    public event SaySomething SaySomethingEvent;

    public void FireSaySomething(string words)
    {
        SaySomethingEvent(words) // not safe, could have no listeners!
        SaySomethingEvent?.Invoke(words);
    }
}
```

Null-Conditional Operator

Events without subscribers are `null`!

Event conventions

```
delegate void SomeEvent(object source, SomeEventArgs e);
```

1. Result type: `void`
2. First parameter: Source of the event
3. Second parameter: Event arguments. The class `SomeEventArgs` must be a subclass of `System.EventArgs` and its name must end with `…EventArgs`

→ Convention to pass context (**source**) and related information (**arguments**) to their listeners

.NET offers the **predefined, generic delegate type** `EventHandler<SomeEventArgs>` for convenience.

Worksheet – Part 2

Predefined delegate types

- `Action<in T1, in T2, ...>`
  Parameters T1...Tn, returns nothing (void)

- `Func<in T1, in T2, ..., out TResult>`
  Parameters T1...Tn, returns `TResult`

- `Predicate<in T>`
  Single parameter T, returns `bool`

Action delegates

- Generic delegate type for methods with any parameters and no return value

```
delegate void Action ();
delegate void Action<in T1> (T1 arg);
delegate void Action<in T1, in T2> (T1 arg1, T2 arg2);
…
```

- Example

```
private static void ActionDelegateExample()
{
    Action<string> act = ShowMessage;
    act("C# language");
}

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

Func delegates

- Generic delegate type for methods with any parameters and a return value

```
delegate TResult Func<out TResult> ();
delegate TResult Func<in T1, out TResult> (T1 arg);
delegate TResult Func<in T1, in T2, out TResult> (T1 arg1, T2 arg2);
…
```

- Example

```
public void FuncDelegateExample()
{
      Func<string, string> convertMethod = UppercaseString;
      Console.WriteLine(convertMethod("Dakota"));
}

private string UppercaseString(string inputString)
{
       return inputString.ToUpper();
}
```

Predicate delegates

- Generic delegate type for methods with a single parameter and a return type bool

```
class List<T> {
    List<T> FindAll(Predicate<T> match);
    T Find(Predicate<T> match);
    //...
}

bool GreaterThan10(int x) {
    return x > 10;
}

void Main() {
    var listOfNumbers = new int[] {1, 2, 25, 3, 11}.ToList();
    var firstMatch = listOfNumbers.Find(GreaterThan10);
    //...
}
```

Worksheet – Part 3