Лекция 9

Объектно-ориентированное программирование – основные принципы.

Концепция ООП

Есть несколько концепций программирования, которые в той или иной степени реализуются в разных языках программирования.

Среди них особым образом выделяется **концепция объектно-ориентированного программирования** (ООП), которая поддерживается всеми (или почти всеми) современными языками программирования.

Концепция ООП – это реакция на значительное усложнение программ и увеличение объема кода. ООП преимущественно используют при написании больших и сложных программ.

(P.S. Программа считается большой, если она содержит несколько тысяч строк кода - это очень условная оценка).

Объектно-ориентированное программирование (в дальнейшем ООП) – ***парадигма программирования***, в которой основными концепциями являются понятия ***объектов и классов***.

Класс и объект

Класс – абстрактный тип данных, определяемый пользователем, и представляет собой модель реального объекта в виде данных и функций для работы с ними.

Данные класса называются **полями** (по аналогии с полями структуры), а функции класса – **методами**. Поля и методы называются **элементами** класса.

Объект – это сущность, конкретный экземпляр класса, которой можно посылать сообщения и которая может на них реагировать, используя свои данные.

Например:

Как вы представляете «цветок»?

Как можно его охарактеризовать:

у него есть лепестки, стебель, листья,

он растет, цветет, дает плоды, вянет.

Класс и объект

Но цветы бывают разные :

Класс и объект

Класс – это «цветок».

Его свойства (поля): количество лепестков, длина стебля, вид листьев; а функции (методы) – это действия «расти», «цвести», «плодоносить», «вянуть».

Объект – это конкретный «цветок»:

- ромашка;
- орхидея;
- фиалка;
- раффлезия;
- мухоловка.

```
class цветок{
свойства:
    количество лепестков, длина стебля, вид листьев
методы:
     расти, цвести, плодоносить, вянуть
};
цветок ромашка, орхидея, фиалка, мухоловка;
```

Класс и объект

Пример описание класса студент:

```
class student{                  // поля
    char name[30];              // имя
    int number;                 // номер зачетки
    int year_birth, year_start; // год рождения и поступления
    int mark_m, mark_inf, mark_rus;// оценки при поступлении
public:      // спецификатор доступа
             // методы
      void inp_data(); // заполнение полей
      void out_data(); // просмотр
       float average_mark()  {// вычисление среднего балла
              return (mark_m + mark_inf + mark_rus)/3;}
             // вычисление возраста поступившего
      int age() {return (year_birth - year_start);}
};
void main(){
student Petrov, Sidorov, ob1, ob2;
         Petrov.age(); ob1.age(); /* … */}
```

Примеры описания классов и объектов

Рассмотрим класс «автомобиль».

Примеры описания классов и объектов

Класс «автомобиль».

Поля: (свойства) масса, количество дверей, размеры, название

Методы: (действия) вычисление каких-то характеристик

Объекты: (конкретные экземпляры)

Феррари, Ламборджини, Ягуар, BMW и т.д.

Примеры описания классов и объектов

Рассмотрим класс «планета».

Примеры описания классов и объектов

Класс «планета».

Поля: размер орбиты, диаметр планеты, масса и т.д.

Методы: вычисление положения относительно чего-нибудь

Объекты: Марс, Венера, Земля, Сатурн

Основные принципы ООП

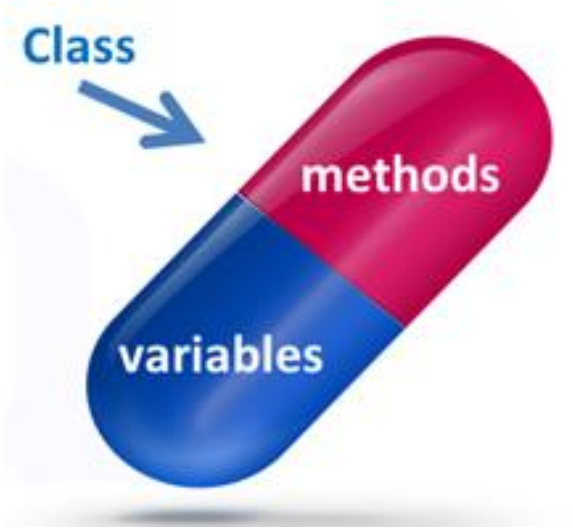
Концепция ООП базируется на трех фундаментальных принципах:

- ***инкапсуляции,***
- ***полиморфизме,***
- ***наследовании.***

Основные принципы ООП: инкапсуляция

Инкапсуляция – это механизм контроля доступа.

Инкапсуляция позволяет скрыть детали реализации, и открыть только то, что необходимо для последующего использования.




```
class student{
private:
    char name[30];
    int number;
    int year_birth, year_start;
    int mark_m, mark_inf, mark_rus;
 public:
      void inp_data();
      void out_data();
       float average_mark() {
            return (mark_m + mark_inf + mark_rus)/3;}
      int age() {return (year_birth - year_start);}
};
```

Основные принципы ООП: инкапсуляция

Существует 3 вида модификаторов доступа:

Основные принципы ООП: инкапсуляция

`public` – уровень предполагает доступ к компоненту с этим модификатором из экземпляра любого класса.

Например:

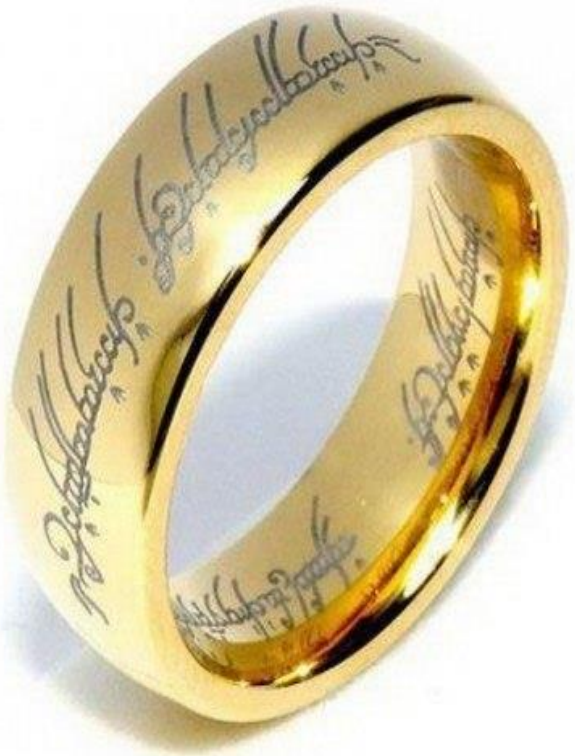
Основные принципы ООП: инкапсуляция

`private` – уровень предполагает доступ к компоненту с этим модификатором только из этого класса.

Например:

Основные принципы ООП: инкапсуляция

`protected` – уровень предполагает доступ к компоненту с этим модификатором из экземпляров родного класса и классов-потомков.
Например:



Основные принципы ООП: инкапсуляция

Пример:

```
class lift {
 private:
      float  v; //Скорость
      float  m; //Грузоподъемность
      int n; //Кол-во людей
      float h, w, d; // Размеры кабины
 public:
       method1(); // Управление алгоритм1
       method2(); // Управление алгоритм2
};
```

При создании объектов класса **lift**:

```
main(){
  lift ob1, ob2;
      cout<< ob1.v; //нет доступа
      cout<< ob2.n; //нет доступа
   cout<< ob1.method1();
   cout<< ob2.method2();
}
```

Основные принципы ООП: наследование

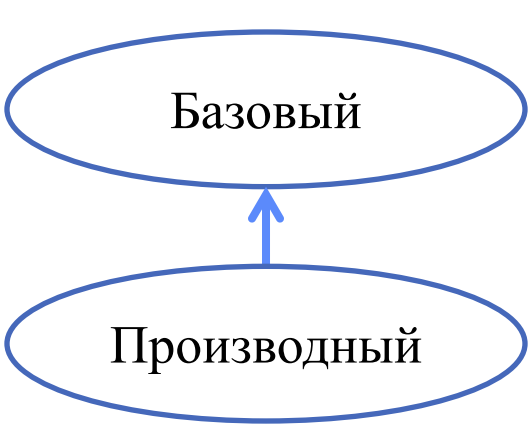
Наследование позволяет создавать новые классы на основе уже существующих классов, что значительно экономит усилия, сокращает объем кода и повышает его надежность.

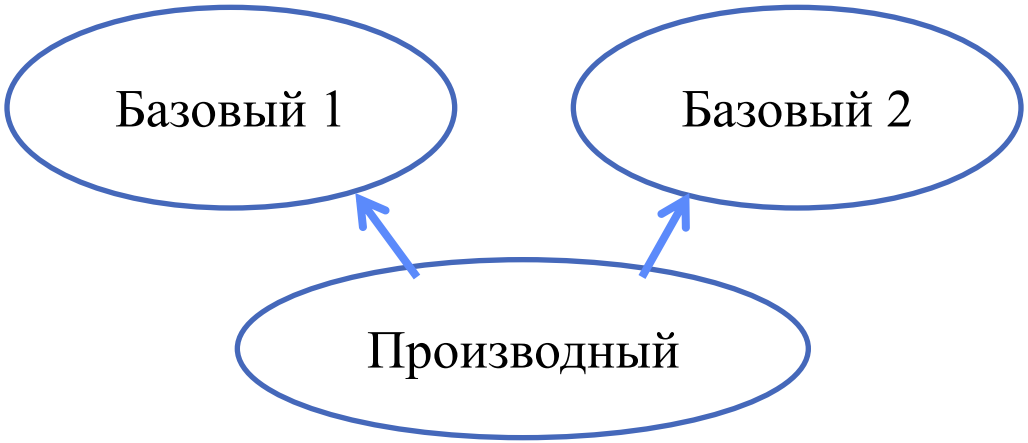
Виды наследования:

- ***простое наследование;***
- ***множественное наследование*.**

При простом наследовании один класс наследует свойства одного класса.

При множественном наследовании один класс наследует свойства от нескольких классов.





Основные принципы ООП: наследование

Пример простого наследования: «семечко» ← «растение»

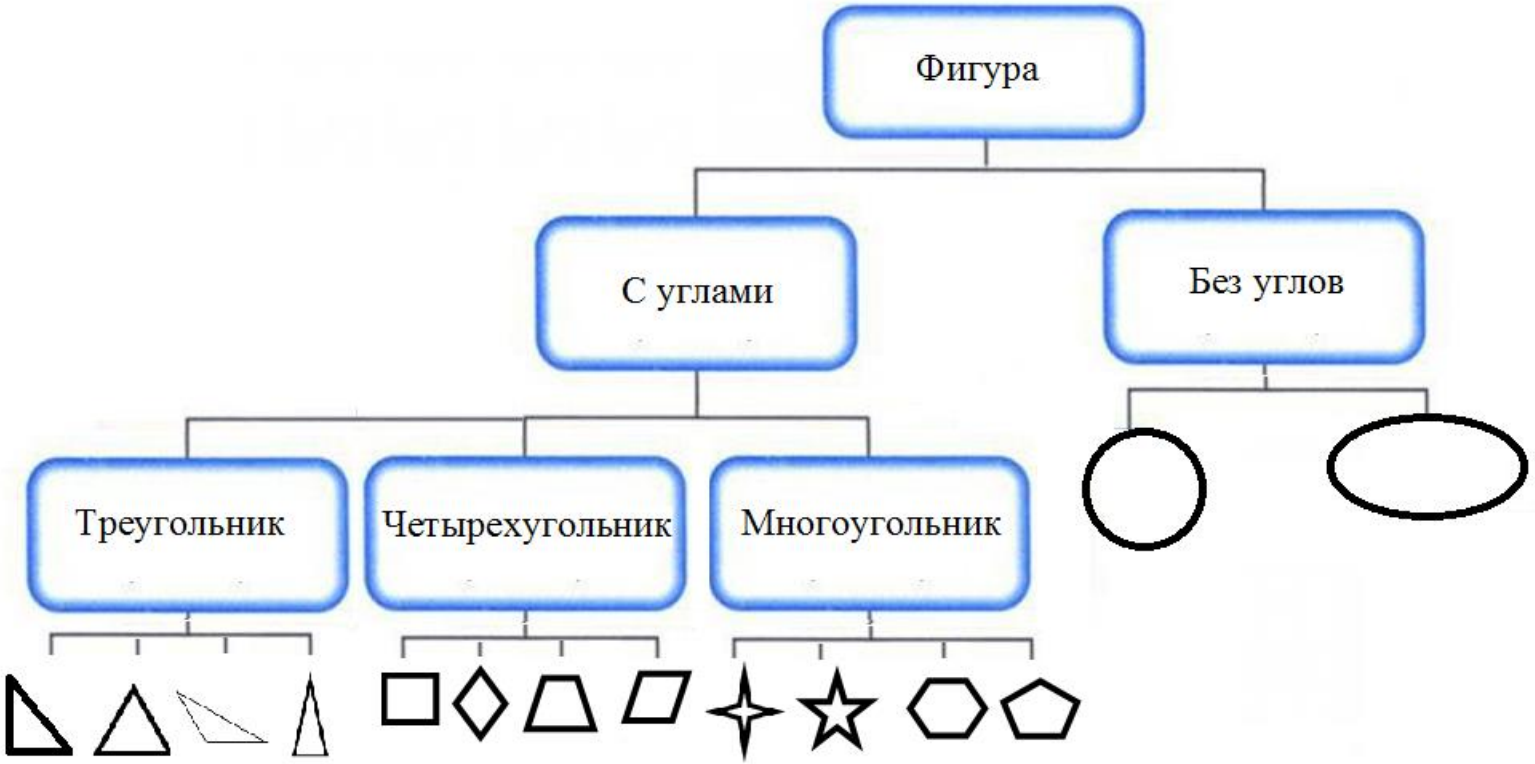
Основные принципы ООП: наследование

Примеры множественного наследования:

Основные принципы ООП: наследование

Применение **иерархии** классов делает управляемыми большие потоки информации.

Например: пусть имеются геометрические фигуры. Для каждой фигуры необходимо описать её свойства и, например, метод перемещения, метод прорисовки на экране. Т.е. сколько фигур – столько и методов. А если фигуры объединить в иерархию по их свойствам, то метод перемещения станет общим для всех – его не нужно переопределять для каждого.



Основные принципы ООП: наследование

Иерархия:

Основные принципы ООП: полиморфизм

Полиморфизм базируется на использовании универсальных интерфейсов для решения однотипных задач.

Полиморфизм: «поли» значит «много», а «морфизм» - «изменение» или «вариативность», таким образом, «полиморфизм» - это свойство одних и тех же объектов и методов принимать разные формы.

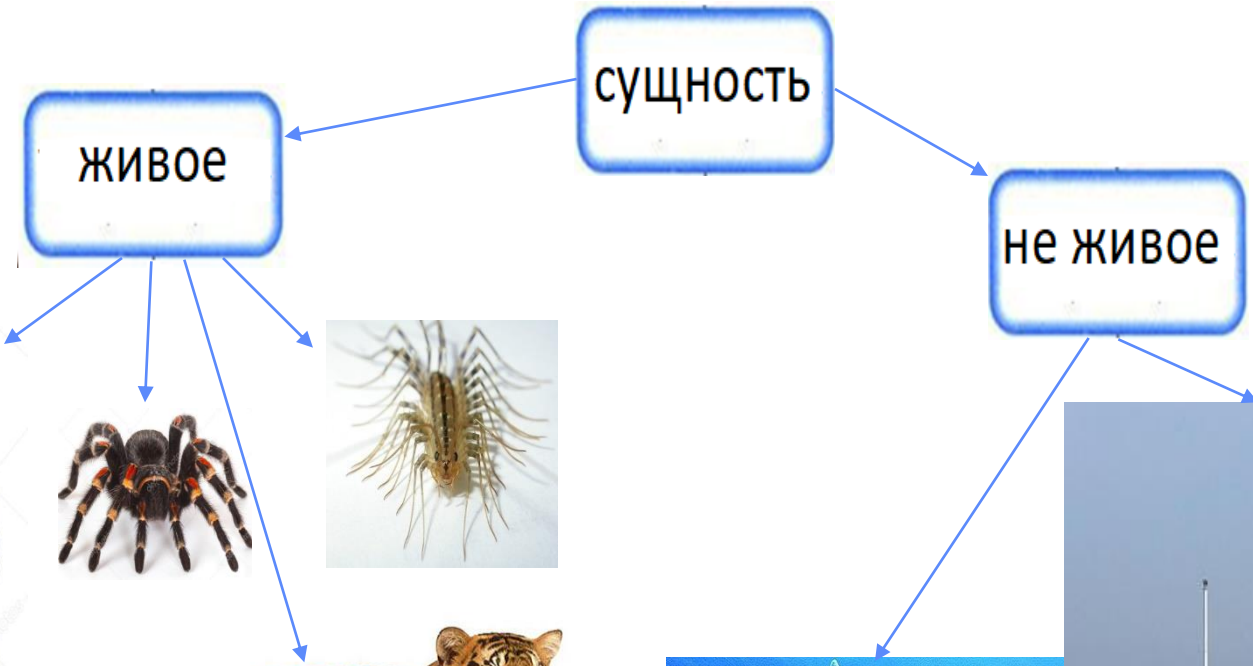
В более общем смысле, концепцией полиморфизма является идея "*один интерфейс, множество методов*". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий. Выбор же конкретного действия, в зависимости от ситуации, возлагается на компилятор.

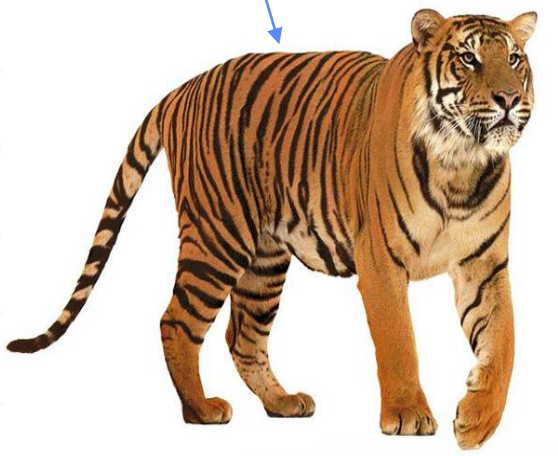
Вам, как программисту, не нужно делать этот выбор самому. Нужно только помнить и использовать общий интерфейс.

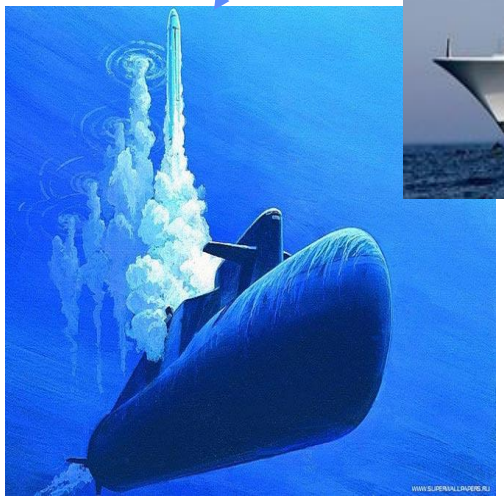
Интерфейс — определяет способ взаимодействия с объектом.

Основные принципы ООП: полиморфизм

Рассмотрим на примере иерархии сущностей - метод «ходить».







```
сущность *ob;
ob= new человек;
ob->ходить();
delete ob;

ob= new тигр;
ob->ходить();
delete ob;

ob= new лайнер;
ob->ходить();
delete ob;
```

Основные принципы ООП

Эти три принципа в той или иной степени присущи любому языку программирования, поддерживающему парадигму ООП. Наиболее популярными объектно-ориентированными языками программирования сегодня принято считать, наряду с C++, языки C# и Java. Исторически первым появился язык C++, ставший существенно усовершенствованной версией C.

Усовершенствования касались главным образом поддержки парадигмы ООП. Именно C++ стал в известном смысле родительским для языков C# и Java. В этом несложно убедиться, если сравнить синтаксисы языков – они очень схожи. Но если в языке C++ можно создавать программы как с использованием классов, так и без них, то в языках Java и C# без ООП уже не обойтись.

Спасибо за внимание!