

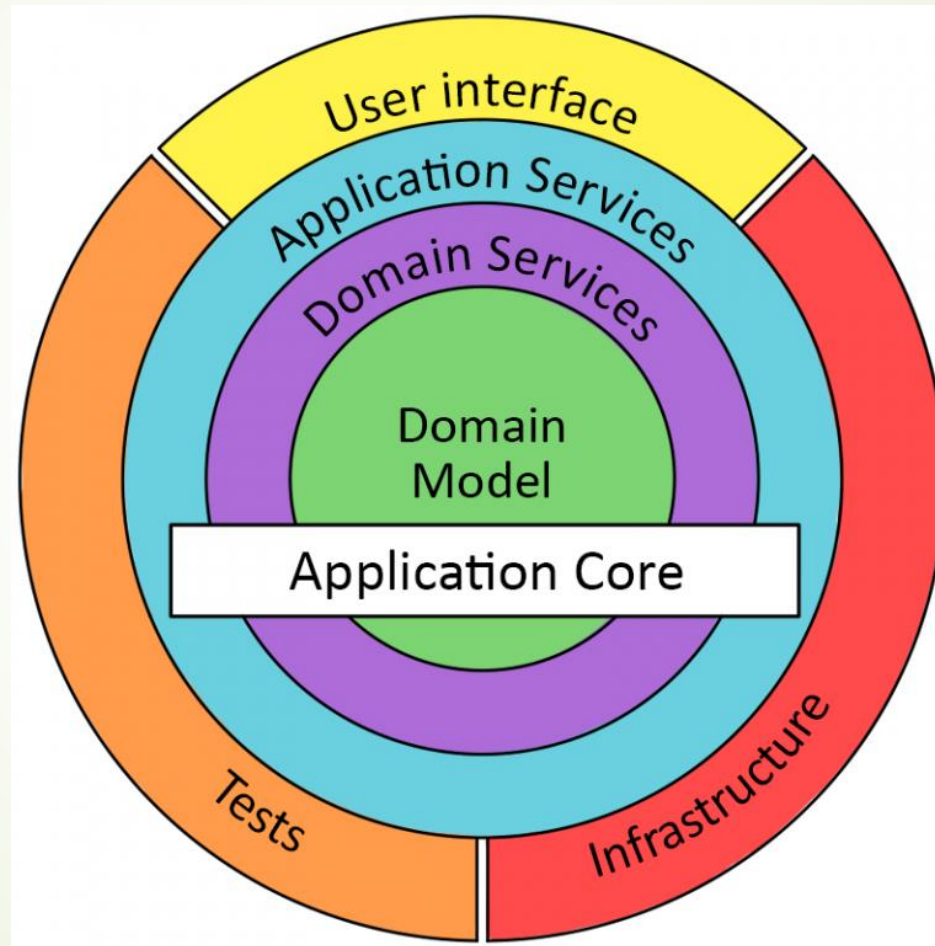


Лекция 11

Атрибуты в C#.

Агентно- и аспектно-ориентированное
программирование

Onion architecture



Атрибуты в С#

- Атрибуты в .NET представляют специальные инструменты, которые позволяют встраивать в сборку дополнительные метаданные. Атрибуты могут применяться как ко всему типу (классу, интерфейсу и т.д.), так и к отдельным его частям (методу, свойству и т.д.).
- В С# атрибуты представляют собой классы, наследующие от базового класса `Attribute`. Любой класс, который наследует от `Attribute`, можно использовать как своего рода "тег" на другие части кода.

```
[Obsolete("Class is deprecated!", true)]  
public class MyClass{  
    ...  
}
```

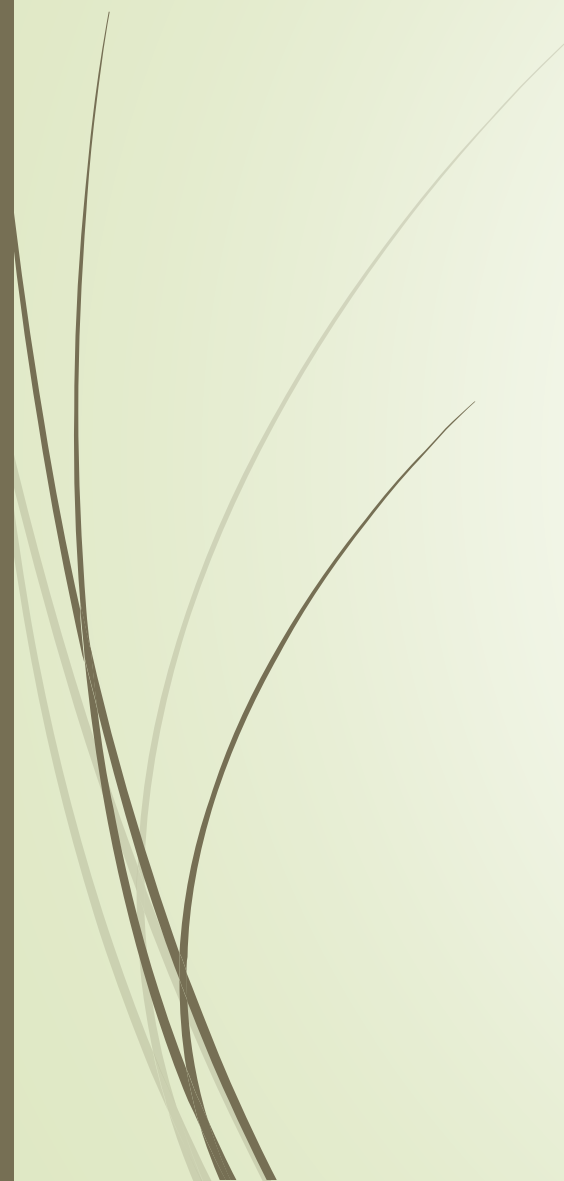

Примеры атрибутов

- **Obsolete** – атрибут, указывающий на то, что вызываемый метод (класс/свойство) является устаревшим
- **Serializable** – указывает на возможность сериализации объектов данного класса (даже если не реализован интерфейс `ISerializable`)
- **NonSerialized** – указывает что поле не участвует в сериализации объекта
- **ThreadStatic** — атрибут позволяющий сделать статическую переменную своей для каждого потока
- **Conditional** – позволяет создавать условные методы, которые вызываются только в том случае, если с помощью директивы `#define` определен конкретный идентификатор, а иначе метод пропускается
- **AssemblyVersionAttribute** - задает версию сборки
- **AttributeUsage** – ограничивает типы, к которым может применяться кастомный атрибут (сборка/класс/структура/метод/свойство/поле/делегат/событие)


Создание собственных атрибутов

```
public class AgeValidationAttribute : System.Attribute{  
    public int Age { get; set; }  
    public AgeValidationAttribute(){ }  
    public AgeValidationAttribute(int age){  
        Age = age;  
    }  
}
```

```
[AgeValidation(18)]  
public class User{  
    public string Name { get; set; }  
    public int Age { get; set; }  
    public User(string n, int a){  
        Name = n;  
        Age = a;  
    }  
}
```



```
static bool ValidateUser(User user)
{
    var t = typeof(User);
    var attrs = t.GetCustomAttributes(false);
    foreach (var attr in attrs)
    {
        if (user.Age >= attr.Age) return true;
        else return false;
    }
    return true;
}
```

Аспектно-ориентированное программирование

- АОП — это парадигма, направленная на повышение модульности различных частей приложения за счет разделения сквозных задач. Для этого к уже существующему коду добавляется дополнительное поведение, без изменений в изначальном коде.
- Основные понятия:
 - Совет (advice) — это дополнительная логика, код, который вызывается из точки соединения
 - Точка соединения (join point) — точка в выполняемой программе (вызов метода, создание объекта, обращение к переменной), где следует применить совет
 - Срез (pointcut) — набор точек соединения. Срез определяет, подходит ли данная точка соединения к данному совету
 - Аспект (aspect) — модуль или класс, реализующий сквозную функциональность. Аспект изменяет поведение остального кода, применяя совет в точках соединения, определенных некоторым срезом. Иными словами, это комбинация советов и точек соединения
 - Цель (target) — объект, к которому будут применяться советы
 - Плетение (weaving) — это процесс связывания аспектов с другими объектами для создания рекомендуемых прокси-объектов. Это можно сделать во время компиляции, загрузки или во время выполнения



Библиотеки и фреймворки для АОП в С#


- PostSharp
- AspectInjector
- AspectCore
- MrAdvice
- Aspectus

и др.



PostSharp

```
[Serializable]
public class LoggingAspect : OnMethodBoundaryAspect{
    public override void OnEntry(MethodExecutionArgs args){
        Console.WriteLine("The {0} method has been entered.", args.Method.Name);
    }
    public override void OnSuccess(MethodExecutionArgs args){
        Console.WriteLine("The {0} method executed successfully.", args.Method.Name);
    }
    public override void OnExit(MethodExecutionArgs args) {
        Console.WriteLine("The {0} method has exited.", args.Method.Name);
    }
    public override void OnException(MethodExecutionArgs args){
        Console.WriteLine("An exception was thrown in {0}.", args.Method.Name);
    }
}
...
[LoggingAspect]
void Foo(){
    Console.WriteLine("Hello, world.");
}
```



Агентно-ориентированное программирование

- Агентно-ориентированное программирование - парадигма программирования, в которой основополагающими концепциями являются понятия агента и его ментальное поведение, зависящее от среды, в которой он находится.
- Концепция была предложена Шохемом в 1990 г. Определение парадигмы, данное автором: агентом является всё, что может рассматриваться как воспринимающее свою среду с помощью датчиков и воздействующее на эту среду с помощью исполнительных механизмов.

- 
- Агент - программная сущность для выполнения поставленных задач. Обладает поведением, а именно: взаимодействует с внешней сложной и динамично-развивающейся средой, способной модифицироваться или быть модифицированной другими агентами в зависимости от конкретных условий. Взаимодействие подразумевает: восприятие динамики среды; действия, изменяющие среду; рассуждения в целях интерпретации наблюдаемых явлений, решения задач, вывода заключений и определения действий.

Примеры

- Моделирование потока транспорта, людей и т.п.
- Боты в компьютерных играх
- Чат-боты
- Прогнозирование потребления ресурсов
и др.

