

# ООП. Лекция \*1

1. Технологии программирования
- 2.ООП. Составные части объектного подхода.
3. Члены класса
4. Делегаты

# 1. Технологии программирования

# Технология программирования – методы + средства для разработки ПО.

В зависимости от назначения программирование подразделяют на:

- **системное** — разработка средств общего ПО (ОС, вспомогательных программ, автоматизированных систем управления, систем управления базами данных и т. д.);
- **прикладное** — разработка и отладка программ для конечных пользователей (бухгалтерских, обработки текстов и т. п.)

В зависимости от метода программирование подразделяют на:

- **процедурное** — программы пишутся как перечни последовательно выполняемых команд.
- **структурное, модульное** — программы пишутся как небольшие независимые структурированные части (модули), каждый из которых связан с какой-либо процедурой или функцией. Результирующая программа = совокупность взаимосвязанных по определенным правилам модулей.
- **декларативное** — решение задач искусственного интеллекта. В указанном контексте программа описывает логическую структуру решения задачи, указывая преимущественно, что нужно сделать, не вдаваясь в детали.
- **параллельное** — разработка программ, обеспечивающих одновременное выполнение нескольких операций, связанных с обработкой данных;
- **ООП**
- **функциональное** — разбиение алгоритма решения задачи на отдельные функциональные модули, а также описания их связей и характера взаимодействия.
- **эвристическое** — моделирование мыслительной деятельности человека. Используется для решения задач, не имеющих строго формализованного алгоритма или связанных с неполнотой исходных данных.
- **компонентное** — сбор объектов-компонентов (физически отдельно существующих частей ПО), взаимодействующих между собой через стандартизованные двоичные интерфейсы, в библиотеки или исполняемые файлы.
- **блочное** – решение задач путем разложения модели на блоки.

# Примеры технологий программирования

# Процедурное программирование //

## Алгоритмическое

- Задача разбивается на шаги и решается шаг за шагом.
- Программа состоит из последовательности операторов (инструкций), задающих процедуру решения задачи.
- Часть кода программы можно объединить в отдельные блок (процедуру). Такой блок команд можно вызывать из любой части программы.
- Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, т. е. значений исходных данных, в результаты.

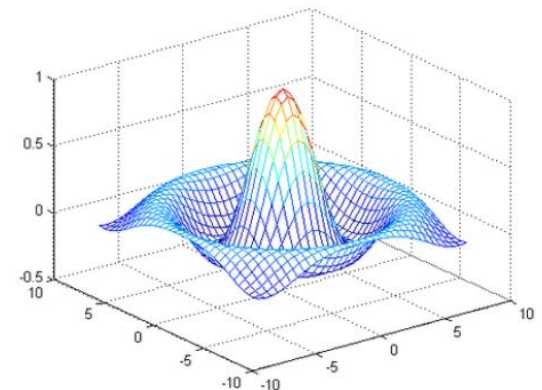
```
#include <iostream>
#include <cstdlib> // для system
using namespace std;

int main()
{
    cout << "Hello, world!" << endl;
    system("pause");
    return 0;
}
```

• Код в C++

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2);
Z = sin(R)./R;
Z(R==0) = 1;
mesh(X,Y,Z);
```

• Код в Matlab



# Функциональное программирование

- Основа – функция = функция в математике.
- Функции не производят никаких побочных действий. Они не изменяют никаких флагов состояния, не записывают никакой информации в глобальные переменные, не пытаются освободить занятую другими объектами память. Никакая функция не сможет вызвать ошибок доступа к памяти.
- Не нужно соблюдать последовательность выполнения операторов в программе - каждая функция может вычислить своё значение в любой момент => легко распараллеливать программы.
- Программы на функциональных языках могут быть в десятки раз короче тех же самых программ, написанных на традиционных языках.
- Имеют плохую совместимость с самыми популярными из императивных языков программирования, плохую переносимость программ на функциональных языках на различные платформы и низкую популярность этих языков
- Работа с функциями (функция от функции возвращает функцию, например, производная, первообразная).

• `(defun sum1 (x y) (+ x y))`

• Код в Lisp

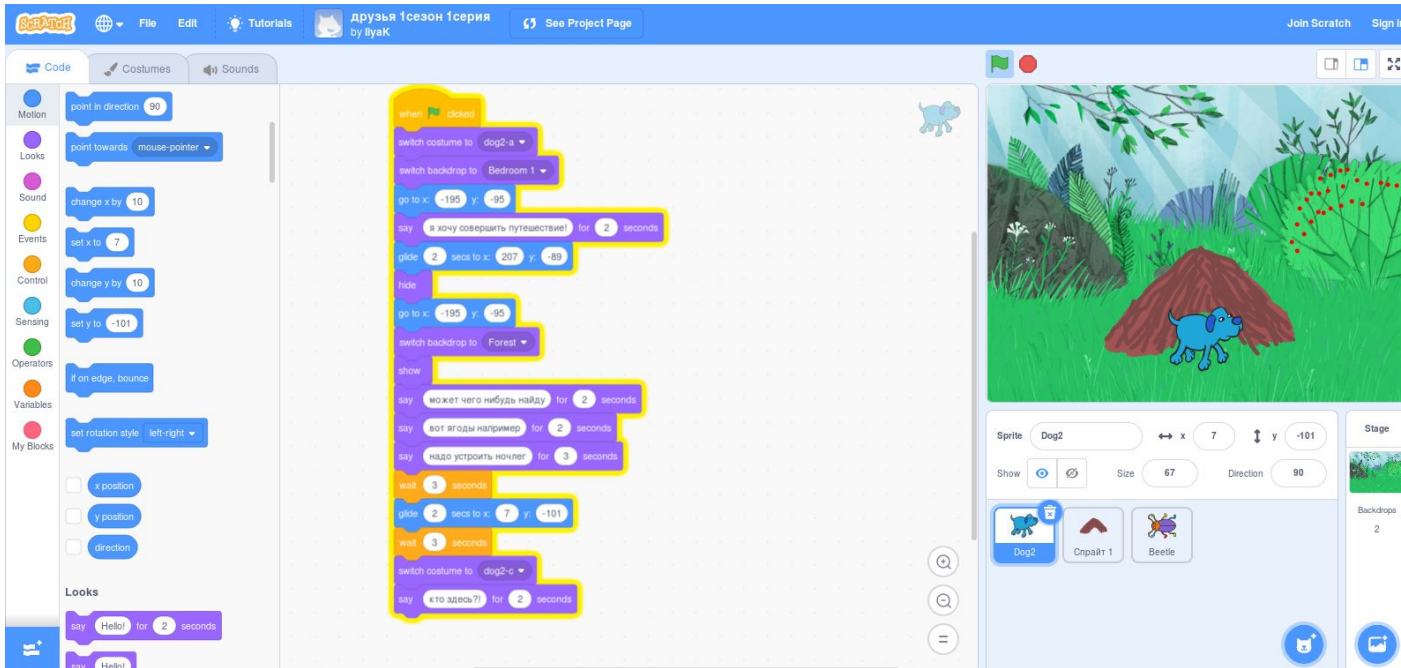
$$Fib(n) = \begin{cases} 1, n = 0 \\ 1, n = 1 \\ Fib(n-2) + Fib(n-1) \end{cases}$$

```
(defun Fib (n)
  (cond
    ((= n 0) 1)
    ((= n 1) 1)
    (t (+ (Fib (- n 2)) (Fib (- n 1))))
  )
)
```

$$n! = \begin{cases} 1, n = 0 \\ n * (n-1)! \end{cases}$$

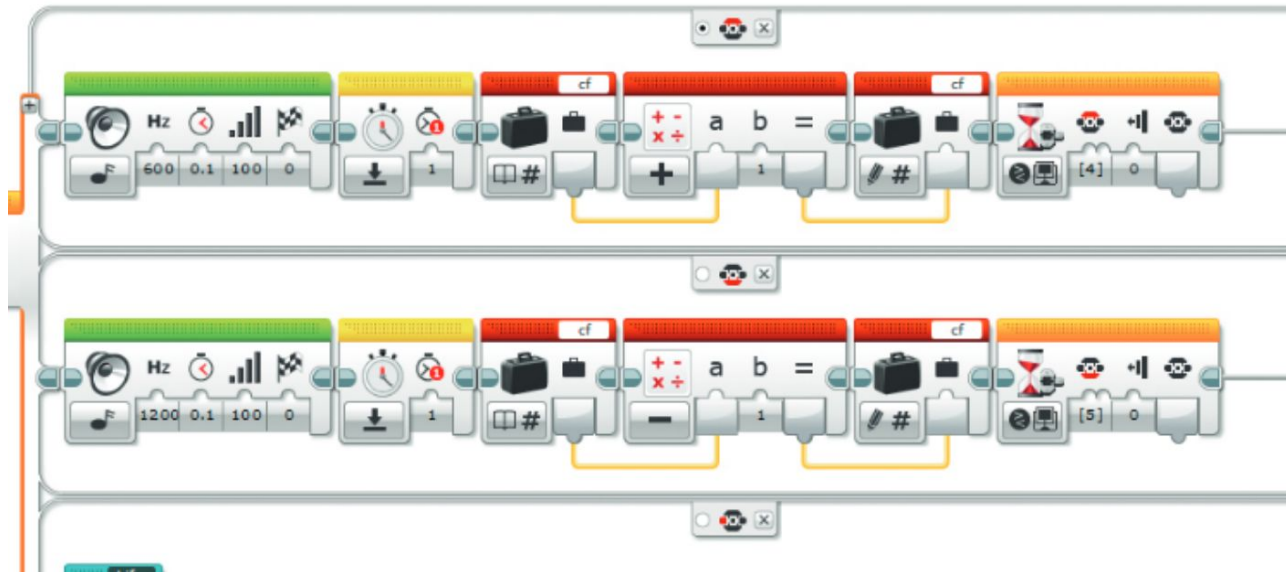
```
(defun Факториал (n)
  (cond
    ((= n 0) 1) ; <- терминальная ветвь
    (t (* n (Факториал (- n 1))))
  )
)
```

# Блочное программирование // Визуальное



- Код в Scratch

- Код в LeGo Education Mindstorms



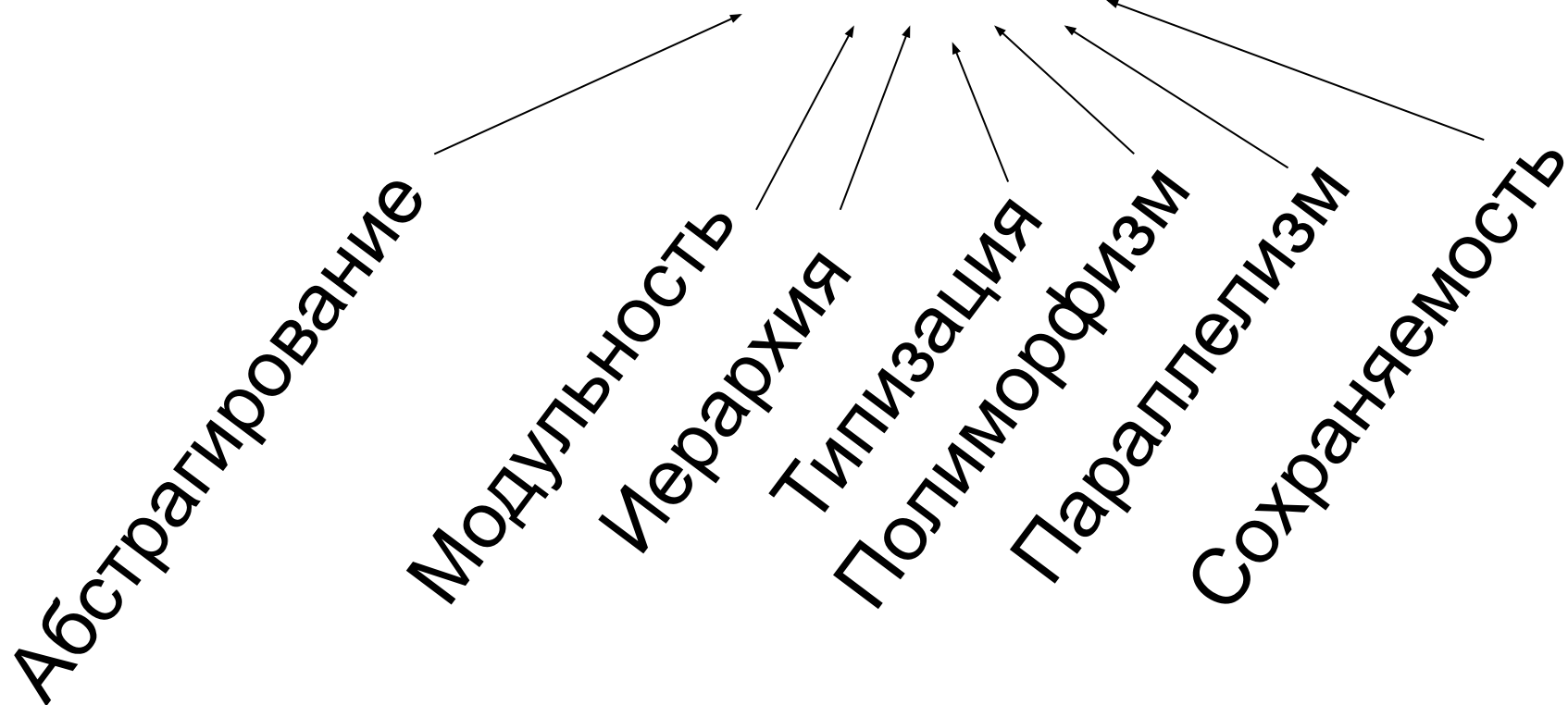
## 2.ООП. Составные части



# ООП



# ООП



# Абстрагирование

- Абстрагирование — отделение существенного от несущественного.
- В ООП это упрощенное описание объекта, при котором одни свойства и детали объекта выделяются, другие опускаются.
- 1. При моделировании некоторого объекта мы можем отказаться от некоторых его частей, не важных в контексте программы.
- 2. Имеется большее описание объекта, чем нужно остальной части программы. (Работа с файлами.)

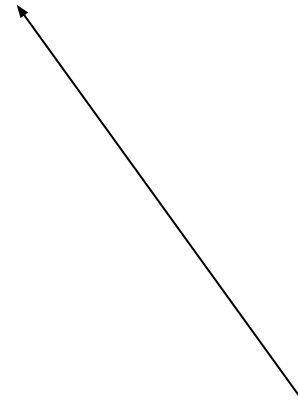
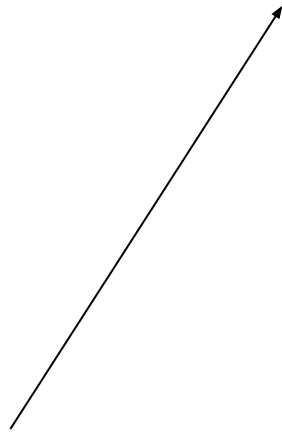
# Инкапсуляция

- Это свойство ЯП, которое позволяет объединить данные и методы, работающие с ними.
- Скрывает детали реализации от пользователя.
- public, protected, private.

# Интерфейс

- Это контракт, который объявляет, что мы должны делать.
- ???

Абстрагирование



Инкапсуляция

Интерфейс

# Модульность

- Разделение программы на сегменты = модули, которые компилируются отдельно.
- Модули могут устанавливать связь между собой.
- Модулем может быть как метод, класс, несколько классов

# Иерархия

- Уровни расположения абстракций.
- Основу составляет наследование.
- Наследование одними классами свойства и поведение других классов.

Иерархия



Наследование

# Типизация

## Типы значений

### Простые типы

sbyte, short, int, long

byte, ushort, uint,ulong

char

float, double

decimal

bool

### Типы перечисления

enum E {...}

### Типы структур

## Типы классов

object

string

class C {...}

### Типы интерфейсов

interface I {...}

### Типы массивов

Одно- и многомерные,  
например, int[] и int[,]

### Типы делегатов

delegate int D(...)



# Типизация

**Способ защитить объекты одного класса от влияния другого класса.**

**Тип  $\Leftrightarrow$  класс.**

**Раннее обнаружение ошибок.**

**Многие компиляторы генерируют более эффективно код, если им известны типы данных.**

# Параллелизм

- Каждая программа имеет по крайней мере 1 поток управления.
- В параллельной системе может быть несколько таких потоков.
- Используется при работе с БД.

# Сохраняемость

- Если объекты присутствуют во время вычисления, а потом удаляются, то это говорит о несохраняемости.
- В БД данные существуют независимо от программы.
- Промежуточные результаты всего выражения.
- Локальные переменные при вызове процедур.
- Глобальные переменные

### 3. Члены класса

# Члены класса



поля, свойства, методы, события

- Поля и свойства – хранят данные для описания объекта или данные, необходимые для работы объекта.
- Методы – производит действия над объектом и возвращает результат своей работы.
- События – уведомляет о том, что изменилось состояние объекта.

# Методы

В C# все функции — это методы

Последовательно выполняют операции и могут возвращать/не возвращать результат работы.

***модификатор\_доступа возвращаемый\_тип имя (параметры\_метода)***

```
{  
тело метода  
}
```

Сигнатура метода — тип и порядок входных значений и тип выходных значений.

```
using System;  
  
class Program  
{  
    static void PrintHelloWorld()  
    {}  
    static void PrintHello(string name)  
    {}  
    static int Cube(int x)  
    {}  
    static int[] GetArrayFromConsole(string arrayName, int elementCount)  
    {  
    static void Test(int i)  
    {}  
    static string ZeroCompare(double number)  
    {}  
    protected void AddGas(int gallons)  
    } }  
}
```

## 4. Делегаты

# Делегаты

Делегат — самостоятельная структура в C#.

Это объект, в который помещается ссылка на метод. Эта ссылка может использоваться для вызова метода.

Делегат — можно упаковать метод в объект и делать с ним все, что хочется: передавать в качестве аргумента другого метода, вызывать этот метод когда угодно и где угодно.

Мы не можем какой угодно метод поместить в делегат, нужно точно указать параметры метода и возвращаемое значение. Т.е. сигнатура метода и делегата должна совпадать.



## 1. Тип делегата:

*delegate string MyDelegate (int paramA, string paramB)*

## 2. Нужен метод, сигнатура которого совпадает с сигнатурой делегата.

*string SomeMethod (int someIntParam, string someStringParam)*

## 3. Создаем экземпляр делегата

*string method = new  
MyDelegate(SomeMethod);  
MyDelegate method2 =  
SomeMethod;*

## 5. Объектная модель