



ОНЛАЙН-ОБРАЗОВАНИЕ

Онлайн-образование

Не забыть включить запись!





Меня хорошо видно && слышно?

Ставьте  , если все хорошо
Напишите в чат, если есть проблемы

Классы как воплощение принципов ООП



Рочев Константин Васильевич

Главный программист Insense Arts LLC, доцент каф. ВТИСИТ УГТУ

k@rochev.ru

+7(904)109-83-18

Преподаватель

Константин Рочев

Опыт программирования на языке C# 10 лет

Немного о текущей деятельности:

Главный программист Insense Arts LLC (разработка MMORPG) (с 02.2018).

Член-корреспондент академии информатизации образования (с 06.2018).

Доцент кафедры вычислительной техники, информационных систем и технологий УГТУ (с 10.2017).

Основатель и главный редактор журнала «Информационные технологии в управлении и экономике» (с 01.2012).

Основатель, ведущий программист и старший научный сотрудник проекта Межвузовской информационной системы оценки деятельности студентов вузов (с 12.2012).

И предыдущей:

Ответственный редактор журнала «Ресурсы Европейского Севера. Технологии и экономика освоения» (05.2015-05.2018).

Супервайзер (руководитель проекта, главный разработчик и администратор) Индексной системы стимулирования трудового коллектива УГТУ (12.2012-08.2018).

Программист-полуприлансер (07.2017-12.2017).

Заведующий кафедрой вычислительной техники, информационных систем и технологий УГТУ (04.2016-10.2017).

Директор экспертно-аналитического центра УГТУ (03.2016-04.2016).

Старший преподаватель кафедры организации и планирования производства Ухтинского государственного технического университета (09.2015-06/2016).

Заведующий лабораторией информационных систем в экономике Ухтинского государственного технического университета (10.2012-02.2016).

Директор Студии мобильных разработок «Л-ИС» (с 2014).

Microsoft student partner (05.2012-06.2015).

Аспирант направления 08.00.05 — Экономика и управление народным хозяйством (10.2011-06.2015).

Ассистент кафедры информационных систем и технологий Ухтинского государственного технического университета (09.2011-07.2015).

Директор направления информационных технологий Ухтинского отделения Ассоциации молодых предпринимателей России (с 03.2012).

Руководитель федеральной темы НИР 14.132.21.1031 «Система индексно-рейтинговой оценки и материального стимулирования студентов вуза» (09.2012-10.2013).

И др.



Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом

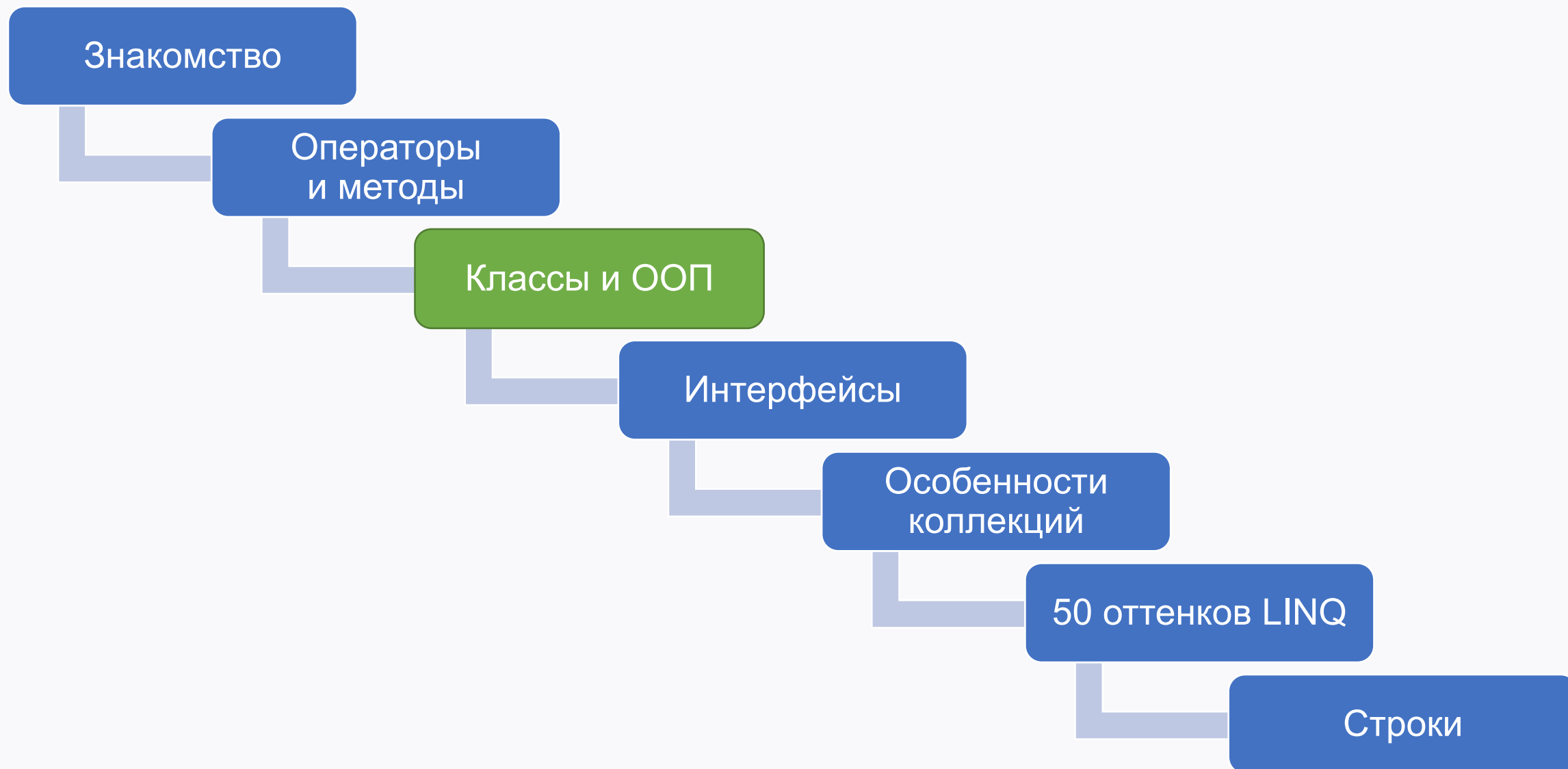


Off-topic обсуждаем в Slack #канал группы или #general



Вопросы вижу в чате, могу ответить не сразу

Карта курса



Цели вебинара | После занятия вы сможете

1

Использовать классы более профессионально

2

Реализовывать основные принципы ООП на C#

3

Разбираться в тонкостях наследования и полиморфизма

Смысл | Зачем вам это уметь

1

Понимать объектно-ориентированный код

2

**Снижать сложность реализации
больших программ с помощью ООП**

3

**Писать код более подходящий
для повторного использования**



1

Небольшой опрос

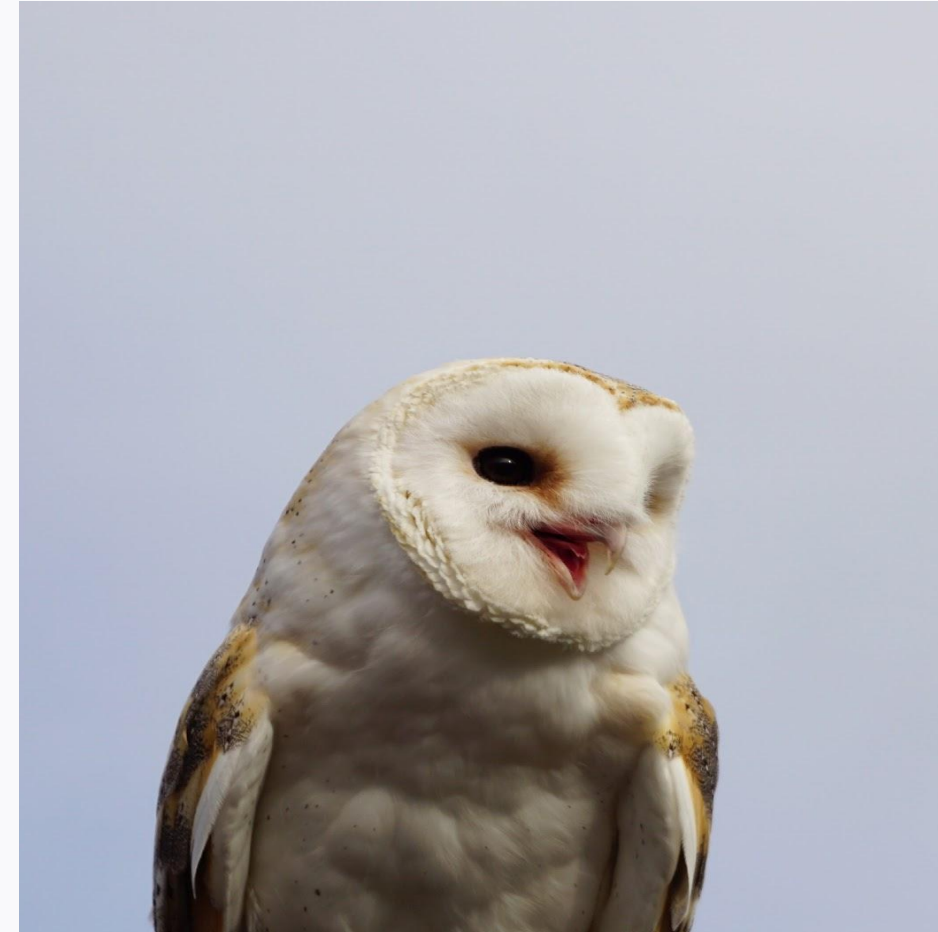
The background of the slide is a blue-tinted aerial photograph of a dense city skyline, likely New York City. Overlaid on this is a semi-transparent network pattern consisting of white dots connected by thin white lines, creating a web-like effect across the center of the image.

2

Немного теории

- C# - объектно-ориентированный язык программирования.

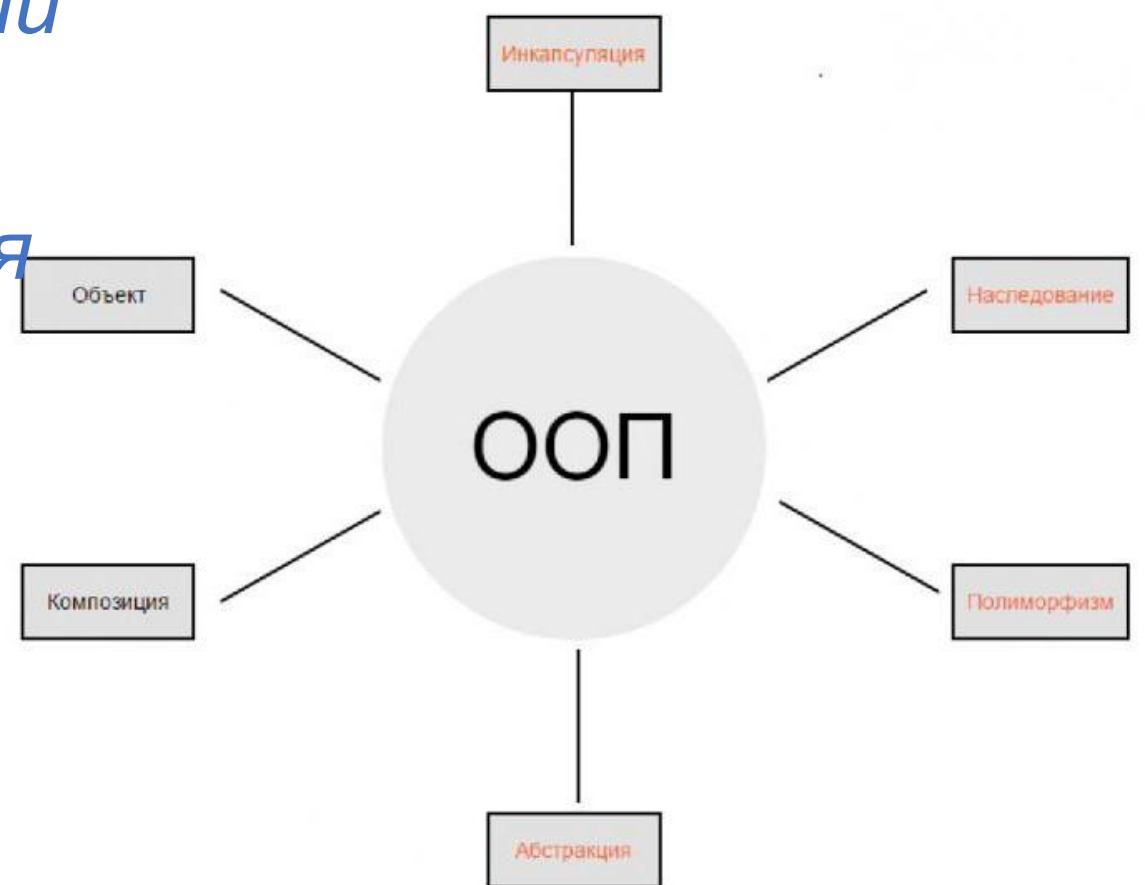
Разработан в 1998–2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270



Объектно-ориентированное программирование

это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Г. Буч



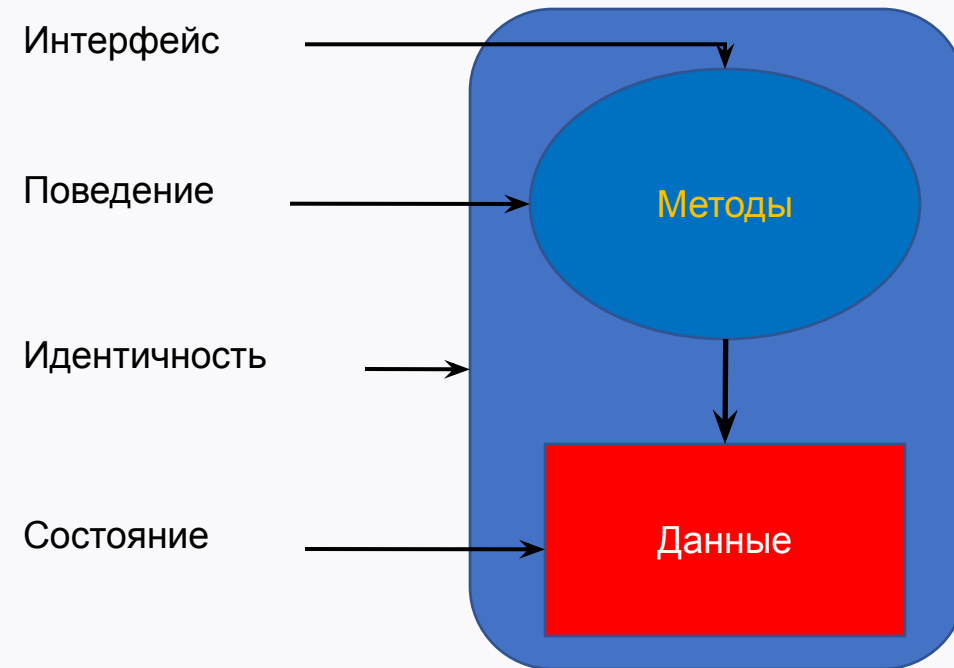
Что такое объект?

Объект обладает

- ▢ состоянием,
- ▢ поведением,
- ▢ идентичностью;

*Структура и поведение
схожих объектов
определяет общий для них
класс.*

*Термины "экземпляр класса" и
"объект"
взаимозаменяемы.*



Состояние

Состояние объекта характеризуется перечнем (обычно статическим) всех *свойств* данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств

Г.Буч

```
1. namespace Otus
2. {
3.     public class Lesson
4.     {
5.         public string Title { get; set; }
6.         public DateTime Date { get; set; }
7.         public string Content { get; set; }
8.     }
9. }

10. var l = new Lesson()
11. {
12.     Title = "ООП",
13.     Date = new DateTime(2019, 11, 11),
14.     Content = "...",
15. };
```


Поведение

Поведение - это то, как объект действует и реагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

□ **Состояние** объекта представляет суммарный результат его поведения.

Г. Буч

```
1. namespace Otus
2. {
3.     public class Lesson
4.     {
5.         public string Title { get; set; }
6.         public DateTime Date { get; set; }
7.         public string Content { get; set; }
8.
9.         public void ChangeDate(int addDays)
10.        {
11.            Date = Date.AddDays(addDays);
12.        }
13.    }
```

Поведение

Метод определяется как *процедура* или *функция*, которая *изменяет состояние объекта* или заставляет объект *отправить сообщение*.

Описание (сигнатура) метода называется *операцией*. Операции показывают, какие сообщения объект может успешно обработать.

```
1. namespace Otus
2. {
3.     public class Lesson
4.     {
5.         public string Title { get; set; }
6.         public DateTime Date { get; set; }
7.         public string Content { get; set; }
8.
9.         public void ChangeDate(int addDays)
10.        {
11.            Date = Date.AddDays(addDays);
12.        }
13.    }
```


Идентичность

**Идентичность
(уникальная
идентичность,
объектная
идентичность) –
это такое свойство
объекта, которое
отличает его от всех
других объектов**
Хошафян, Коуплэнд

- Мир
 - Положение в пространстве и времени.
- Реляционные модели
 - Понятие первичного ключа.
- Объектно-ориентированные модели:
 - Понятие уникального идентификатора объекта (UUID, например).
 - Уникальное размещение объектов в памяти.

Идентичность

Идентичность
(уникальная
идентичность,
объектная
идентичность) –
это такое свойство
объекта, которое
отличает его от всех
других объектов
Хошафян, Коуплэнд

```
1. public class Lesson
2. {
3.     static int _idIncrement = 0;
4.     public int Id { get; set; }
5.     public Guid GlobalId { get; set; }

6.     public Lesson()
7.     {
8.         var l = new Lesson()
9.         {
10.             Id = _idIncrement,
11.             GlobalId = Guid.NewGuid(),
12.         };
13.     }
14. }
```


Класс

*Класс - это некое
множество объектов,
имеющих **общую**
структуру и общее
поведение.*

Г. Буч

```
1.  public class Lesson
2.  {
3.      public Guid Id { get; set; }
4.      public string Title { get; set; }
5.      public DateTime Date { get; set; }
6.      public string Content { get; set; }
7.
8.      public void ChangeDate(int addDays)
9.      {
10.         Date = Date.AddDays(addDays);
11.     }
```




LIVE

Давайте напишем класс
Live-coding



Давайте напишем класс... но начнем не с него 😊

```
// структуры - они как классы,  
// но типы-значения и не наследуются  
public struct Point  
{  
    public double X, Y;  
}  
  
// интерфейсы - тоже как классы,  
// но без реализации и всё публично  
public interface IFigure  
{  
    List<Point> Points { get; }  
    double GetPerimetr();  
}  
  
public abstract class Figure : IFigure { }  
  
public class Round : Figure { }  
  
public class Triangle : Figure { }
```

Фундамент объектно-ориентированных методов

- Абстрагирование
- Модульность
- Иерархия

- Наследование
- Инкапсуляция
- Полиморфизм

- Типизация
- Параллелизм
- Сохраняемость

```
// структуры - они как классы,  
// но типы-значения и не наследуются  
public struct Point  
{  
    public double X, Y;  
}
```

```
// интерфейсы - тоже как классы,  
// но без реализации и всё публично  
public interface IFigure  
{  
    List<Point> Points { get; }  
    double GetPerimetr();  
}
```

```
public abstract class Figure : IFigure { }
```

```
public class Round : Figure { }
```

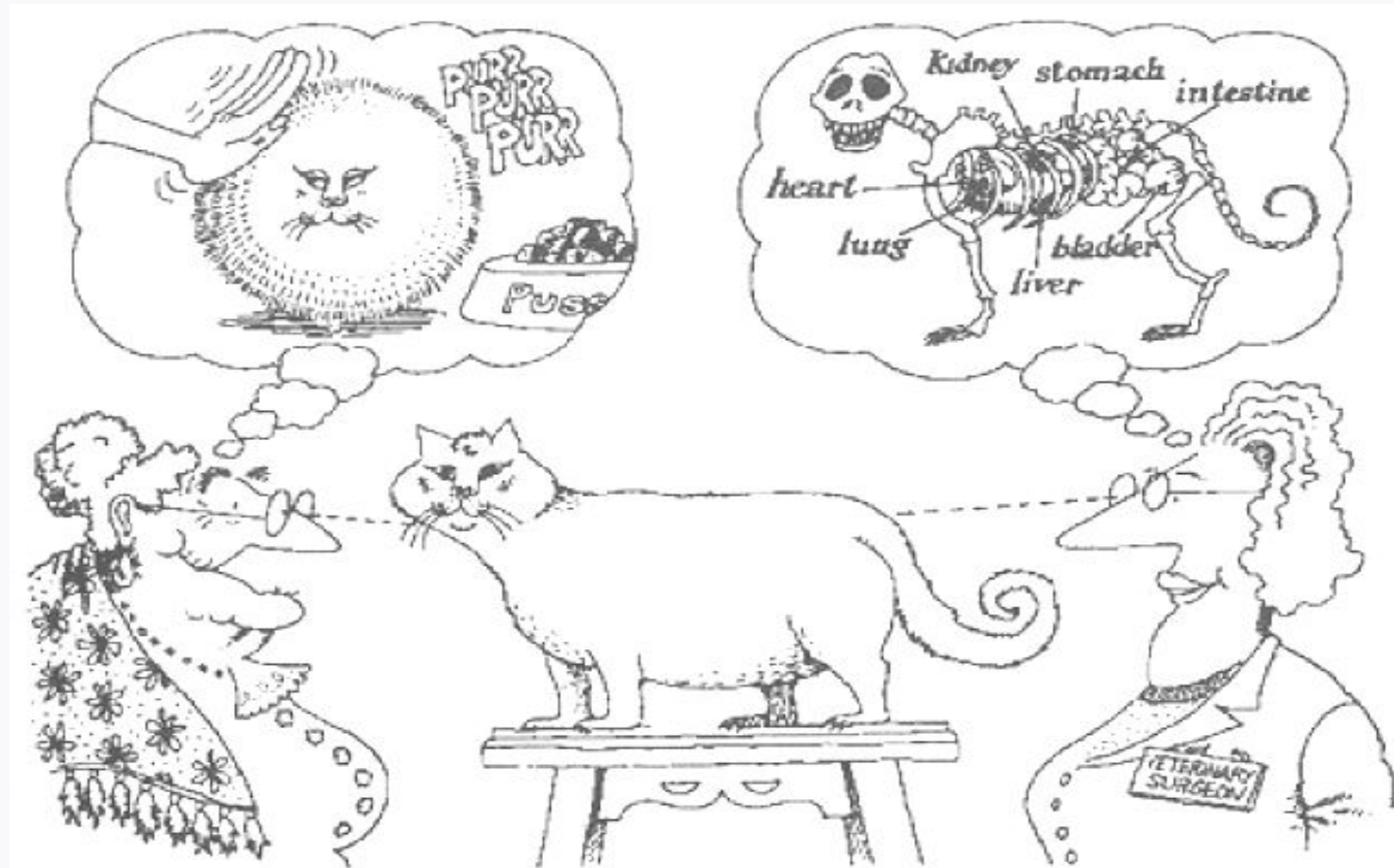
```
public class Triangle : Figure { }
```


Абстракция

Абстракция выделяет *существенные характеристики* некоторого объекта, отличающие его от всех других видов объектов и, таким образом, четко *определяет* его концептуальные *границы с точки зрения наблюдателя*
Г. Буч.

```
abstract class Cat
{
    Color CatColor;
    public void Caress();
    public void Feed();
}

abstract class Cat
{
    double Weight;
    int Age;
    List<string> CaseHistory;
    public void OpenCaseHistory();
    public void CloseCaseHistory();
}
```



Инкапсуляция

Инкапсуляция — это процесс отделения друг от друга *элементов объекта*, *определяющих* его *устройство и поведение*; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации

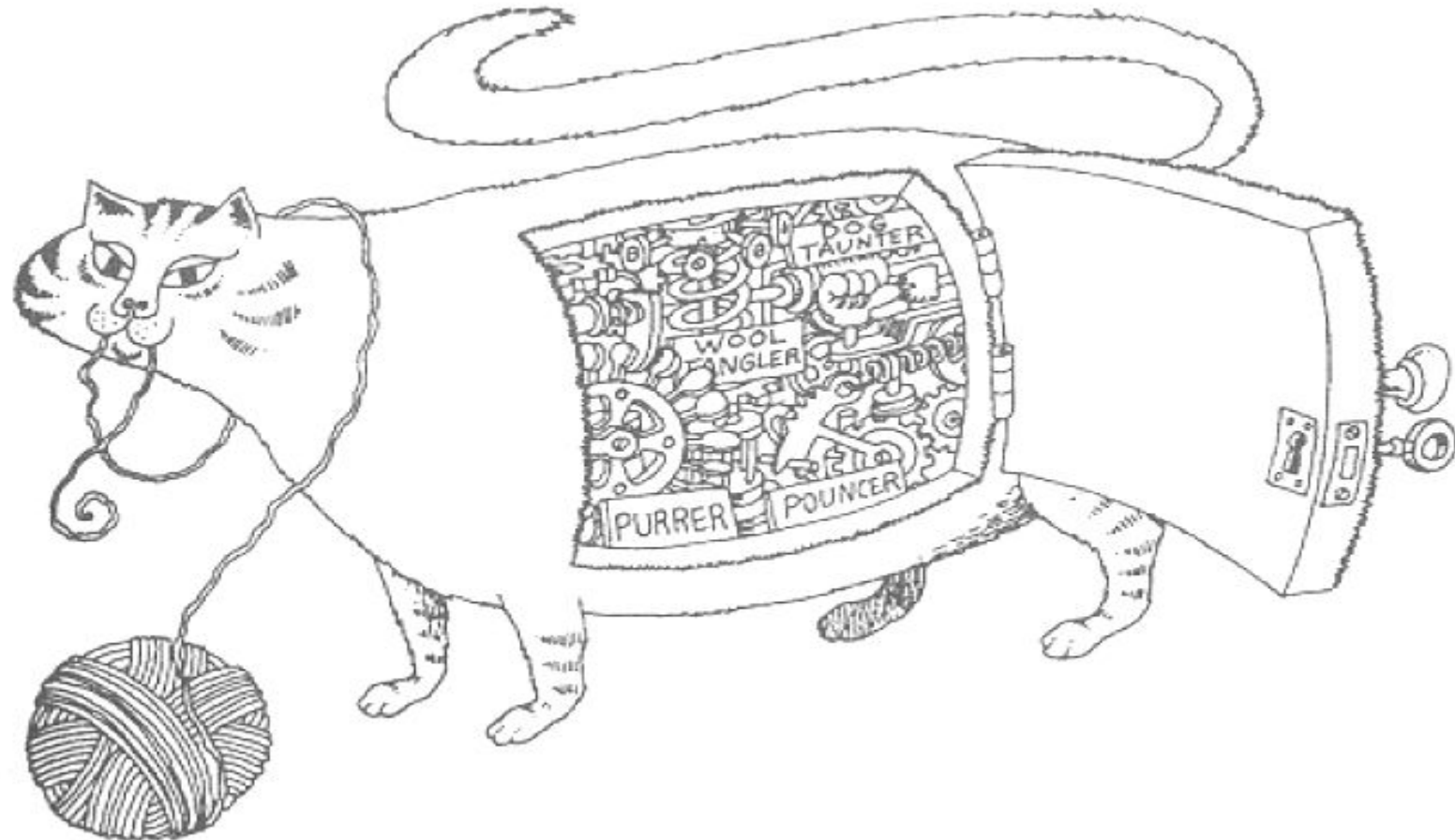
Г. Буч

```
class Cat
{
    private Color CatColor;

    private void Purr() { DoPurr(); }

    protected virtual void DoPurr ();

    public void Caress() { Purr(); }
};
```



Модульность

Модульность — это свойство системы, которая была *разложена на внутренне связанные, но слабо связанные между собой модули*.

Г. Буч.

```
class Cat { }
```

```
class Tooth { }
```

```
class Wool { }
```

```
class Nail { }
```

```
class ...
```



Иерархия

Иерархия — это упорядочение абстракций, расположение их по уровням.

Виды иерархий

Одиночное наследование («is kind of»)

```
class Cat : Animal { ... }
```

Множественное наследование («is kind of»)

```
class Cat : Animal, IFeline, IPet { ... }
```

Агрегация («is part of»)

```
class Cat
```

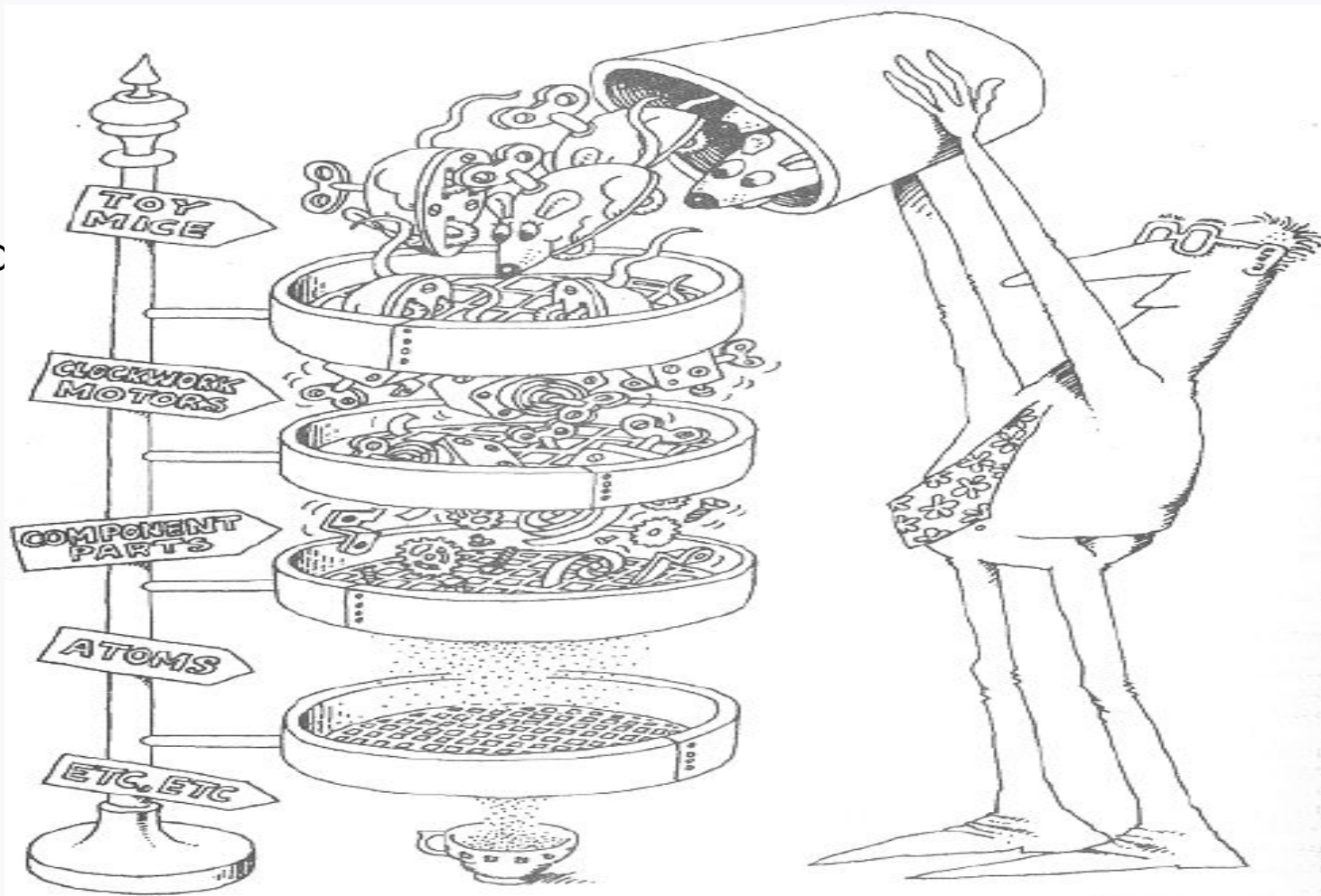
```
{
```

```
    Wool Wool { get; set; }
```

```
    List<Tooth> Teeth { get; set; }
```

```
    List<Nail> Nails { get; set; }
```

```
}
```



Наследование

Наследование (отношение «обобщение/специализация», АКО – A kind of) – *возможность класса наследовать структуру и поведение другого класса* (для одиночного наследования) или нескольких классов (множественное наследование).

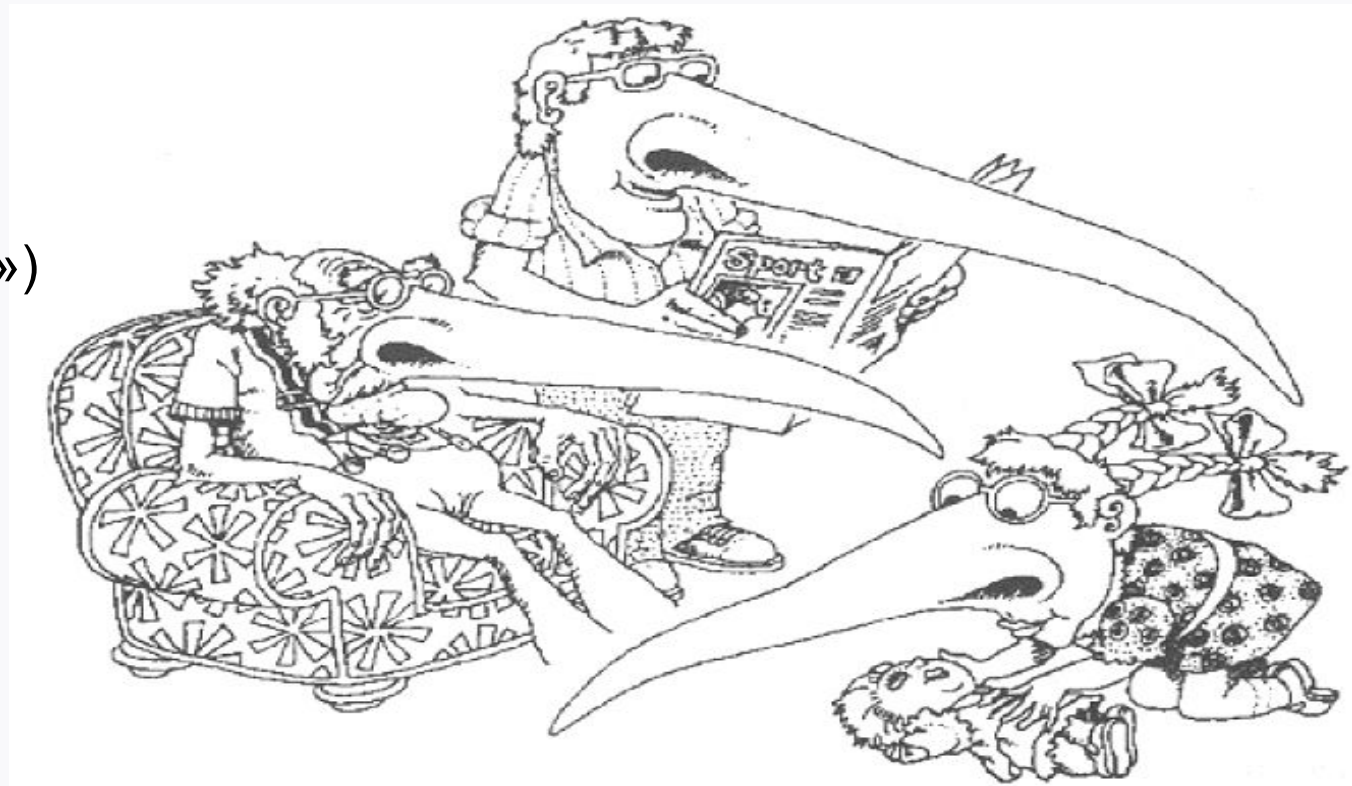
Виды иерархий

Одиночное наследование («is kind of»)

```
class Cat : Animal { ... }
```

Множественное наследование («is kind of»)

```
class Cat : Animal, IFeline, IPet { ... }
```



Полиморфизм

Полиморфизм (*Polymorphe* греч.) Дословно - множественность форм. *Способность объекта или оператора ссылаться на объекты разных классов на стадии выполнения. Таким образом, полиморфные сообщения могут интерпретироваться по-разному, если они получены разными объектами или в разных контекстах.*

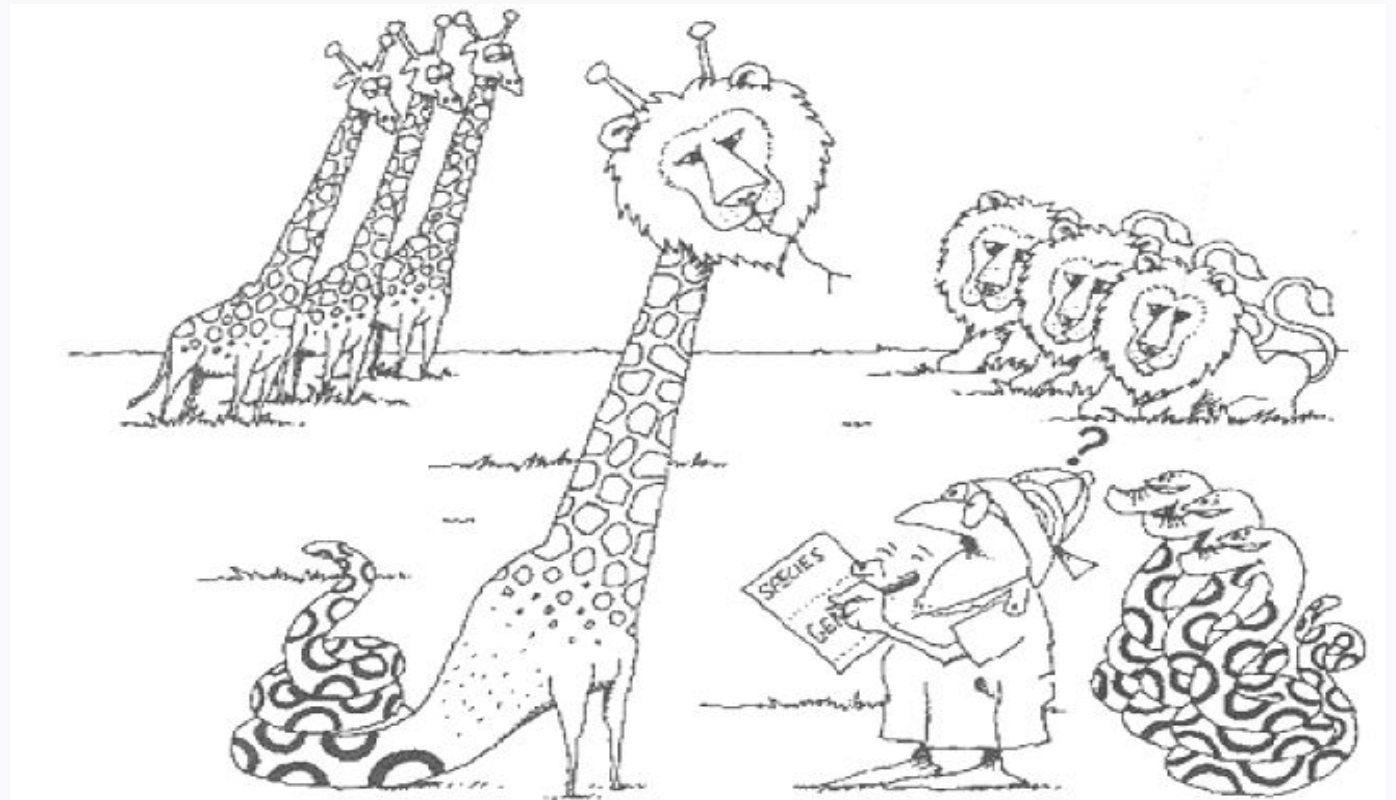
И. Грэхем

```
interface class Animal
{
    void Goto(float x, float y, float z);
}

class Snake : Animal
{
    public void Goto(float x, float y, float z) { ползём }
}

class Cat : Animal
{
    public void Goto(float x, float y, float z) { бежим }
    public void Goto(Point3 target) { бежим }
}

Animal c = new Cat();
c.Goto(1,2,4);
c.Goto(new Point3());
```



Полиморфизм

```
abstract class Animal
{
    internal virtual void Goto(float x, float y, float z);
}

class Snake : Animal
{
    // полиморфизм наследования, ползем
    internal void override Goto(float x, float y, float z);
}

class Cat : Animal
{
    // полиморфизм наследования, бежим
    internal void override Goto(float x, float y, float z);
    public void Goto(Point3 target); // перегрузка
}
```

```
Animal c = new Cat();
Animal s = new Snake();
c.Goto(1,2,4); // бежим
c.Goto(new Point3()); // не доступен
```

```
// Приведение типов
var cat = (cat)c; // +
var cat2 = (cat)s; // ошибка
cat.Goto(new Point3()); // +

или
var cat = c as cat; // +
var cat2 = s as cat; // s == null;
```

Специальный полиморфизм	Универсальный полиморфизм
Перегрузка (времени компиляции)	Параметрический (времени компиляции)
Приведение типов (времени компиляции) (времени выполнения)	Полиморфизм включения (наследования) (времени выполнения)

Параллелизм

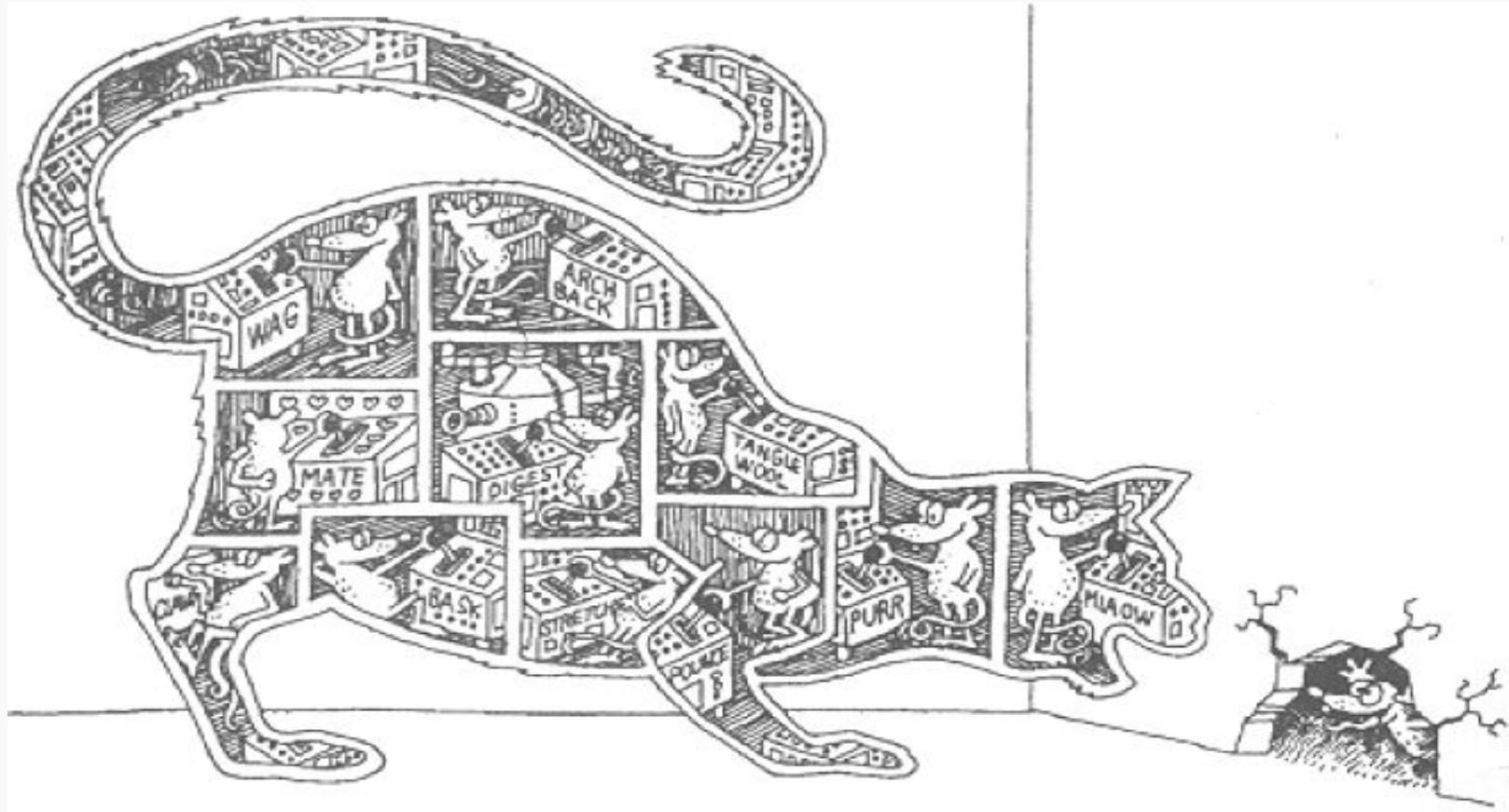
Параллелизм — это свойство, отличающее активные объекты от пассивных.

Г. Буч

Активные объекты имеют собственный поток (thread) управления.

Процесс (поток управления) — это фундаментальная единица действия в системе. Каждая программа имеет по крайней мере один поток управления, параллельная система имеет много таких потоков.

```
public void Parallel()  
{  
    Thread thread = new Thread(() =>  
    {  
        // Задача, выполняемая  
        // в другом потоке  
    });  
    thread.Start();  
}
```



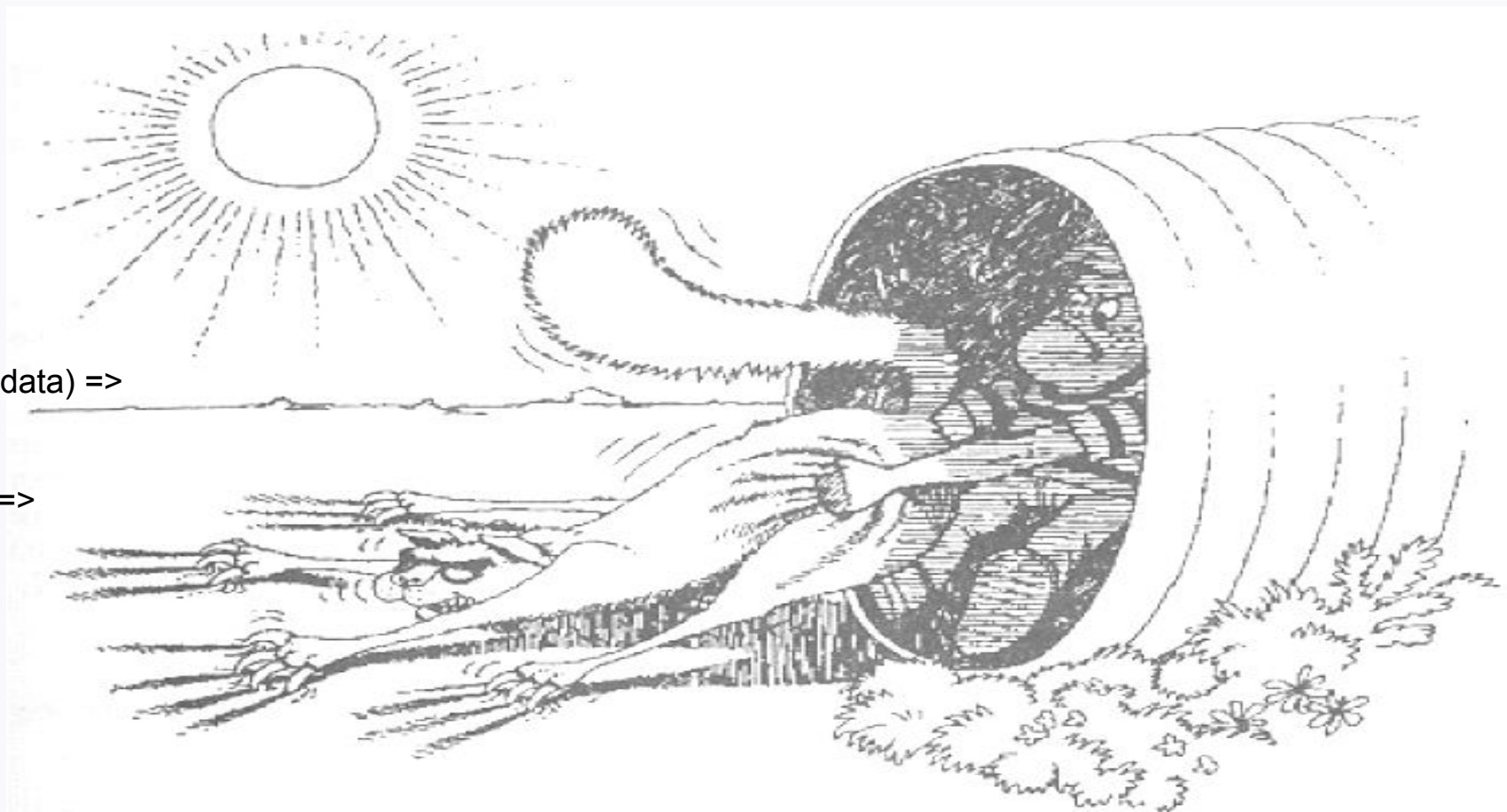
Сохраняемость

Сохраняемость — способность объекта существовать во времени, переживая породивший его процесс, и (или) в пространстве, перемещаясь из своего первоначального адресного пространства.

```
using Newtonsoft.Json;

namespace Common
{
    public static class SerializationExt
    {
        public static T FromJson<T>(this string data) =>
            JsonConvert.DeserializeObject<T>();

        public static string ToJson(this object t) =>
            JsonConvert.SerializeObject();
    }
}
```



Типизация

Типизация — это способ защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.

Г. Буч

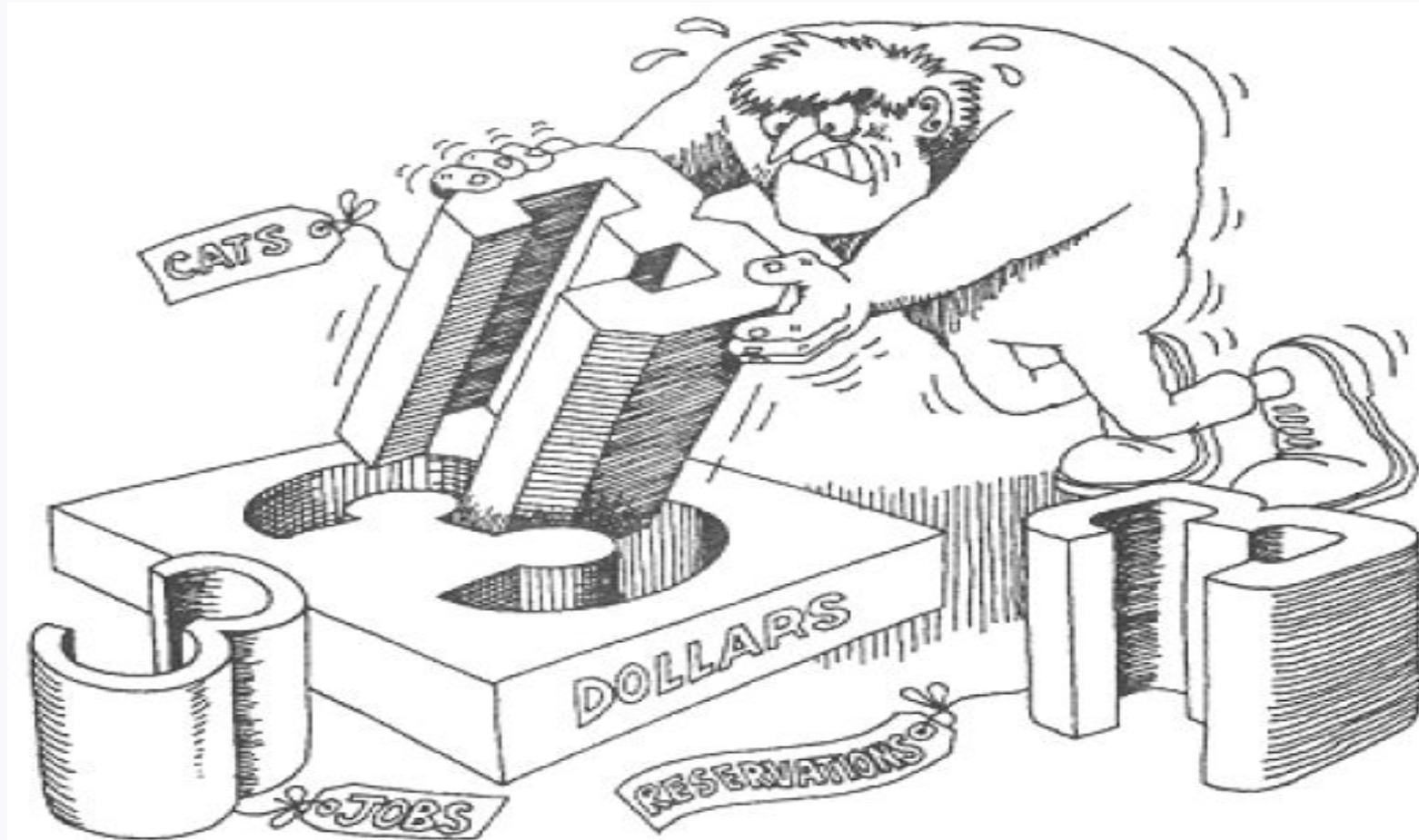
```
class Animal { }  
class Cat : Animal { }  
class Bot { }
```

```
List<Animals> aa = new List<Animals>();
```

```
aa.Add(new Animal()); // Класс подходит  
aa.Add(new Cat()); // Наследники тоже  
aa.Add(new Bot()); // Ошибка
```

C# - строго типизированный язык

- Все операции подвержены строгому контролю типов данных
- Операции «привязаны» к типам
- Все типы данных наследуются от типа object и обладают его базовым функционалом



Типизация в языке C#

- Типы данных
 - Типы значений
 - Хранятся в стэке
 - Передаются по значению
 - Ссылочные типы
 - Хранятся в управляемой куче
 - Передаются по ссылке
 - Могут ссылаться на один и тот же объект

Типы данных языка C#

Типы значений

Простые типы

- Целочисленный со знаком: sbyte, short, int, long
- Целочисленный без знака: byte, ushort, uint, ulong
- Символы Юникода (UTF-16): char
- Бинарный оператор IEEE с плавающей запятой: float, double
- Десятичное значение с повышенной точностью и плавающей запятой: decimal
- Логическое значение: bool

Типы перечисления

- Пользовательские типы в формате enum E {...}

Типы структур

- Пользовательские типы в формате struct S {...}

Типы значений, допускающие значение NULL

- Расширения других типов значений, допускающие значение null

Ссылочные типы

Типы классов

- Исходный базовым классом для всех типов: object
- Классы CLR-библиотек:
 - Строки Юникода (UTF-16): string и др.
- Пользовательские типы в формате class C {...}

Типы интерфейсов

- Пользовательские типы в формате interface I {...}

Типы массивов

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

Типы делегатов

- Пользовательские типы в формате delegate int D(...)

Тип Object. Упаковка / Boxing

```
using System;
class BoxingExample
{
    static void Main()
    {
        int i = 123;
        object o = i; // Boxing
        int j = (int)o; // Unboxing

        Console.WriteLine(object.ReferenceEquals(o, i));
        Console.WriteLine(object.Equals(o, i));
        Console.WriteLine(o.Equals(i));
        Console.WriteLine(o == i);
    }
}
```

Содержимое структур и классов

Константы

- Константные значения, связанные с классом.
 - `const int X = 10;`

Поля

- Переменные класса.
 - `int X; float Y = 10; string Z;`

Методы

- Вычисления и действия, которые может выполнять класс.
 - `int Count() { return 5; }` `float Kvadrat(float x) => x*x;`

Свойства

- Действия, связанные с чтением и записью данных класса.
 - `int X { get; set; }` `int X => 5;`

Индексаторы

- Действия, реализующие индексирование экземпляров класса, чтобы обращаться к ним как к массиву.

События

- Уведомления, которые могут быть созданы этим классом.
 - `event Action OnClick;`

Операторы

- Поддерживаемые классом операторы преобразования и выражения.

Конструкторы

- Действия, необходимые для инициализации экземпляров класса или класса в целом.

Методы

- Действия, выполняемые перед окончательным удалением экземпляров класса.

законч

Типы

- Вложенные типы, объявленные в классе.



LIVE

Давайте доведем классы
Live-coding



Давайте продолжим писать наши классы

```
public interface IFigure // Абстракция
{
    List<Point> Points { get; }
    double GetPerimetr();
}

internal abstract class Figure : IFigure // Наследование
{
    protected List<Point> _points; // Инкапсуляция
    public List<Point> Points => _points; // Реализация
    public virtual double GetPerimetr(); // Абстракция
}

public class Round : Figure
{
    public double Radius { get; set; }
    public Point Centr => Points[0];
    public override double GetPerimetr() => Radius * 2 * Math.PI; // Реализация, полиморфизм
    public Round(double x, double y, double radius) {
        Radius = radius, Points.Add(new Point() { x = x, y = y });
    }
}
```




3

Пара задач на типизацию,
наследование и полиморфизм

Задача 1. Что получится в консоли?

```
public void Show()
{
    A a = new A(), b = new B(), c = new C();
    I ia = new A(), ib = new B(), ic = new C();
    Console.WriteLine($"{a.P}, {b.P}, {c.P}, {ia.P}, {ib.P}, {ic.P}");
}

interface I
{
    int P { get; }
}

class A : I
{
    public virtual int P => 0;
}

class B : A
{
    public override int P => 1;
}

class C : B, I
{
    public int P => 2;
}
```

Ответ: 0, 1, 1, 0, 1,
2

Задача 2. Что получится в консоли?

```
public void Show()
{
    A a = new B();
}

public class A
{
    public A()
    {
        Console.WriteLine("Constructor A called");
        Console.WriteLine($"B value = {B.value}");
    }
}

public class B : A
{
    public B() => Console.WriteLine("Constructor B called");
    static B()
    {
        value = 0;
        Console.WriteLine("Static constructor B called");
    }
    public static int value = 1;
}
```

Ответ: Static constructor B called
Constructor A called
B value = 0
Constructor B called

Задача 3. Что получится в консоли?

```
class A
{
    public virtual void Foo() => Console.WriteLine("A");
}

class B : A
{
    public override void Foo() => Console.WriteLine("B");
}

class C : A
{
    public new void Foo() => Console.WriteLine("C");
}

class D : A
{
    public void Foo() => Console.WriteLine("D");
}

class E : B
{
    public new void Foo() => Console.WriteLine("E");
}
```

```
A a = new A();
a.Foo();
```

A

```
B b1 = new A();
b1.Foo();
```

Ошибка компиляции

```
A b2 = new B();
b2.Foo();
```

B

```
A c = new C();
c.Foo();
```

A

```
B b3 = new C();
b3.Foo();
```

Ошибка компиляции

```
D d = new D();
d.Foo();
```

D

```
E e = new E();
e.Foo();
```

E

И под занавес ещё одна задачка на полиморфизм

```
class A
{
    public virtual int Calc() => 10 * Gen();
    protected int Gen() => 10;
}

class B : A
{
    public override int Calc() => 20 * Gen();
    protected int Gen() => 20;
}

class C : B
{
    public override int Calc() => 30 * Gen();
    protected int Gen() => base.Gen();
}

static void Main(string[] args)
{
    A a = new B();
    A a1 = new C();
    Console.WriteLine(a.Calc() + a1.Calc());
}
```

Создание своих типов данных

```
public struct Coords
{
    public int x, y;

    public Coords(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

```
var coords = new Coords(10, 30);
Console.WriteLine($"x = {coords.x}, y = {coords.y}");
```


Перечислимый тип

```
enum Day { Sat, Sun, Mon, Tue, Wed, Thu, Fri };
```

// можно указывать наследование от типа и значения

```
enum Day : byte { Sat = 6, Sun = 7, Mon = 1, Tue, Wed };
```

```
enum Range : long { Max = 2147483648L, Min = 255L };
```

// можно преобразовывать к числовым типам, переводить в строку, парсить

```
int x = (int) Day.Sun;
```

```
Day day; Day.TryParse("Mon", out var day);
```

// можно указывать битовые флаги

```
[Flags] public enum Options { Sun = 0x01, Spoiler = 0x02, Fog = 0x04, }
```

// соединять их, проверять их вхождение, сравнивать

```
var options = Options.Sun | Options.Fog;
```

```
options.HasFlag(Options.Sun);
```

Рефлексия



Дайте понятие иерархии и перечислите виды иерархий. С каким фундаментальным свойством тесно связано это понятие?



В чем отличия между значимыми и ссылочными типами данных?



Перечислите фундаментальные понятия объектно-ориентированных методов

Следующий вебинар

Тема:



Интерфейсы и их особенности



Ссылка на вебинар будет в ЛК за 15 минут



Материалы к занятию
в ЛК — можно изучать



Обязательный
материал обозначен
красной лентой

The background of the image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white geometric network pattern of interconnected dots and lines. The text is centered within this blue layer.

Заполните, пожалуйста,
опрос о занятии по ссылке в чате



До новых встреч! Приходите на следующие занятия

Константин Рочев

Главный программист Insense Arts LLC,
доцент каф. ВТИСиТ УГТУ

E-mail: k@rochev.ru

Телефон: 9041098318