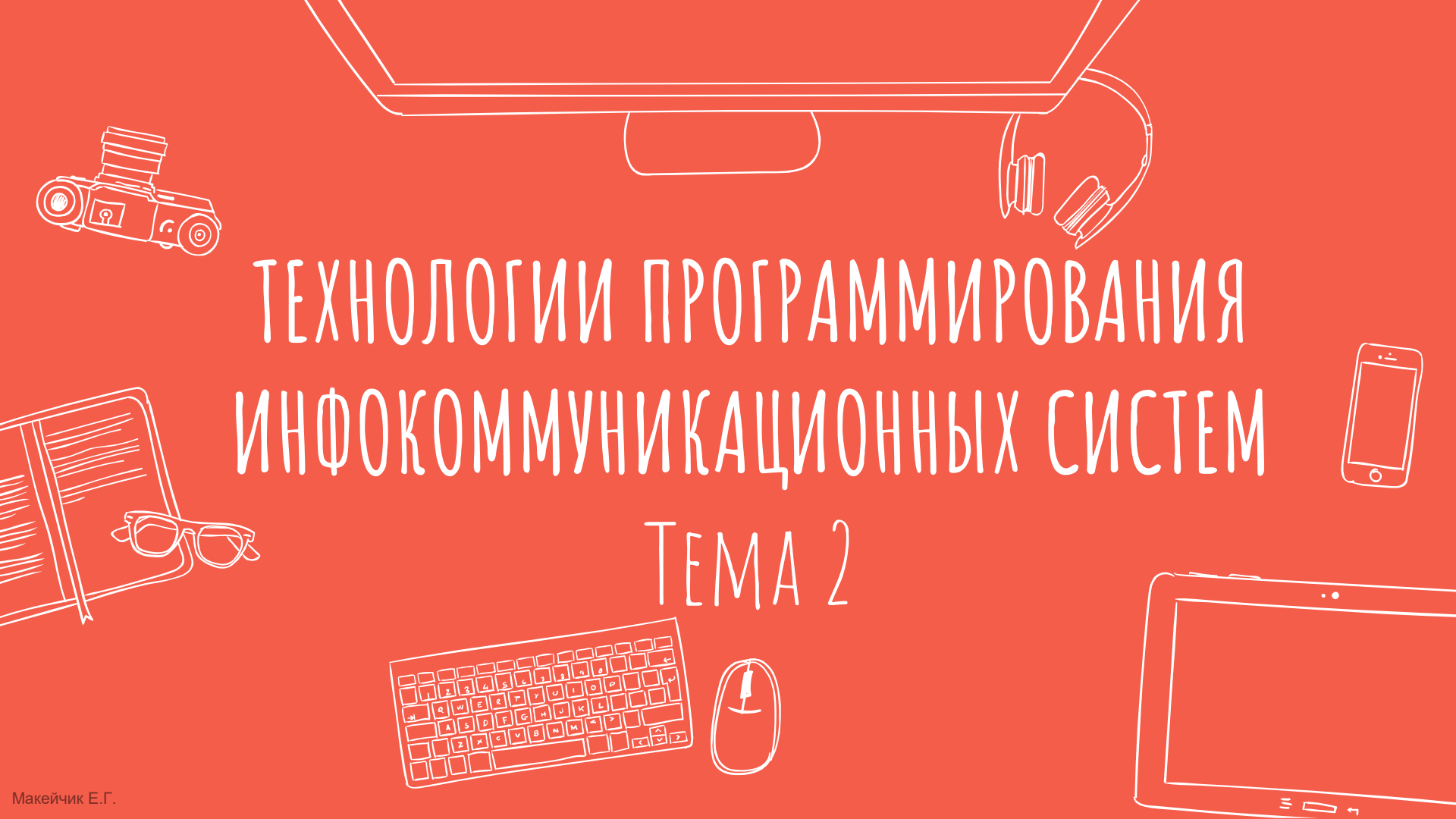


Технологии программирования инфокоммуникационных систем

Тема 2

Макейчик Е.Г.

UML ДИАГРАММА КЛАССОВ

UML происходит от Unified Modeling Language, унифицированный язык моделирования, который, как гласит легенда, разработали, когда серьезные дяди и тети в конец задолбались плавать в разнообразии кружочков, черточек и облачков.

UML ДИАГРАММА КЛАССОВ

UML ДИАГРАММА КЛАССОВ

Словарь UML включает три вида строительных блоков:

- диаграммы;
- сущности;
- связи.

UML ДИАГРАММА КЛАССОВ

Диаграммы классов оперируют тремя видами сущностей UML:

- структурные;
- поведенческие;
- аннотирующие.

Структурные сущности – классы

Графически класс изображается в виде прямоугольника, разделенного на три блока горизонтальными линиями:

- имя класса;
- атрибуты (свойства/поля) класса;
- операции (методы) класса.

Структурные сущности – классы

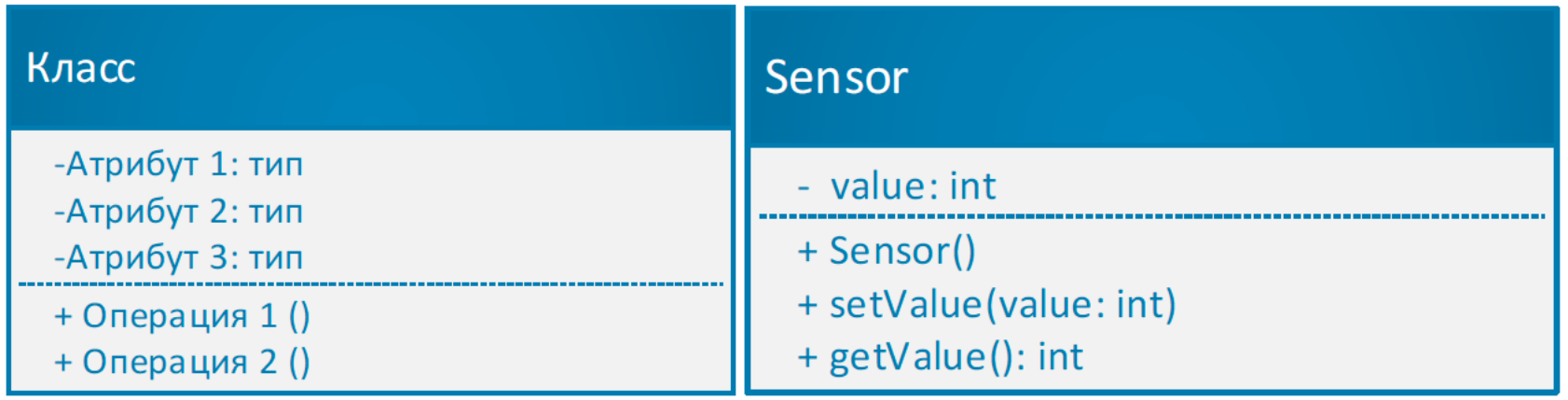
Для атрибутов и операций может быть указан один из трех типов видимости:

– – private (закрытый).

– protected (защищенный).

\+ – public (открытый).

UML ДИАГРАММА КЛАССОВ



UML ДИАГРАММА КЛАССОВ

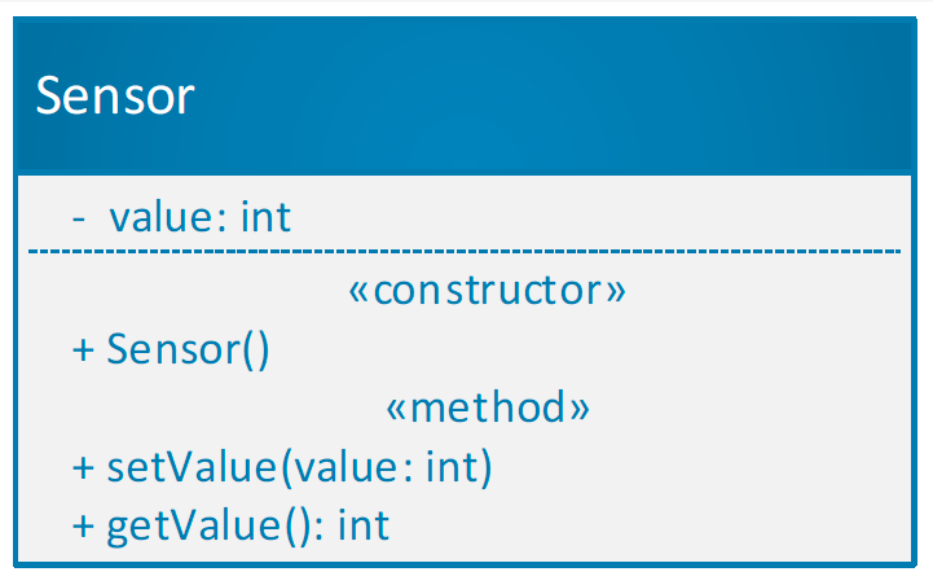
Имя класса пишется в самом верхнем делении, затем идут атрибуты класса, типы которых записываются после двоеточия и, наконец, в нижнем делении идут методы.

Тип, который может возвращать метод, записывается после двоеточия в самом конце сигнатуры метода. Модификаторы области видимости изображены перед атрибутами класса и методами.

UML ДИАГРАММА КЛАССОВ

Чтобы легче воспринимать длинные списки атрибутов и операций, желательно снабдить префиксом (именем стереотипа) каждую категорию в них. В данном случае *стереотип* – это слово, заключенное в угловые кавычки, которое указывает то, что за ним следует.

UML ДИАГРАММА КЛАССОВ



Отношения между классами

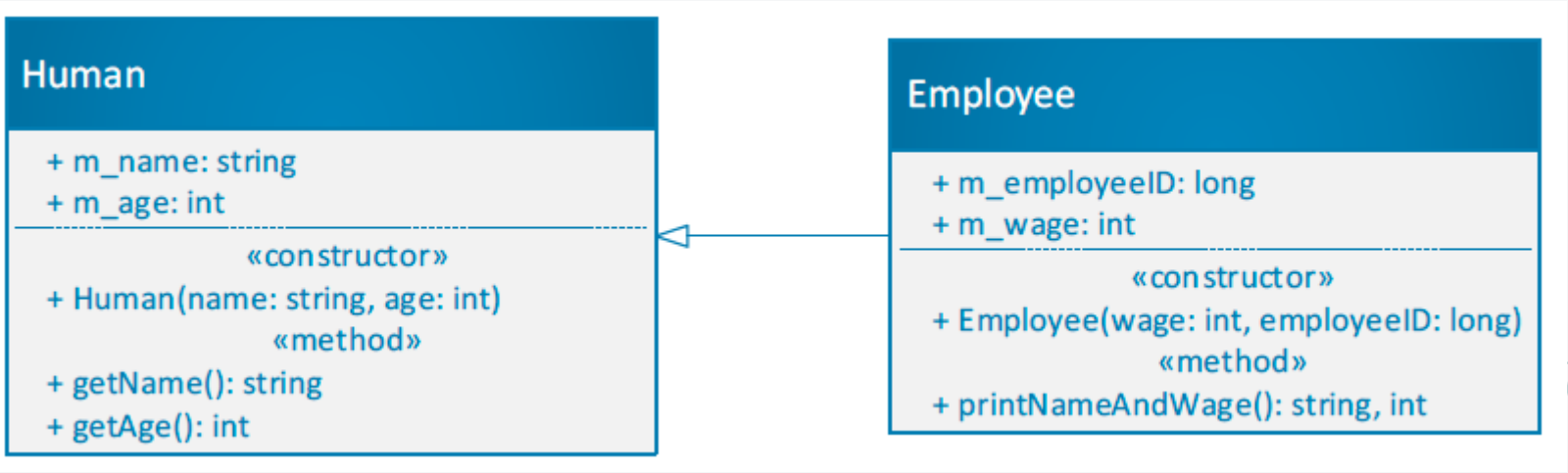
Существует четыре типа связей в UML:

- зависимость;
- ассоциация;
- обобщение;
- реализация.

Отношения между классами

Обобщение – выражает специализацию или наследование, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя). Потомок разделяет структуру и поведение родителя. Графически обобщение представлено в виде сплошной линии с пустой стрелкой, указывающей на родителя.

Отношения между классами



Блок-схемы

Блок-схемы – это диаграммы, используемые для графического описание процесса.

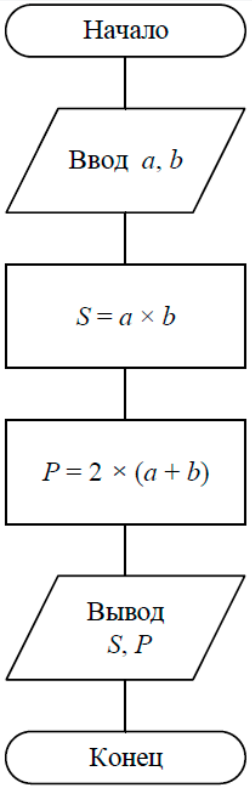
Обычно блок-схемы создаются перед началом процесса или написанием приложения, чтобы проверить логику рабочего процесса и выявить возможные проблемы, прежде чем решение будет разработано и внедрено.

Блок-схемы

Дано: a, b
Найти: S, P

$S = a \times b$
$P = 2 \times (a + b)$

Структура блок-схемы:
1 Начало
2 Ввод значений сторон прямоугольника
3 Расчет площади прямоугольника
4 Расчет периметра прямоугольника
5 Вывод значений площади и периметра
6 Конец



Блок-схемы

Число a меняется от −10 до 10 с шагом 5.

Это означает, что число является переменной цикла. Изначально равно −10 – это первоначальное задание переменной цикла. Затем, a будет изменяться с шагом 5, и т.д. пока не будет достигнуто значение 10 – это соответствует изменению переменной цикла. Итерации нужно повторять, пока выполняется условие $a \le 10$. Итак, a будет принимать следующие значения: −10, −5, 0, 5, 10. Число b не будет являться переменной цикла, т.к. $b = 7$, и не изменяется по условию задачи.

Структура блок-схемы:

1 Начало

2 Ввод значений $a = -10$, $b = 7$

3 Расчет суммы S и разности R

$S = a + b = -10 + 7 = -3$

$R = a - b = -10 - 7 = -17$

4 Вывод S, R

$S = -3$

$R = -17$

5 Расчет a

$a = a + 5 = -10 + 5 = -5$

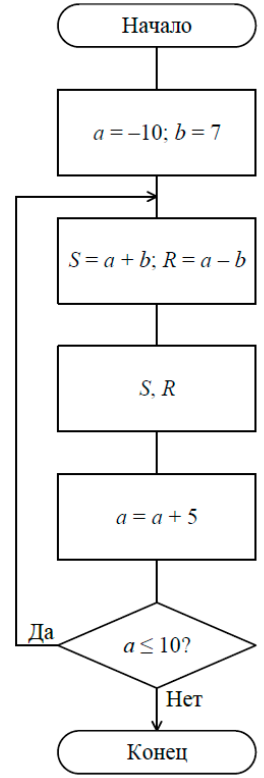
6 Проверка $a \le 10$

6.1 $-5 \le 10$ Да, верно. Идем по стрелке вверх к шагу 3

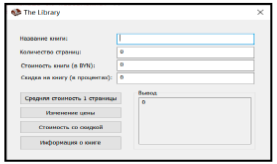
...

6.2 $15 \le 10$ Нет, ложно. Выходим из цикла. Идем по стрелке вниз

7 Конец



Блок-схемы



Структура блок-схемы:

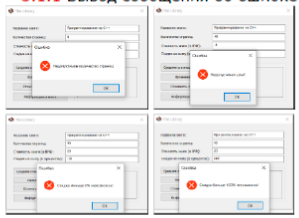
1 Начало

2 Ввод данных "Название книги", "Количество страниц", "Стоимость книги (в BYN)", "Скидка на книгу (в процентах)"

3 Проверка на корректность введенных значений

3.1 Данные введены корректно? Нет, ложь. Идем по стрелке вверх

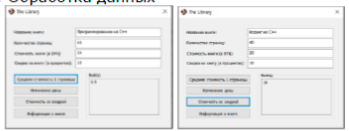
3.1.1 Вывод сообщения об ошибке



3.1.2 Переход к шагу 2

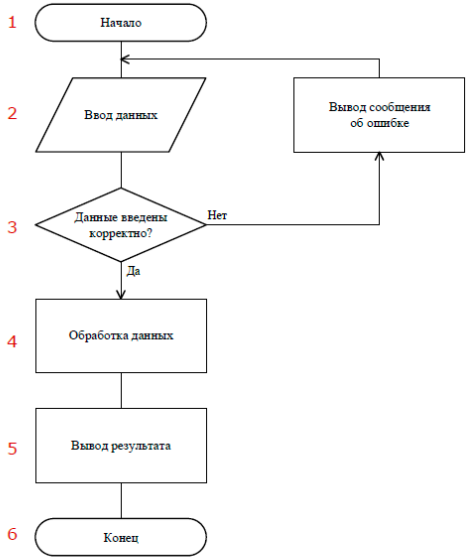
3.2 Данные введены корректно? Да, верно. Идем по стрелке вниз

4 Обработка данных



5 Вывод результата

6 Конец



Организация памяти в программе

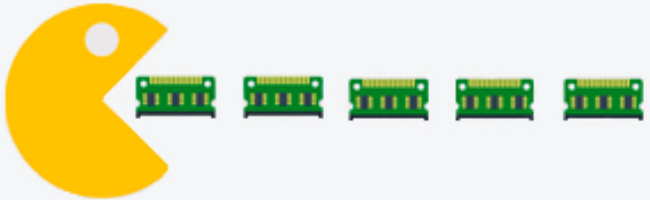
- Статическое выделение памяти выполняется для статических и глобальных переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.

Организация памяти в программе

- Автоматическое выделение памяти выполняется для параметров функции и локальных переменных. Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.

Организация памяти в программе

- Динамическое выделение памяти – это способ запроса памяти из операционной системы запущенными программами по мере необходимости. Эта память не выделяется из ограниченной памяти стека программы, а выделяется из гораздо большего хранилища, управляемого операционной системой – кучи.



Организация памяти в программе

Как работает динамическое выделение памяти?

Когда вы динамически выделяете память, то вы просите операционную систему зарезервировать часть этой памяти для использования вашей программой. Если ОС может выполнить этот запрос, то возвращается адрес этой памяти обратно в вашу программу.

Организация памяти в программе

Как работает динамическое выделение памяти?

С этого момента и в дальнейшем ваша программа сможет использовать эту память, как только пожелает. Когда вы уже выполнили с этой памятью всё, что было необходимо, то её нужно вернуть обратно в операционную систему, для распределения между другими запросами.

Работа с динамической памятью

Управление программным размещением объектов в памяти, т.е. выделение памяти в процессе работы программы под переменные и массивы, осуществляется с помощью операций new и delete.

Операция new определяет размер выделяемой памяти для указанного типа и позволяет инициализировать создаваемый объект.

Работа с динамической памятью

Формат операций:

```
Тип *id_указателя; // Объявление указателя
id_указателя = new Тип(значение); // Установка указателя на динам. область памяти
delete ID_указателя; // Удаление указателя
```

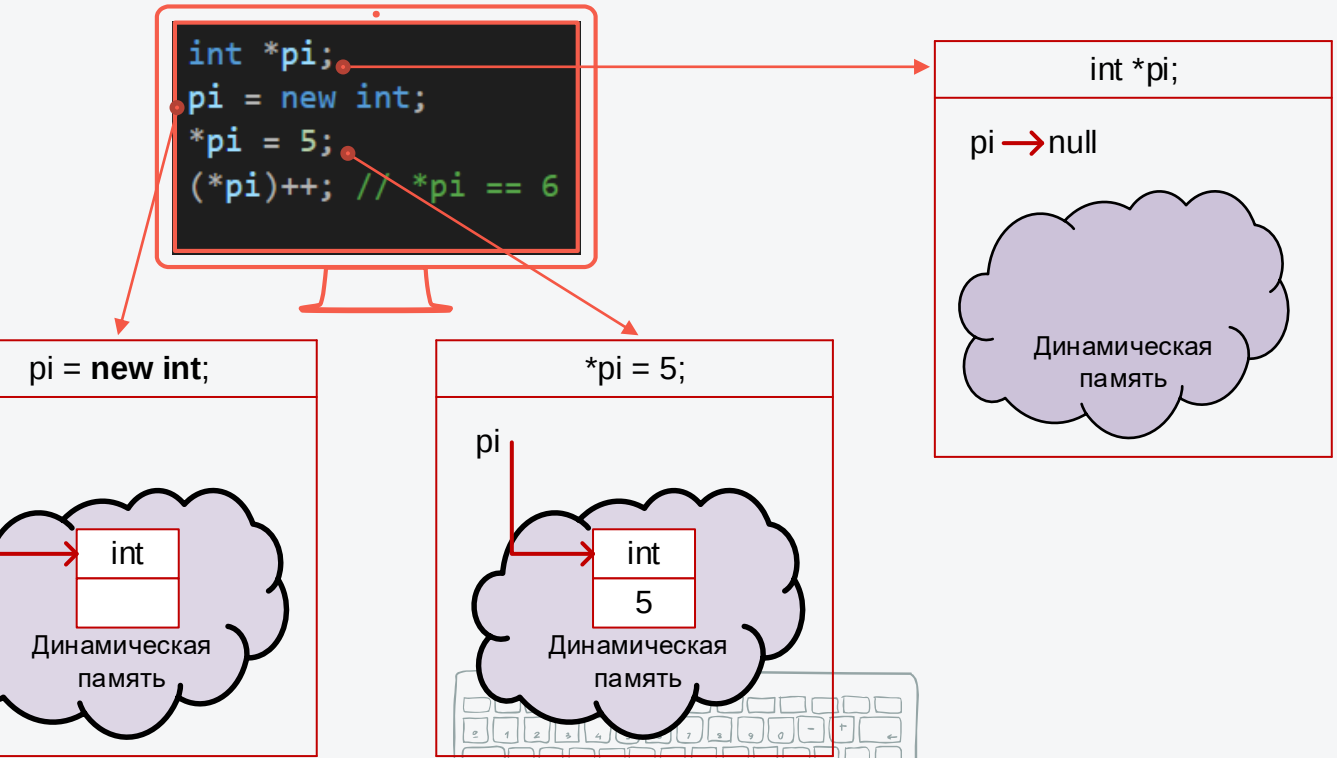
Работа с динамической памятью

где **тип** – тип значений, для которых выделяется память; **значение** – константа (необязательный атрибут), определяющая начальное значение динамической переменной.

Работа с динамическим объектом производится только через косвенную адресацию – операция «*».

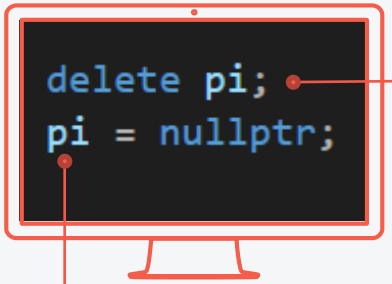
Указатели и динамическая память

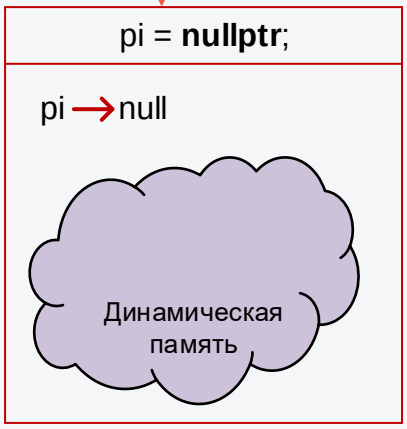
Управление динамической памятью с помощью указателей



Указатели и динамическая память

Освобождение динамической памяти





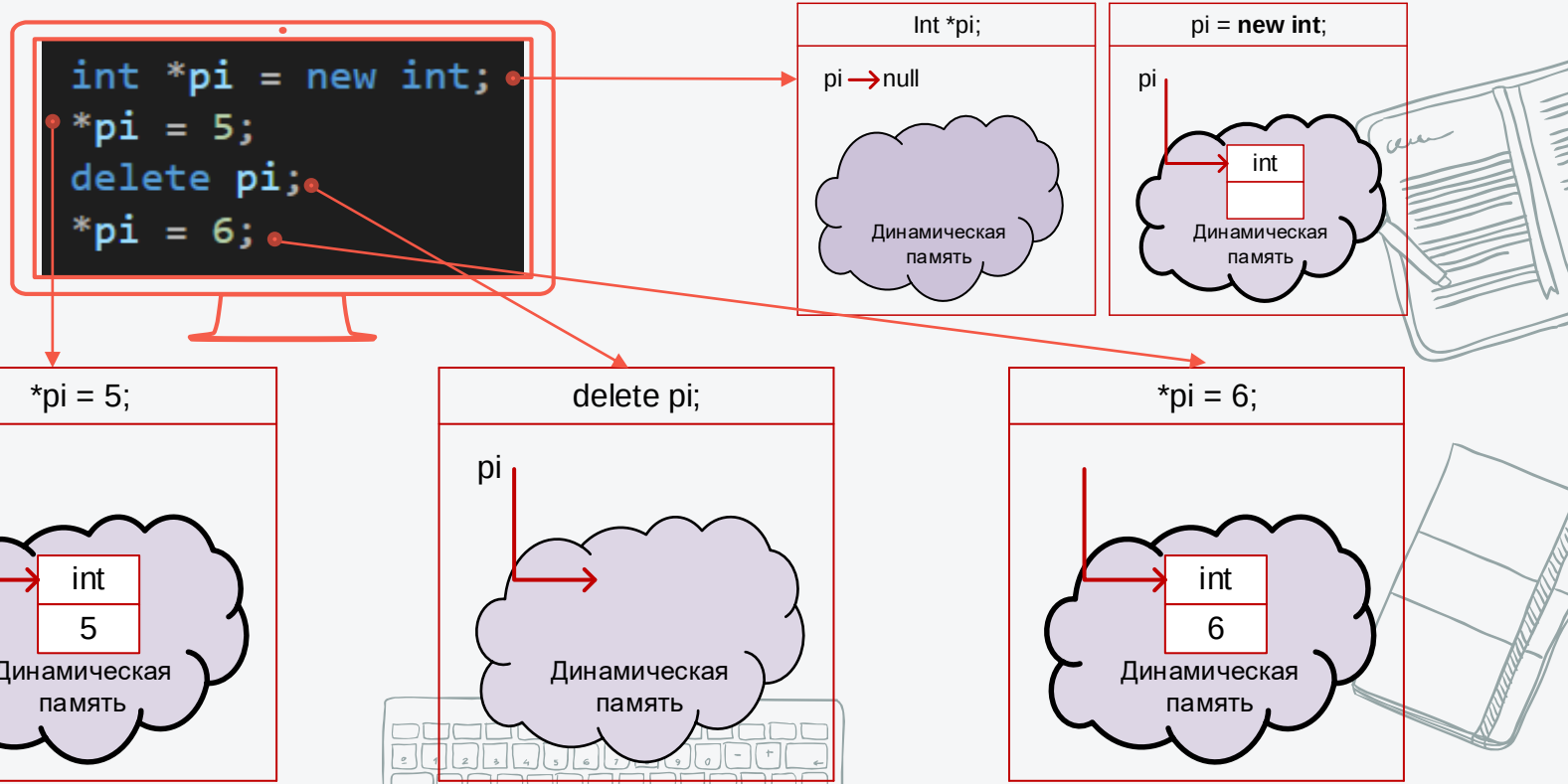
Делаем pi нулевым указателем (вместо 0 nullptr начиная с C++11)

Возвращаем память, на которую указывал pi, обратно в операционную систему

Указатели и динамическая память

Обращение к освобожденной памяти

```
int *pi = new int;
*pi = 5;
delete pi;
*pi = 6;
```



Указатели и динамическая память

Массивы в динамической памяти (динамические массивы)

```
int *pi = new int[10]; // Выделение памяти для 10 элементов
pi[0] = 5;
pi[1] = 3;
//...
delete []pi; // Возврат этой память
```

Двумерные динамические массивы

Выделение памяти для статических массивов

```
int a[3][4];
// Размеры - это константы,
// поскольку память выделяется на этапе компиляции

void f(int a[][4], int m, int n) {
    //...
}
```

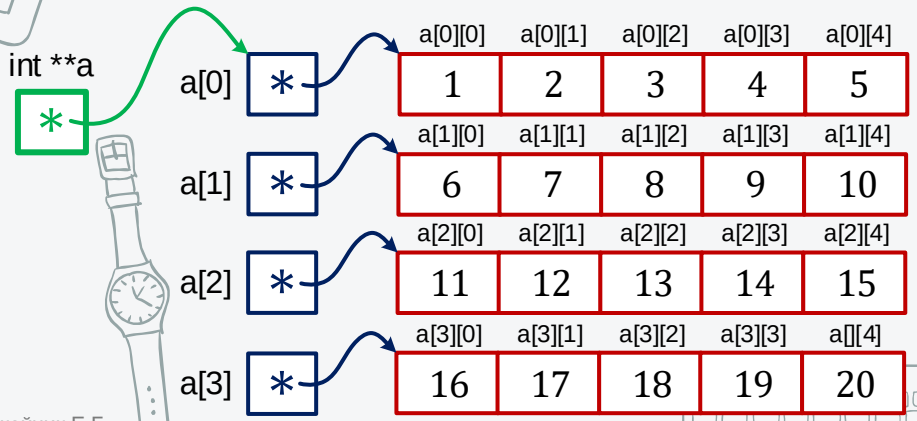
«Вот так вот не надо!»

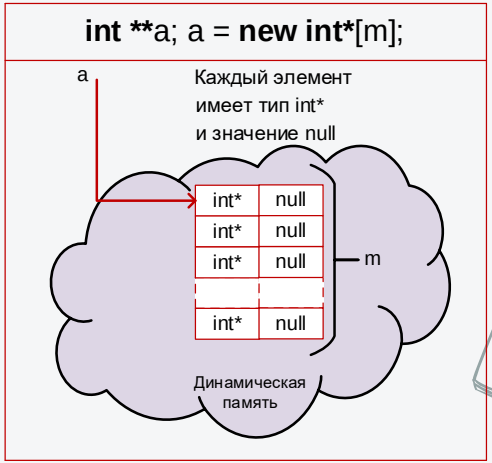
В таких случаях лучше использовать двумерные динамические массивы

Двумерные динамические массивы

Выделение памяти для динамических массивов

```
int **a; // a - указатель на начало массива из int*
int m, n;
cin >> m >> n; // Здесь размеры определяются на этапе выполнения программы
a = new int*[m];
```





Дружественные функции

Дружественная функция – это функция, которая не является членом класса, но имеет доступ к закрытым (private) и защищенным (protected) членам класса

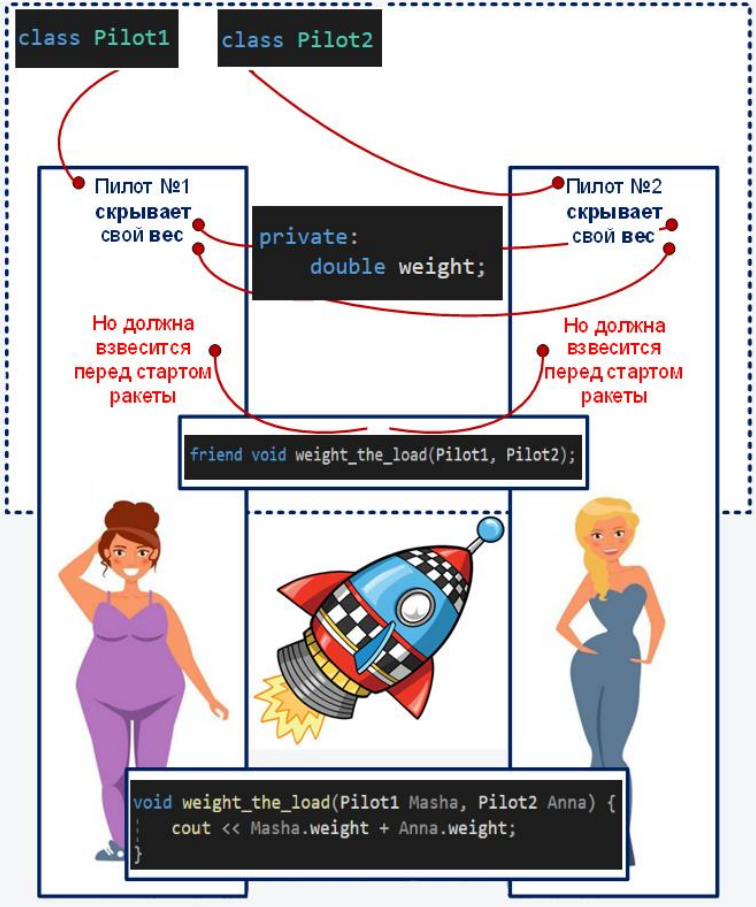
Дружественные функции

- Дружественная функция объявляется с помощью ключевого слова friend в классе (-ах), который (-ые) предоставляет (-ют) доступ к своим данным
- Объявление friend помещается в любом месте описания класса. На него не влияют ключевые слова управления доступом

Дружественные функции

- Дружественные функции описываются вне класса, однако при этом в виде параметров им передаются указатель или ссылка на объект класса и его элементы
- При вызове дружественных функций не надо использовать операцию привязки к классу или объекту (.) или (->)

Дружественные функции



ПРИМЕРЫ

Дружественная функция

```cpp
#include <iostream>
using namespace std;
class Year { // Класс с закрытым полем year
    double year;
public:
    Year(double y) // Конструктор класса с параметром
    {
        year = y;
    }
    friend void display(Year god); // Дружественная функция
};
// Реализация дружественной функции
void display(Year god) {
    cout << "Year = " << god.year << endl;
}
int main() {
    Year this_year(2022); // Объект this_year класса Year
    display(this_year); // Дружественная функция имеет доступ к закрытым элементам
}
```

```
Year = 2022
```

Перегрузка операторов

Перегрузка операторов — это возможность назначать новый смысл операторам при использовании их с определенным классом

Перегрузка операторов

- Переопределяемые операции реализуются как особый вид функции со специальным именем **operator ++** (где ++ – символ переопределяемой операции)
- Перегрузка операции действует только для класса, в котором она определяется

Перегрузка операторов

❌ C++ позволяет перегружать большинство операторов, за исключением:

.	точка (выбор элемента класса)
*	звездочка (определение указателя)
::	двойное двоеточие (область видимости метода)
?:	знак вопроса с двоеточием (тернарный оператор сравнения)
#	диез (символ препроцессора)
##	двойной диез (символ препроцессора)
sizeof	оператор нахождения размера объекта в байтах

❌ С помощью перегрузки невозможно создавать новые символы для операций

ПРИМЕРЫ

```cpp
// Решение задачи с использованием класса string
#include <iostream>
#include <string>
using namespace std;
class Stroka
{
public:
    string text; // Создание объекта text класса string
    void print() // Метод для вывода содержимого поля text
    {
        cout << text << endl;
    }
    void operator=(Stroka& str) // Перегрузка оператора "=". str – объект стоящий справа от "="
    {
        for (int i = 2; i < str.text.length() - 2; i++) // Прохождение по строке в объекте str. Обход начинается сразу с третьего символа, не доходя до конца строки на два символа
            {
                text += str.text[i]; // Добавление в строку text i-го символа из строки объекта str
            }
    }
};
int main()
{
    Stroka s1, s2; // Создание объектов
    getline(cin, s1.text); // Помещение строки введенной пользователем в поле text объекта s1
    cin.clear(); // Очищение потока для ввода
    s1.print(); // Вывод строки
    s2 = s1; // Запись в поле text объекта s2 строки из объекта s1 без двух первых и последних символов, при помощи перегрузки оператора "="
    s2.print(); // Вывод строки
}
```

```
Ко...
programming
programming
ogrammi
```

ПРИМЕРЫ

Operator – метод класса

```
class Person
{
    int age;
public:
    void operator++()
    {
        ++age;
    }
// ...

};
int main()
{
    Person John;
    ++John;
// ...
}
```

Operator – дружественная функция

```
class Person
{
    int age;
public:
    friend void operator++(Person&);
    // ...
};

void operator++(Person& obj)
{
    ++obj.age;
}

int main()
{
    Person John;
    ++John;
// ...
}
```