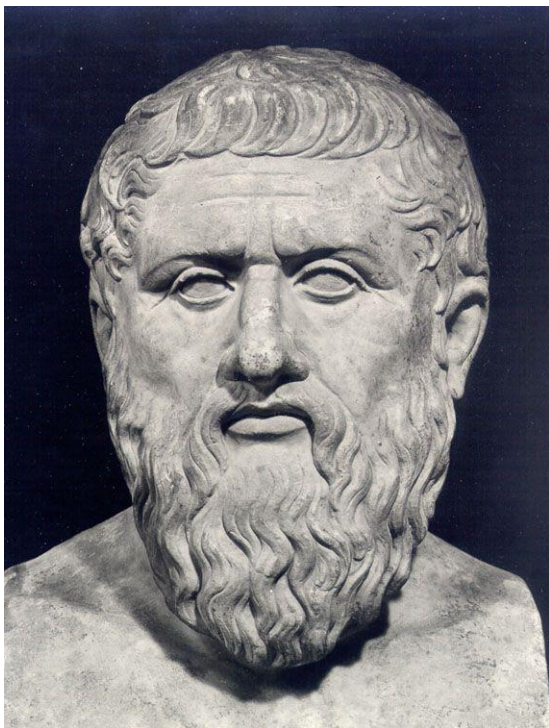


06. 00

*Лекции по информатике для студентов
первого курса Высшей школы ИТИС
2019 год*

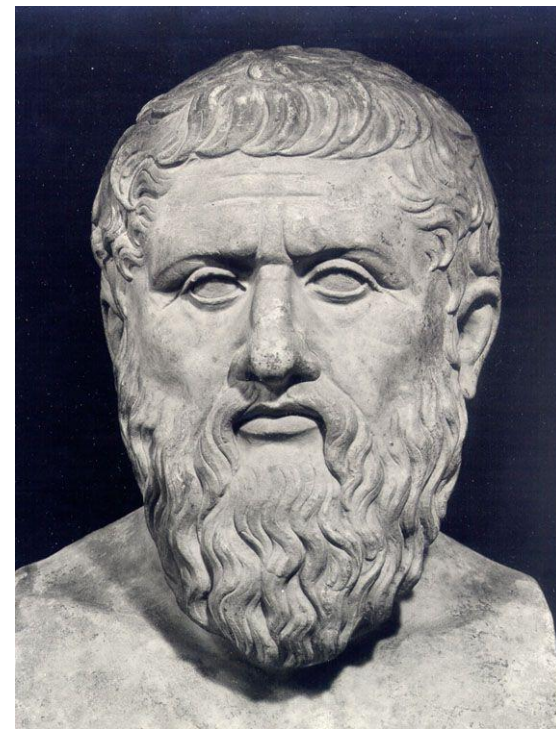
МИХАИЛ АБРАМСКИЙ
старший преподаватель
Высшая школа ИТИС КФУ



Кто это?

Платон

Во-первых, есть [...] **идея**, [...] **незримая** и никак иначе **не ощущаемая**, но отданная на попечение мысли. Во-вторых, есть **нечто подобное этой идее** и носящее **то же имя** — **ощутимое, рождённое**, вечно движущееся, **возникающее в некоем месте [...]**.



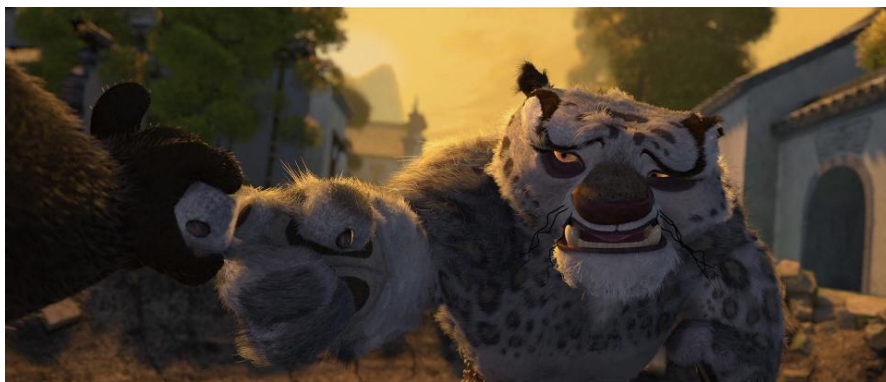
428-348 BC



Что за произведение?

Кунг-фу панда

Помните этот момент?



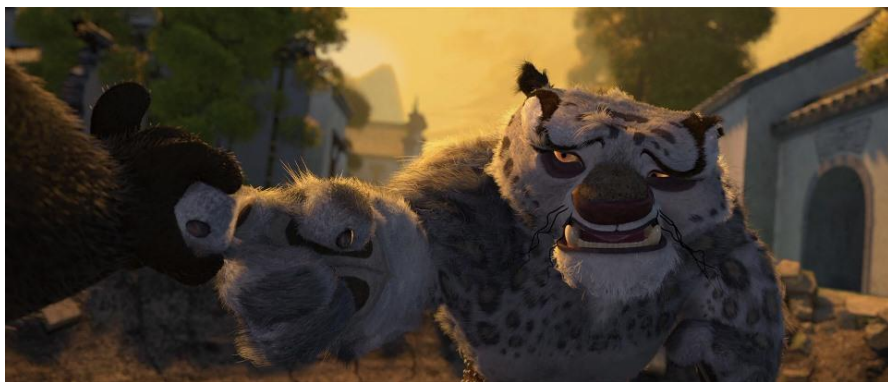
- Ты не можешь победить! Ты всего лишь обычная большая и жирная панда!



*- Да, я большая и жирная панда!
Но не совсем обычная!*

Kung Fu Panda

In English:



- *You can not win! You are just **a** big and fat panda!*



- *No! I am **the** big and fat panda!*

**ПРИШЕЛ ЗАКАЗЧИК,
ЗАКАЗАЛ ДВА ПРИЛОЖЕНИЯ**

Описание продукта #1

Компания оказывает консалтинговые услуги по общим вопросам.

Требуется разработать систему управления договорами компании. Договор может быть с физическим лицом (человеком), может быть с юридическим лицом (другой компанией). У каждого договора есть предмет, сумма и сроки. Сроки и сумма могут быть изменены. У договора должен быть статус, а также должна быть возможность узнать, кто из сотрудников компании является ответственным за договор. У физического лица должны быть известны ФИО, паспортные данные, адрес прописки, у юридического – наименование, адрес, банковские реквизиты, директор. Все договора хранятся в некоем хранилище, должна быть возможность искать в нем по физ.лицу или юр.лицу.

Описание продукта #2

Требуется разработать текстовую игру, где есть два игрока, которые наносят друг другу удары по очереди. Игроки вводят силу удара от 1 до 9, с увеличением силы возрастает вероятность промахнуться. При успешном ударе у противника уменьшаются очки здоровья (health points, hp). Когда hp одного из игроков становится ≤ 0 , этот игрок проиграл.

#1

Вроде ясно как делать:

- Типы данных – числа и строки
- Хранить можно в файле, читать из файла.
- Набор договоров – массив.

Но что такое договор?

Договор

Набор разнотипных переменных.

- String client;
- Date dueTo;
- double price;
- ...

Просто несколько переменных – несогласованные данные. Нужно хранить их «под одной крышей»

- По другому: нужно хранилище разнотипных данных. Массив – хранилище однотипных, а нужно – разнотипных.
 - Так появились record в Pascal, struct в C.
 - *Но это еще не вся проблема.*

weakness of function-based approach

- Разрабатывать эту систему в процедурном стиле (единым алгоритмом): как?
 - договор создать(физ.лицо, предмет)
 - ударить(игрок1, игрок2)
 - ...
- Это мало эффективно (невозможно).
 - Наши будущие приложения не может быть просто суперпозицией функций (как это характерно для процедурного подхода)
 - Более того, они не являются типичным алгоритмом «запустил, отработал, закончил работать». Они потенциально работают бесконечно. Только их кусочки работают как функции.

Вспоминаем школу, русс.яз.

- Функция – набор операторов, оператор – действие.
- Получается, функция – тоже действие. А что в любом естественном языке выражает действие?

Глагол/сказуемое

- Привычные нам алгоритмы формулируются именно так, вспомните!
 - прибавить последние разряды
 - если сумма больше 10, запомнить единичку
 - перейти к соседнему разряду слева
- Попытка программировать систему на уровне только функций – все равно, что описывать окружающий мир только инфинитивными глаголами.
 - Несколько ограничено, как вы понимаете.

С другой стороны

Нам понятны все инструменты, типы данных и операции в обоих примерах.

- **Типы данных:**

int, String, double, Date, boolean, массивы

- **Операции:**

#1 – изменить сумму/сроки – присваивание нового значения переменной

#2 – уменьшение hr при ударе – обычное вычитание

Итак!

Мы не выдумаем новых способов обработки данных.

Просто нужен новый подход к разработке (новая ***парадигма***), когда приложение не может быть представлено как алгоритм, работа которого описывается вызовом функций.

Продолжаем заниматься русским языком! Псевдосинтаксический разбор #1 (неполный, неправильный)

Договор может быть с физическим лицом или юридическим лицом. У каждого договора есть предмет, сумма и сроки. Сроки и сумма могут быть изменены. У договора должен быть статус, а также должна быть возможность узнать, кто из сотрудников компании является ответственным за договор. У физического лица должны быть известны ФИО, паспортные данные, адрес прописки, у юридического – наименование, адрес, банковские реквизиты, директор. Все договора хранятся в некоем хранилище, должна быть возможность искать в нем по физ.лицу или юр.лицу.

Псевдосинтаксический разбор #2 (неполный, неправильный)

Два игрока наносят удары друг другу по очереди.

Игроки вводят силу удара от 1 до 9, с увеличением силы
возрастает вероятность промахнуться. При успешном
ударе у противника уменьшаются очки здоровья (health
points, hp). ...

Подлежащие и дополнения

- *Договор, срок, сумма, предмет, хранилище, адрес, сотрудник.*
- *Игрок, hp, сила удара.*

Это все **существительные**

1. Некоторые – данные примитивного типа (сумма, hp)
2. Но некоторые – договор, игрок – нет.

Существительные 2го типа

(игрок, договор, сотрудник, хранилище)

Не простые! Все будто крутится вокруг них, все будто строится на них!

- Все данные будто хранятся в НИХ
 - Сумма у договора, hp у игрока, договоры у хранилища.
- Все действия привязываются к НИМ
 - Игрок бьет, в хранилище ищем, у договора меняем данные.

Я, объект

Основная единица, юнит, сущность в разрабатываемой системе.

- Содержит в себе данные (статика)
- Совершает действия, обладает поведением (динамика)

Я, не объект

Приведенные две характеристики иллюстрируют, почему не все существительные объекты:

- *Не все имеют поведение (название компании, hr)*
- *Не все являются набором данных (hr, сумма)*
- *Адрес – промежуточные пример*
 - *Поведения не имеет, но...*
 - *Хранит набор данных.*
 - **вердикт?**

Данные (статика, состояние объектов)

- Какие данные хранятся в объекте?
 - Значения примитивных типов + *понятные* ссылочные (String, Date, ...)
 - Другие объекты (договоры у хранилища, ответственный сотрудник у договора)
- Такие вещи мы создавать умеем, т.к. умеем объявлять переменные.
- Такие данные называются атрибутами (полями, свойствами) объектов.

Динамика

Действия, которые совершаются объектами, их поведение.

- Ничего не поменялось, синтаксически это почти все еще функции.
- НО! Теперь все действия привязаны к объектам – поэтому теперь они называются *методами*.
- Методы очевидно меняют атрибуты объектов, значит методам понадобится доступ к атрибутам других объектов.
 - Уменьшение `hp` одного игрока после удара другого (выполнения метода `kick` другим игроком)

Что есть объект

- Атрибуты
- Методы

Вместе это называется **членами класса**.

Получается, нужно просто взять и определить атрибуты и методы для всех объектов...

Не так быстро

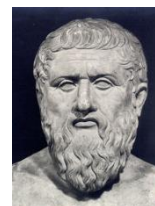
Количество «уникальных» сущностей
отличается от реального количества объектов:

- Уникальные сущности – Договор, Сотрудник, Игрок
- Но сколько будет договоров? Сколько будет сотрудников?
А игроков?
- Есть объекты, составляемые по некоторому
единому подобию, шаблону, каркасу... ничего
не напоминает?

Не так быстро

Количество «уникальных» сущностей
отличается от реального количества объектов:

- Уникальные сущности – Договор, Сотрудник, Игрок
- Но сколько будет договоров? Сколько будет сотрудников?
А игроков?
- Есть объекты, составляемые по некоторому
единому подобию, шаблону, каркасу...



Класс

- ~~Абстрактный объект~~
- Тип данных, состоящий из набора атрибутов и методов
 - *Каркас, шаблон, чертеж объекта*
- Объект – переменная ЭТОГО типа.
 - **Экземпляр (instance) класса (хорошо)**
 - Объект класса (плохо)
 - У класса есть атрибуты и методы, у него нет объектов

Объектно-ориентированный подход к разработке приложений

1. Спроектировать классы

- определить атрибуты
- реализовать методы

- *при этом в методах (этих классов или main) создать объекты, которые выполняют нужный функционал, описанный в своих классах.*

2. ?????????

- исправить ошибки проектирования, ошибки реализации, ошибки компиляции, runtime-ошибки

3. PROFIT

ВПЕРЕД КОДИТЬ!

Player

```
class Player {}
```

Уже правильный класс:

- название – CamelCase
- в теле – члены класса, которых может и не быть
- [почти] каждый класс объявляется в отдельном файле, имя которого совпадает с именем класса (**Player.java**)

Player.java

```
class Player {  
    int hp;  
}
```

hp – атрибут (**camelCase**)
он будет доступен всем методам Player

Contract.java

- Начинаем проектировать.

```
class Contract {  
    String subject;  
    Date dueTo;  
    double cost;  
    ...  
}
```

ПРИШЕЛ ЗАКАЗЧИК ПОМЕНЯЛ ТРЕБОВАНИЯ

- Слушай, это, уточнили у генерального, ответственный должен быть только у договора с физиками. Учтите это у себя.

Contract

Понимаем, что у двух типов контрактов различаются наборы атрибутов, поэтому **пока** придется делать два класса. Пример физ.лица:

```
class IndividualContract {  
    String subject;  
    Date dueTo;  
    double cost;  
    ...  
}
```

Не хватает ответственного,
самого физ.лица...
А чем они являются?

Классов все больше

```
class IndividualContract {  
    String subject;  
    Date dueTo;  
    double cost;  
    Individual individual;  
    Employee responsible;  
}
```

```
class PassportInfo {...}
```

```
class Address {...}
```

```
class Individual {  
    String fio;  
    PassportInfo passportInfo;  
    Address address;  
}
```

```
class Employee {...}
```

Обратите внимание на название полей,
типами которых являются классы

Что писать в Employee

- Заказчик не пришел и ничего не сказал.
 - *Тупить перед ним не хотим.*
- Аналитик: Employee – человек.
 - Какие атрибуты у человека?

Человек

(собрано за 6 лет от студентов)

- ФИО
- Год рождения
- Образование
- Группа крови
- Семейное положение
- Любимая музыка
- Наличие музыкального образования
- Пол
- Месяц рождения
- Рука
- Предпочтения в еде
- Внешние данные
- Пароль от ВК
- Количество лайков на фотку Васи Пупкина в инстаграм.

Тысячи их! Все ли нужны?

Абстракция

Принцип ООП.

В класс добавляются только те атрибуты и методы, которые действительно необходимы в рамках предметной области и разрабатываемой системы.

- Если человек - клиент интернет-магазина книг – то атрибут «группа крови» ему не нужен
- Если человек – клиент в медицинской информационной системе, то атрибут «группа крови» очевидно нужен.

Employee.java

```
public class Employee {  
    String fio;  
    Department department;  
    Employee chief;  
}
```

- Никаких проблем с созданием атрибутов того же класса, который мы проектируем

Метод

- Ну пусть игрок перед началом битвы хочет произнести свой боевой клич!
 - Заказчик пришел, сказал, что не против.
 - Новый атрибут battleCry

```
class Player {  
    int hp;  
    String battleCry;  
  
    void shoutBattleCry() {  
        System.out.println(battleCry);  
    }  
}
```

battleCry глобален для всех методов

Классы создали, теперь давай объекты

Если не оговорено другого, считайте, что код пишется в main.

```
Player p1 = new Player();
```

Уже знакомы с ЭТИМ синтаксисом:

- Player – ссылочный тип,
 - отдельно создаем ссылку p1,
 - отдельно создаем объект с помощью оператора new (который выделяет память под Player)
- Что есть Player()?

Конструктор

- Метод, вызывающийся при создании объекта класса.
 - Там можно инициализировать значения атрибутов, дать им актуальные значения.
 - иначе они все будут null, 0, false,...
 - Что очевидно можно инициализировать в конструкторе для каждого игрока?

Конструктор по умолчанию

- Без параметров
 - Явно не создавали, но он всегда есть, если нет другого.
- Ничего не делает, если его явно не определим.
- `hr` не зависит ни от каких параметров, поэтому его корректно там определить.

Player.java

```
class Player {  
    int hp;  
    String battleCry;  
  
    Player() {  
        hp = 100;  
    }  
  
    void shoutBattleCry() {  
        System.out.println(battleCry);  
    }  
}
```

← void, но не пишем

Наш герой... все еще немой...

```
Player p1 = new Player();  
p1.shoutBattleCry();
```

выдаст null

*точка – оператор доступа
к членам класса!*

Хотим определить его боевой клич! И еще имя, нужно дать ему имя!

- Заказчик приходил, сказал, что думал, что мы сами до этого додумаемся.

Constructor Almighty

- При инициализации необходимо передать данные извне в атрибуты!
- Но ведь у нас уже есть средство это сделать!

Разработка с конца

- Сначала мы определяем поведение объекта, его *интерфейс*.
 - *Слово интерфейс здесь – чисто смысловое. Читать как «как он себя ведет»*
- А потом уже определяем то, что за ЭТИМ поведением стоит (реализуем метод в классе).

Я хочу, чтобы это выглядело так

```
Player p1 = new Player("Wasya", "Leerooooy Jenkinsss");  
p1.shoutBattleCry();
```

Ок, какие вопросы...ааа

```
class Player {  
    int hp;  
    String name;  
    String battleCry;  
    Player() {  
        hp = 100;  
    }  
    Player(String name, String battleCry) {  
        hp = 100;  
        name = name;  
        battleCry = battleCry;  
    }  
    void shoutBattleCry() {  
        System.out.println(battleCry);  
    }  
}
```

добавили имя

передали
параметры

Пытаемся их присвоить...хаха
Ошибки нет, просто параметры присваиваются
параметрам, а не атрибутам (затенение)

Выход

- Переименовать параметры
 - Можно, но не круто.
 - Ведь battleCry – он и везде battleCry
- Другой способ – сказать классу, что это его атрибуты.

this

```
Player(String name, String battleCry) {  
    this.name = name;  
    this.battleCry = battleCry;  
}  
  
void shoutBattleCry() {  
    System.out.println(name + ": " + battleCry);  
}
```

this – это ссылка объекта на самого себя
можете в голове проговаривать слово «МОЙ»

не требуется там, где очевидно

Дублирование кода

```
Player(String name) {  
    hp = 100;  
    this.name = name;  
}  
  
Player(String name, String battleCry) {  
    hp = 100;  
    this.name = name;  
    this.battleCry = battleCry;  
}
```

Может думать так

```
Player(String name) {  
    hp = 100;  
    this.name = name;  
    this.battleCry = "Leroooooy Jenkinsss";  
}  
  
Player(String name, String battleCry) {  
    hp = 100;  
    this.name = name;  
    this.battleCry = battleCry;  
}
```

Кто чей частный случай?

this – не ТОЛЬКО СЛОВО

```
Player(String name) {  
    this(name, "Leroooooy Jenkinsss");  
}  
Player(String name, String battleCry) {  
    hp = 100;  
    this.name = name;  
    this.battleCry = battleCry;  
}
```

Еще раз про классы и объекты

- Классы проектируются от первого лица, объекты используются от третьего!
 - Сравните класс Player и объект p1
- Проектирование класса и использование его экземпляров – два разных процесса разработки (разное место, разное время)
 - Мы это делаем в случае игры вместе, т.к. объекты Player – это фактически неявно поля класса Game.
 - Вот вы используете объекты String, Scanner, Date – а вы хоть раз в реализацию смотрели?
 - » Поэтому надо мыслить создание класса и использование класса **раздельно!**

Доступ к полям

Вернемся к договорам. Нам сказали, что у договоров «можно менять сроки и сумму». Казалось бы, бери и меняй (это же переменные)

```
IndividualContract ic1 =  
    new IndividualContract(  
        "Development",  
        new Date(2016, 3, 15),  
        100000);  
  
...  
ic1.cost = 200000;
```

НО ЭТО ЖЕ ПРОСТО ПЕРЕМЕННЫЕ, ЗНАЧИТ МОЖНО...

```
ic1.subject = "ФИГНЯ всякая";
```

Но в требованиях не было написано, что так
МОЖНО.

Заказчик пришел, сказал, что можем начинать нести
деньги чемоданами.

Немного в философию.

- Какого цвета глаза?
- Как узнать?



Пример с глазами

- Какого цвета глаза?
- Как узнать?
 - ~~Хакнем бытие!~~
 - ТОЛЬКО МЕТОДОМ!

Модификаторы доступа

- Определяют возможность прямого доступа к членам класса и к самим классам
 - **public** – доступ всем отовсюду
 - **private** – прямой доступ только внутри класса, в котором находится данный атрибут/метод
 - ...
 - ...

Инкапсуляция

- Скрытие реализации объекта (*читай, атрибутов и тел его методов*)
- «Мне не нужно знать устройство машины, чтобы ее водить»
 - Водитель не может «вращать двигатель»

IndividualContract.java

```
public class IndividualContract {  
    private String subject;  
    private Date dueTo;  
    private double cost;  
    private Individual individual;  
    private Employee responsible;  
  
    public IndividualContract(String subject,  
                             Date dueTo, double cost) {  
        this.subject = subject;  
        this.dueTo = dueTo;  
        this.cost = cost;  
    }  
}
```

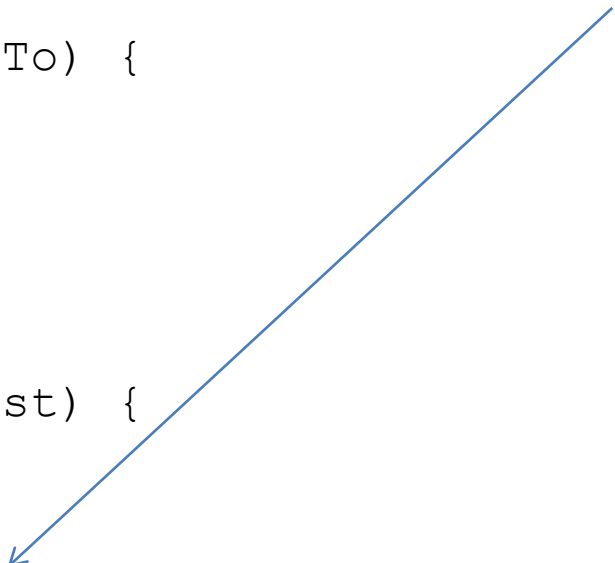
Set-, get- методы

- Методы, с помощью которых мы получаем доступ к атрибутам (get) и изменяем их (set).
- В средах разработки могут быть сгенерированы автоматически.

set, get в Contract

```
private String subject;  
private Date dueTo;  
private double cost;  
  
public Date getDueTo() {  
    return dueTo;  
}  
  
public void setDueTo(Date dueTo) {  
    this.dueTo = dueTo;  
}  
  
public double getCost() {  
    return cost;  
}  
  
public void setCost(double cost) {  
    this.cost = cost;  
}  
  
public String getSubject() {  
    return subject;  
}
```

На Subject только get




заказчик пришел, сказал,
так и быть, штрафовать не будет

Статические поля и методы

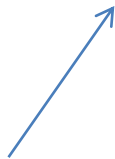
- У каждого объекта – свой набор значений полей:
 - У каждого договора – свои поля
 - У каждого игрока – свои hp, клич
- Но иногда есть необходимость в атрибутах класса, общих на всех
 - Что если хочу сделать сквозную нумерацию договоров IndividualContract

Static

```
public class IndividualContract {  
  
    private static int numberOfContracts = 0;  
  
}
```



модификатор для таких данных



static данные общие для всех объектов,
они должны существовать, даже если ни одного объекта не создано (ни одного конструктора не вызвано). Поэтому инициализируем вот так.

Работа с этими данными обычна

```
public IndividualContract(String subject,  
                           Date dueTo, double cost) {  
    this.subject = subject;  
    this.dueTo = dueTo;  
    this.cost = cost;  
    numberOfContracts++;  
}
```


Стоп-стоп

- Но если есть данные, привязанные к классам, а не к объектам, то наверное, есть и методы, привязанные только классам.
 - Не имеющие смысла для отдельных объектов
 - *Щелкает?*

Примеры

- `Math.cos`, `Math.sin`, ...
- `Integer.parseInt`, `Double.parseDouble`
- А какой еще метод должен (ОБЯЗАН) работать, даже тогда, когда еще ни одного объекта не создано?

Разгадка main!

```
public static void main(String[] args)
```

Точка запуска программы

- **public** – чтобы ее могли запустить извне
- **static** – потому что main не принадлежит конкретному объекту. Метод main должен запуститься, когда еще ни один объект не создан.

В статических методах

- Могут использоваться только другие статические методы/статические атрибуты класса
 - Опять, потому что должны работать, когда ни одного объекта не создано.
 - Поэтому все методы рядом с `main` писались тоже как `public static`.

**ПЕРЕРЫВ НА НЕДЕЛЬКУ,
ЗАКАЗЧИК УЕХАЛ В ОТПУСК**