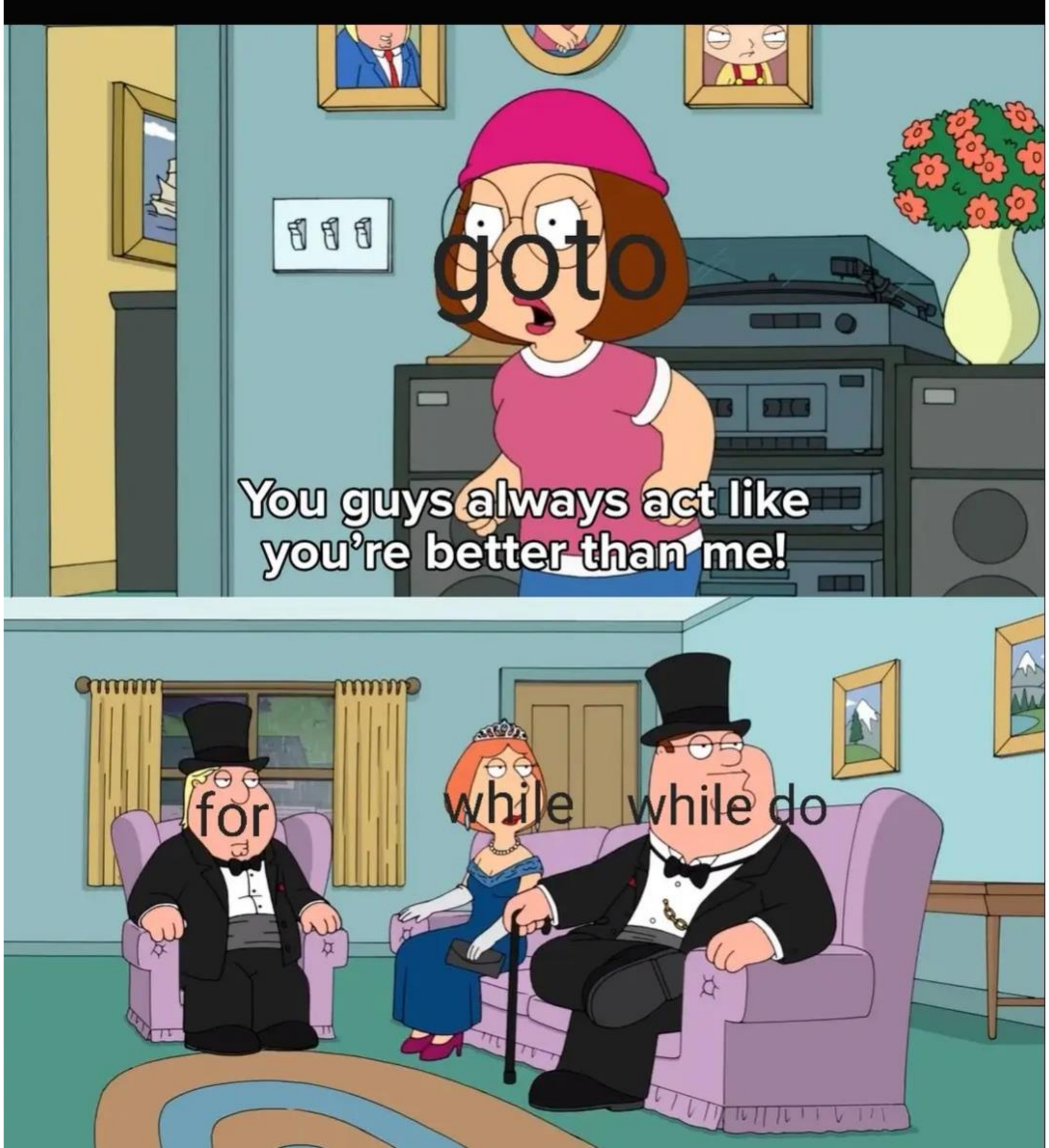
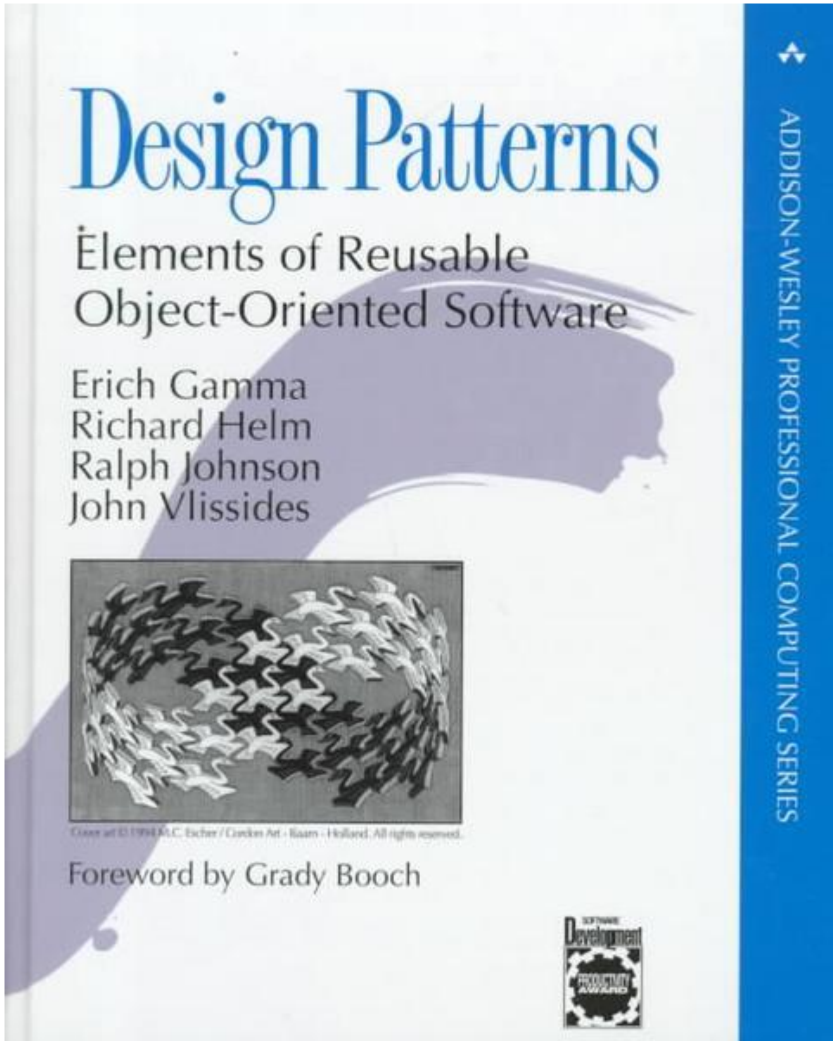
Design patterns

Paolo Burgio
paolo.burgio@unimore.it


goto
You guys always act like you're better than me!
for
while
while do


Design Patterns

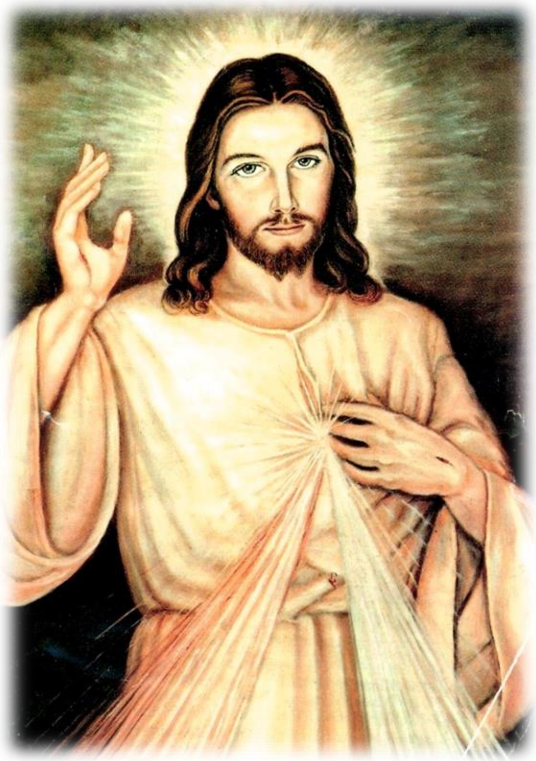
Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



The Gang of Four

Elements of reusable Object Oriented Software

Elements

of reusable

Object Oriented

Software

Elements of reusable Object Oriented Software

Elements

- Simple, basic parts of

of reusable

- We did mistake, we learned from them

Object Oriented

- Years of mistakes

Software

- ....

As simple as that

Your parents, granparents, teachers, ancestors faced problems

They found solutions

› ..smart solutions...

This is their (our) legacy

› Hundreds of know problems, with known solutions
› All of them build upon basic principles
› Sync/vs async, de-coupling, SOLID, etc

Ok, let's be clear

What design patter can give you

- A common, known vocabulary
- Solve complex problems way ahead of time
- Provide solid ground to motivate your design choices

What they cannot give you

- Exact solution: each problem/project is unique
- Full-fledged solution for every design/programming problem

But they can save you a lot of headaches!

The typical structure

1. Purpose
2. Motivation (why the hell should I do so?)
3. Applicability (where it applies, and where it doesn't)

=> What to do

(Personal note: even if you don't know why...use them!)

A full set of example/code snippets to implement it

› With known examples
› With related patterns (everything is part of a bigger picture!)

The bad news

› I will only teach you 3-4 four of them
› Advanced (...?) courses can give you a full
› BUY-THE-***-BOOK/COURSES

(Incomplete) taxonomy of design patterns

Creational

- **Factory**
- **Singleton**
- Builder
- Prototype

Structural

- **Adapter**
- Bridge
- Composite
- Façade
- Proxy
- Decorator
- FlyWeight

Behavioral

- Chain of Responsibility
- *Command*
- *Iterator*
- Interpreter
- *Mediator*
- Memento
- Observer
- State
- Strategy
- Template Method
- *Visitor*

Singleton

The singleton pattern

Purpose

- Make sure that there is only one instance (object) of a class active in the whole system

Motivation

- You might need to abstract single resources (e.g., printing queues, DBMS, ...)
- The class itself shall be responsible to instantiate the singleton
- No other instance (i.e., object of the same class) shall exist

Applicability

- When you need a single point of access to an instance of a class

Singleton pattern

Let's code!

- I won't give you any practical example
- Just, make sure it is possible to instantiate only only one object of any class

What are the challenges?

- Hint: think of multi-thread programs
- What “entity” does this model?
- Which data structures would you need to support this paradigm?

Factory

Adapter

References

Course website

- http://hipert.unimore.it/people/paolob/pub/ProgSW/index.html

Course website

- Gamma, et.al «Design Patterns – Elements of reusable Object Oriented Software», Addison Wesley

My contacts

- paolo.burgio@unimore.it
- http://hipert.mat.unimore.it/people/paolob/
- https://github.com/pburgio