Тема 12. Операторы и события

Парадигмы программирования. C#

План

- Введение в операторы
- Перегрузка операторов
- Создание и использование делегатов
- Создание и использование событий

Введение в операторы

Часть 1

Операторы и методы

- Использование методов вместо операторов
  - Сокращает чёткость
  - Повышает риск синтаксических и семантических ошибок

```
// Внимание! Этот пример приведён
// только для демонстрации, он не компилируется
int myIntVar1, myIntVar2, myIntVar3;
int myIntVar4 = Int32.Add(myIntVar1,
    Int32.Add(Int32.Add(myIntVar2,
    myIntVar3), 45));
```

- Использование операторов
  - Делает выражения понятными

```
int myIntVar4 = myIntVar1 + myIntVar2 + myIntVar3 + 45;
```

Встроенные операторы C#

Арифметические	+, -, *, /, %	Доступа к членам	.
Логические	&, |, ^, !, ~, &&, ||, true, false	Индексации	[]
Конкатенации строк	+	Приведения типа	()
Инкрементации и декрементации	++, --	Условный	?:
Сдвига	<<, >>	Конкатенации и удаления делегатов	+, -
Сравнения	==, !=, <, >, <=, >=	Создания объектов	new
Присваивания	=, +=, -=, *=, /=, %=, &=, |=, <<=, >>=	Информации типов	is, sizeof, typeof
Управления переполнением	checked, unchecked	Работы с адресами	*, ->, [ ], &

Перегрузка операторов

Часть 2

Введение в перегрузку операторов

- Перегрузка операторов
  - Определяйте собственные операторы только при необходимости
- Синтаксис операторов
  - operator**op**, где **op** – это перегружаемый оператор
- Пример

```
public static Time operator+(Time t1, Time t2)
{
    int newHours = t1.hours + t2.hours;
    int newMinutes = t1.minutes + t2.minutes;
    return new Time(newHours, newMinutes);
}
```

Перегрузка операторов сравнения

- Операторы сравнения должны переопределяться парами
  - < и >
  - <= и =>
  - == и !=
- Переопределяйте метод Equals при перегрузке операторов == и !=
- Переопределяйте метод GetHashCode при переопределении метода Equals

Перегрузка операторов равенства

```
public static bool operator
  ==(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) == 0;
}
public static bool operator
  !=(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) != 0;
}
```

Перегрузка сравнения

```
public static bool operator <(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) < 0;
}
public static bool operator >(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) > 0;
}
public static bool operator <=(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) <= 0;
}
public static bool operator >=(Time lhs, Time rhs)
{
    return lhs.Compare(rhs) >= 0;
}
```

Дополнительно перегружаемые методы

```
public override bool Equals(object obj)
{
    return obj is Time && Compare((Time)obj) == 0;
}
public override int GetHashCode()
{
    return TotalMinutes();
}
private int Compare(Time other)
{
    int lhs = TotalMinutes();
    int rhs = other.TotalMinutes();
    int result;
    if (lhs < rhs)
        result = -1;
    else if (lhs > rhs)
        result = +1;
    else
        result = 0;
    return result;
}
```

Перегрузка логических операторов

- Операторы && и || не могут перегружаться непосредственно
  - Они вычисляются через &, |, true и false
  - x && y вычисляется как T.false(x) ? x : T.&(x, y)
  - x || y вычисляется как T.true(x) ? x : T.|(x, y)

Перегрузка операторов приведения типа

```
public static explicit operator
  Time (float hours)
{ ... }
public static explicit operator
  float (Time t1)
{ ... }
public static implicit operator
  string (Time t1)
{ ... }
```

- Если класс определяет преобразование в строку, нужно переопределить метод `ToString()`

Многократная перегрузка операторов

- Один и тот же оператор может быть перегружен несколько раз

```
public static Time operator +(Time t1,
  int hours)
{...}
public static Time operator +(Time t1,
  float hours)
{...}
public static Time operator -(Time t1,
  int hours)
{...}
public static Time operator -(Time t1,
  float hours)
{...}
```

Найдите ошибки

- `public bool operator != (Time t1, Time t2) { ... }`
- `public static operator float(Time t1) { ... }`
- `public static Time operator += (Time t1, Time t2) { ... }`
- `public static bool Equals(Object obj) { ... }`
- `public static int operator implicit(Time t1) { ...}`

Создание и использование делегатов

Часть 3

Сценарий: атомная электростанция

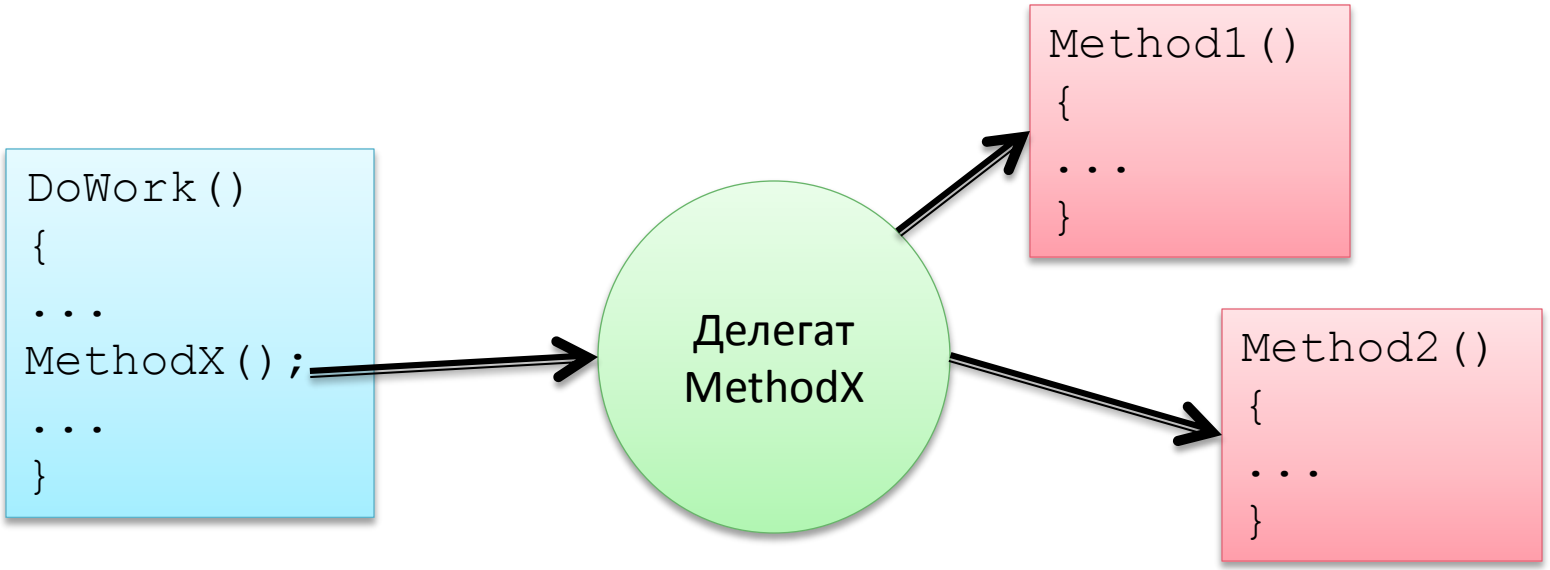
- Проблема
  - Необходимо реагировать на события изменения температуры реактора
  - Если температура реактора повышается выше заданного значения, все охлаждающие насосы должны быть включены
- Возможные решения
  - Каждый насос должен постоянно циклически отслеживать температуру реактора посредством поллинга и включаться при необходимости
    - Слишком частое измерение температуры – большие вычислительные затраты
    - Слишком редкое измерение температуры – высока вероятность пропуска выхода температуры за допуск
  - Для слежения за изменением температуры реактора использовать отдельный компонент, который будет извещаться термодатчиками о изменении температуры и будет включать все насосы при необходимости
    - При использовании различных типов насосов для каждого из них придётся разрабатывать отдельный модуль включения
    - При большом количестве типов насосов количество таких модулей будет велико и программу будет трудно поддерживать

Анализ проблемы

- Соображения о настоящем
  - Могут существовать несколько разных насосов, поставляемых разными производителями
  - Каждый насос может иметь свой собственный метод включения с уникальным идентификатором
- Соображения о будущем
  - Для добавления нового насоса приходится перерабатывать весь код
  - Для каждого такого добавления стоимость реализации будет высока
- Решение
  - Использовать делегаты

Создание делегатов

- Делегат позволяет вызывать методы косвенным способом
  - Делегат содержит ссылки на вызываемые им методы
  - Все методы, вызываемые тем же делегатом, должны иметь одинаковые параметры и возвращаемое значение
  - Альтернативное название делегата – метод обратного вызова



Использование делегатов

- Для вызова делегатов используется синтаксис методов

```
public delegate void StartPumpCallback( );
...
StartPumpCallback callback;
...
callback = new
StartPumpCallback(ed1.StartElectricPumpRunning);
callback.Add(pd1.SwitchOn);
...
callback( );
```

Объявление делегата

Отсутсвует тело метода

Объявление объекта делегата

Получение экземпляра делегата

Добавление метода обратного вызова

Вызов делегата

Создание и использование событий

Часть 4

Работа событий

- Издатель
  - Вызывает событие для извещения всех заинтересованных объектов
- Подписчик
  - Предоставляет метод для обратного вызова при возникновении события

Использование событий

- Объявление события

```
public delegate void StartPumpCallback();
private event StartPumpCallback CoreOverheating;
```

- Добавление подписчика к событию

```
PneumaticPumpDriver pd1 = new PneumaticPumpDriver();
ElectricPumpDriver ed1 = new ElectricPumpDriver();
...
CoreOverheating += new StartPumpCallback(pd1.SwitchOn);
CoreOverheating += ed1.StartElectricPumpRunning;
```

- Уведомление подписчиков о возникновении события

```
public void SwitchOnAllPumps()
{
    if (CoreOverheating != null)
    {
        CoreOverheating();
    }
}
```

Передача параметров в событии

- Параметры события должны передаваться объектом класса EventArgs или его наследником
- Один и тот же метод подписчика может вызываться разными событиями
  - Дополнительно к параметру события следует передавать ссылку на издателя (отправителя) события – sender

Класс параметра события

```
public class CoreOverheatingEventArgs :
  EventArgs
{
    private readonly int temperature;

    public CoreOverheatingEventArgs(
        int temperature)
    {
        this.temperature = temperature;
    }
    public int GetTemperature()
    {
        return temperature;
    }
}
```

Передача параметра в событие

```
public delegate void StartPumpCallback(
    object sender, CoreOverheatingEventArgs e);
private event StartPumpCallback CoreOverheating;
...
CoreOverheating += pd1.SwitchOn;
CoreOverheating += ed1.StartElectricPumpRunning;
...
public void SwitchOnAllPumps()
{
    int temperature = GetTemperature();
    if (CoreOverheating != null)
    {
        CoreOverheating(this,
            new CoreOverheatingEventArgs(
                temperature));
    }
}
```

Получение в методе обратного вызова параметров события

```
public class ElectricPumpDriver
{
    public void StartElectricPumpRunning(
        object sender,
        CoreOverheatingEventArgs args)
    {
        // Получение температуры из
        // параметра события
        int currentTemperature =
            args.GetTemperature( );
        ...
    }
    ...
}
```

Заключение

- Спасибо за внимание!