Казанский федеральный УНИВЕРСИТЕТ
ИНСТИТУТ информационных технологий и интеллектуальных систем

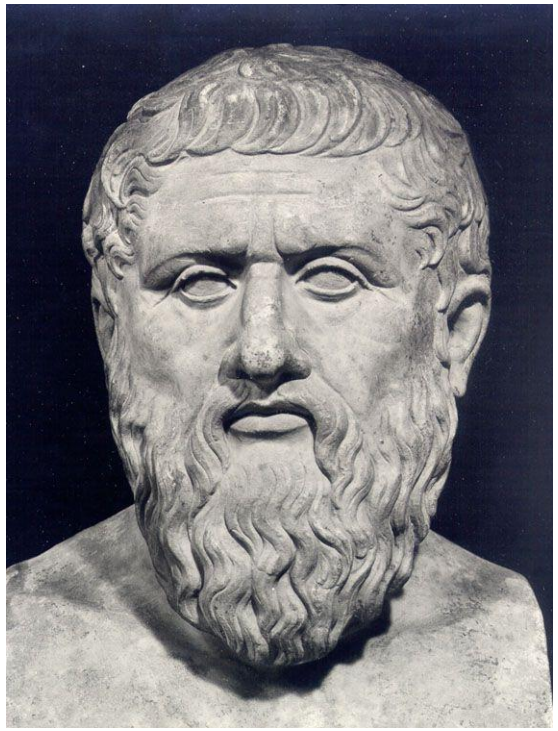
07. ВВЕДЕНИЕ В ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

курс лекций по информатике и программированию
для студентов первого курса ИТИС КФУ (java-поток)
2023/2024

М.М. Абрамский

кандидат технических наук, доцент кафедры программной инженерии

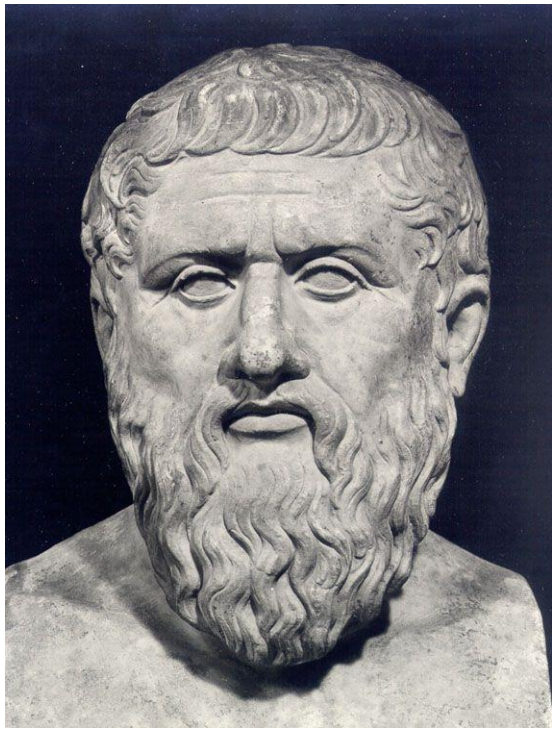
ИСТОКИ КОНЦЕПЦИИ



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 ИЗ КОМПАНИИ N, ЗАКАЗАЛ ДВА ПРИЛОЖЕНИЯ

Описание приложения #1

Компания **X** оказывает микрокредитные услуги.

Требуется разработать систему управления договорами компании.

Договор может быть заключен с физическим лицом (человеком) или юридическим лицом (другой компанией).

У каждого договора есть сумма, процент и сроки погашения. Они могут быть изменены.

У договора должен быть статус, а также 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 договоры хранятся в некоем хранилище, должна быть возможность искать в нем договора по признакам.

Описание приложения #2

Еще компания **N** иногда занимается разработкой компьютерных игр.

Требуется разработать игру, где два игрока наносят друг другу удары по очереди. Силу удара на каждом ходу может быть от 1 до 9

С увеличением силы возрастает вероятность промаха.

При успешном ударе у противника уменьшаются очки здоровья (health points, hp).

Когда hp одного из игроков становится <= 0, он проигрывает.

Вроде ясно как делать:

– Нет непонятных типов данных (почти все – числа и строки);
– Операции – тоже понятные (найти, изменить, подсчитать, вычесть)

Но

– что такое Договор? Физическое лицо? Игрок?

Договор

Набор разнотипных переменных.

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

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

Массив – хранилище однотипных, а нужно – разнотипных.

- Так появились record в Pascal, struct в C.

- *Но это еще не вся проблема.*

Функция

- 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

- Такие алгоритмы (как и все функции) не хранят ***состояние***

ЧТО ОПЯТЬ ЗА «СОСТОЯНИЕ»??

Пример

Ключевой функционал приложения микрокредитования:

Где-то в коде приложения

*В слое **бизнес-логики***

- *Пройдем, что это, чуть позже, а пока просто отсылка*

Функция **дать_деняк**

Какие у нее могут быть параметры и возвращаемое значение?

Варианты

int дать_деняк() {…}

или

boolean дать_деняк(**int** сколько_деняк_надо) {…}

В любом случае **очевидно**: одна и та же функция при одинаковых параметрах может давать разные результаты.

Из-за чего?

Зависит от клиента

- У одного клиента есть доход, у другого нет дохода
- Уже давали – и не вернул
- Подозрительный

Как это запрограммировать?

Варианты

```
boolean дать_деняк(int сколько_деняк_надо, boolean уже_давали_деняк,
boolean подозрительный, int доход, …) {…}
```

? А где хранятся эти значения, чтобы их передать в вызов функции?

- ~~Глобальные переменные~~
- ~~Файлы~~

? Если у нас появится новый критерий, мы все время будем переписывать заголовок функции

Другой взгляд

- Клиент – набор характеристик, параметров и др. Их совокупность – **состояние (state)**.
  - Как во фразе «его состояние – такое-то» (и про здоровье, и про деньги, и про психологическое состояние)
- В зависимости от **состояния (во всех смыслах)** клиента мы ему даем / не даем кредит
  - А в коде как?

Варианты

```
boolean дать_деняк(int сколько_деняк_надо, клиент)
{…}
```

Клиент хранит **состояние**. Набор характеристик, каждая из которых имеет понятный тип данных

клиент.доход

клиент.подозрительный

…

Кто «дает деняк». А они вообще есть?

Состояние есть не только у клиента, но и у фирмы. Как функция узнает о ее состоянии?

```
boolean дать_деняк(int сколько_деняк_надо, клиент, контора) {…}
```

Вызов: `дать_деняк(10000, вася, рога_и_копыта)`

Но говорим мы по-другому – «контора дает»:

```
рога_и_копыта.дать_деняк(вася, 10000)
```

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, а ближе к естественному описанию предметной области.

Где все крутится вокруг особых **«существительных».** Игрок бьет, в хранилище ищем, у договора меняем данные.

Я, объект

- 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, hp)*
- *Не все являются набором данных (hp, 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)** объектов.

Динамика

Действия, которые совершаются объектами, их поведение.

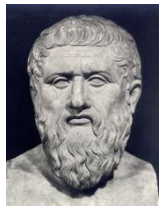
- Синтаксически это почти все еще функции.
- НО! Теперь все действия привязаны к объектам – поэтому теперь они называются *методами*.
- Методы очевидно меняют атрибуты объектов, значит методам понадобится доступ к атрибутам других объектов.
  - Уменьшение **hp** одного игрока после удара другого (выполнения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

Внимание

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?



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 (*ну такое*)
    - У класса есть атрибуты и методы, у него нет объектов

Что есть класс (объект)

- Атрибуты
- Методы

Вместе это называется **членами класса.**

Получается, нужно просто взять и определить атрибуты и методы для всех 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. Неопределенное количество раз повторяем шаг 2 (а если не везет, то и шаг 1)

4. ????????

5. УСПЕХ!

ВПЕРЕД КОДИТЬ!

Player

```
class Player {}
```

уже правильный класс:

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

Player.java

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

hp – атрибут (**c**amelCase)

Допустимо, но с умом:

```
class Player {
    int hp = 10;
}
```

Contract.java

Начинаем проектировать.

```
class Contract {
    String subject;
    Date dueTo;
    int money;
    double percent;
    …
}
```

Классов все больше

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

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

```
class PassportInfo {…}
```

```
class Address {…}
```

```
class Employee {…}
```

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 10 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 всех видео Васи Пупкина во всех соц.сетях.

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

Классы создали, теперь 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,…

Конструктор по умолчанию

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

Player.java

```
class Player {

    int hp;

    Player() {
        hp = 100;
    }

}
```

void, но не пишем

```
class Player {
    int hp = 100;
}
```

В чем отличие от инициализации?

https://stackoverflow.com/questions/36244977/creating-constructor-for-attributes-vs-just-creating-attributes-and-using-the-d

Метод

- 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 тоже глобален для всех методов класс

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", "Всё нормально,
                    всё хорошо, сейчас будем драться");
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 = "Всё нормально, всё
            хорошо, сейчас будем драться";
}
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, "Всё нормально, всё
          хорошо, сейчас будем драться");
}
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**!


To Be Continued