Основы программирования

Самойлов Михаил Юрьевич

Модуль. Модульное программирование

Модуль - функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной её части.

Модульное программирование — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок.

Функциональные модули

Первыми формами модульности, появившимися в языках программирования, были процедуры и функции.

Позволяли:

- задавать определенную функциональность
- многократно выполнять один и тот же параметризованный программный код при различных значениях параметров.

Процедуры и функции

В математике функции начали использоваться давно, как следствие появление их в языках программирования было закономерным.

Процедуры и функции позволяли решать одну из важнейших задач, стоящих перед программистами - задачу повторного использования программного кода.

Встроенные в язык функции давали возможность существенно расширить возможности языка программирования. Важным шагом в автоматизации программирования было появление библиотек процедур и функций, доступных из используемого языка.

Процедуры и функции

Долгое время процедуры и функции играли не только функциональную, но и архитектурную роль. Весьма популярным при построении программных систем был метод функциональной декомпозиции "сверху вниз".

С появлением объектно-ориентированного программирования архитектурная роль функциональных модулей отошла на второй план. Для ООП-языков, к которым относится и язык C#, в роли архитектурного модуля выступает класс.

Программная система строится из модулей, роль которых играют классы, но каждый из этих модулей имеет содержательную начинку.

Процедуры и функции

Процедуры и функции связываются теперь с классом и они обеспечивают функциональность данных класса и называются методами класса.

Главную роль в программной системе играют данные, а функции лишь служат данным.

В C# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса.

Процедуры и функции

Процедуры и функции связываются теперь с классом и они обеспечивают функциональность данных класса и называются методами класса.

Главную роль в программной системе играют данные, а функции лишь служат данным.

В C# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса.

Процедуры и функции

В языке C# нет специальных ключевых слов - **procedure** и **function**, но присутствуют сами эти понятия.

Синтаксис объявления метода позволяет однозначно определить, чем является метод - процедурой или функцией.

Прежнюю роль библиотек процедур и функций теперь играют библиотеки классов.

Процедуры и функции. Отличия

Функция отличается от процедуры двумя особенностями:

- *всегда вычисляет некоторое значение, возвращаемое в качестве результата функции;*
- *вызывается в выражениях.*

Процедура C# имеет свои особенности:

- *возвращает формальный результат void, указывающий на отсутствие результата ;*
- *вызов процедуры является оператором языка;*
- *имеет входные и выходные аргументы, причем выходных аргументов - ее результатов - может быть достаточно много.*

Описание методов. Синтаксис

Синтаксически в описании метода различают две части - описание заголовка и описание тела метода:

- заголовок_метода
- тело_метода

Синтаксис заголовка метода:

[модификаторы] {void | тип_результата_функции}

имя_метода([список_формальных_аргументов])

Модификатор доступа

Модификатор public показывает, что ***метод открыт*** и доступен для вызова клиентами и потомками *класса*.

Модификатор private говорит, что метод предназначен для внутреннего использования в *классе* и доступен для вызова только в теле методов самого *класса*.

Если модификатор доступа опущен, то по умолчанию предполагается, что он имеет значение private и метод является ***закрытым*** для клиентов и потомков *класса*.

Методы

Обязательным при описании заголовка является указание типа результата, имени метода и круглых скобок, наличие которых необходимо и в том случае, если сам список формальных аргументов отсутствует.

Формально тип результата метода указывается всегда, но значение void определяет, что метод реализуется процедурой.

Тип результата, отличный от void, указывает на функцию.

Методы

Вот несколько простейших примеров описания методов:

- void A() {...};
- int B(){...};
- public void C(){...};

Методы A и B являются *закрытыми*, а метод C - *открыт*.
Методы A и C реализованы *процедурами*, а метод B - *функцией*, возвращающей целое значение.

Список формальных аргументов

Как уже отмечалось, список формальных аргументов метода может быть пустым, и это довольно типичная ситуация для методов *класса*. Список может содержать фиксированное число аргументов, разделяемых символом запятой.

Тело метода

Синтаксически тело метода является блоком, который представляет собой последовательность операторов и описаний переменных, заключенную в фигурные скобки.

Если речь идет о теле функции, то в блоке должен быть хотя бы один оператор перехода, возвращающий значение функции в форме return (выражение).

Переменные, описанные в *блоке*, считаются локализованными в этом *блоке*. В записи операторов *блока* участвуют имена локальных переменных *блока*, имена полей класса и имена аргументов метода.

Пример

Например:

```
void S(int p1, int p2)
{
        int p3 = p1 + p2;
        Console.WriteLine(p3);
}
```

Вызов метода. Синтаксис

Метод может вызываться в выражениях или быть вызван как оператор. В качестве оператора может использоваться любой метод - как процедура, так и функция.

Если же попытаться вызвать процедуру в выражении, то это приведет к ошибке еще на этапе компиляции.

Возвращаемое процедурой значение void несовместимо с выражениями. Так что в выражениях могут быть вызваны только функции.

Сам вызов метода, независимо от того, процедура это или функция, имеет один и тот же синтаксис:

- имя_метода([список_фактических_аргументов])

Main

```
static void Main(string[] args)
{
    Console.WriteLine("привет мир!");
}
```

Ключевое слово static является модификатором. Далее идет тип возвращаемого значения. В данном случае ключевое слово void указывает на то, что метод ничего не возвращает. Такой метод еще называется процедурой.

Далее идет название метода - Main и в скобках параметры - string[] args. И в фигурные скобки заключено тело метода - все действия, которые он выполняет.

Пример:

```
void S(int p1, int p2)
{
    int p3 = p1 + p2;
    Console.WriteLine(p3);
}

static void Main(string[] args)
{
    int a = 2;
    int b = 3
    S(a, b) //5
}
```

Функции

В отличие от процедур функции возвращают определенное значение. Например, определим пару функций:

```
int Factorial()
{
    return 1;
}


string Hello()
{
    return "Hell to World";
}
```

Функции. Использование

```
string Hello()
{
    return "Hell to World";
}
static void Main(string[] args)
{
      string message = Hello(); // вызов первого метода
Console.WriteLine(message);
      Console.ReadLine();
}
```

Пример

Функция не возвращающая значения и не принимающая аргументы:

```
void printError()
{
    Console.Write("Error! Press Key...");
    Console.ReadKey();
}
```

Пример

Функция не возвращающая значения, но принимающая аргумент:

```
void printError(string s)
{
    Console.Write("Error! " + s + "Press Key...");
    Console.ReadKey();
}
```

Пример

Функция не возвращающая значения, но принимающая аргументы:

```
void printError(string s, int i)
{
    Console.Write("Error! " + s + " " + i + "Press Key...");
    Console.ReadKey();
}
```

Пример

Функция возвращающая значения, и принимающая аргументы:

```
int shifr(int x, int shifr)
{
	x = x + shifr;
	return x;
}
```

Задания

Задание 1. Написать метод для нахождения максимального из двух чисел и продемонстрировать его работу.

Задание 2. Написать метод для вывода на экран чисел от 1 до n чисел и продемонстрировать его работу.

Задание 3. Написать метод для расчета суммы всех чисел от 1 до n чисел и продемонстрировать его работу.

Задание 4. Написать метод для нахождения минимального числа из n случайных чисел в интервале [1; 100] и продемонстрировать его работу.

Задание 5. Написать метод для нахождения суммы цифр трехзначного числа и продемонстрировать его работу.