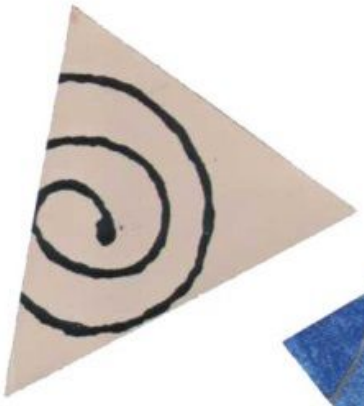
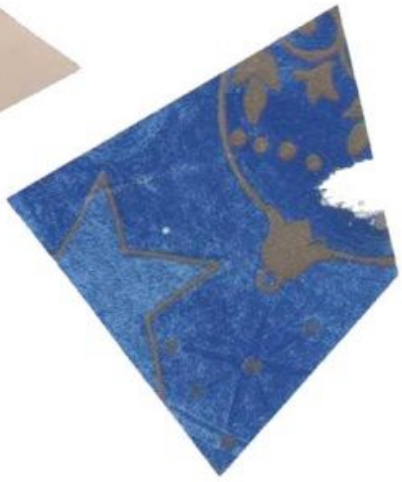




ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ.


Филимонова Д.В.
40-102С.


Москва, 2013 г.

Данное пособие не дает возможности читателю абсолютно овладеть языком программирования. Задача этой книги - дать общее ассоциативное представление об отношениях основных объектов языка.

Поэтому данное издание рекомендуется либо ещё не приступавшему к изучению программирования ученику, либо уже опытному и владеющему основными навыками студенту для обобщения и систематизации изученного им материала.

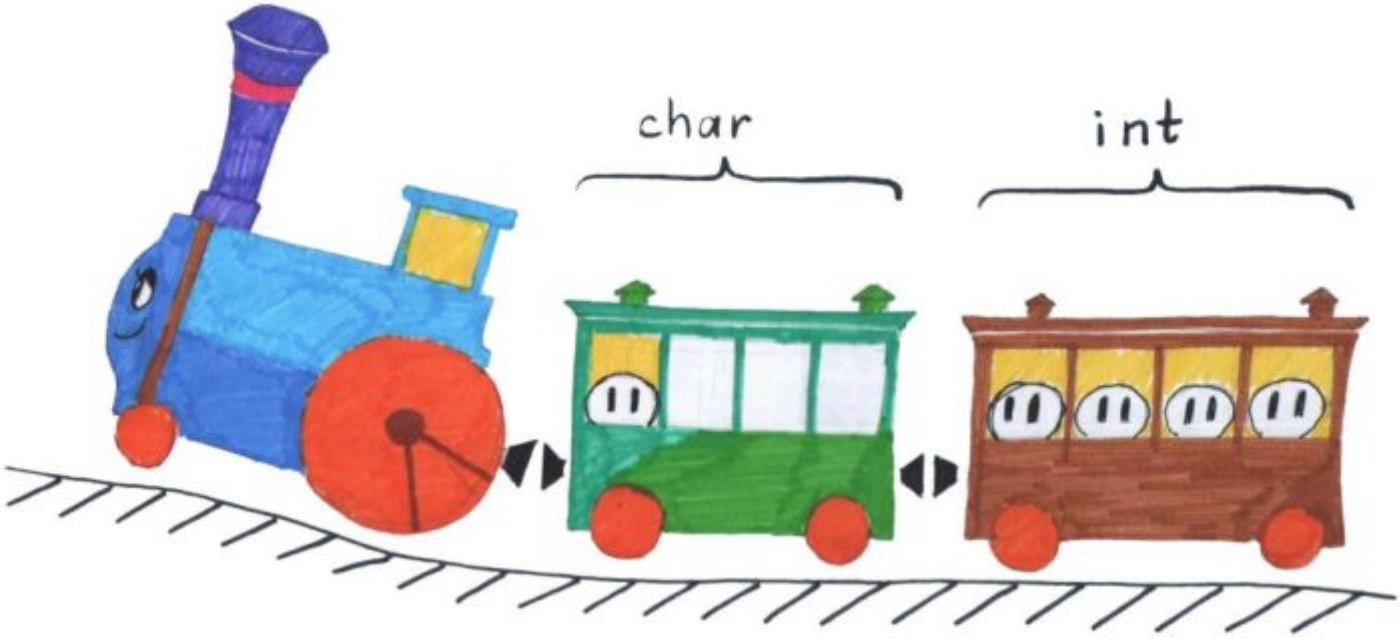
Типы данных в C/C++.

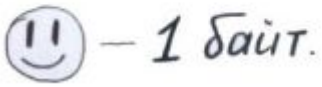
Память состоит из пронумерованных ячеек.

Ячейки в памяти располагаются последовательно.

Типы данных определяют:

- количество байт памяти, отводимое на переменную (поле) в программе;





- формат представления данных в памяти;

- возможные операции для выполнения над данными.

Типы данных

базовые производные

1. Базовые типы данных.

Определены стандартом языка Си.

Способ хранения и записи значения в памяти компьютера стал называться "типом" этого данного или этой величины. Существует пять ***базовых типов данных***::

char	символьный тип величины
int	целочисленный тип данных
float	данное с плавающей точкой
double	данное с плавающей точкой двойной длины
void	пустое данное (без значения)

Существует возможность создавать собственные типы данных (производные от стандартных).

2. Производные типы данных.

Определяются программистом.

Были введены с целью облегчения разработки программ. Строятся на основе базовых типов, которые являются как бы кирпичиками для их построения.

1) Переименование типа (**typedef**).

Одному из существующих типов данных присвоить произвольное имя.

typedef имя_старого_типа имя_нового_типа;

typedef unsigned int ВАСЯ;

2) Перечисление (**enum**).

Это тип, значения которого ограничены конечным набором констант.

enum [имя] {список_констант_через_запятую};

enum Counter{one=1;two=2;three=3;};

3) Структура (**struct**).

Это объединенное в единое целое множество поименованных элементов (компонентов) данных.

struct [имя]{

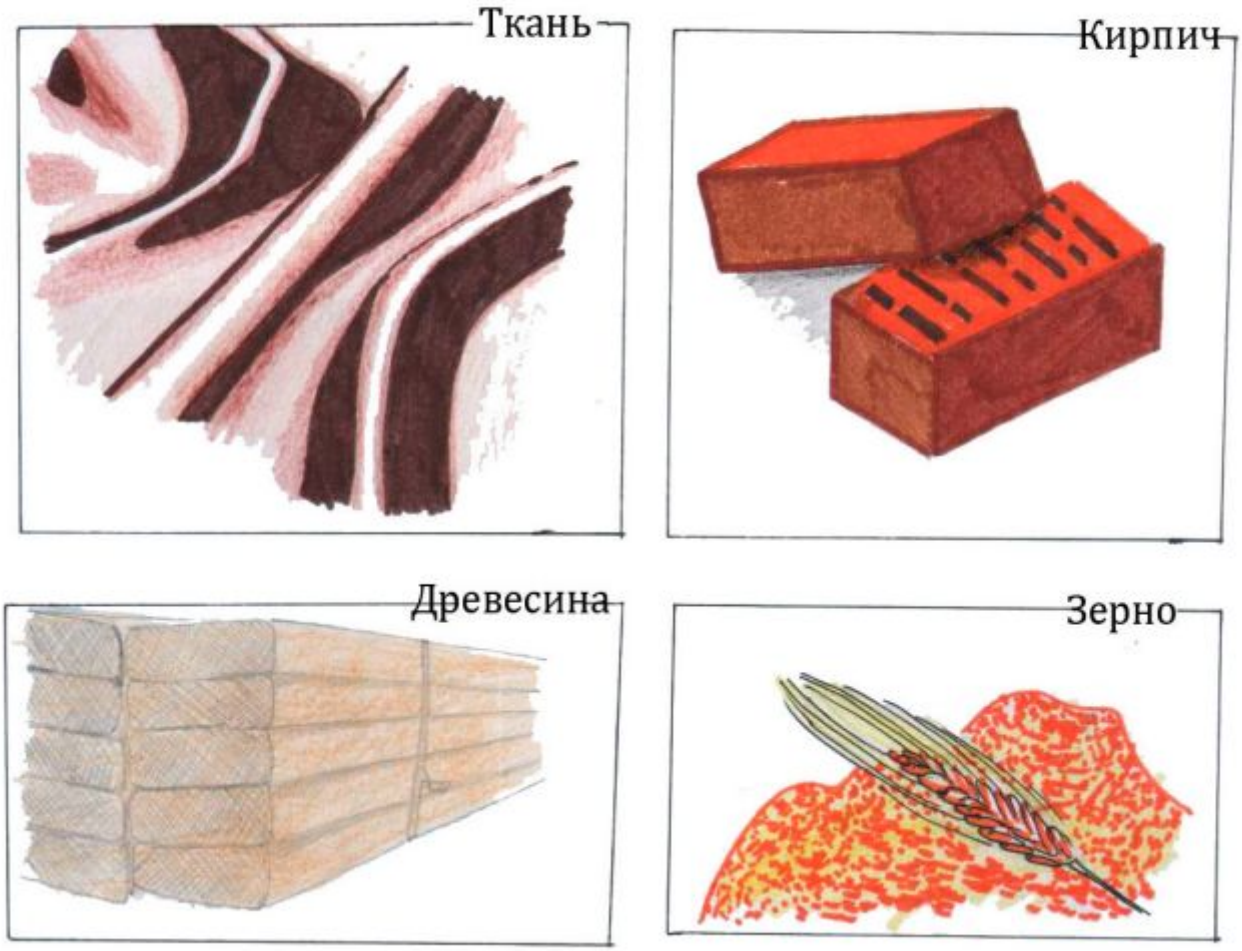
  тип_1 имя_поля_1;

  тип_2 имя_поля_2;

};

Изобразим её в виде набора материалов, используемых для создания чего-либо (переменных, над которыми нужно производить действия).

Поля <Материалы>




```
struct ОрудияТруда{
   char ткань;
   int кирпич;
   float древесина;
   float зерно;
};
```

4) Объединение (**union**).

Это частный случай структуры: все её поля хранятся по одному и тому же адресу. Может одновременно хранить значение только *одного* поля.

union [имя]{

  тип_1 имя_поля_1;

  тип_2 имя_поля_2;

};

Проиллюстрируем это как выбор только одного из возможных вариантов.

5) Битовые поля.

Это поля структуры или объединения, заданные в битах.

int имя_поля: число_бит;

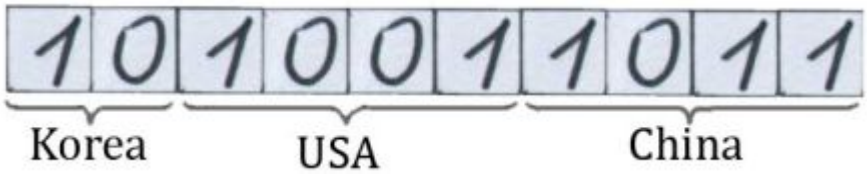
Таблица. Медали, полученные различными странами на олимпиаде в Лендоне-2012.

Rank	Country	Gold	Silver	Bronze	Total
1	People's Republic of China	18	11	5	34
2	United States of America	18	9	10	37
3	Republic of Korea	7	2	5	14

Чтобы данные по медалям для каждой страны занимали как можно меньше места, каждое поле для этих стран определим в битах с длинной, необходимой для хранения заработанных медалей.

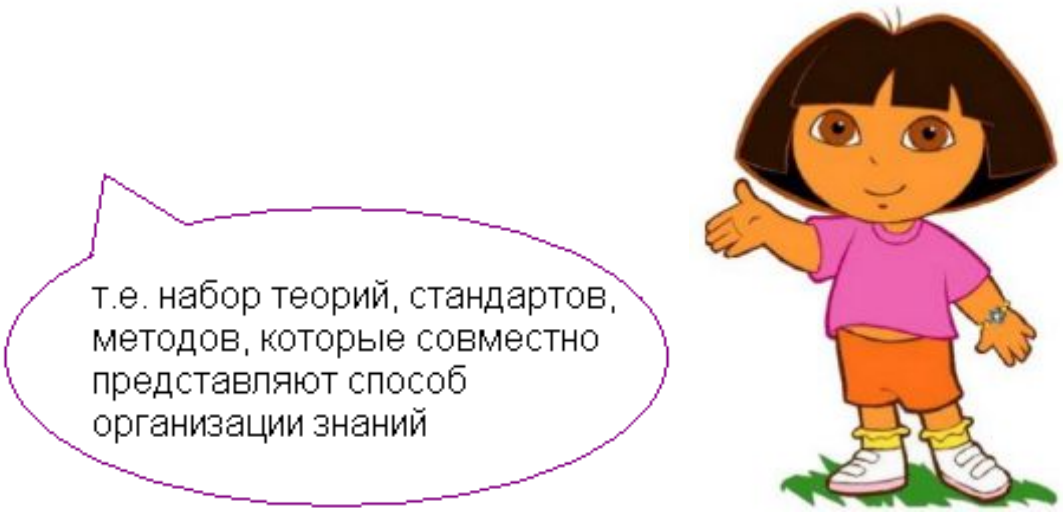
```
struct{
   int China:4;
   int USA:4;
   int Korea:2;
}Silver={11,9,2};
```

Расположение информации в памяти:

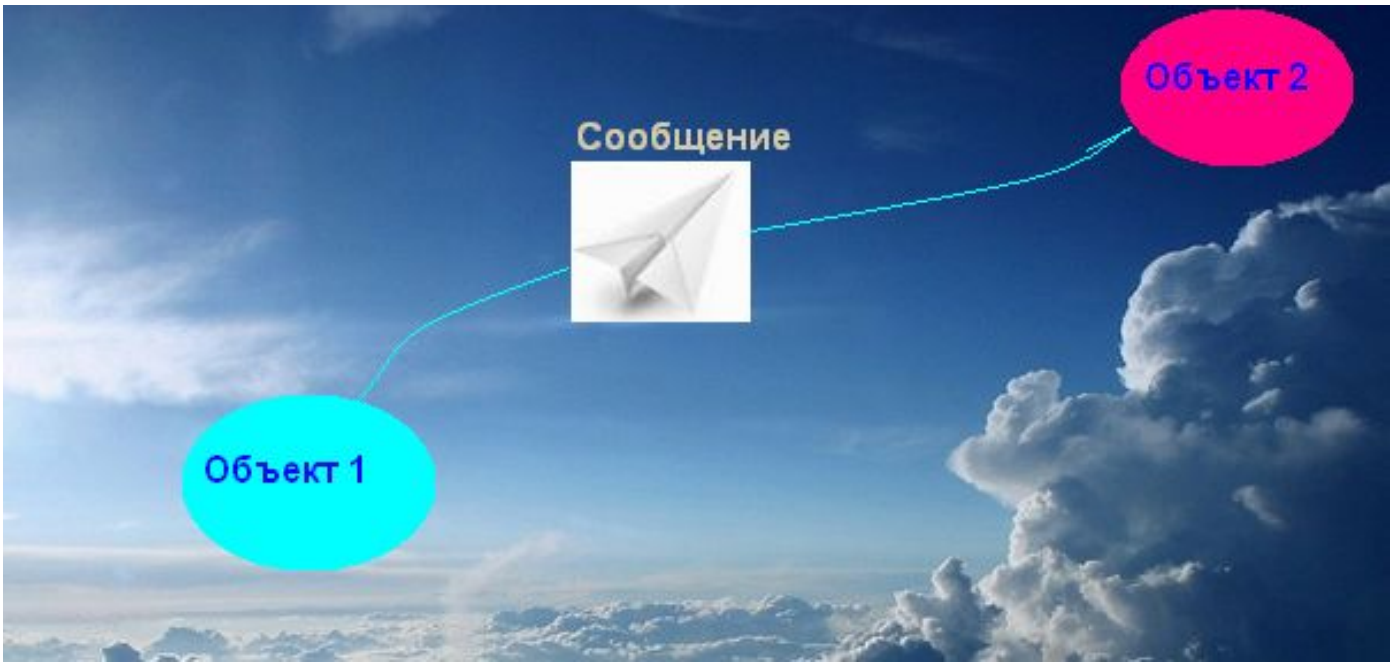


ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ.

Это ***парадигма***



программирования, строящаяся на терминах *объектов* и их взаимосвязей.



По мнению Алана Кея, создателя языка Smalltalk, которого считают одним из «отцов-основателей» ООП, объектно-ориентированный подход заключается в следующем наборе основных принципов (цитируется по книге Т. Бадда).

1. *Всё* является объектом.
2. Вычисления осуществляются путём взаимодействия (обмена данными) между объектами, при котором один объект *требует*, чтобы другой объект *выполнил* некоторое действие. Объекты взаимодействуют, посылая и получая сообщения. Сообщение — это *запрос* на выполнение действия, дополненный набором *аргументов*, которые могут понадобиться при выполнении действия.
3. Каждый объект имеет независимую память, которая состоит из других объектов.
4. Каждый объект является представителем класса, который выражает общие свойства объектов.
5. В классе задаётся поведение (функциональность) объекта.
6. Классы организованы в единую древовидную структуру с общим корнем, называемую *иерархией наследования*.

6) Класс (**class**).

class [имя_класса]{

<поля>;

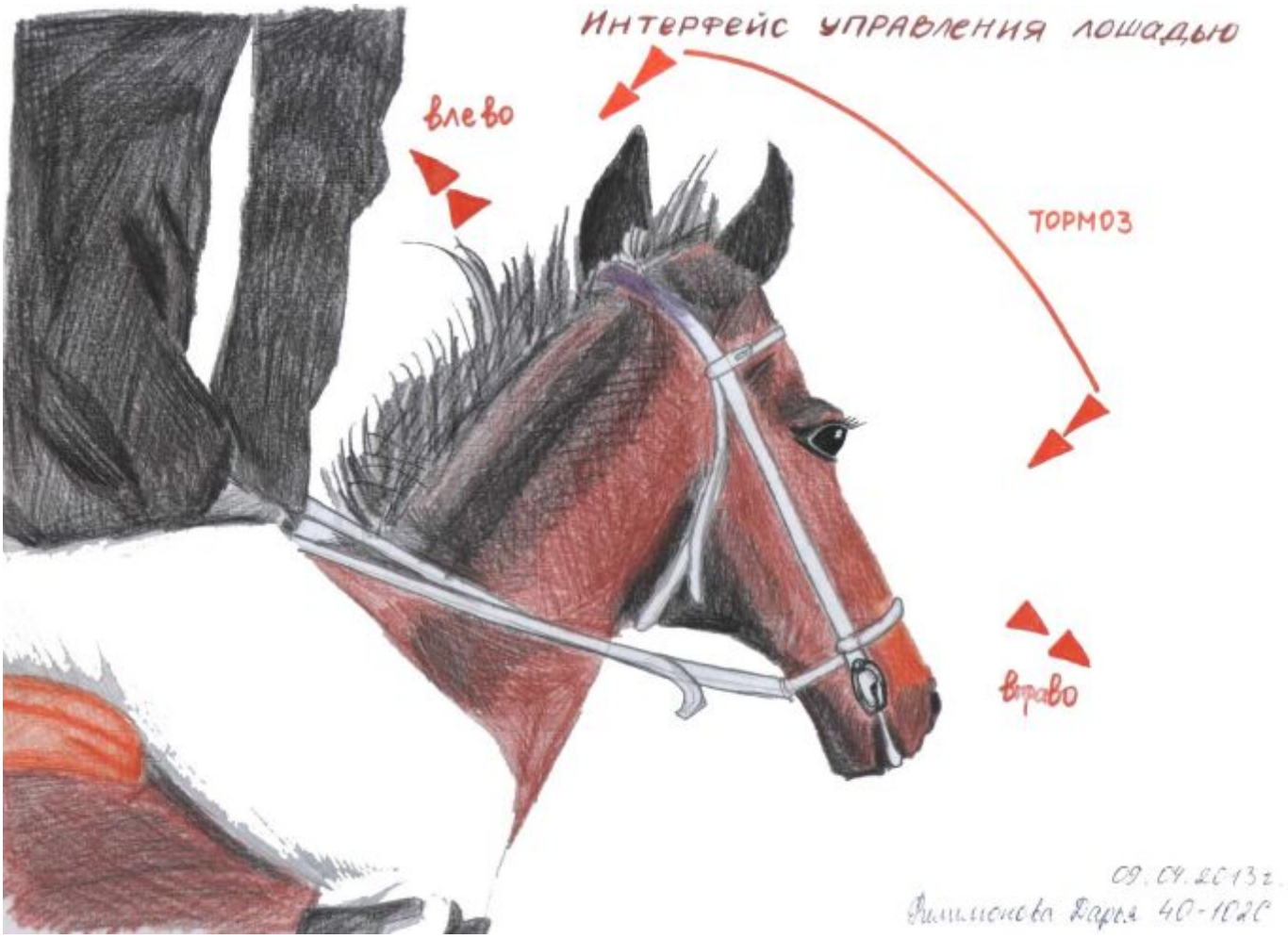
<методы>;

};

В отличие от структуры класс содержит не только поля, над которыми можно производить действия, но и методы – набор самих действий.

Изобразим методы в виде орудий труда, с помощью которых можно производить действия над материалами, рассматриваемыми в разделе «Структура».

Те методы, которые объявлены в области **public**, образуют *интерфейс* класса.



Пусть **class** Лошадь скрывает в себе различные «методы» кровообращения, дыхания, пищеварения… Объект *Ганька* является объектом этого класса. Пользователю, как наезднику, необходимы только методы управления объектом класса для передвижения, т.е. такие как Влево, Вправо, Тормоз и т. д…

Диаграмма UML (Unified Modeling Language – унифицированный язык моделирования) полей и методов классов.

Имя класса	Производство
поля	-ткань -кирпич +зерно -древесина
методы	-резать #пилить +косить

Пример класса.

```
class Производство{
  public:
    float зерно;
    void косить(){…};
  protected:
    void пилить(){…};
  private:
```

```
    char ткань;
    int кирпич;
    float древесина;
    void резать(){…};
};
```

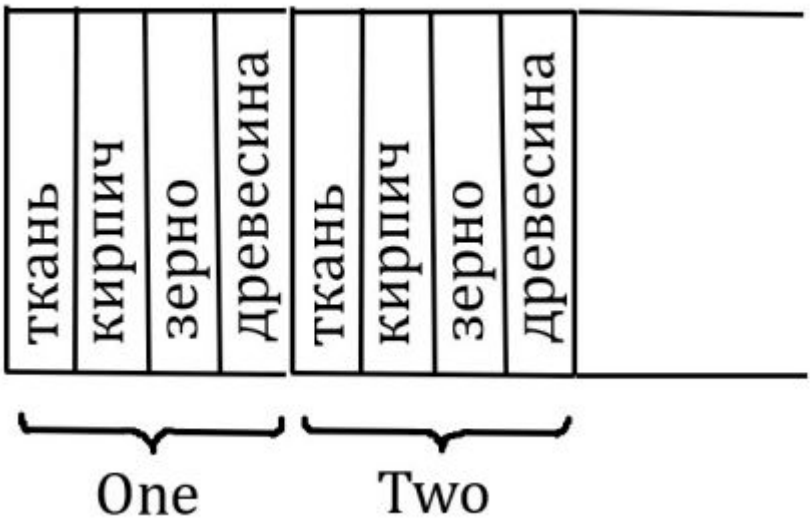
Создание объектов класса.

Объект класса – это реальная именованная сущность, они создаются «по образцу», такому, что описан в классе.

Производство One, Two;

При создании объекта класса под все поля для каждого объекта индивидуально выделяется память. Память под методы для всех объектов выделяется в единственном

числе.

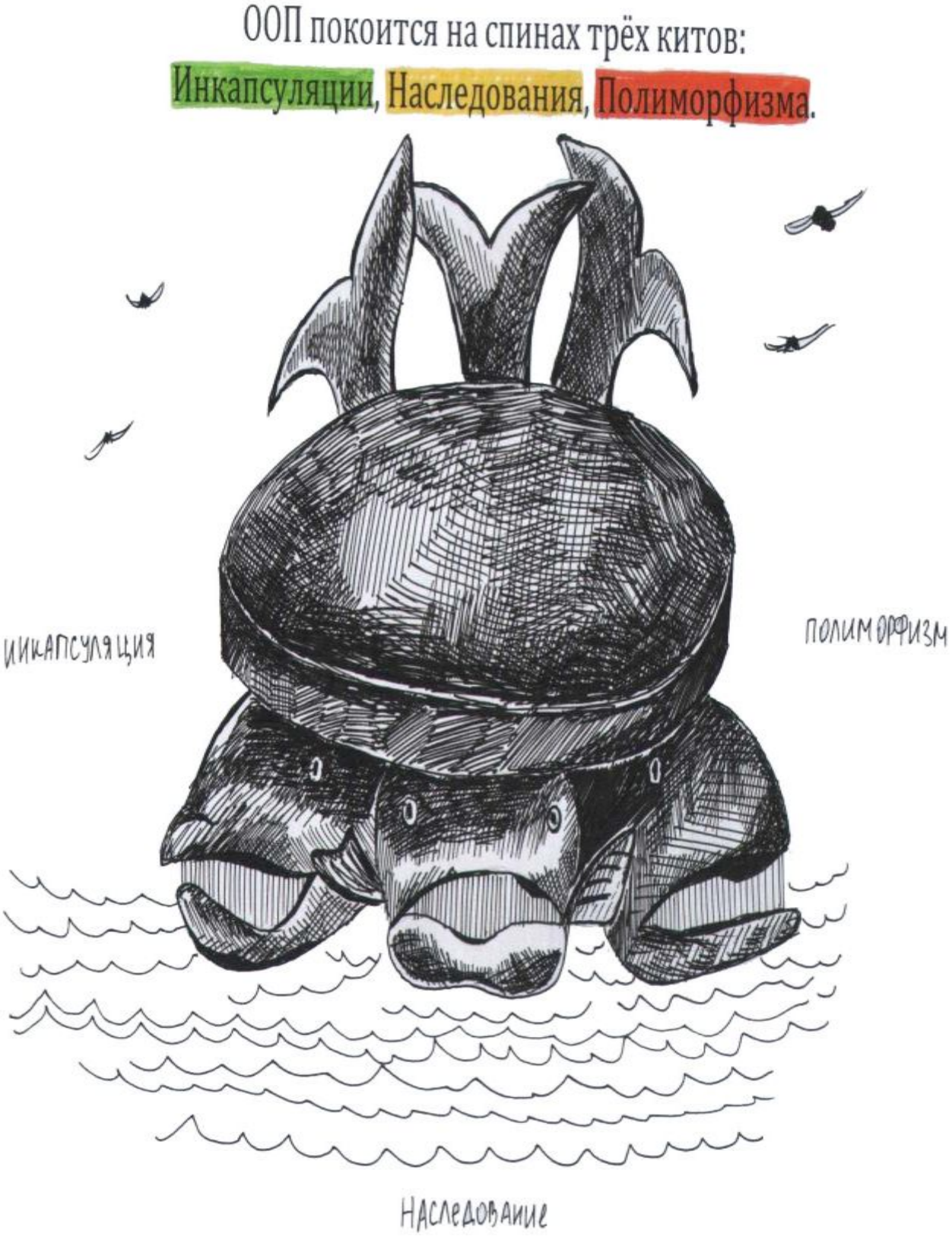


Класс, моделирующий постройку дома и обустройство сада.

class HomeSweetHome

```
{
  int height;
  int brick, tree;
 public:
  HomeSweetHome(int he=0, int br=1000, int tr=9)
  {
    height=he;brick=br;tree-tr;
  }
  void put_tree(int tree)
  {
    //посадить дерево – заготовленных деревьев
    // остаётся меньше на 1.
    tree--;
  }
  void put_brick(int brick,int height)
  {
    //положить кирпич – в резерве кирпичей остаётся
    //меньше на 1.
    brick-=5;
  }
  //узнать высоту дома можно так:
```

```
    void get_height(){return height;}
};
```



I. Инкапсуляция –

это сокрытие данных и методов класса от остальной части программы.

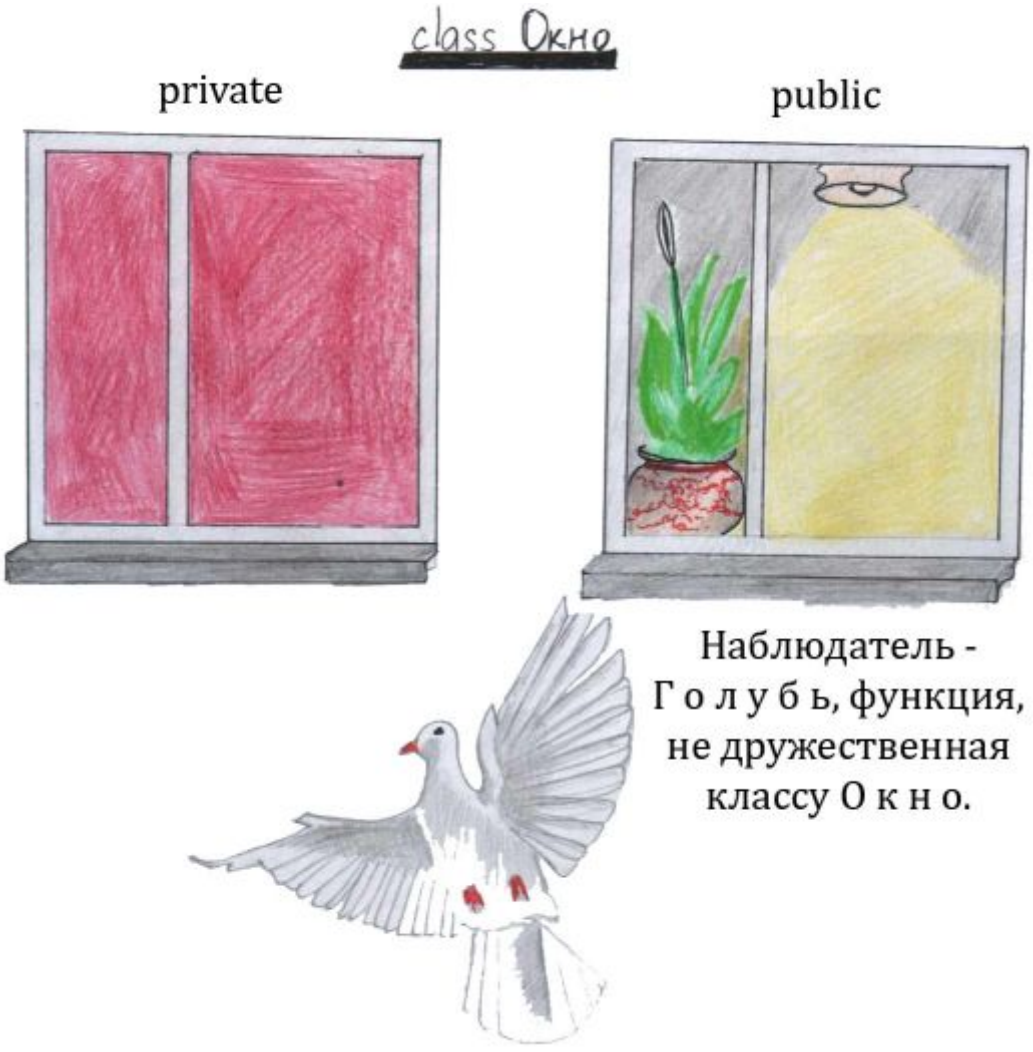
Для её реализации используют спецификаторы доступа к элементам класса (**private**, **public**, **protected**).

Видимость при объявлении в области:

- **private** – внутренним методам, функциям-друзьям класса,

\# **protected** – тем же + методам наследников данного класса,

\+ **public** – всем методам в любой точке кода.



class Окно{

//если область видимости не указана, то по

//умолчанию она является **private**

```
    int flower;    //некоторое количество
                   //цветков
    bool light;    //логический тип – свет либо
                   //горит, либо нет
};
void Голубь(){…};
```

В этом случае функция не имеет доступа к классу Окно.

А иногда технологии программирования оказываются примененными в обычной жизни!

Отношения между классами.

Отношение включения.

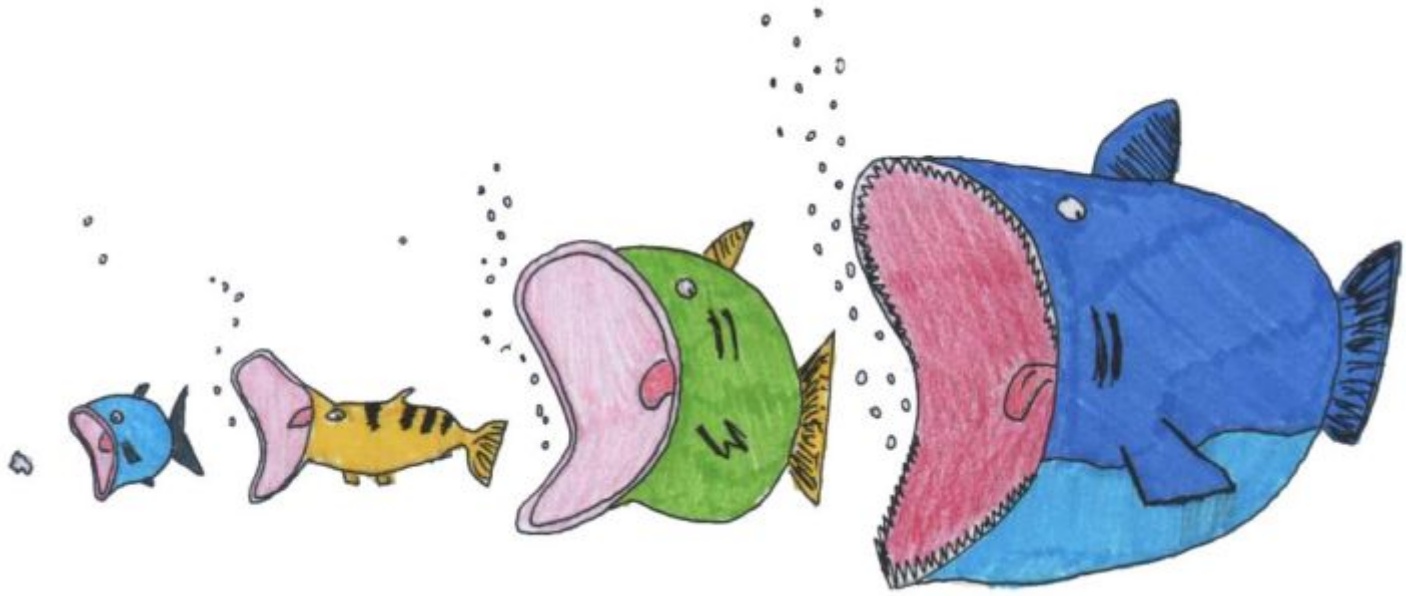
В этом случае объект одного класса состоит из объектов другого класса.

Бывает:

композиционное – объект является неотъемлемой частью целого, принадлежит только ему;

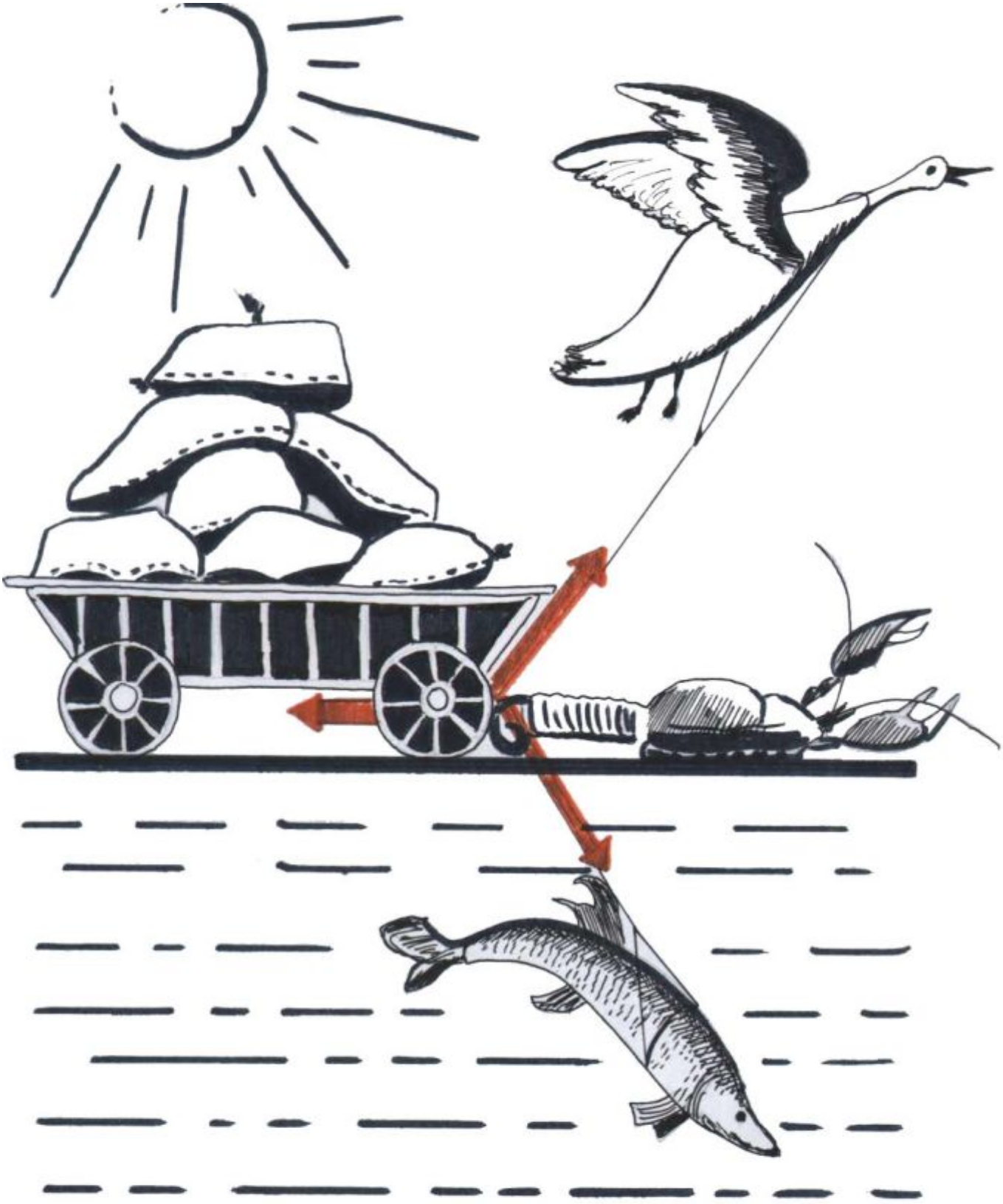
множественное – один и тот же объект может входить в различные объекты.

Композиционное включение в виде пищевой цепи экосистемы Океан…



… и в виде матрешек:

При множественном включении объект может одновременно входить в состав различных объектов. Поэтому возможна ситуация, описанная в басне И.В. Крылова «Лебедь, Щука и Рак».



Объект класса Т е л е г а одновременно входит в классы Л е б е д ь, Р а к и Щ у к а.

```
class Лебедь
{
  public:
    …
};
```

```
class Рак
{
  public:
    …
};
```

```
class Щука
{
  public:
    …
};
```

```
class Телега
{
  public:
    Лебедь L;
    Рак R;
    Щука S;
};
```

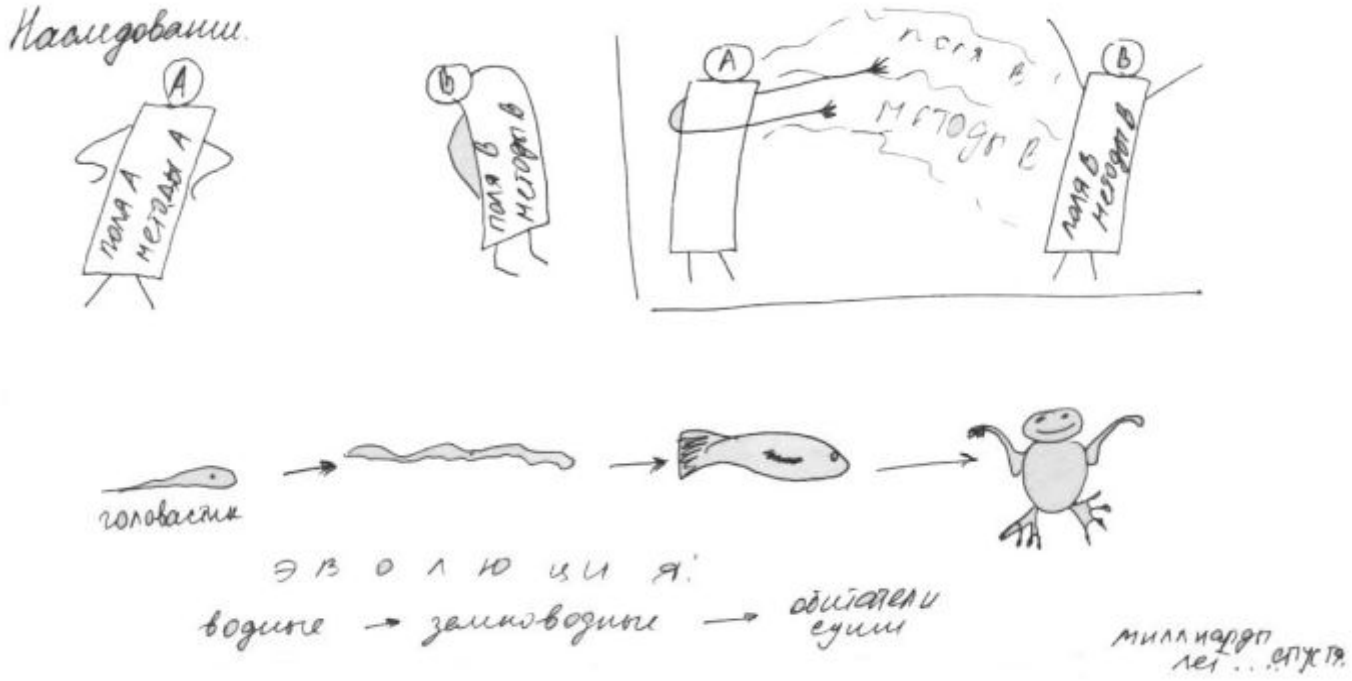
II. Наследование –

это создание нового (производного) класса на основе существующего (базового). Бывает:

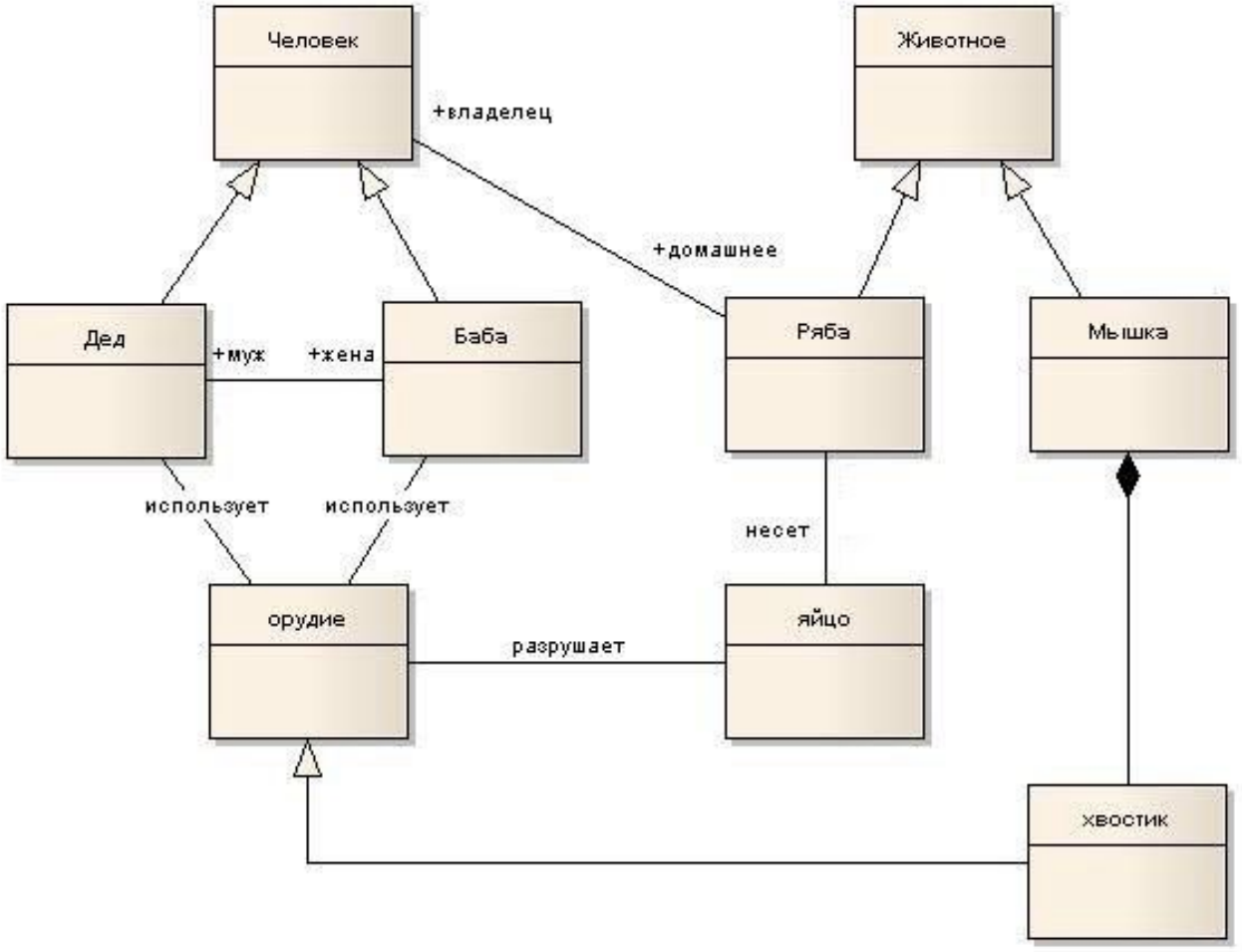
одиночное – у производного класса один базовый класс,

множественное – у производного класса несколько базовых.

При наследовании производный класс от базового получает в наследство его поля и методы.

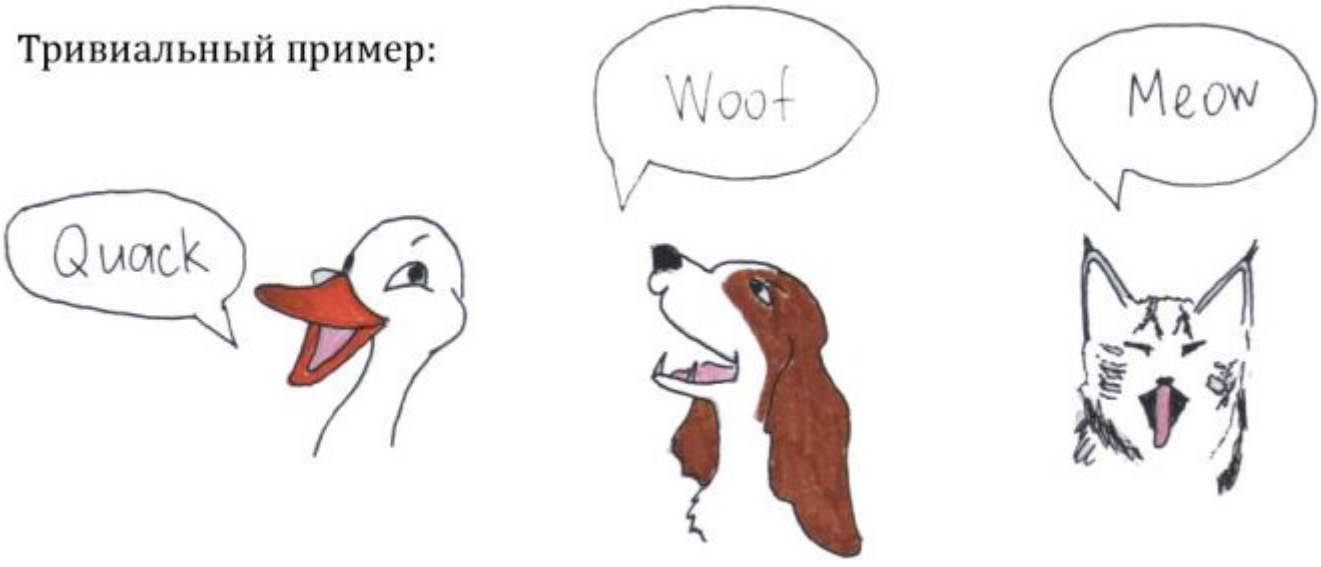


Пример иллюстрации сюжета сказки «Курочка Ряба» с помощью диаграммы UML.



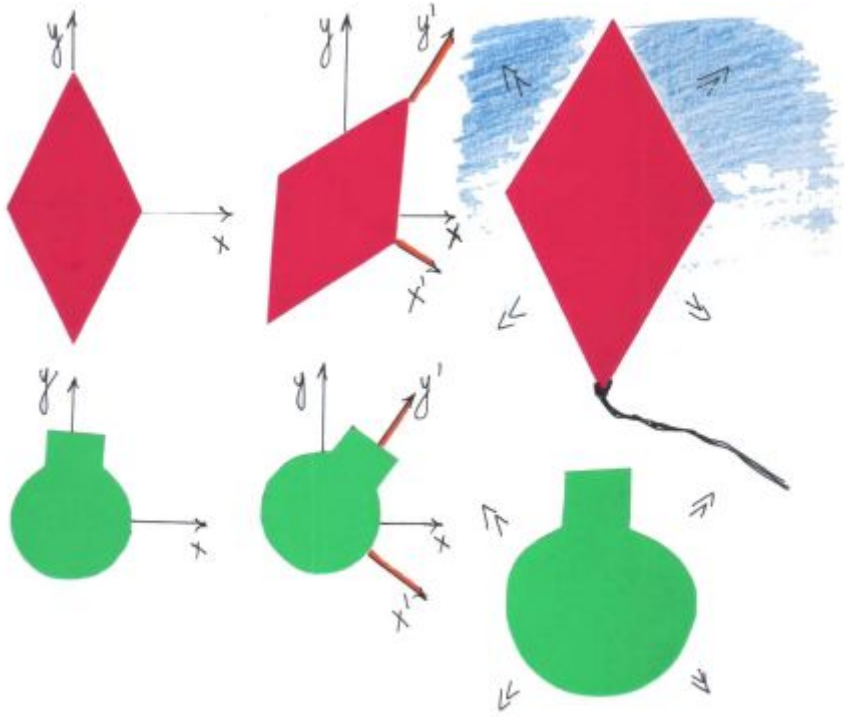
III. Полиморфизм.

Кратко смысл можно выразить фразой: «Один интерфейс, множество реализаций». Объекты имеют эквивалентный набор методов одинаковой функциональности.



P. S. Ещё пример. Класс потоков имеет методы для последовательной передачи данных. Потоком может быть информация, вводимая пользователем с терминала, обмен данными по компьютерной сети, файл.

P. P. S. Классический пример. Геометрические фигуры.



В ООП соблюдается концепция абстракции данных.

Абстракция данных

Фундаментальная идея состоит в разделении:

- несущественных деталей реализации подпрогр.

и

- характеристик, существенных для её использования.

Пример: сферическая корова в вакууме.

Spherical cow

Свойства:

- имеет шарообразную форму,
- дышит идеальным газом;
- пасётся на однородных полях;
- движется равномерно по прямой.
- выполняет "работу" абсолютно черного тела.

Существенные характеристики: ЭТО КОРОВА!