Methods, Fields, Events, and Properties

Methods

- Methods define behavior
- Every method has a return type
  - `void` if no value returned
- Every method has zero or more parameters
  - Use `params` keyword to accept a variable number of parameters
- Every method has a signature
  - Name of method + parameters

```
public void WriteAsBytes(int value)
{
      byte[] bytes = BitConverter.GetBytes(value);

      foreach(byte b in bytes)
      {
          Console.Write("0x{0:X2} ", b);
      }
}
```

Method Overloading

- **Define multiple methods with the same name in a single class**
  - Methods require a unique signature
- **Compiler finds and invokes the best match**

```
public void WriteAsBytes(int value)
{
    // ...
}

public void WriteAsBytes(double value)
{
    // ...
}
```

Methods - Review

- **Instance methods versus static methods**
  - Instance methods invoked via object, static methods via type
- **Abstract methods**
  - Provide no implementation, implicitly virtual
- **Virtual methods**
  - Can override in a derived class
- **Partial methods**
  - Part of a partial class
- **Extension methods**
  - Described in the LINQ module

Fields

- **Fields are variables of a class**
  - Can be read-only

```
public class Animal
{
    private readonly string _name;

    public Animal(string name)
    {
        _name = name;
    }
}
```

Properties

- A property can define a get and/or set accessor
  - Often used to expose and control fields
- Auto-implemented properties use a hidden field

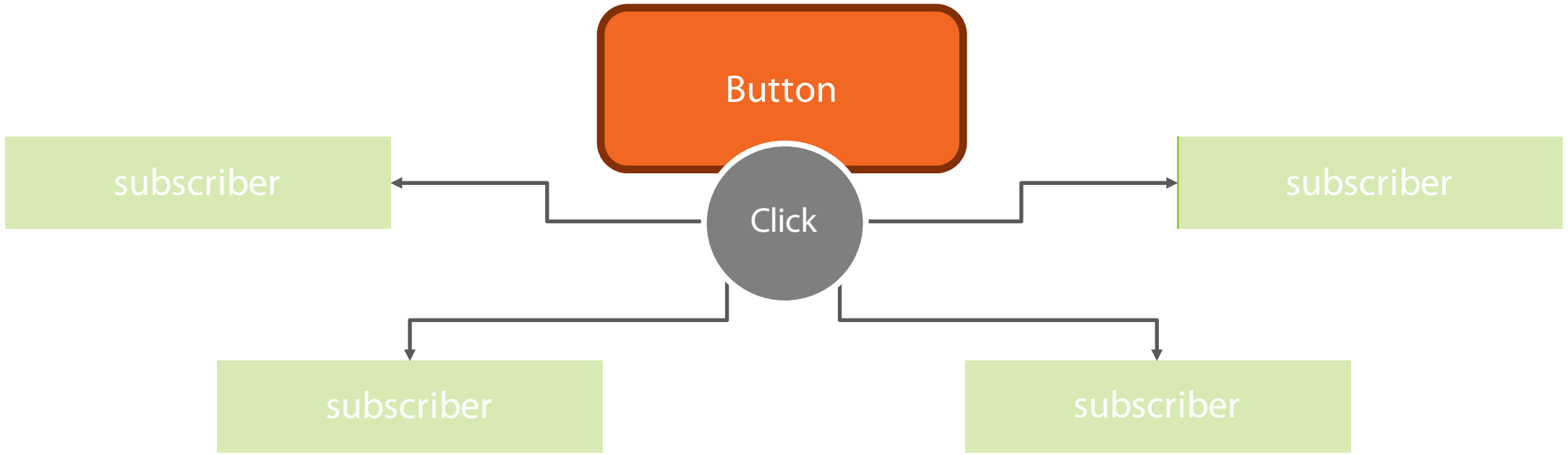
```
public string Name
{
    get;
    set;
}
```

```
private string _name;

public string Name
{
    get { return _name; }
    set
    {
        if(!String.IsNullOrEmpty(value))
        {
            _name = value;
        }
    }
}
```

Events

- **Allows a class to send notifications to other classes or objects**
  - Publisher raises the event
  - One or more subscribers process the event



Delegates

- I need a variable that references a method
- A delegate is a type that references methods

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

```
Logger logger = new Logger();
Writer writer = new Writer(logger.WriteMessage);
writer("Success!!");
```

```
{
    public void WriteMessage(String message)
    {
        Console.WriteLine(message);
    }
}
```

Subscribing To Events

- **Use the += and -= to attach and detach event handlers**
  - Can attached named or anonymous methods

```
public static void Initialize()
{
    _submitButton.Click += new RoutedEventHandler(_submitButton_Click);
}

static void _submitButton_Click(object sender, RoutedEventArgs e)
{
    // ... respond to event
}
```

Publishing Events

- Create custom event arguments (or use a built-in type)
  - Always derive from the base EventArgs class
- Define a delegate (or use a built-in delegate)
- Define an event in your class
- Raise the event at the appropriate time

```
public event NameChangingEventHandler NameChanging;

private bool OnNameChanging(string oldName, string newName)
{
    if(NameChanging != null)
    {
        NameChangingEventArgs args = new NameChangingEventArgs();
        args.Cancel = false;
        args.NewName = newName;
        args.OldName = oldName;
        NameChanging(this, args);
    }
}
```

Summary

- **Members are used to craft an abstraction**
  - Fields and properties for state
  - Methods for behavior
  - Events for notification