Факултет Техничких Наука
Универзитет у Новом Саду

Виртуелизација процеса
Delegate-и и *Event*-и

Нови Сад, 2023.

Delegates

- Делегати представљају показиваче на методе које имају исте аргументе и повратну вредност као сам делегат
- Делегати се користе за прослеђивање метода као аргумента другим методама
- Пример декларације делегата:

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

- Било која метода из било које доступне класе или структуре која одговара типу делегата може бити додељена делегату

Delegates

- Метода може бити статичка или метода инстанце
- Метода не мора да има повратну вредност:

```
public delegate void Del(string message);
```

- Параметри које је позивалац проследио делегату ће се проследити методи
- Ако постоји повратна вредност из методе делегат је враћа позиваоцу

Delegates

```
public delegate void Del(string message);

// Create a method for a delegate.
public static void DelegateMethod(string message)
{
        Console.WriteLine(message);
}

// Instantiate the delegate.
Del handler = DelegateMethod;

// Call the delegate.
handler("Hello World");
```

Delegates

- Делегат не мора да буде једини аргумент прослеђен методи:

```
public static void MethodWithCallback(int param1, int param2, Del callback)
{
        callback("The number is: " + (param1 + param2).ToString());
}

MethodWithCallback(1, 2, handler);
```

- Шта је резултат овог позива?
- Често се користе у алгоритмима за сортирање

Delegates

- Делегат има само информације о методи на коју показује
- Делегат може да се позива на било који тип објекта све док постоји метод на томе објекту који одговара потпису делегата
- Делегат може да позове више од једног метода
- Позивање више метода зе назива *multicasting*
- Проширивање/скраћивање листе метода које се позивају кроз позив делегата врши се помоћу оператора + и -

Delegates

- Ако поред `public delegate void Del(string message);` имамо и следеће методе:

```
public class MethodClass
{
        public void Method1(string message) { }
        public void Method2(string message) { }
}
```

- Прављење листе метода се врши на следећи начин:

```
var obj = new MethodClass();
Del d1 = obj.Method1;
Del d2 = obj.Method2;
Del d3 = DelegateMethod;
//Both types of assignment are valid.
Del allMethodsDelegate = d1 + d2;
allMethodsDelegate += d3;
```

Delegates

- Када се позива *allMethodsDelegate* позивају се све три методе редом
- Свака промјена претходне методе је видљива у наредним методама
- Ако нека од њих баци изузетак он се прослеђује позиваоцу делегата
- Уклањање метода из листе делегата:

```
//remove Method1
allMethodsDelegate -= d1;
// copy AllMethodsDelegate while removing d2
Del oneMethodDelegate = allMethodsDelegate - d2;
```

Delegates

```
delegate void Del(int i, double j);
class MathClass
{
        static void Main()
        {
                MathClass m = new MathClass();
                Del d = m.MultiplyNumbers;
                Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");
                        for (int i = 1; i <= 5; i++) { d(i, 2); }
                Console.WriteLine("Press any key to exit.");  Console.ReadKey();
        }
        void MultiplyNumbers(int m, double n)
         {
                Console.Write(m * n + " ");
        }
 }
```

- Шта ће бити испис?

Delegates

```
delegate void Del();
class SampleClass
{
        public void InstanceMethod()
        {
                Console.WriteLine("A message from the instance method.");
        }
         static public void StaticMethod()
        {
                Console.WriteLine("A message from the static method.");
        }
}
class TestSampleClass
 {
        static void Main()
         {
                var sc = new SampleClass();
                Del d = sc.InstanceMethod; d();
                d = SampleClass.StaticMethod; d();
        }
}
```

- Шта ће бити испис?

Delegates

```
delegate void CustomDel(string s);
class TestClass
{
        static void Hello(string s)
        {
                Console.WriteLine($" Hello, {s}!");
        }
        static void Goodbye(string s)
        {
                Console.WriteLine($" Goodbye, {s}!");
        }
        static void Main()
        {
                …
        }
}
```

```
…
static void Main()
{
        CustomDel hiDel, byeDel, multiDel, multiMinusHiDel;
        hiDel = Hello;
        byeDel = Goodbye;
        multiDel = hiDel + byeDel;
        multiMinusHiDel = multiDel - hiDel;
        Console.WriteLine("Invoking delegate hiDel:");
        hiDel("A");
        Console.WriteLine("Invoking delegate byeDel:");
        byeDel("B");
        Console.WriteLine("Invoking delegate multiDel:");
        multiDel("C");
        Console.WriteLine("Invoking delegate multiMinusHiDel:");
        multiMinusHiDel("D");
}
```

- Шта ће бити испис?

Delegates

```
// A set of classes for handling a bookstore:
namespace Bookstore
{
    using System.Collections;

    // 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 ProcessBookCallback(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(ProcessBookCallback processBook)
        {
            foreach (Book b in list)
            {
                if (b.Paperback)
                    // Calling the delegate:
                    processBook(b);
            }
        }
    }
}
```

```
// Using the Bookstore classes:
namespace BookTestClient
{
    using Bookstore;

    // 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;
        }
    }

    // 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.
            bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, tr
            bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);
            bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true)
        }
```

- Шта ће бити испис?

Events

- У свету конзолних апликација редослед интеракције са корисником је био унапред дефинисан приликом писања програма
- Након појаве графичке корисничке спреге (*Graphical User Interface – GUI*) корисник добија слободу интеракције са програмом (нпр. може да кликне на било које дугме)
- Појавио се проблем у комуникацији програмских компоненти
- Намеће се питање како обавестити компоненте шта је корисник урадио

Events

- Догађаји (*events*) служе да се објекат неке класе обавести да се десило нешто од интереса за тај објекат

- Код догађаја увек имамо:
  - Класу која генерише догађај (генератор догађаја, *publisher*)
  - Касу која жели да буде обавештена о неком догађају (потрошач, *subscriber*)

- Класа која имплементира контролу корисничког интерфејса може дефинисати догађај који се догоди када корисник направи леви клик мишем

- Класу не занима шта ће се десити након клика корисника, али је дужна да обавести друге класе, потрошаче, да се догађај десио

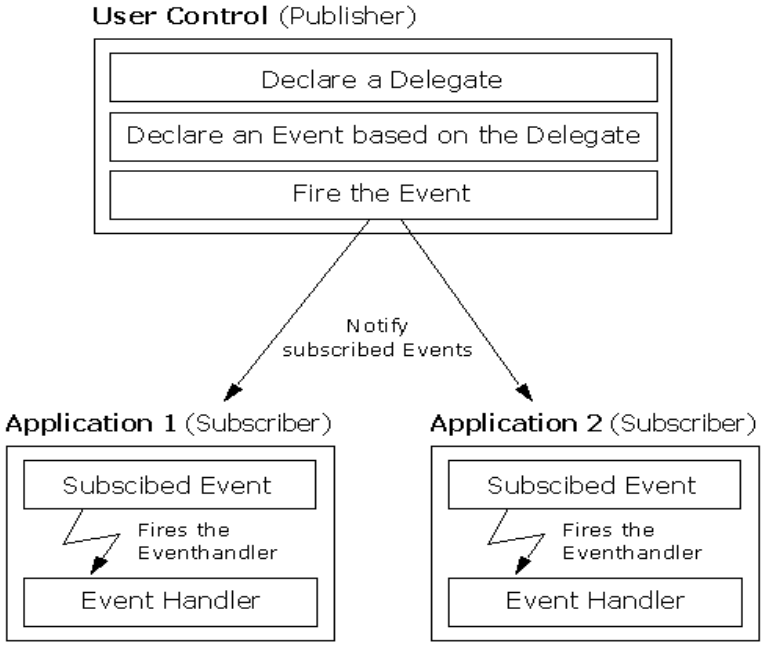
- Обавештене класе треба да имплементирају даљу логику обраде догађаја

Events

- Догађаји могу да буду и различите системске нотификација на коју апликација треба да одреагује
- Механизам догађаја користи *publisher-subscriber* развојни узорак (*design pattern*)
- *Publisher* је објекат који садржи дефиницију догађаја и делегат који имплементира руковаоца догађајем (*event handler*)
- *Subscriber* је објекат који прихвата догађај и врши његову имплементацију
- Догађаји се дефинише делегатом:

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

Events


User Control (Publisher)
Declare a Delegate
Declare an Event based on the Delegate
Fire the Event
Notify
subscribed Events
Application 1 (Subscriber)
Subscibed Event
Fires the
Eventhandler
Event Handler
Application 2 (Subscriber)
Subscibed Event
Fires the
Eventhandler
Event Handler

Events

- Приликом клика на дугме у оквиру ваше апликације креира се метода за обраду догађаја:

```
private void Form1_Load(object sender, System.EventArgs e)
{
        // Add your form load event handling code here.
}
```

- У методи *InitializeComponent* се аутоматски генерише ред који у суштини претставља саму претплату:

```
this.Load += new System.EventHandler(this.Form1_Load);
```

Events

- Ако је догађај заснован на типу делегата *EventHandler* имамо методу за обраду догађаја:

```
void HandleCustomEvent(object sender, CustomEventArgs a)
{
        // Do something useful here.
}
```

- Помоћу += вршимо претплату на догађај

- Ако имамо објекат са именом *publisher* који креира догађај под називом *RaiseCustomEvent* претплата се врши на следећи начин:

```
publisher.RaiseCustomEvent += HandleCustomEvent;
```

Events

- Сви догађаји у *.Net*-у засновани су на делегату *EventHandler* који је дефинисан на следећи начин:

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

- Да би отказали претплату на догађај позивамо следећу линију кода:

```
publisher.RaiseCustomEvent -= HandleCustomEvent;
```

- Док год не откажемо претплату *Garbage Collector* неће покупити ове ресурсе и имаћемо цурење меморије

- Када сви претплатници откажу претплату, инстанца догађаја у класи издавача треба да буде подешена на *null*

Events

```
// Define a class to hold custom event info
public class CustomEventArgs : EventArgs
{
    public CustomEventArgs(string message)
    {
        Message = message;
    }

    public string Message { get; set; }
}
```

```
// Class that publishes an event
class Publisher
{
    // Declare the event using EventHandler<T>
    public event EventHandler<CustomEventArgs> RaiseCustomEvent;

    public void DoSomething()
    {
        // Write some code that does something useful here
        // then raise the event. You can also raise an event
        // before you execute a block of code.
        OnRaiseCustomEvent(new CustomEventArgs("Event triggered"));
    }

    // Wrap event invocations inside a protected virtual method
    // to allow derived classes to override the event invocation behavior
    protected virtual void OnRaiseCustomEvent(CustomEventArgs e)
    {
        // Make a temporary copy of the event to avoid possibility of
        // a race condition if the last subscriber unsubscribes
        // immediately after the null check and before the event is raised.
        EventHandler<CustomEventArgs> raiseEvent = RaiseCustomEvent;

        // Event will be null if there are no subscribers
        if (raiseEvent != null)
        {
            // Format the string to send inside the CustomEventArgs parameter
            e.Message += $" at {DateTime.Now}";

            // Call to raise the event.
            raiseEvent(this, e);
        }
    }
}
```

```
//Class that subscribes to an event
class Subscriber
{
    private readonly string _id;

    public Subscriber(string id, Publisher pub)
    {
        _id = id;

        // Subscribe to the event
        pub.RaiseCustomEvent += HandleCustomEvent;
    }

    // Define what actions to take when the event is raised.
    void HandleCustomEvent(object sender, CustomEventArgs e)
    {
        Console.WriteLine($"{_id} received this message: {e.Message}");
    }
}

class Program
{
    static void Main()
    {
        var pub = new Publisher();
        var sub1 = new Subscriber("sub1", pub);
        var sub2 = new Subscriber("sub2", pub);

        // Call the method that raises the event.
        pub.DoSomething();

        // Keep the console window open
        Console.WriteLine("Press any key to continue...");
        Console.ReadLine();
    }
}
```

- Шта ће бити испис?

Events

```
namespace BaseClassEvents
{
    // Special EventArgs class to hold info about Shapes.
    public class ShapeEventArgs : EventArgs
    {
        public ShapeEventArgs(double area)
        {
            NewArea = area;
        }

        public double NewArea { get; }
    }

    // Base class event publisher
    public abstract class Shape
    {
        protected double _area;

        public double Area
        {
            get => _area;
            set => _area = value;
        }

        // The event. Note that by using the generic EventHandler<T> event type
        // we do not need to declare a separate delegate type.
        public event EventHandler<ShapeEventArgs> ShapeChanged;

        public abstract void Draw();

        //The event-invoking method that derived classes can override.
        protected virtual void OnShapeChanged(ShapeEventArgs e)
        {
            // Safely raise the event for all subscribers
            ShapeChanged?.Invoke(this, e);
        }
    }
```

```
public class Circle : Shape
{
    private double _radius;

    public Circle(double radius)
    {
        _radius = radius;
        _area = 3.14 * _radius * _radius;
    }

    public void Update(double d)
    {
        _radius = d;
        _area = 3.14 * _radius * _radius;
        OnShapeChanged(new ShapeEventArgs(_area));
    }

    protected override void OnShapeChanged(ShapeEventArgs e)
    {
        // Do any circle-specific processing here.

        // Call the base class event invocation method.
        base.OnShapeChanged(e);
    }

    public override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}
public class Rectangle : Shape
{
    private double _length;
    private double _width;

    public Rectangle(double length, double width)
    {
        _length = length;
        _width = width;
        _area = _length * _width;
    }

    public void Update(double length, double width)
    {
        _length = length;
        _width = width;
        _area = _length * _width;
        OnShapeChanged(new ShapeEventArgs(_area));
    }

    protected override void OnShapeChanged(ShapeEventArgs e)
    {
        // Do any rectangle-specific processing here.

        // Call the base class event invocation method.
        base.OnShapeChanged(e);
    }

    public override void Draw()
    {
        Console.WriteLine("Drawing a rectangle");
    }
}
```

```
public class ShapeContainer
{
    private readonly List<Shape> _list;

    public ShapeContainer()
    {
        _list = new List<Shape>();
    }

    public void AddShape(Shape shape)
    {
        _list.Add(shape);

        // Subscribe to the base class event.
        shape.ShapeChanged += HandleShapeChanged;
    }

    // ...Other methods to draw, resize, etc.

    private void HandleShapeChanged(object sender, ShapeEventArgs e)
    {
        if (sender is Shape shape)
        {
            // Diagnostic message for demonstration purposes.
            Console.WriteLine($"Received event. Shape area is now {e.NewArea}");

            // Redraw the shape here.
            shape.Draw();
        }
    }
}
class Test
{
    static void Main()
    {
        //Create the event publishers and subscriber
        var circle = new Circle(54);
        var rectangle = new Rectangle(12, 9);
        var container = new ShapeContainer();

        // Add the shapes to the container.
        container.AddShape(circle);
        container.AddShape(rectangle);

        // Cause some events to be raised.
        circle.Update(57);
        rectangle.Update(7, 7);

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to continue...");
        Console.ReadKey();
    }
}
```

- Шта ће бити испис?