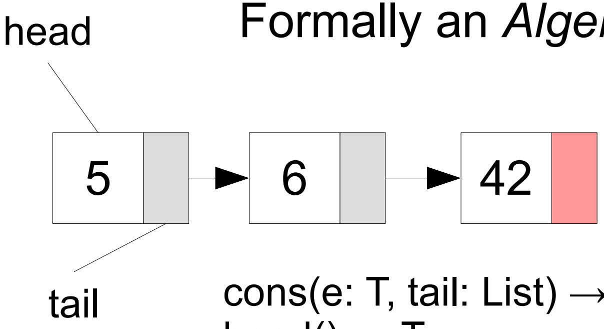
Object-Oriented Design

- Class-Based OOP
  - Classes as Abstract Data Types and Domain Models
  - Encapsulation, Inheritance and Polymorphism
- Iterative Design Process
- CRC Cards (Class-Responsibility-Collaboration)
- UML 101
  - Class Diagrams. Association, Aggregation, Composition
  - Sequence Diagrams and State Machines
- OO Design Principles
  - SOLID (esp. LSP)
  - Low Coupling, High Cohesion
  - Fun Acronyms: KISS, DRY, YAGNI…

Classes and Objects

Abstract Data Types
(Data + Operations)
Formally an *Algebra*



cons(e: T, tail: List) → List(e: T, tail)
head() → T
tail() → List
prepend(lst: List, e: T) → List
concat(a: List, b: List) → List

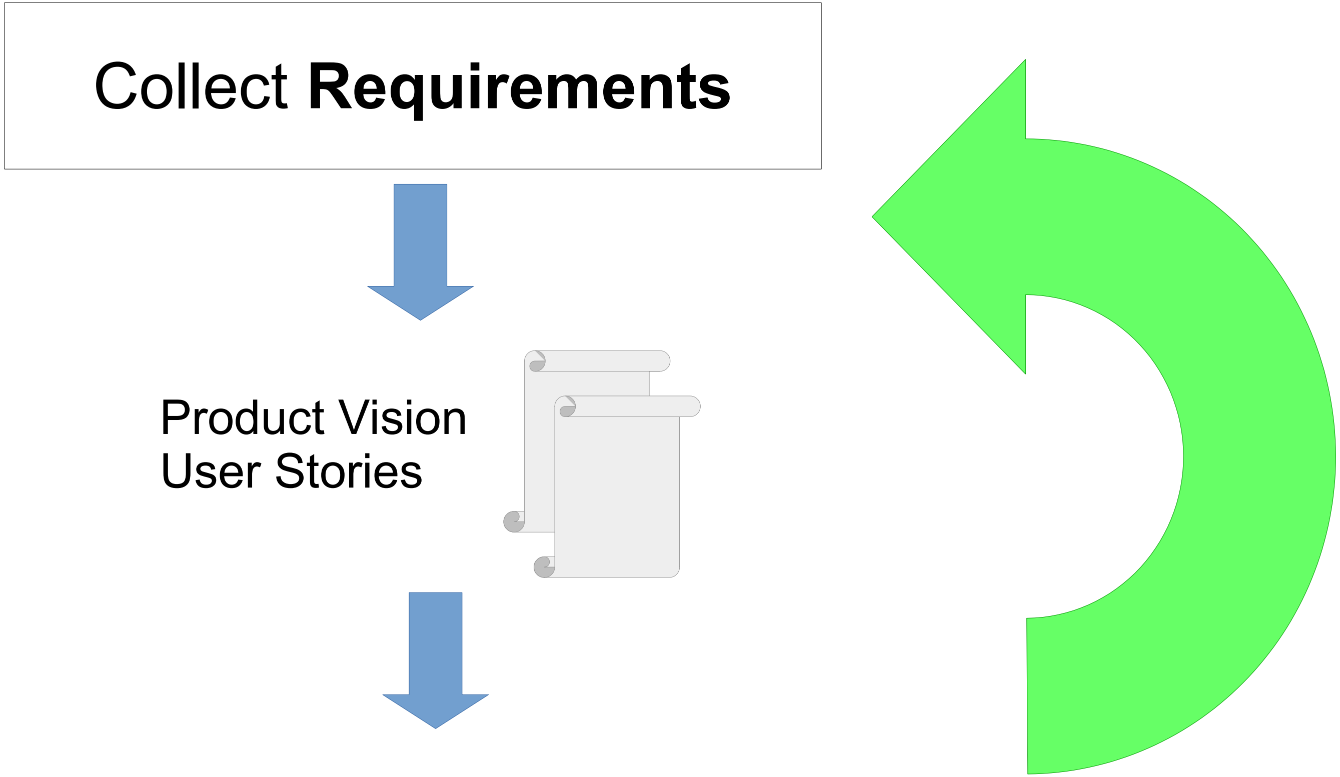
Model of the Real World™
Metamodel: a Model of a Model

Ceci n'est pas une pipe.
Magritte

Class-Based OOP (Java, C#, C++)

- **Encapsulation** (*aka Data Hiding*)
  - *@see* Visibility Modifiers (`public`, `protected`, `private`, package-private)
- **Inheritance**
- **Polymorphism**
  - That is, **Subtype** Polymorphism (*cf.* Generics: **Parametric** Polymorphism)
  - Virtual dispatch, specifically Single Dispatch
- Contrast other OOP styles:
  - Prototype-Based OOP (JavaScript, Self)
  - Message Passing (Obj-C, Lua, Smalltalk)
  - Dynamic Class-Based (Python, Ruby, Perl, Common Lisp)

Iterative Design Process (1)



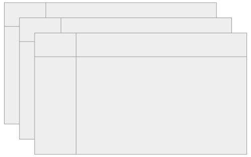
Collect **Requirements**

Product Vision
User Stories

Iterative Design Process (2)

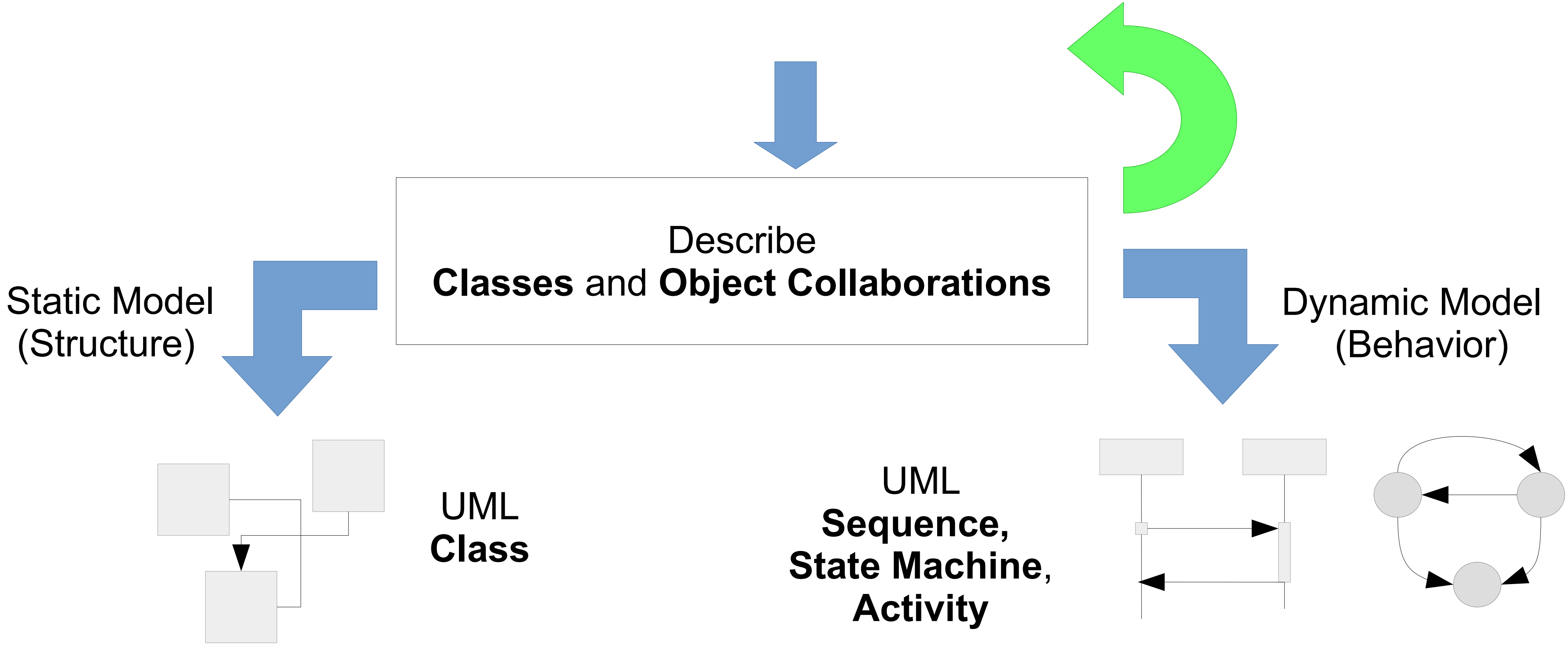
Identify Classes and Objects

CRC Cards
(Class-Responsibility-Collaboration)



Nouns → **Classes**
Verbs → **Responsibilities**
(Class × Class) → **Collaborations**

Iterative Design Process (3)



Describe
Classes and **Object Collaborations**

Static Model
(Structure)

Dynamic Model
(Behavior)

UML
Class

UML
Sequence,
State Machine,
Activity

CRC Cards

(front)

Class: Resource Pool

Responsibilities (*i.e.* Public Methods):	**Collaborators**:
Borrow Resource	Allocator, Resource
Return Resource	Allocator, Resource
Print Statistics	–

(back)

Attributes:

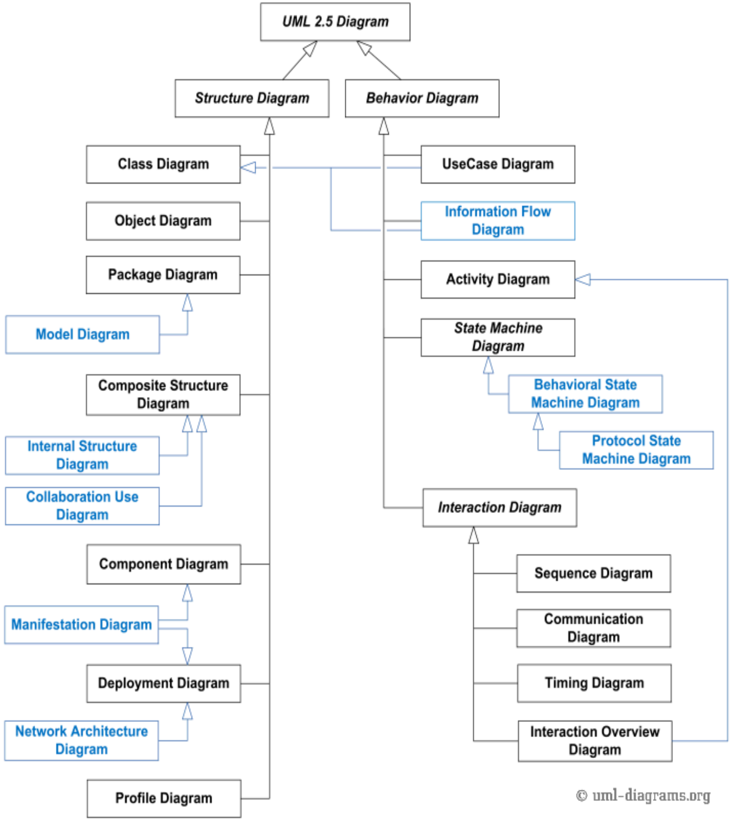
- LRU Queue
- Max Resource Count

Resource Pool allocates expensive **Resources** and keeps them for a while, to amortize resource creation cost. Resource Pool might pre-allocate resources. Borrowing from the pool returns an LRU resource.

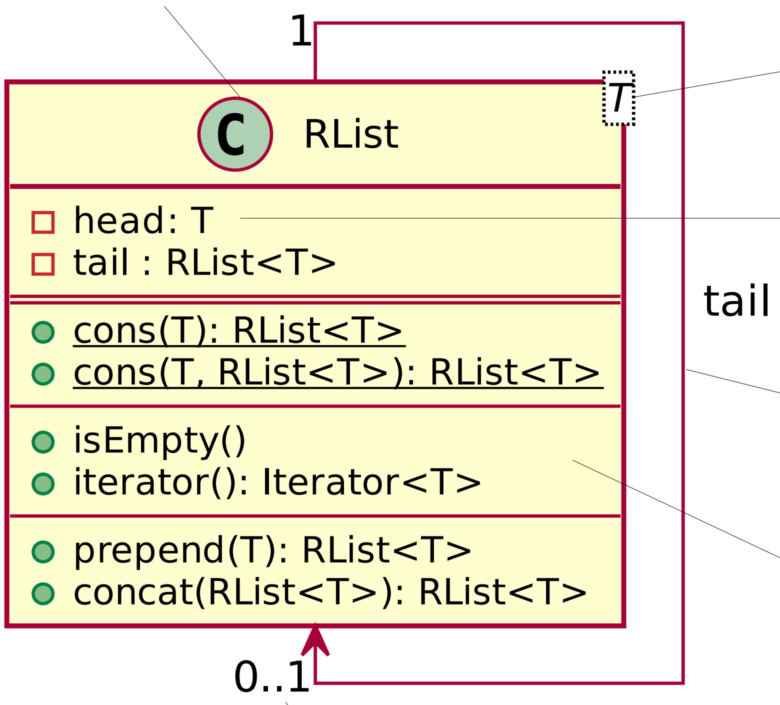
UML

https://www.uml-diagrams.org

- Structure (=Static)
  - **Class**
  - Package
  - Component, Deployment
  - Object, Collaboration Use
- Behavior (=Dynamic)
  - Use Case
  - **Sequence**
  - **State Machine**
  - Activity



UML Class Diagram: Members



This is a Class!

Generics

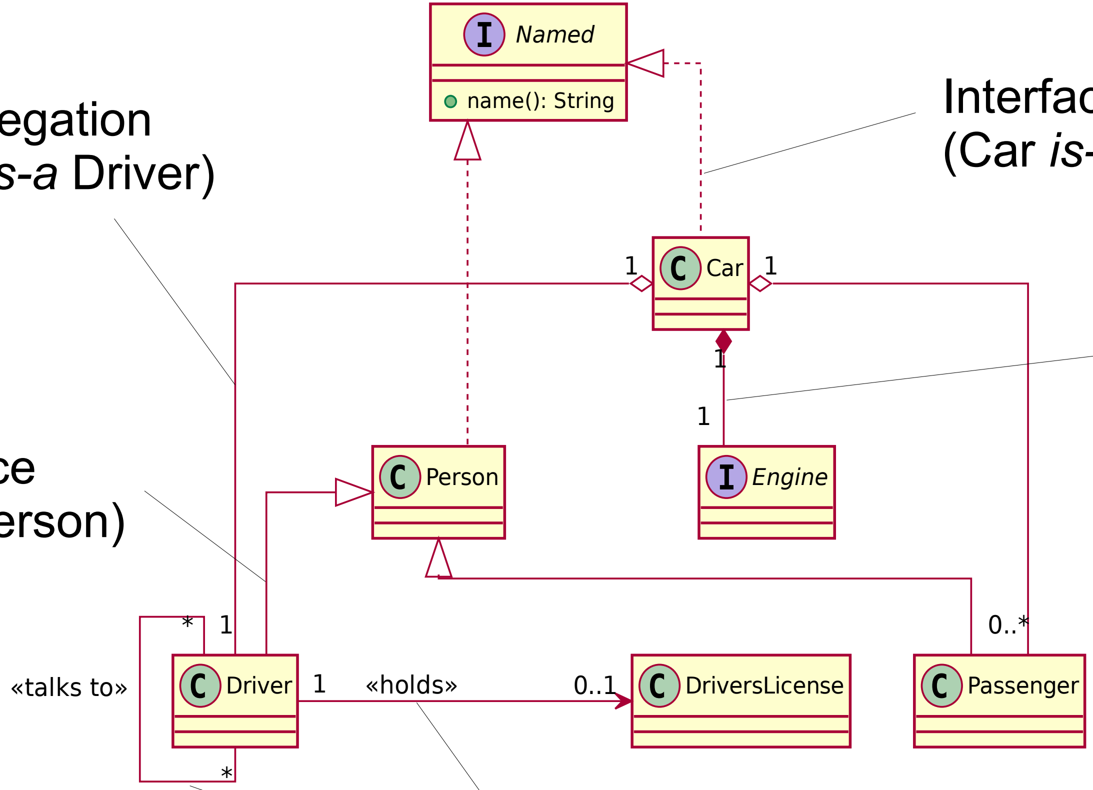
Private Fields:
List is represented recursively as list node **RList**(*head*, *tail*) where *tail* is also an **RList**

Relationship:
One list node can have either no tail, or a single tail

Public Methods
<u>static</u>
abstract

Cardinality

UML Class Diagram: Relationships



Aggregation
(Car *has-a* Driver)

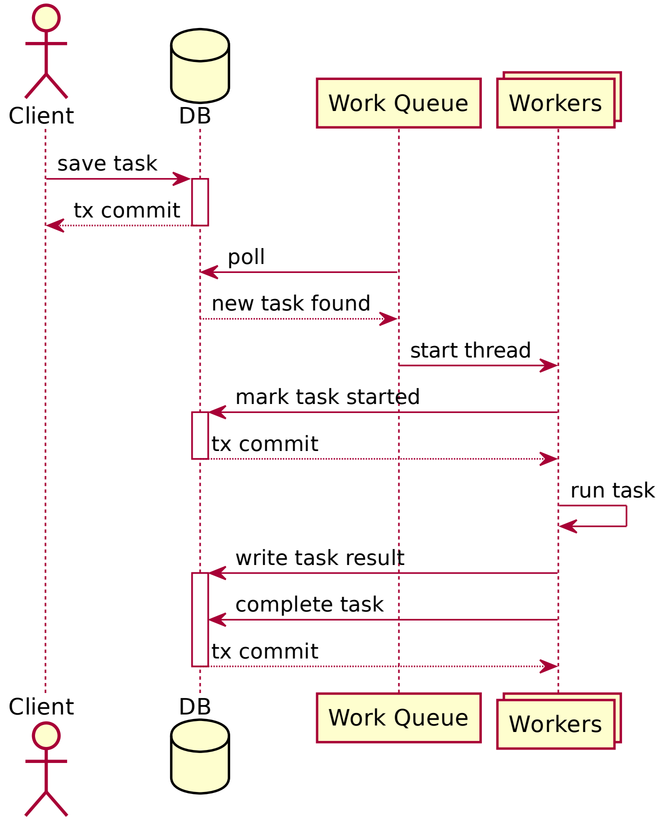
Interface Implementation
(Car *is-a* Named entity)

Composition
(Car *has-a* Engine)

Inheritance
(Driver *is-a* Person)

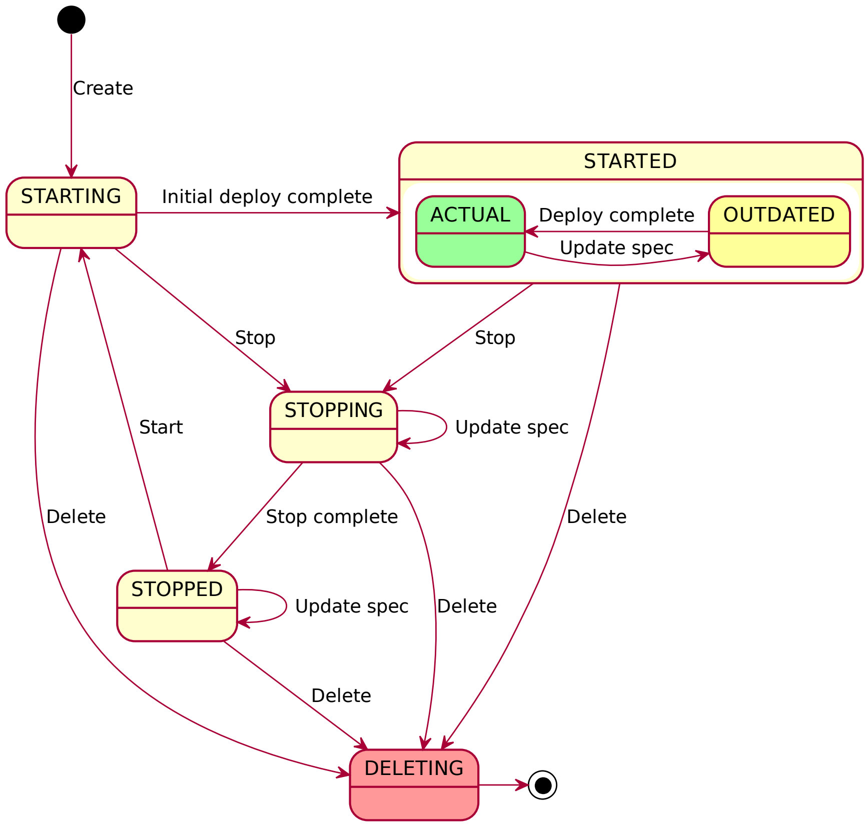
Association

UML Sequence Diagram



- **Client** *saves* a Task to **DB**
- **Work Queue** *polls* **DB**
- When a *new task is found*, **Work Queue** starts a **Worker**
- **Workers** on multiple hosts *race to mark task started* in the DB
- The winning **Worker** *runs* the task
- When the task is *completed*, **Worker** *writes result* to the DB and *marks the task* complete

UML State Machine



- When Instance is *Created*, it is put into **STARTING** state
- When the *initial deploy is complete*, Instance becomes **STARTED**…
- A **STARTED** Instance can be either **ACTUAL** or **OUTDATED**
  - **STARTED/OUTDATED** Instances become **STARTED/ACTUAL** when spec changes are applied to them
- etc.

SOLID OO Design Principles

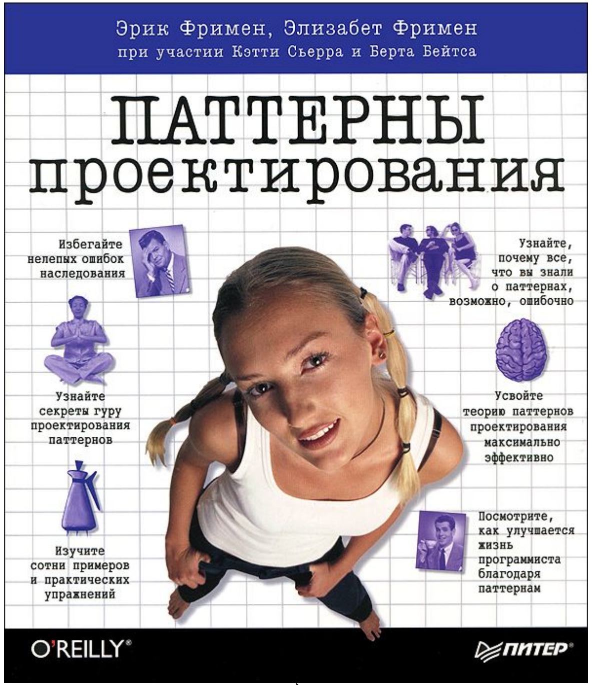
- **S**ingle Responsibility
  - Class must have a Single reason for Change
  - *E.g.,* in Logging frameworks: separate *Logger* vs *Appender* vs *Layout*
- **O**pen-Closed
  - Open for Extension (well-defined extension points)
  - Closed for Modification (well-defined public interface)
- **L**iskov Substitution Principle
  - Subtypes must be *substitutable* for supertypes without altering *program correctness*
  - *Class invariants* and method *pre-* and *postconditions*
- **I**nterface Segregation: Smaller client-specific interfaces vs God Object. *Also*: API & SPI
- **D**ependency Inversion: Depend upon abstractions, not concrete classes

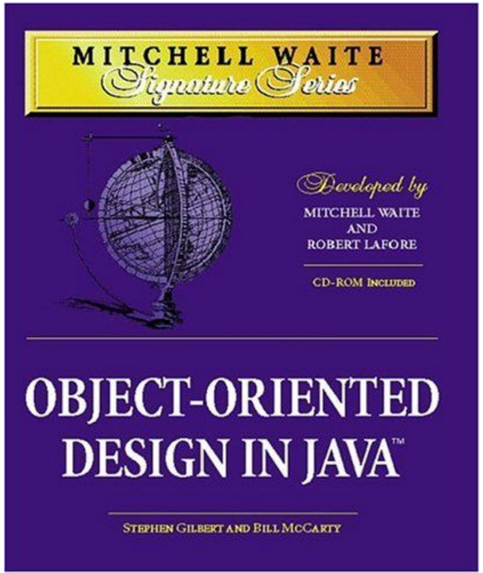
OO Design Principles (*Contd.*)

- Low Coupling + High Cohesion (from GRASP Patterns)
- Prefer Composition to Inheritance
  - …and Interface Inheritance to Implementation Inheritance
- API Design: Design for both Extension and Backward Compat
- DRY (Do not Repeat Yourself)
- YAGNI (Agile vs BDUF, Big Design Up Front)
- KISS (Keep it Simple Stupid)

Recommended Reading

С. Макконнелл
МАСТЕР КЛАСС
СОВЕРШЕННЫЙ КОД
ПРАКТИЧЕСКОЕ РУКОВОДСТВО ПО РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
bhv
Microsoft


Эрик Фримен, Элизабет Фримен
при участии Кэтти Сьерра и Берта Бейтса
ПАТТЕРНЫ проектирования
Избегайте нелепых ошибок наследования
Узнайте, почему все, что вы знали о паттернах, возможно, ошибочно
Узнайте секреты гуру проектирования паттернов
Усвойте теорию паттернов проектирования максимально эффективно
Изучите сотни примеров и практических упражнений
Посмотрите, как улучшается жизнь программиста благодаря паттернам
O'REILLY
ПИТЕР


MITCHELL WAITE Signature Series
Developed by MITCHELL WAITE AND ROBERT LAFORE
CD-ROM Included
OBJECT-ORIENTED DESIGN IN JAVA™
STEPHEN GILBERT AND BILL MCCARTY

Chapters 5..8

For the Next Seminar

Recommended Reading (*Contd.*)

- *Head First Patterns* by Eric & Elizabeth Freeman
- *Code Complete* by Steve McConnell
- *Object-Oriented Design in Java*
  by Gilbert & McCarty (https://www.amazon.com/MWSS-Object-Oriented-Design-Mitchell-Signature/dp/1571691340)
  @*see* Chapters 5..8