Pain of OOP

Series of lectures by Yegor Bugayenko

Abstract:
The course is a critical review of the current situation in object-oriented programming, especially in Java, C++, Ruby, and JavaScript worlds. At the course, certain programming idioms, which sometimes are called "best practices," are criticized for their negative impact on code quality, including static methods, NULL references, getters and setters, ORM and DTO, annotations, traits and mixins, inheritance, and many others. Much "cleaner" object-oriented programming practices will be proposed instead. Most lectures are organized as reviews of existing code snippets from well-known open source libraries.

What is the goal?

The primary objective of the course is to help students understand the difference between "object thinking," originally motivated the appearance of OOP, and the modern practices that often severly impact the quality of code in a negative way.

Who is the teacher?

Yegor is developing software for more than 30 years, being a hands-on programmer (see his GitHub account: @yegor256) and a manager of other programmers. At the moment Yegor is a director of an R&D laboratory in Huawei. His primary research focus is software quality problems. Some of the lectures he has recently presented at some software conferences could be found at his YouTube channel. Yegor also published a few books and wrote a blog about software engineering and object-oriented programming. Yegor previously tought a few courses in Innopolis University (Kazan, Russia) and HSE University (Moscow, Russia), for example, Software Systems Design and Ensuring Quality in Software Projects (all videos are available).

Why this course?

Maintainability of object-oriented software that most of us programmers write these days is way below our expectations. Two main reasons for that is our misunderstanding of what objects are. This course may help clear things up.

What's the methodology?

Each lecture is an overview of existing source code from well-known open source libraries and frameworks, mostly from Apache. In an interactive discussion mode we will go through their code and find out whether their programming practices are in line with "object thinking" .

Course Structure

Prerequisites to the course (it is expected that a student knows this):

- How to write code
- How to design software

After the course a student *hopefully* will understand:

- What is the difference between objects and data?
- Why static methods are bad?
- What is immutability and why it's good?
- How to design a constructor?
- How to handle exceptions right?
- What data hiding is for?
- What's wrong with Printers, Writers, Scanners, and Readers?
- Why NULL references are a billion-dollar mistake?
- What's wrong with mixins and traits?
- How to avoid getters and setters?
- How declarative programming is better than imperative?
- Why composition is better than inheritance?
- Where to store the data if DTO is a bad practice?
- How to avoid ORM design pattern?
- Why long names of variables is bad design?
- Why type casting and type checking are against OOP?
- What is cohesion and why it matters?
- How to apply SOLID and SRP principles?
- What Inversion of Control is for and why DI Containers are evil?
- Why MVC is a bad design idea?

Grading

[45%]
Students form groups of 1–3 people. Each group presents its own public GitHub repository with a software module inside. It may be any piece of object-oriented software as long as it has more than 5,000 lines of code written personally by the students. Higher grades will be given for better object-oriented practices used in the code. Lower grades will be given for the precense of static methods, NULLs, getters, setters, mutability, ORM, DTO, factories, type casting, and other anit-OOP practices.

[30%]
At the laboratory classes each group will have to complete three home works and defend them verbally on-site.

[25%]
The attendance is tracked at the lectures.

Learning Material

The following books are highly recommended to read (in no particular order):

David West, *Object Thinking*
David West, *Design Thinking*
Robert Martin, *Clean Code*
Steve McConnell, *Code Complete*
Yegor Bugayenko, *Elegant Objects*