

Object-Oriented Programming

Flocea Dominic

Technical University of Moldova

October 5, 2025

Lecture 1: Course Intro

Let's go back to the 70s

- Disco Music, *Led Zeppelin*, *Pink Floyd* and *Queen* were destroying the charts;
- Muhammad Ali was the heavyweight champion of the world;
- Atari released *Pong*;
- Silicon Valley startup called Intel created the first commercial microprocessor;

Cue in The legendary - Xerox PARC

3 major innovations

Which would you pick
for **biggest impact**?

- The Graphical User Interface (GUI)
- The Mouse (The Mouse)
- Object-Oriented Programming (OOP)

About tutor



Flocea Dominic - University
Assistant

- BSc @ TUM FAF Alumni
- Software Developer @ Orange Systems
- University Assistant @ TUM
- MSc @ TUM

About tutor



Flocea Dominic - University
Assistant

Languages

- Java (native speaker)
- Kotlin (native speaker)
- C/C++ (recreational)
- Python (recreational)
- JS/TS (recreational)
- Romanian, English, Russian

Course disclaimer

The lecture slides and materials, including the content and theme were created by the author. Please do not distribute and use without prior permission.

Now I want to hear about you!

Please introduce yourself (not mandatory):



- Anonymous;
- 5 questions;
- 2 mandatory, 3 optional;
- It will help me help the group;

Prerequisites

- Basic understanding of programming constructs (variables, loops, conditionals);
- Course assumes you have no experience with OOP;
- Knowledge of a High-Level programming language is not required;

Course goals

By the end of the course you will have:

- Fundamentals of Object-Oriented Programming;
- Apply OOP in building modular systems using abstraction, and modularization;
- Practical & conceptual knowledge in multiple high-level programming languages;
- Entry experience for future courses (Design Patterns, Game development, Computer Graphics);

Lecture Goals

By the end of the lecture you will be able to:

- List some of the general benefits of OOP;
- Explain what a paradigm is, and define some popular programming paradigms;
- Understand some of the historical and cultural context of OOP;
- Navigate and progress the course materials and structure;

Object-Oriented Programming

Hmm, I heard about *Object-Oriented Programming*, but what is it really? Some sort of way of doing programming.

- **First what is programming?**

- The process of automating and managing complex systems of subproblems, through written computer instructions;
- Like making coffee: there are multiple ways to make coffee, *same goes about about programming software*;
- Programming is about *designing, organizing, and controlling complexity* so that the machine and humans both understand the system.

Object-Oriented Programming

Hmm, I heard about *Object-Oriented Programming*, but what is it really? Some sort of way of doing programming.

- **Yes, OOP is a specific way of writing software. Duh, this much I get, but why do we need multiple ways of writing software, could we agree on one, like only one way of making coffee?**
 - Making *good* coffee is a complex process, and there are many ways to do it. A machine could brew coffee, but you choose a café.
 - A skilled barista knows multiple brewing *tools and methods* and chooses the *right* one for the right bean, taste or customer. The same applies to every craft.
 - Writing software is also a craft. Different domains, scopes and problems require different *tools and methods*.
 - A skilled engineer knows which one to choose, and can apply them.

Describing software

I know how to describe my coffee, but how would you describe software? I just ask for a cappuccino and I go with my day.

- True, as a customer, you do not explain the recipe step by step, but you could tell if your coffee is bad - *sour, cold, weak, or wrong order*;
- But a question to you is how can you describe software?
 - **FAF24**: Fast/Slow, How much resource it is using, **whether it works**, User Friendly, **does it crash?**, **Cross platform**, Good Structure, Reliable, Useful, Extendible, Flexible

Describing software

Software qualities that leave a nice taste:

- **Correctness** - weather fast, or pretty - if the result is wrong other qualities do not matter;
- **Robust** - withstands abnormal situations;
- **Extendible** - easy to update and adapt to new specifications;
- ...
- **Readable**
- **Modular**
- **Structured**

Programming paradigms

Hmm... I think I know now how good software "tastes like", how can I make it?

- There are multiple ways to write software, called *programming paradigms*;
- A paradigm is the technical term for "way of going about something";

Programming paradigms

Programming paradigm - A programming paradigm is a relatively high-level way to conceptualize and structure the implementation of a computer program. A programming language can be classified as supporting one or more paradigms (Wiki).

Programming paradigms

So which is the correct paradigm to write good software? Is it OOP?

- Just as there are different ways of brewing - all aim to make good coffee.
- There is no correct paradigm, but some are better suited for certain problems;

Programming paradigms

Popular programming paradigms:

- Procedural Programming;
- Functional Programming;
- Logic Programming;
- Actor Model;
- Event-Driven/Reactive Programming;
- Object-Oriented Programming;

Programming paradigms

Procedural programming

- Based on the concept of procedure calls, in which statements are structured into procedures (also known as routines or functions);
- Procedures contain a series of computational steps to be carried out;
- Examples: C, Fortran, Pascal, BASIC;
- Still widely used, especially in embedded systems, game development, and high-performance computing;
- Hard to manage complexity in large codebases;

Programming paradigms

Functional programming

TRANSFORM



- Based on mathematical functions that use conditional expressions and recursion to perform computation;
- The idea is that of transformation: $f(x) = y$ [or: $f(a, b, c) = y$];
 - x - is the input
 - y - is the output
- Examples: Lisp, Haskell, Erlang, Scala;
- Hard to map complex domains and read.
- People are bad at math

Programming paradigms

Procedural programming

- Hard to manage above 1000 lines of code;
- No functionality to enforce boundaries between units of knowledge;
- Difficult to map complex domains;

Functional programming

- Hard to map complex domains and read;
- People are bad at math;
- Difficult to manage state and side-effects in large applications;

Cue in The legendary - Xerox PARC

3 major innovations

Which would you pick
for **biggest impact**?

- The Graphical User Interface (GUI)
- The Mouse (The Mouse)
- Object-Oriented Programming (OOP)

Object-Oriented Adoption

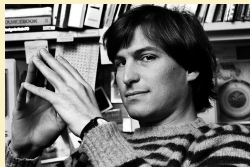
Choosing OOP



- NeXT - a computer company that made the first OOP-based OS - NeXTSTEP;
- Tim Berners-Lee created the first web browser on a NeXT machine;
- Apple acquired NeXT in 1997, and used NeXTSTEP as the basis for MacOS X;
- Today, all Apple OSes (iOS, macOS, watchOS, tvOS) are based on NeXTSTEP
=> OOP;

Object-Oriented Adoption

Choosing OOP



- NeXT - a computer company that made the first OOP-based OS - NeXTSTEP;
- Tim Berners-Lee created the first web browser on a NeXT machine;
- Apple acquired NeXT in 1997, and used NeXTSTEP as the basis for MacOS X;
- Today, all Apple OSes (iOS, macOS, watchOS, tvOS) are based on NeXTSTEP => OOP;

Object-Oriented Adoption

Choosing OOP



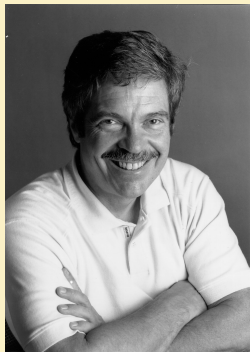
- NeXT - a computer company that made the first OOP-based OS - NeXTSTEP;
- Tim Berners-Lee created the first web browser on a NeXT machine;
- Apple acquired NeXT in 1997, and used NeXTSTEP as the basis for MacOS X;
- Today, all Apple OSes (iOS, macOS, watchOS, tvOS) are based on NeXTSTEP => OOP;

Object-Oriented Adoption

“In my 20 years in this industry, I have never seen a revolution as profound as this.”

“You can build software literally five to ten times faster, and that software is much more reliable, much easier to maintain and much more powerful.” — Steve Jobs, Rolling Stone, 1994

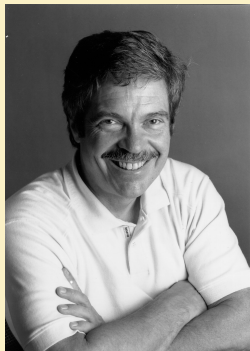
Object-Oriented Adoption



Alan Kay

- Coined the term "Object-Oriented Programming" in 1966;
- Often quoted as "Father of OOP", though he dislikes the term OOP;
- He does not attribute the invention of OOP to himself;

Alan Kay



- Created smalltalk, the first "true" OOP language;
- He envisioned OOP as a way to model biological systems;
- He believes that OOP has been misused and misunderstood, "OOP is not about objects, it is about messaging";
- He had a huge impact on the development of personal computing, GUIs, and educational technology;



Alan Kay

- Dynabook concept - a personal, portable computer for learning and creativity for kids to learn;
- Influenced the development of laptops, tablets, and educational software;
- His vision of computing as a medium for human expression and learning continues to inspire technologists and educators today;

Object-Oriented Adoption

“The best way to predict the future is to invent it”

— Alan Kay

Who Invented What?

- **Simula (1967-1968, Norway)** — first language with classes and objects (Ole-Johan Dahl Kristen Nygaard)
- **Smalltalk (1970s, Xerox PARC)** — Alan Kay, Dan Ingalls, Adele Goldberg; first “true OOP environment” with messaging, inheritance, and dynamic objects
- **C++ (1980s, USA)** — Bjarne Stroustrup; combined OOP with systems programming
- **Objective-C (1980s, USA)** — Brad Cox Tom Love; added messaging to C
- **NeXT / Apple adoption** — Steve Jobs popularized OOP concepts for commercial software with NeXTStep

Object-Oriented Programming

What is Object-Oriented Programming?

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). —

Wikipedia

Object-Oriented Programming

What is Object-Oriented Programming?

OOP is a way of organizing code that emphasizes the use of objects to represent real-world entities and their interactions.

The 4 ideas that resulted in OOP:

- **Structuring method (1)**
- **Enforce Reliability (2)**
- **Epistemological Principle (3)**
- **A Classification Technique (4)**

Object-Oriented Programming

The 4 ideas that resulted in OOP:

- **Structuring method (1)**

- How can we decompose software?
- How can we reuse existing code?

- **2. Enforce Reliability (2)**

- A small modular component is easier to maintain
- Treat the system as a collection of these reliable small components
- Each component has clear responsibility and benefits for the other components

- **Classes and Objects**

Object-Oriented Programming

The 4 ideas that resulted in OOP:

- **Epistemological Principle (3)**

- Nature of knowledge or knowledge modelling
- Logic that is used to model knowledge
- This resulted in the concept of **Abstraction**

- **A Classification Technique (4)**

- Grouping components based on shared characteristics
- Changing and replicating behaviour based on the context
- Reusing behaviour through composition or inheritance
- This resulted in the concept of Polymorphism and Inheritance

- **Inheritance** - mechanism for creating new classes based on existing ones;

- **Polymorphism** - ability to present the same interface for different underlying data types.

Object-Oriented Programming

The class and the object

- **Class** - a blueprint or template for creating objects, defining their properties (attributes) and behaviors (methods);
- **Object** - an instance of a class, representing a specific entity with its own state and behavior;
- Classes encapsulate data and methods, providing a structured way to model real-world entities and their interactions;
- Objects interact with each other through methods, enabling communication and collaboration within the system.

Object-Oriented Programming

The class and the object



The 4 building blocks of OOP

The 4 ideas resulted in the 4 building blocks of OOP:

- **Encapsulation** - data and methods that operate on that data within a single unit or class, with defined boundaries;
- **Abstraction** - simplifying complexity and hiding details through simple interfaces;
- **Inheritance** - mechanism for creating new classes based on existing ones (Often a bad idea);
- **Polymorphism** - ability to present the same interface in different contexts with different outcomes.

What is next?

- Course structure and materials;
- Grading and evaluation;
- Tools and setup;

Any questions so far?

Course Structure and Materials

- All the materials will be available on the ELSE (Moodle) page for OOP;
- Code examples will be on the course Github repository;
- Expect a Quiz for every lecture - not a promise, but a possibility;
- Chance to earn extra points for active participation and contributing;
- **Contributing will be rewarded;**

Grading and Evaluation

Individual Work, Lecture Grade, Midterm 1, Midterm 2, Exam, Course Grade, Final Grade:

- **Individual Work** - Your laboratories assignments;
- **Lecture Grade** - Quizzes, participation, attendance;
- **Midterm 1 & 2** - Laboratory Work Avg + Midterm Quiz Grade;
- **Exam** - Final written exam;
- **Course Grade** - $\text{avg}(\text{IW}, \text{LG}, \text{M1}, \text{M2})$;
- **Final Grade** - $.4 * \text{Exam} + .6 * \text{Course Grade}$;

Tools and Setup

You will need:

- A computer capable of running a modern IDE (IntelliJ IDEA, VSCode, etc.);
- Git for version control;
- A GitHub account for accessing course materials and submitting assignments;
- Google Docs/MS Word for writing your report on labs;

Tools and Setup

Allowed languages

- Java - The OOP language
- Kotlin - modern OOP language, rich in features and a pleasure to write
- C++ - THE programming language. You can write low-level or high-level code, meaning you can do everything.
- Python - Less OOP features supported. *So learn to use the tools that makes it OOP, and conventions for private variables, and typization
- C - The Microsoft OOP language
- TypeScript - JS with types, and OOP features (also Microsoft)

Tools and Setup

Personal recommendation

- Do not use Rust/Go for OOP;
- If you are familiar with an OOP language, try a new one;
- Stick to one language for the whole course;

Special Assignment

For a small bonus:



- Contribute: improve, enhance, OOP-ify;
- Present changes, articulating differences, improvements;
- Add features, and own hooks - present how OOP was used;
- *How would you go about doing it in Procedural language (optional, but very welcome - try it I dare you);

Special Assignment

For a small bonus

- **Codeskulptor:** <https://py3.codeskulptor.org/>
- **Initial code:** ask for repository rights from Dom
- **Flocking and Boids paper:**
<http://www.red3d.com/cwr/boids/>

Object-Oriented Programming



Or click me! (Not a scam)