# Модуль 11: Операторы, делегаты и события

# Обзор

- Обзор операторов
- Перегрузка операторов
- Создание и использование делегатов
- Определение и использование событий

# Обзор операторов

- Операторы и методы
- Операторы, определенные в C#

# Операторы и методы

- **Использование методов**
  - Громоздкая, неудобная запись выражений
  - Увеличивается вероятность возникновения ошибок

```
myIntVar1 = Int.Add(myIntVar2,
            Int.Add(Int.Add(myIntVar3,
                            myIntVar4), 33));
```

- **Использование операторов**
  - Упрощается запись выражений

```
myIntVar1 = myIntVar2 + myIntVar3 + myIntVar4 + 33;
```

# Перегрузка операторов

- Основы перегрузки операторов
- Перегрузка операторов сравнения
- Перегрузка операторов преобразования типов
- Многократная перегрузка операторов
- Тест: Найдите ошибки

# Возможности перегрузки операторов

| | |
|---|---|
| +, –, !, ~, ++, —, true, false | Унарные символы операций, допускающие перегрузку. true и false также являются операциями |
| +, –, *, /, %, &, \|, ^, <<, >> | Бинарные символы операций, допускающие перегрузку |
| ==, !=, <, >, <=, >= | Операции сравнения перегружаются |
| &&, \|\| | Условные логические операции моделируются с использованием ранее переопределенных операций **&** и **\|** |
| [] | Операции доступа к элементам массивов моделируются за счет индексаторов |
| () | Операции преобразования реализуются с использованием ключевых слов **implicit** и **explicit** |
| +=, –=, *=, /=, %=, &=, =, ^=, <<=, >>= | Операции не перегружаются, по причине невозможности перегрузки операции присвоения |
| =, ., ?:, –>, new, is, sizeof, typeof | Операции, не подлежащие перегрузке |

# Основы перегрузки операторов

- **Перегрузка операторов**
  - Перегружайте операторы только когда это действительно необходимо
- **Синтаксис перегрузки операторов**
  - operator **op**, где **op** – это оператор, который перегружается
- **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 необходимо перегружать попарно**
  - < и >
  - <= и >=
  - == и !=
- **При перегрузке операторов == и != настоятельно рекомендуется перегружать метод Equals**
- **Вместе с методом Equals необходимо перегрузить также метод GetHashCode**

# Перегрузка операторов преобразования типов

- **Перегруженные операторы преобразования типов**

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

- **Если в классе используется преобразование типа в строку**
  - В классе должен быть переопределен метод ToString

# Многократная перегрузка операторов

- Один и тот же операторов можно перегрузить 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)
{...}
```

# Тест: Найдите ошибки

```
public bool operator != (Time t1, Time t2)
{ ... }
```

1

```
public static operator float(Time t1) { ... }
```

2

```
public static Time operator += (Time t1, Time t2)
{ ... }
```

3

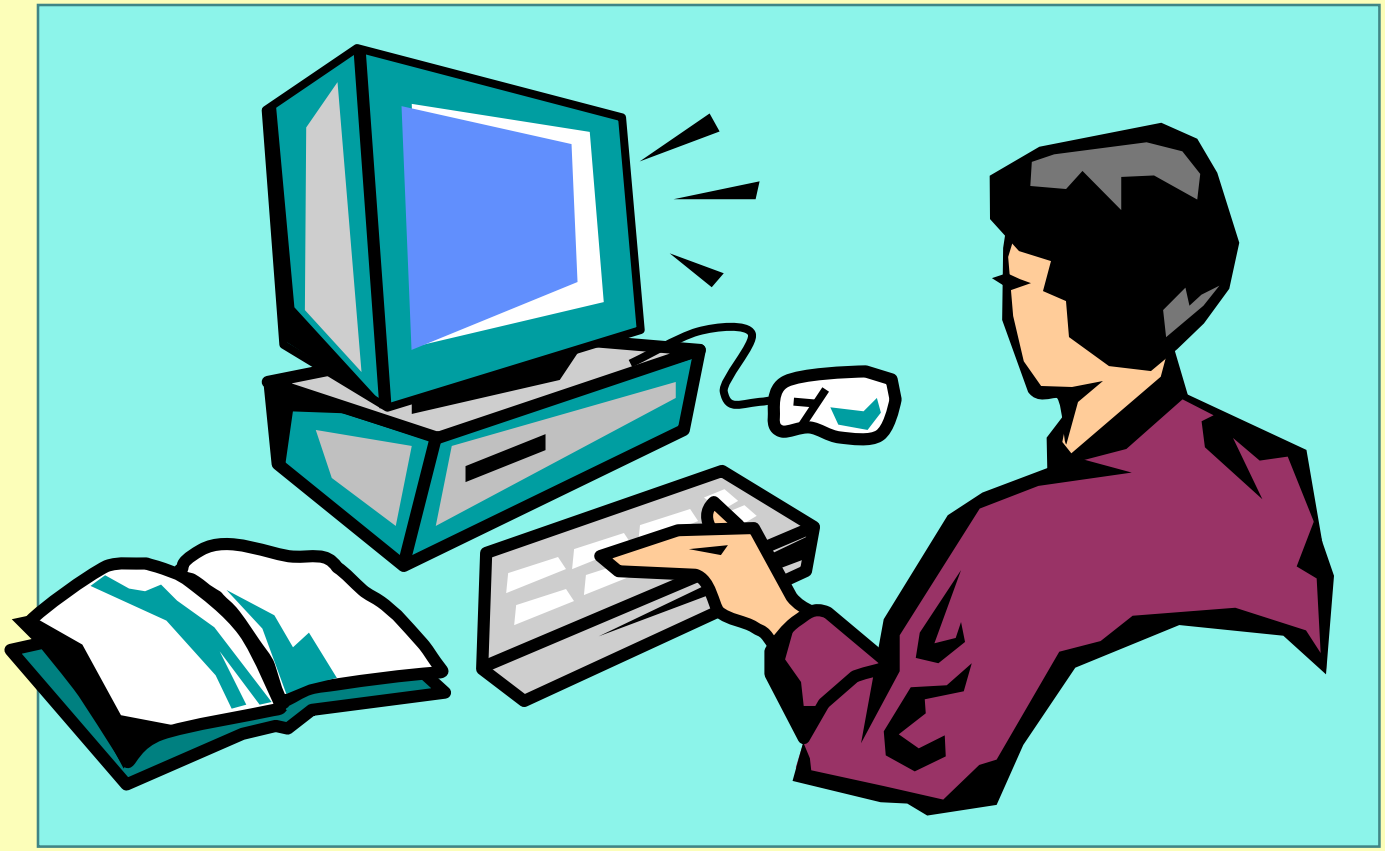
```
public static bool Equals(Object obj) { ... }
```

4

```
public static int operator implicit(Time t1)
{ ...}
```

5

# Лабораторная работа 11.1: Перегрузка операторов



# Создание и использование делегатов

- Сценарий: Атомная электростанция
- Анализ проблемы
- Создание делегатов
- Использование делегатов

# Сценарий: Атомная электростанция

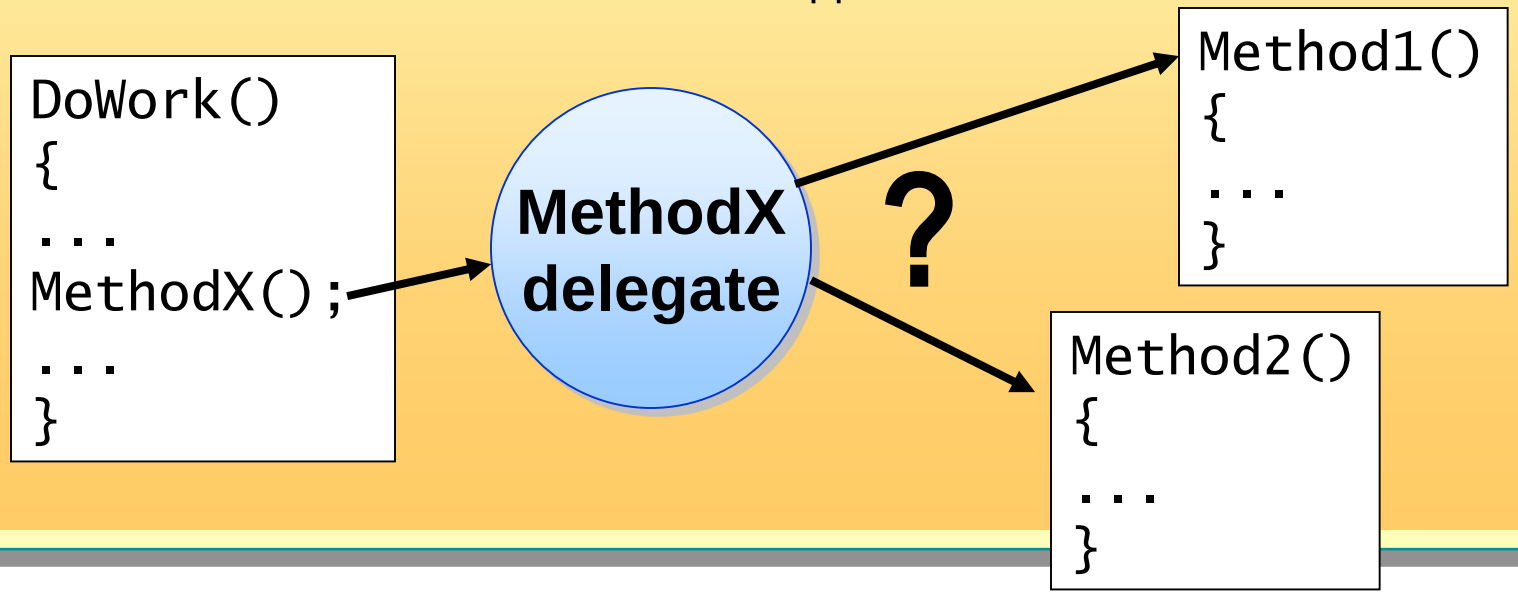
- **Проблема**
  - Как реагировать на изменения температуры на атомной электростанции
  - Если температура активной зоны реактора станет выше определенной температуры, необходимо включить охлаждающие насосы
- **Возможные решения**
  - Все охлаждающие насосы должны постоянно отслеживать температуру активной зоны реактора
  - При критическом изменении температуры специальный компонент, отслеживающий температуру активной зоны, должен включить необходимые насосы

# Анализ проблемы

- **Имеющиеся трудности**
  - Имеются различные типы насосов, произведенные различными заводами
  - У каждого насоса свой метод для его активации
- **Возможные трудности в будущем**
  - При добавлении нового насоса, необходимо переписать весь код
  - При каждом таком добавлении существенные накладные расходы
- **Решение**
  - Используйте в своем коде делегаты

# Применение делегатов

- **Делегат позволяет вызывать метод косвенно**
  - Содержит ссылку на метод
  - Делегат может вызывать только такие методы, у которых тип возвращаемых значений и список параметров совпадают с соответствующими элементами объявления делегата



# Создание делегатов

- **Синтаксис объявления делегата**

```
[<спецификатор доступа>] delegate <тип результата > <имя класса> (<список аргументов>);
```

- **Размещать объявление делегата :**
  - непосредственно в пространстве имен,
  - внутри другого класса,
    - ✓ такое объявление рассматривается как объявление вложенного класса.

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

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

# Свойства делегатов

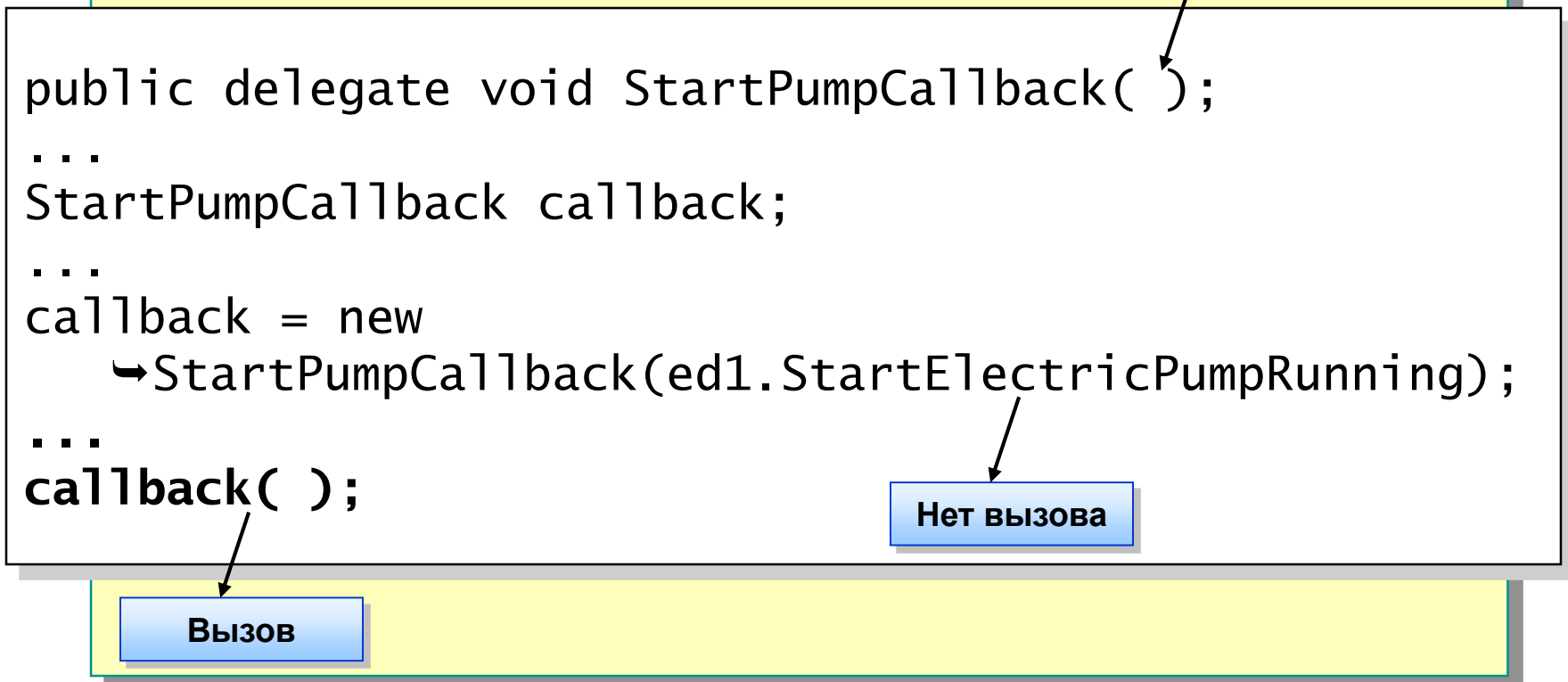
- Делегаты похожи на указатели функций в C++, но являются типобезопасными.
- Делегаты допускают передачу методов в качестве параметров.
- Делегаты можно использовать для задания функций обратного вызова.
- Делегаты можно связывать друг с другом; например, несколько методов можно вызвать по одному событию.
- Точное соответствие методов сигнатуры делегата не требуется (вариативность в делегатах).

# Где следует размещать объявление делегата?

- **Размещают делегаты:**
  - непосредственно в пространстве имен, наряду с объявлениями других классов, структур, интерфейсов;
  - внутри другого класса, наряду с объявлениями методов и свойств.

# Использование делегатов

- **Делегаты вызываются также как и методы**



# Делегат с именованным методом

- При создании экземпляра делегата с помощью именованного метода этот метод передается в качестве параметра

```
// Declare a delegate:
delegate void Del(int x);
// Define a named method:
void DoWork(int k) { /* ... */ }
// Instantiate the delegate using
//the method as a parameter:
Del d = obj.DoWork;
```

# Делегат с анонимным методом

- Создание анонимных методов является, по существу, способом передачи блока кода в качестве параметра делегата

```
// Create a delegate instance
delegate void Del(int x);
// Instantiate the delegate using an anonymous method
Del d = delegate(int k) { /* ... */ };
```

- Использование анонимных методов позволяет сократить издержки на кодирование при создании делегатов, поскольку не требуется создавать отдельный метод

# Определение и использование событий

- Как работают события
- Определения событий
- Передача параметров в события
- Демонстрация: Обработка событий

# Как работают события

- **Издатель**
  - Генерирует событие, оповещающее все заинтересованные объекты (подписчики)
- **Подписчик**
  - Предоставляет метод, вызываемый при генерации события
- **Форма объявления события:**

```
event событийный_делегат объект;
```

# Определения событий

- **Определение события**

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

- **Подпись на событие**

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

- **Уведомление подписчиков о событии**

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

# Передача параметров в события

- **Параметры в события должны передаваться как EventArgs**
  - Создайте класс, унаследованный от EventArgs, который будет служить контейнером для параметров события
- **Один и тот же метод-подписчик может вызываться несколькими событиями**
  - Первым параметром, передаваемым в метод, всегда должен быть издатель события (sender)

# Лабораторная работа 11.2: Определение и использование событий