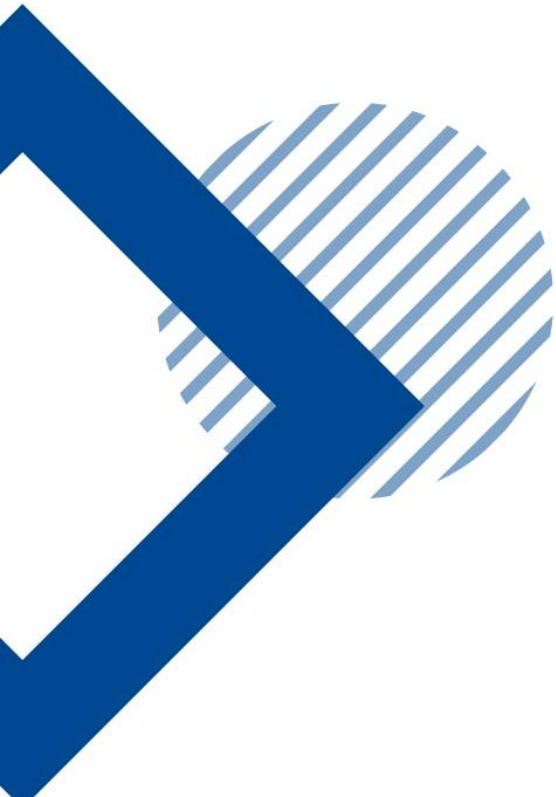
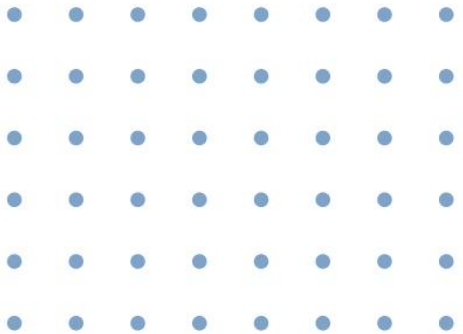
Лекция 1

Парадигмы программирования
Концепции ООП
Классы и объекты
Модификаторы доступа





2022

Парадигмы программирования

Парадигмы программирования

Парадигмы программирования

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование **различных** парадигм (мультипарадигменное программирование). Также существующие парадигмы зачастую пересекаются друг с другом в деталях (например, модульное и объектно-ориентированное программирование).

Парадигмы программирования

Парадигмы программирования

- Императивное программирование
- Декларативное программирование

- Структурное программирование
- Функциональное программирование
- Логическое программирование
- Объектно-ориентированное программирование
  - Компонентно-ориентированное программирование
  - Прототипно-ориентированное программирование
  - Агентно-ориентированное программирование
  - Событийно-ориентированное программирование

Парадигмы программирования

Императивное программирование

парадигма программирования для которой характерно последовательное выполнение инструкций (команд) в исходном коде программы, при этом данные, полученные при выполнении инструкции, могут записываться в память и могут читаться из памяти последующими инструкциями.

Декларативное программирование

парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается ожидаемый результат, а не способ его получения. Такие программы не используют понятия состояния, в частности, не содержат переменных и операторов присваивания.

Парадигмы программирования

Структурное программирование

Структурное программирование – парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков. В соответствии с парадигмой выделяют 3 ключевых аспекта:

- программа строится **без** использования оператора **goto**;
- состоит из трёх базовых управляющих конструкций:
  - **последовательность**,
  - **ветвление**,
  - **цикл**;
- используются подпрограммы.

При этом разработка программы ведётся пошагово, методом «сверху вниз».

Парадигмы программирования

Функциональное программирование

Функциональное программирование – парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм).
Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

Парадигмы программирования

Логическое программирование

Логическое программирование – парадигма программирования, основанная на автоматическом доказательстве теорем, а также раздел дискретной математики, изучающий принципы логического вывода информации на основе заданных фактов и правил вывода.
Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций.

Парадигмы программирования

Объектно-ориентированное программирование

Объектно-ориентированное программирование – методология программирования, основанная на представлении программы в виде совокупности **объектов**, каждый из которых является экземпляром определённого **класса**, а классы образуют иерархию наследования.
ООП возникло в результате развития идеологии процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально **не связаны**.

Концепции ООП

Концепции ООП

Концепции ООП

- Инкапсуляция
- Наследование
- Полиморфизм

- Абстракция

Концепции ООП

Абстракция

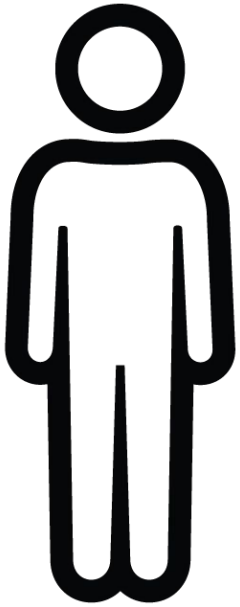
Абстракция – это использование только тех **характеристик** объекта, которые с достаточной точностью **представляют** его в данной системе. Основная идея состоит в том, чтобы представить объект **минимальным** набором полей и методов и при этом с **достаточной** точностью для решаемой задачи.

Концепции ООП

Абстракция

Человек-студент:

- ФИО
- Дата рождения
- Имеющееся образование
- Направление обучения
- Курс
- Группа
- Оценки



Человек-преподаватель:

- ФИО
- Дата рождения
- Должность
- Ученое звание
- Ученая степень
- Трудовой стаж
- Преподаваемые дисциплины

Концепции ООП

Абстракция

Автомобиль:

- Марка
- Модель
- Тип кузова
- Мощность двигателя
- Гос. номер
- VIN номер
- Владелец

- Вес
- Максимальная скорость
- Расход топлива
- Пробег
- Грузоподъемность
- Динамика разгона
- Вместимость

Концепции ООП

Абстракция

Абстракция данных связывает лежащий в основе тип данных с набором операций над ним.
Преимущество абстракции данных в разделении операций над данными и внутреннего представления этих данных, что позволяет изменять реализацию, не затрагивая пользователей типа данных. Такое разделение может быть выражено через специальный «интерфейс», сосредотачивающий описание всех возможных применений программы.

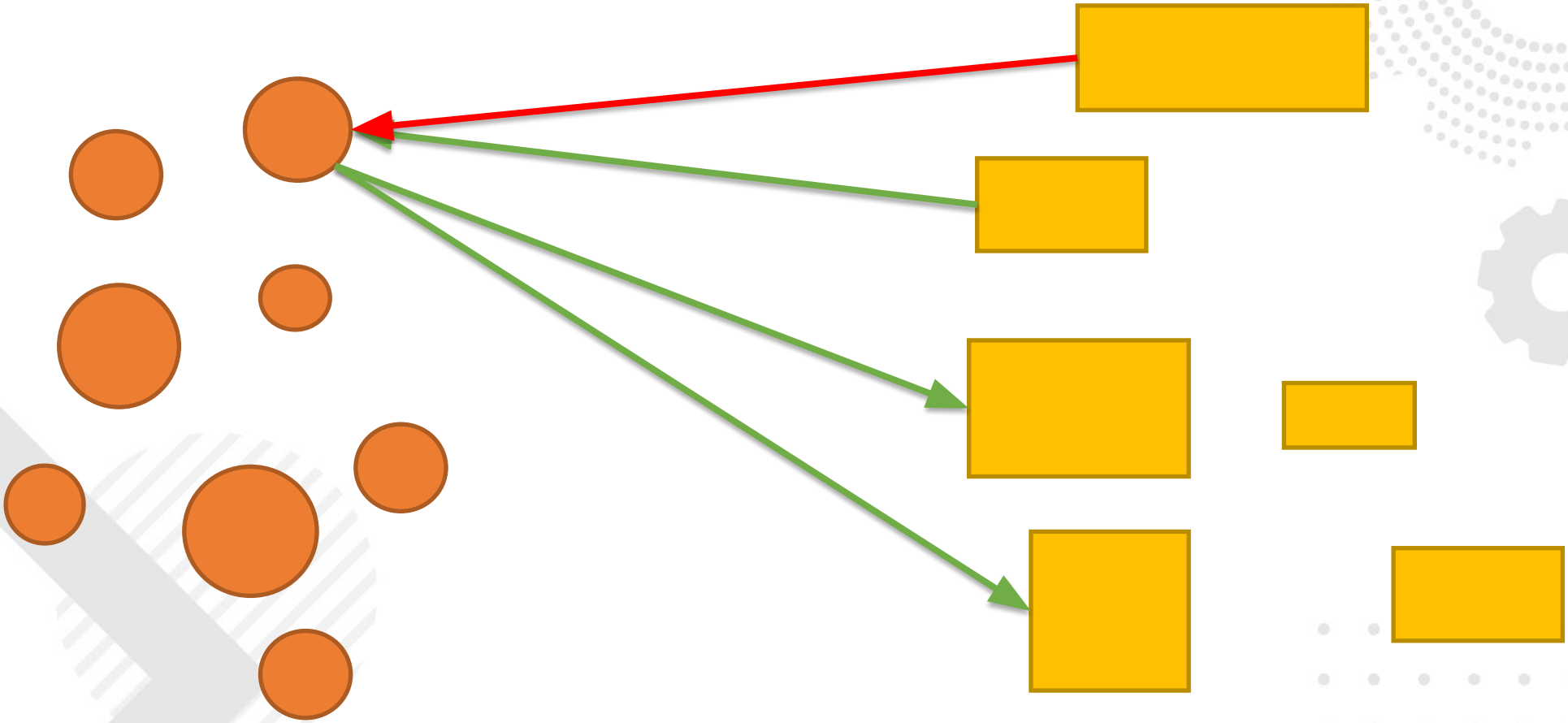
Концепции ООП

Инкапсуляция

Инкапсуляция – в информатике размещение в **одном** компоненте **данных** и **методов**, которые с ними работают.
В реализации большинства языков программирования (C++, C#, Java и другие), обеспечивает механизм **сокрытия**, позволяющий разграничивать **доступ** к различным частям компонента.

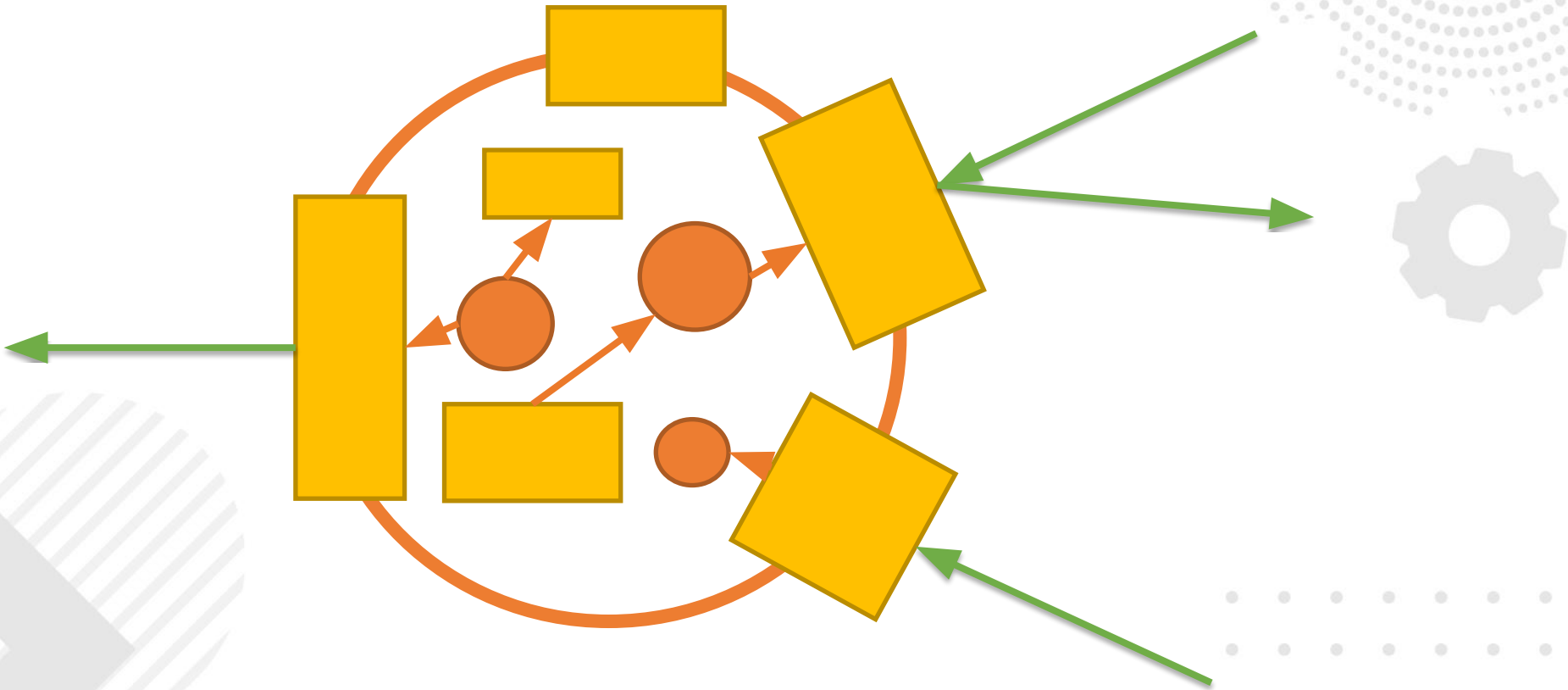
Концепции ООП

Инкапсуляция



Концепции ООП

Инкапсуляция



Концепции ООП

Инкапсуляция

В общем случае в разных языках программирования термин «инкапсуляция» относится к одной или обеим одновременно следующим нотациям:

- механизм языка, позволяющий **ограничить** доступ одних компонентов программы к другим;
- языковая конструкция, позволяющая **связать** данные с методами, предназначенными для обработки этих данных.

Классы и объекты

Классы и объекты

Класс

Класс – это элемент, описывающий абстрактный тип данных и его частичную или полную реализацию.
В ООП представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов.
Класс является ключевым понятием в ООП.
На практике ООП сводится к созданию некоторого количества классов, включая интерфейсы и их реализации, и последующему их использованию.

Классы и объекты

Создание класса

```
namespace Lection01
{
    Ссылок: 0
    class ClassA
    {
    }
}
```

Классы и объекты

Класс включает в себя

- Поля
- Константы
- Методы
- Свойства
- Конструкторы и деструктор
- События
- Индексаторы
- Операторы
- Вложенные типы

Классы и объекты

Поля

Поле – элемент класса для хранения данных.
Поля инициализируются непосредственно при вызове конструктора для экземпляра объекта.

<модификаторы> <тип данных> <имя поля (имя переменной)>;

Классы и объекты

Поля

```
namespace Lection01
{
    Ссылок: 0
    class ClassA
    {
        int _a;

        double d;

        bool f = true;

        ClassB classB = new ClassB();
    }
}
```

Классы и объекты

Типы данных

Данные, с которыми работает программа, хранятся в оперативной памяти. При запуске программы необходимо точно знать, сколько места они займут. Также при сборке компилятор проверяет, что для данных применяют только те операции, которые с ними можно выполнять. Все это задается при описании данных с помощью типа.

Тип данных позволяет определять:

- внутреннее представление данных (множество допустимых значений, занимаемый размер);
- допустимые действия над данными (операции и функции).

Классы и объекты

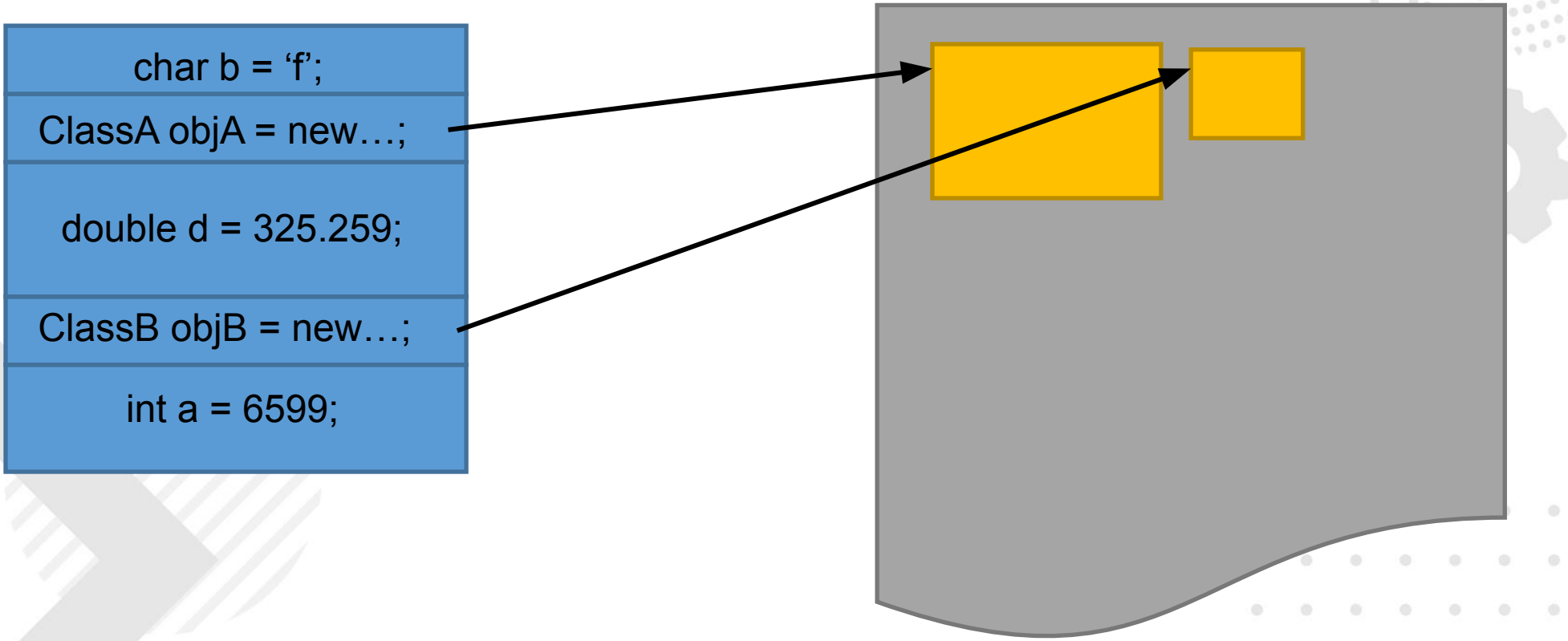
Типы данных. Классификация

Классификации типов данных:

- По строению на простые (не имеют внутренней структуры, например, int, bool, char) и сложные, структурированные (состоят из элементов других типов).
- По своему "создателю" типы можно разделить на встроенные (стандартные, в стандартных библиотеках созданные) и создаваемые программистом (пользовательские типы).
- По способу хранения значений типы делятся на значимые (простые типы, структуры, перечисления), или типы-значения, и ссылочные (классы, массивы, интерфейсы, делегаты).

Классы и объекты

Типы данных. Классификация по способу хранения значений



Классы и объекты

Типы данных. Классификация по способу хранения значений

```
using Lection01;

Console.WriteLine("Hello, World!");

int a1 = 10;

int a2 = a1;

ClassA objA = new ClassA();

ClassA objACopy = objA;

ClassA objA2 = new ClassA();

Console.WriteLine("a1: {0}, a2:{1}, objA:{2}, objACopy:{3}, objA2:{4}", a1, a2, objA._a, objACopy._a, objA2._a);

a2 = 5;

objACopy._a = 5;

Console.WriteLine("a1: {0}, a2:{1}, objA:{2}, objACopy:{3}, objA2:{4}", a1, a2, objA._a, objACopy._a, objA2._a);

objA2._a = 5;
Console.WriteLine("a1 == a2 - {0}", a1 == a2);
Console.WriteLine("objA == objACopy - {0}", objA == objACopy);
Console.WriteLine("objA == objA2 - {0}", objA == objA2);

Console.ReadKey();
```

```
U:\Teaching\RPPO\FirstSemester\FirstSemesterLections\Lection01\bin\Debug\net6.0\Lection01.exe
Hello, World!
a1: 10, a2:10, objA:0, objACopy:0, objA2:0
a1: 10, a2:5, objA:5, objACopy:5, objA2:0
a1 == a2 - False
objA == objACopy - True
objA == objA2 - False
```

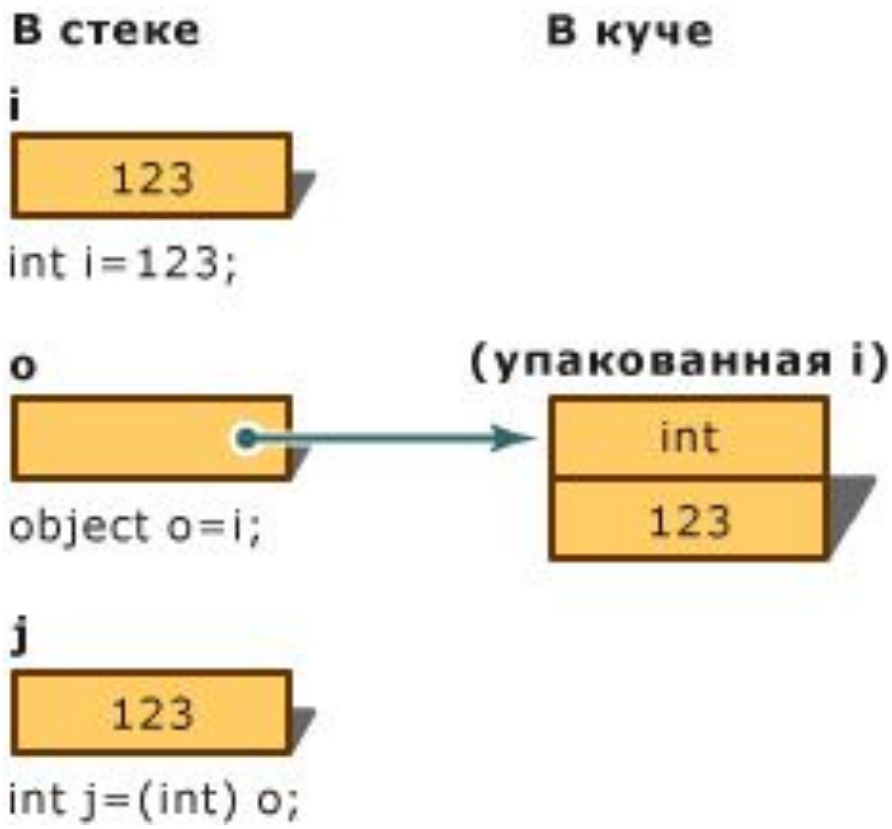
Классы и объекты

Упаковка/распаковка

Упаковка представляет собой процесс преобразования типа **значения** в тип **object** или в любой другой тип интерфейса, реализуемый этим типом значения.
Когда тип значения упаковывается, то создается оболочка значения внутри object и сохранятся в управляемой куче.
Распаковкой называют обратное преобразование из ссылочного типа в тип-значение.
Если величина значимого типа используется в том месте, где требуется ссылочный тип, автоматически выполняется создание промежуточной величины ссылочного типа. При необходимости обратного преобразования с величины ссылочного типа "снимается упаковка", и в дальнейших действиях участвует только ее значение.

Классы и объекты

Упаковка/распаковка



Классы и объекты

Константы

Константы предназначены для описания таких значений, которые **не должны изменяться** в программе. Для определения констант используется либо ключевое слово const, либо ключевое слово readonly. Модификатор readonly указывает на то, что присвоение значения полю может происходить только **при объявлении** или **в конструкторе** (для const только при объявлении) этого класса.
Константное поле **нельзя изменять**.

const <тип данных> <имя поля (имя переменной)>;

readonly <тип данных> <имя поля (имя переменной)>;

Классы и объекты

Константы

```
const int sdfds = 10;

readonly double sdf = 2342.2433;

Ссылок: 2
public ClassA()
{
    sdfds = 234;
    sdf = 65.87;
}

Ссылок: 0
void method()
{
    sdfds = 234;
    sdf = 65.87;

    _a = sdfds;
}
```

Классы и объекты

Методы

Метод – это блок кода, содержащий ряд инструкций. Программа инициирует выполнение инструкций, **вызывая** метод и указывая все аргументы, необходимые для этого метода. Все инструкции выполняются в контексте метода.

Методы объявляются в классе или в структуре путем указания модификаторов доступа, необязательных модификаторов, (abstract или sealed), **возвращаемого значения**, **имени метода** и **передаваемых в метод параметров**. Все эти части вместе представляют собой **сигнатуру метода**.

В качестве возвращаемого значения может любой тип данных, либо ключевое слово **void**.

<модификаторы> <тип данных или void> <имя метода> (передаваемые параметры)

Классы и объекты

Методы

```
void Method()
{
    sdfds = 234;
    sdf = 65.87;

    _a = sdfds;
}

Ссылок: 0
int MethodInt(int f, double dd)
{
    d = dd;
    _a = f;
    return _a + sdfds;
}
```

```
Ссылок: 0
void MethodVoid(int f, double dd)
{
    if (f == 0)
    {
        return;
    }

    d = dd;
    _a = f;
}
```

Классы и объекты

Методы. Передаваемые параметры

Передача параметров переменной длины.
В метод можно передавать переменное число параметров с рядом ограничений:

- все параметры одного типа;
- аргумент в сигнатуре метода, обозначающий переменной число параметров задается последним, после него уже не будет аргументов.

C#: модификатор params после которого указывается одномерный массив.

Java: указывается тип, за которым следует многоточие (...).

Классы и объекты

Методы. Передаваемые параметры

```
Ссылок: 2
void MethodParamList1(params int[] f)
{
    for(int i = 0; i < f.Length; ++i)
    {
        Console.WriteLine(f[i]);
    }
}

Ссылок: 2
void MethodParamList2(double dd, bool flag, params int[] f)
{

}

Ссылок: 0
void MethodParamList3(double dd, params int[] f, bool flag)
{

}
```

```
Ссылок: 0
int MethodInt(int f, double dd)
{
    d = dd;
    _a = f;
    MethodParamList1(_a, f, 35, 4, 23,45,57);
    MethodParamList2(dd, true, _a, f, 35, 4, 23, 45, 57);
    MethodParamList2(dd, true);
    return _a + sdfds;
}
```

Классы и объекты

Методы. Передаваемые параметры

Передача значимых типов по ссылке (упаковка):
В случае, когда необходимо передать переменную значимого типа в метод по ссылке (чтобы, например, поменять там значение, хранимое в переменной, или по каким-то причинам чтобы не создавался дубликат этой переменной), то существует ряд механизмов для этого:

- оператор ref просто передает значимый тип как ссылочный;
- оператор in, передает значимый тип как ссылочный, но при этом не позволяет менять значение в вызываемом методе;
- оператор out передает значимый тип как ссылочный и обязывает присвоить ему значение в вызываемом методе.

Классы и объекты

Методы. Передаваемые параметры

```
Ссылок: 1
public void MethodLink(int f, in int d, ref int r, out int a)
{
    _a = d;
    f = 45;
    d = 23;
    _a = r;
    r = 34;
    if (_a > 34)
    {
        a = 345456;
    }
    else
    {
        a = -45;
    }
}
```

```
int f1 = 345;
int f2 = 245;
int f3 = 567;
int f4;

objA.MethodLink(f1, in f2, ref f3, out f4);
Console.WriteLine("f1: {0}, f2:{1}, f3:{2}, f4:{3}", f1, f2, f3, f4);
```

```
U:\Teaching\RPPO\FirstSemester\FirstSemesterLections\Lection01\bin\Debug\net6.0\Lection01.exe
f1: 345, f2:245, f3:34, f4:345456
```

Классы и объекты

Методы. Передаваемые параметры

В случае, когда необходимо расширить существующий метод новым параметром, но при это не менять уже имеющиеся места вызова этого метода можно использовать необязательные параметры. Для таких параметров необходимо объявить значение по умолчанию.
Также существуют понятие именованных параметров. Этот механизм позволяет игнорировать установленный порядок передачи параметров в метод и передавать их в ином порядке, либо пропускать часть необязательных параметров.

Классы и объекты

Методы. Передаваемые параметры

```
Ссылок: 4
public void Method345(int f, double dd, int sdfd = 0, char c = ' ')
{
    if (sdfd != 0)
    {
        // логика
    }
    // большая логика + sdfd
    _a = f + sdfd;
}
```

```
Ссылок: 0
void MethodVoid(int f, double dd)
{
    if (f == 0)
    {
        return;
    }

    Method345(4, dd);
    Method345(4, dd, 34);
    Method345(4, dd, c: 'e');
    Method345(dd: d, f: 4, c: 'e');

    d = dd;
    _a = f;
}
```

Классы и объекты

Свойство

Свойство – это элемент класса, предоставляющий гибкий механизм для чтения, записи или вычисления значения закрытых полей. Свойства фактически представляют собой специальные методы, называемые методами доступа. Это позволяет легко получать доступ к данным и помогает повысить безопасность и гибкость методов.
У свойства могут быть два ключевых слова get и set. Первый используется для чтения поля, второй для записи в поле. Свойство может иметь только одно из ключевых слов.

<модификаторы> <тип данных> <имя свойства> { get { } set { }}

Классы и объекты

Свойство

```
public int P { get; set; }

private int _p;

public int get_p() { return _p; }

public void set_p(int value) { _p = value; }

double DD1 { get; }

double DD2 { set { } }

DateTime date = DateTime.Now;

public string Date
{
    get { return date.ToString("dd.MM.YYYY"); }
    set { date = Convert.ToDateTime(value); }
}
```

Классы и объекты

Объект

Объект – это некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы). Как правило, при рассмотрении объектов выделяется то, что объекты принадлежат одному или нескольким классам, которые определяют поведение (являются моделью) объекта. Термины «**экземпляр** класса» и «**объект**» взаимозаменяемы.

Классы и объекты

Объект

```
namespace Lection01
{
    internal class Car
    {
        public string Marka { get; set; }

        public string Model { get; set; }

        public string TypeOfBoard { get; set; }

        public string VIN { get; set; }

        public string GosNumber { get; set; }

        public int MaxSpeed { get; set; }

        public void Move() { }

        public void SitDown() { }
    }
}
```

```
Car car = new Car()
{
    Marka = "Lada",
    Model = "Vesta",
    TypeOfBoard = "Sedan",
    VIN = "54356464435436364",
    GosNumber = "r233awe73",
    MaxSpeed = 100
};
car.SitDown();
car.Move();

Car car2 = new Car();
car2.Marka = "Kia";
car2.Model = "Rio";
car2.TypeOfBoard = "Sedan";
car2.VIN = "345687977776542";
car2.GosNumber = "r224awe73";
car2.MaxSpeed = 130;

car2.SitDown();
car2.Move();
```

Модификаторы доступа

Модификаторы доступа

Модификаторы доступа

Поскольку методы класса могут быть как чисто внутренними, обеспечивающими логику функционирования объекта, так и внешними, с помощью которых взаимодействуют объекты, необходимо обеспечить скрытость первых при доступности извне вторых. Для этого в языки вводятся специальные синтаксические конструкции, явно задающие область видимости каждого члена класса.

Традиционно это модификаторы public, protected и private, обозначающие, соответственно, открытые члены класса, члены класса, доступные внутри класса и из классов-потомков, и скрытые, доступные только внутри класса.

Конкретная номенклатура модификаторов и их точный смысл различаются в разных языках.

Модификаторы доступа

public

Общий доступ является уровнем доступа с **максимальными** правами. Ограничений доступа к общим членам **не существует**.

private

Закрытый доступ является уровнем доступа с **минимальными** правами. Доступ к закрытым членам можно получить только **внутри тела класса**, в которой они объявлены. Если у элементов класса не указывать модификатор доступа, то по умолчанию у них будет проставляться private.

Модификаторы доступа

public и private

```
namespace Lection01
{
    class ClassB
    {
        private int field;

        public int Field
        {
            get { if (field >= 0) return field; return -1; }
            set { if (value <= 0) field = value * D(); }
        }

        int D() { return -1; }
    }
}
```

```
ClassB objB = new ClassB();
objB.Field = -12;
Console.WriteLine("f1: {0}", objB.Field);
```

```
U:\Teaching\RPPO\FirstSemester\FirstSemesterLections\Lection01\bin\Debug\net6.0\Lection01.exe
f1: 12
```

Спасибо за внимание!

Рады видеть Вас на наших мероприятиях!

Ульяновский государственный
технический университет

2021