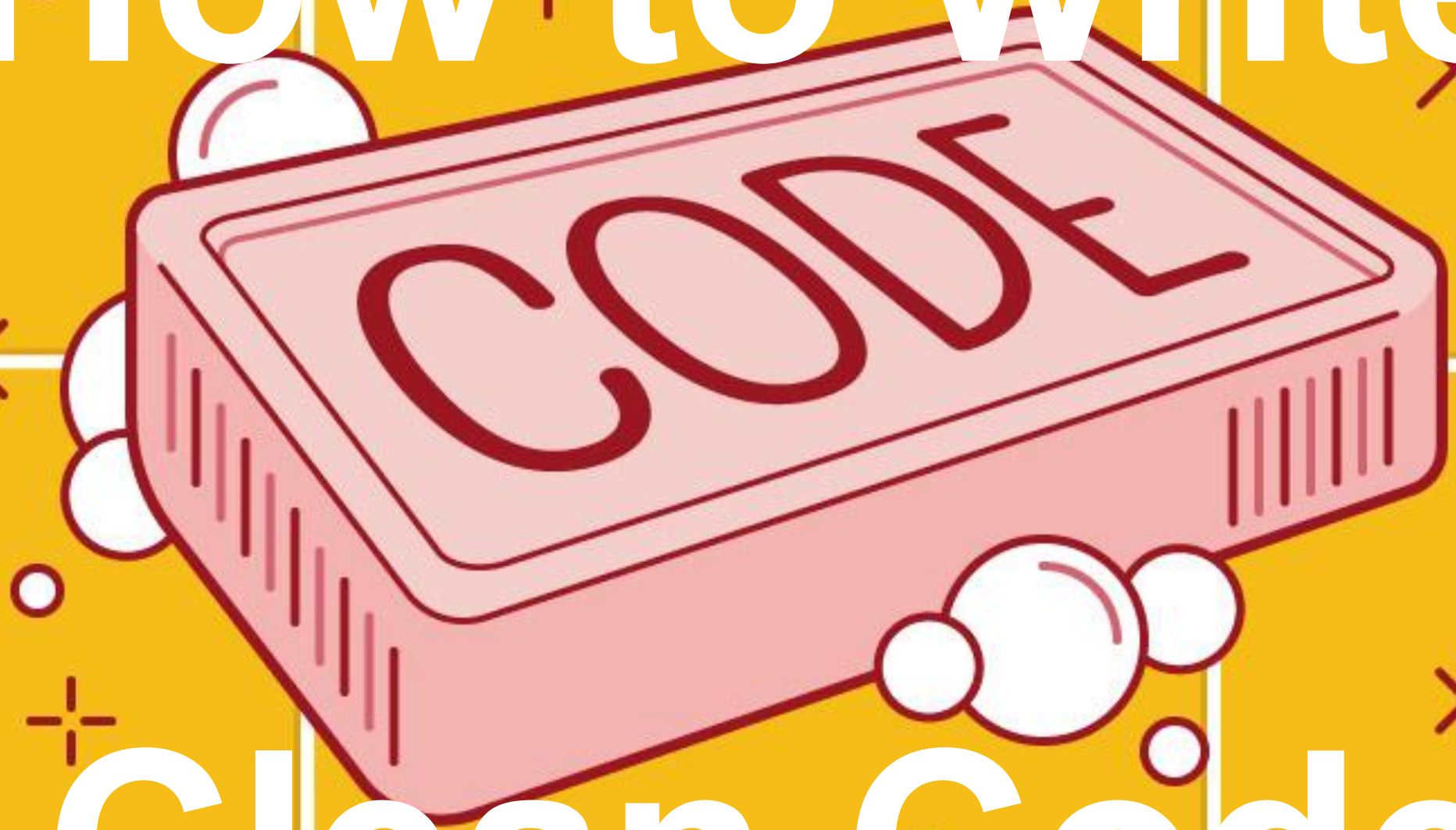
ЛЕКЦИЯ I

JAVA OOP: Качествен код 2

Теодор Костадинов

How to write Clean Code



SOFTWARE ACADEMY
Educational Center Net It

Какво е лош код?

Лично пространство

- Дръжте всичко private, не дефинирайте getters & setters автоматично

- Смисълът на криенето на имплементация не е в това просто да добавим слой методи между юзъра и данните, а в това да създадем такава абстракция, че лесно да можем да променяме имплементацията след това

```
public class Point {
    public double x;
    public double y;
  }
```

```
public interface Point {
    double getX();
    double getY();
    void setCartesian(double x, double y);
    double getR();
    double getTheta();
    void setPolar(double r, double theta);
  }
```

```
public interface Vehicle {
    double getFuelTankCapacityInGallons();
    double getGallonsOfGasoline();
}
```

```
 public interface Vehicle {
    double getPercentFuelRemaining();
  }
```

SOFTWARE ACADEMY
Educational Center Net It

Какво е структура от данни?

Обекти и структури

- Обектите крият своите данни зад абстракции и разкриват функции, които оперират над тези данни

- Структурите разкриват своите данни и нямат значими функции

- Какво ще стане ако добавя функцията периметър?
- Какво ще стане ако добавя нова фигура?

```
public class Square {
  public Point topLeft;
  public double side;
}

public class Rectangle {
  public Point topLeft;
  public double height;
  public double width;
}

public class Circle {
  public Point center;
  public double radius;
}
```

```
public class Geometry {
  public final double PI = 3.14159265358979;

  public double area(Object shape) throws
                                    NoSuchShapeException {
      if (shape instanceof Square) {
          Square s = (Square)shape;
          return s.side * s.side;
      } else if (shape instanceof Rectangle) {
          Rectangle r = (Rectangle)shape;
          return r.height * r.width;
      } else if (shape instanceof Circle) {
          Circle c = (Circle)shape;
          return PI * c.radius * c.radius;
      }
      throw new NoSuchShapeException();
  }
```

```
public class Rectangle implements Shape {
  private Point topLeft;
  private double height;
  private double width;

  public double area() {
    return height * width;
  }
}

  public class Circle implements Shape {
    private Point center;
    private double radius;
    public final double PI = 3.1415926535897793;

    public double area() {
      return PI * radius * radius;
    }
}
```

- Какво трябва да променя ако добавя нова фигура?
- Какво трябва да променя ако добавя функция за параметъра?

Procedural code (code using data structures) makes it easy to add new functions without changing the existing data structures. OO code, on the other hand, makes it easy to add new classes without changing existing functions.

Или обратното:

Procedural code makes it hard to add new data structures because all the functions must change. OO code makes it hard to add new functions because all the classes must change.

THE LAW OF DEMETER

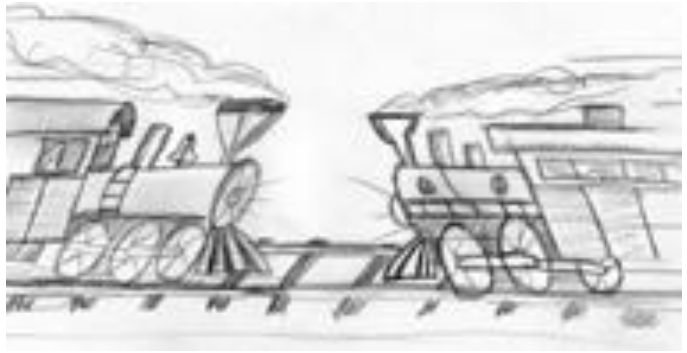
SOFTWARE ACADEMY
Educational Center Net It

A module should not know about the innards of the *objects* it manipulates

Метод м от класа К може да извиква само:

- К
- Обект, създаден в м
- Обект, подаден като аргумент на м
- Обект, който е поле на К

Методът не трябва да вика методи на обекти, които са върнати като резултат на някои от позволените функции



```
final String outputDir =
ctxt.getOptions().getScratchDir().getAbsolutePath();
final String outputDir = ctxt.options.scratchDir.absolutePath;
 ctxt.getAbsolutePathOfScratchDirectoryOption();
 ctx.getScratchDirectoryOption().getAbsolutePath()
```

```
String outFile = outputDir + “/” + className.replace('.', '/') + “.class”;
FileOutputStream fout = new FileOutputStream(outFile);
BufferedOutputStream bos = new BufferedOutputStream(fout);
```

```
BufferedOutputStream bos = ctxt.createScratchFileStream(classFileName);
```

Хибриди

- Полу-обект и полу-структура
- Те имат значими функции, но също така и публични полета или мутатори, които на практика правят полетата публични
- При тях добавянето на нови функции е трудно
- При тях добавянето на нови структури е труд

Класове

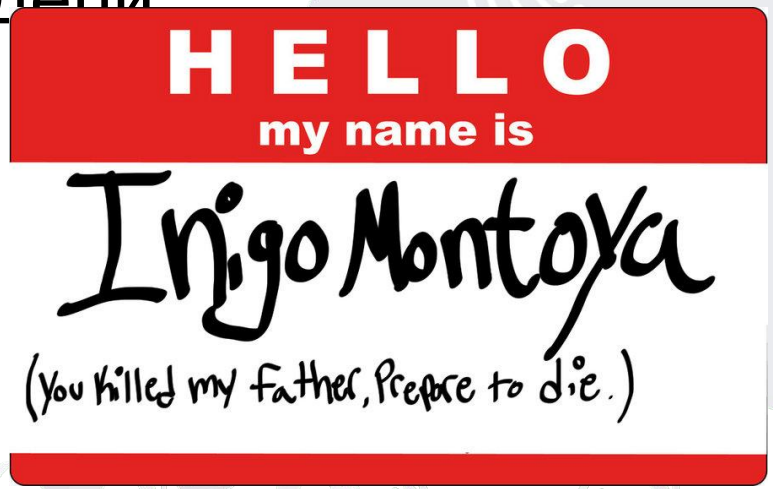
- Класът трябва да има подредба: първо полетата - public statics, private statics, private instances, после методите - publics first, private after each public they are referenced into
- Всичко което може да е private, трябва да е, освен ако тест не изисква друго. За тестовете си позволяваме свобода.
- Класовете трябва да са малки. За функциите използвахме брой редове, тук използваме брой отговорности на класа

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public String getCustomizerLanguagePath()
    public void setSystemConfigPath(String systemConfigPath)
    public String getSystemConfigDocument()
    public void setSystemConfigDocument(String systemConfigDocument)
    public boolean getGuruState()
    public boolean getNoviceState()
    public boolean getOpenSourceState()
    public void showObject(MetaObject object)
    public void showProgress(String s)
    public boolean isMetadataDirty()
    public void setIsMetadataDirty(boolean isMetadataDirty)
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public void setMouseSelectState(boolean isMouseSelected)
```

```
public class SuperDashboard extends JFrame implements MetaDataUser
    public Component getLastFocusedComponent()
    public void setLastFocused(Component lastFocused)
    public int getMajorVersionNumber()
    public int getMinorVersionNumber()
    public int getBuildNumber()
}
```

Класове

- Името подсказва отговорността на класа. Ако не може да се избере хубаво име или името е обобщено или съдържа думи като Manager, Processor, Super, значи вероятно класа трябва да се раздели на по-малки
- Ако в описанието на класа използваме думите АКО, ИЛИ, НО то това е знак че класа трябва да се раздели



SOFTWARE ACADEMY
Educational Center Net It

The Single Responsibility Principle

The Single Responsibility Principle (SRP) states that a class or module should have one, and only one, *reason to change*.

SRP

- Един от основните ОО дизайн принципи
- Един от най-пренебрегваните
- Да подкараш един код и да го направиш чист са две различни неща и често мислим само за първото
- Много, но малки класове няма да направи системата по-трудна за разбиране, точно обратното
- Ако имате кутия с инструменти, предпочитате да имате много чекмеджета, надписани с точно какво има в тях ИЛИ две-три чекмеджета, в които всичко е наблъскано и трябва да ровите в тях за да намерите каквото ви трябва

Кохезия

- Класовете трябва да имат малък брой полета. Всеки метод трябва да манипулира едно или няколко от тях.
- Колкото повече полета използва един метод, толкова по-силна е кохезията между него и класа
- Когато кохезията е силна, значи класа и методите му са силно свързани логически и трябва наистина да бъдат заедно

Поддържайки кохезивност, държим класовете малки

- Имаме голяма функция с много променливи в нея.
- Екстрактваме част от функцията в друга функция, но кодът на новата използва четири променливи на голямата. Правим променливите полета.
- Екстрактваме друга част от голямата функция, новата използва пак същите новосъздадени полета
- Сега имаме много полета, но четири от тях се използват само то две функции.
- Това е нов клас. Изваждаме четирите полета и двете нови функции в него.

SOFTWARE ACADEMY
Educational Center Net It

Пример с прости числа: Listing 10-5 `PrintPrimes.java (P142)`

https://pastebin.com/zCeaZdbG

SOFTWARE ACADEMY
Educational Center Net It

Какво е boundary?

Граници

Понякога използваме външни библиотеки. Понякога очакваме друг екип от нашата компания да ни предостави някой модул или под-система.

Какъвто и да е случая, ние трябва качествено да интегрираме чуждия код с нашия.

Когато работим с чужд код, трябва да държим границата с нашия код ясно разграничима.

Граници

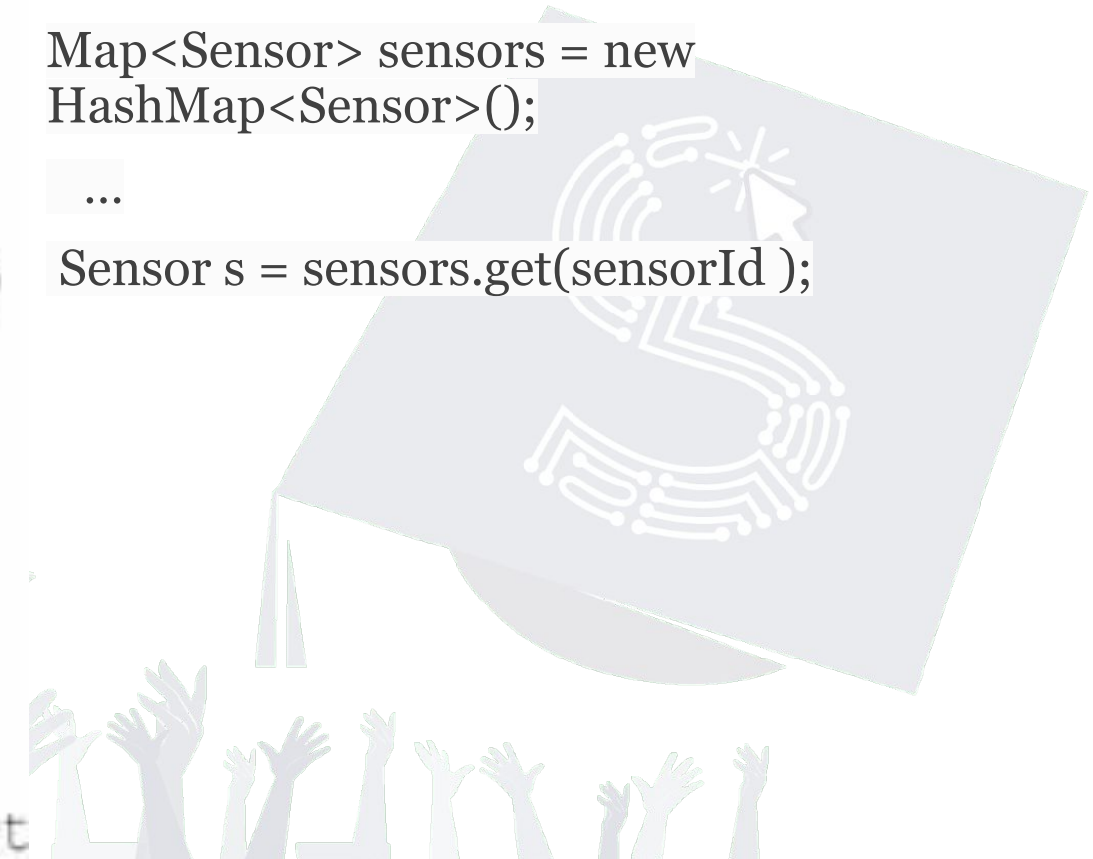
- Има разминаване между какво иска създателя на една библиотека и какво иска потребителя й. Създателя иска възможно най-голяма приложимост, потребителя иска възможно по-голяма специфичност.

Пример: Искаме да имаме Map с един конкретен тип обект, и Map-а да не може да се изтрива

- clear() void - Map
- containsKey(Object key) boolean - Map
- containsValue(Object value) boolean - Map
- entrySet() Set - Map
- equals(Object o) boolean - Map
- get(Object key) Object - Map
- getClass() Class<? extends Object> - Object
- hashCode() int - Map
- isEmpty() boolean - Map
- keySet() Set - Map
- notify() void - Object
- notifyAll() void - Object
- put(Object key, Object value) Object - Map
- putAll(Map t) void - Map
- remove(Object key) Object - Map
- size() int - Map
- toString() String - Object
- values() Collection - Map
- wait() void - Object
- wait(long timeout) void - Object
- wait(long timeout, int nanos) void - Object

```
Map sensors = new HashMap();
Sensor s = (Sensor)sensors.get(sensorId );
```

```
Map<Sensor> sensors = new
HashMap<Sensor>();
  ...
Sensor s = sensors.get(sensorId );
```



```
public class Sensors {
  private Map sensors = new HashMap();

  public Sensor getById(String id) {
    return (Sensor) sensors.get(id);
  }
}
```

Граници

- Не всеки Map трябва да се енкапсулира в наш интерфейс, но когато Map-а е на граница, то това е почти задължително
- Искате да имате пълен контрол над това, което дават вашите методи на потребителите на вашия код. Ако им подадете Map те лесно могат да го манипулират и да компрометират системата ви
- Ако обаче Map-а се използва само вътрешно от вашите класове, то тогава няма проблем

Exploring Boundaries

- Най-добрия начин да разберете как работи един външен компонент е да напишете тестове за него
- **LEARNING TESTS ARE BETTER THAN FREE**
- Чрез тестовете потвърждаваме, че външния код прави това което искаме
- Чрез тестовете се научаваме как да използваме външния код

Използване на несъществуващ код

- Друг вид граница, разделящ познатото от неизвестното
- Когато кодът който трябва да използваме още не е готов, просто сложете слой абстракция между него и вашия код
- Като си дефинирате свой интерфейс и използвате него вместо чуждия код, вие запазвате вашия собствен код чист
- Също така вие сте този който определя как да се използва чуждия код, а не обратното

ВЪПРОСИ?

Резюме

- Винаги мислете за качеството на кода.
- Един код трябва да се рефакторира постоянно, за да се държи в добра форма.
- За да може това да се прави безопасно, трябва да има юнит тестове.

Ресурси

- Clean Code Book
- Single Responsibility Principle | Object Oriented Design
- Кохезия

SOFTWARE ACADEMY
Educational Center Net It

Детектор за плагиатство

- Условието е тук: https://github.com/fmi/java-course/blob/ab5f6a55fce87696aca7f4edc55d30b60e92b71a/homeworks/02-authorship-detection/README.md

- Не е задължително да спазвате класовете и интерфейсите дефинирани в условието.

Ами сега?

Задача за упражнение

БЛАГОДАРЯ
ЗА ВНИМАНИЕТО!