OOP KEY PRINCIPLES

Например, в тестировании* у нас есть 7 ключевых принципов:

1. Тестирование показывает лишь наличие дефектов
2. Исчерпывающее тестирование невозможно
3. Раннее тестирование экономит время и деньги
4. Дефекты имеют особенность группироваться (эффект соседей)
5. Существует парадокс пестицида (устойчивость дефектов к тестам)
6. Тестирование должно зависеть от контекста
7. Отсутствие дефектов – заблуждение

<u>В ООП тоже существуют свои принципы:</u>

1. **<u>Абстракция</u>** — ключевой принцип, позволяющий нам справляться со сложностью внешнего мира.

С помощью абстракции мы выделяем наиболее важные характеристики и информацию об объекте. Таким образом, мы отсекаем «всё ненужное», оставляя только самое необходимое для решения задачи.

2. **<u>Инкапсуляция</u>** — объединение данных и методов доступа к ним для

предотвращения прямого доступа.

Мы уже обсуждали на примере кондиционера, что приватные поля + методы доступа (get/set) должны предотвратить установку

температуры выше максимальной/ниже минимальной

Методы доступа, которые мы предоставляем другим объектам, называют ещё интерфейсом* нашего объекта (API, которое мы

обсуждали на лекции по JUnit).

Благодаря тому, что мы предоставляем только методы доступа, мы можем менять внутреннюю реализацию (так как она скрыта от

остальных), не нарушая нашего с ними взаимодействия.

3. наследование и полиморфизм

4. **<u>Композиция</u>** — это подход, при котором мы строим объекты, собирая

их (как конструктор) из других объектов.

Для отражения текущих тенденций мы добавим ещё один важный

для нас принцип ООП — **Testability.**

ID (идентификатор)

Мы дадим вам совет, который сэкономит вам время: привыкните, что у каждого объекта, представляющего собой абстракцию данных,

должно быть поле id.

id — это уникальный идентификатор, позволяющий найти этот

объект среди других объектов.

Во всех системах, хранящих данные, есть идентификаторы: артикул, инвентарный номер и т.д. Их можно называть по-разному, но они должны быть.

id не обязательно должен быть числом, хотя так делают достаточно

часто.

Например, в качестве id для кино можно выбрать строку из ссылки

(если кликнуть на тизер):

вперёд — vpered-2020

отель Белград — otel-belgrad

остров фантазий — ostrov-fantazii-2020

Мы можем попытаться предугадать возможные варианты

(построить гипотезы) на основании:

- своего опыта (если мы являемся экспертами этой области)
- анализа конкурентов
- общения с заказчиками, пользователями и другими экспертами

Конечно же, если есть ТЗ, то нужно использовать информацию из

него (но не факт, что составитель ТЗ тоже предусмотрел всё).

Из ответа на вопрос следуют выводы:

1. Нужно наращивать экспертизу в конкретной области

2. Нужно быть готовым к изменениям и уметь производить их

Безопасно:

Нужно проверить, что после наших изменений всё работает «как

нужно»: тесты (или автотесты) и testability.

МЕНЕДЖЕР

Договоримся, что у нас будут

объекты, которые содержат только данные (+ get/set к ним), и

объекты, которые управляют ими.

Тогда наша работа делится на:

1.объекты с данными (классы, описывающие их называются *data-classes*)

2.объекты-менеджеры

Таким образом, мы приходим к следующему:

1) если объекты data-классов содержат только приватные поля + get/set к ним и не содержат логики*, то тесты к ним не нужны
2) объектам data-классов не нужно знать об окружающем мире, за это теперь будут отвечать менеджеры
3) но для объектов-менеджеров тесты нужны, поскольку именно в них теперь будет сосредоточена вся бизнес-логика

```
public class MovieManager {
    private Movie[ ] movies;

    public Movie[ ] getMoviesForFeed() {
        // TODO: add logic
        return null;
    }
}
```

Обратите внимание: никаких getter'ов и setter'ов здесь нет.

Менеджер на данном этапе будет отвечать за:

1) хранение списка (в нашем случае массива) объектов

- добавление объекта в список
- удаление объекта из списка
- поиск объекта в списке
- другие операции (сортировка и т.д.)

Метод generateBlock - генерирует свой «кусочек» страницы.

Полиморфизм и наследование позволяют нам осуществить этот трюк в Java: мы можем называть всех менеджеров, кроме главного, BlockManager. У всех менеджеров должен быть метод generateBlock, но

при этом этот метод будет у каждого свой.

Тогда задача главного менеджера страницы предельно проста:

1. собрать всех менеджеров
2. сказать каждому: «генерируй блок»
3. включить сгенерированный блок в страницу

Ему не нужно знать, что один генерирует фильмы, а другой — новости.

```
public class MainPageManager {
   private BlockManager[] managers;

   /**
    * Main Page generation
    */
   public String generate() {
      for (BlockManager manager : managers) {
         String block = manager.generateBlock();
      }
      // TODO: add logic
      return null;
   }
}
```

сюда можно класть любых менеджеров

но метод у каждого свой (со своей логикой)

Ключевое упрощение: мы можем при необходимости делить классы по назначению (менеджеры и data-классы).

Самое важное для вас как для тестировщика, программиста и автоматизатора — обязательно нужно смотреть на:

- реальные системы, существующие в объектах поля
- связи между объектами
- механику взаимодействия

Мы верхнеуровнево рассмотрели ключевые принципы ООП на примерах. Дальше мы их будем внедрять в практику.

Важно, чтобы вы понимали, что эти принципы нужны для обеспечения:

- простоты
- гибкости
- безопасности

При правильном применении они также обеспечат и тестируемость.