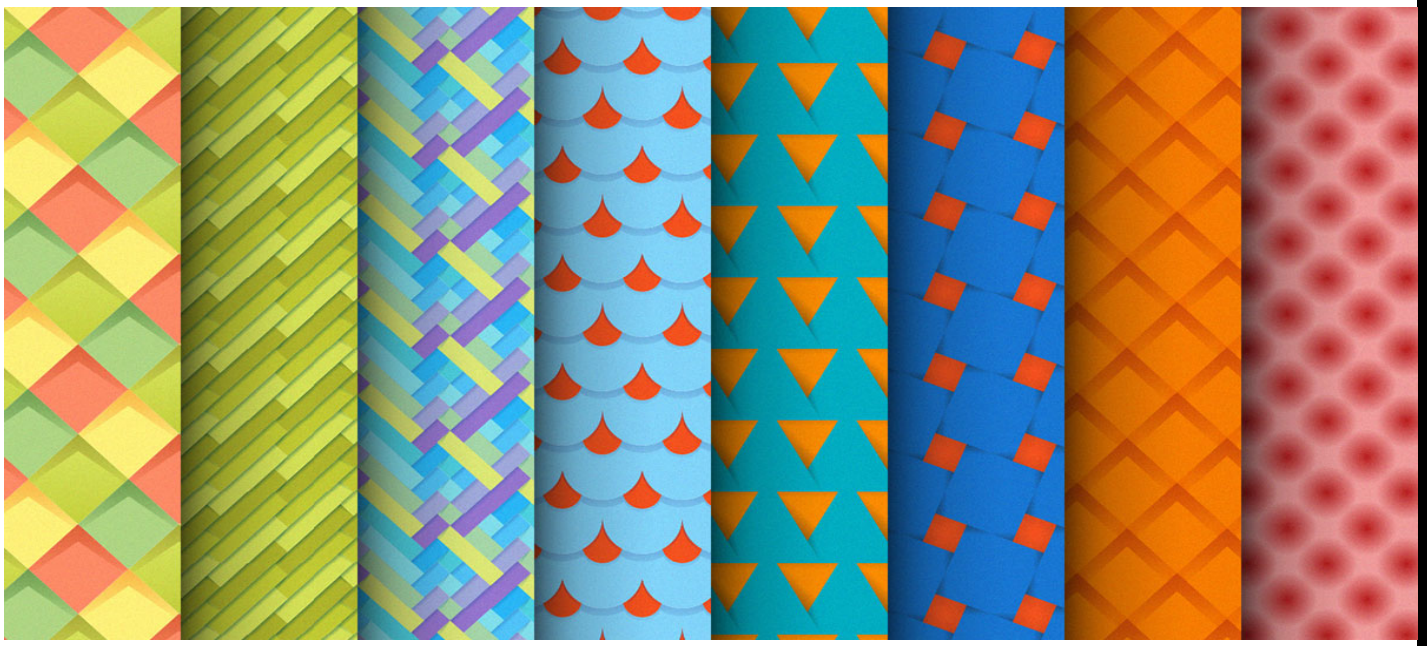


DESIGN PATTERNS

COURSE 12

PREVIOUS COURSE

- **Refactoring**
  - Way refactoring
  - Some refactoring examples

CURRENT COURSE

- **Anti – patterns**
  - The blob
  - Poltergeist
  - Golder Hamer
  - Spagetty

ANTI-PATTERNS

- **Pattern: good ideas**
- **Refactoring: better ideas**
- **Anti-Patterns: bad ideas**
  - A literary form that describes a commonly occurring solution to a problem that generates decidedly negative consequences.
  - May be the result of a manager or developer not knowing any better, not having sufficient knowledge or experience in solving a particular type of problem, or having applied a perfectly good pattern in the wrong context.

ANTI-PATTERNS

- Anti-pattern is a pattern that may commonly used but is ineffective and/or counterproductive in practice
- Provide a method of efficiently mapping a general situation to a specific class of solutions
- Provide real world experience in recognizing recurring problems in the software industry
- Provide a common vocabulary for identifying problems and discussing solutions.

ANTI-PATTERNS

- **Software Refactoring**
  - A form of code modification, used to improve the software structure in support of subsequent extension and long-term maintenance.
- **AntiPatterns**
  - Define a migration (or refactoring) from negative solutions to positive solutions.
  - Not only do they point out trouble, but they also tell you how to get out it.

ANTI-PATTERNS. TYPES

- **Software development**
  - Technical problems and solutions encountered by programmers
- **Architectural**
  - Identify and resolve common problems in how systems are structured.
- **Software project management**
  - Address common problems in software processes and development organizations.

CAUSES. ANTI-PATTRENS

- **Haste**
  - Aggressive project deadlines and budget
  - Lower acceptance levels for code quality
  - Insufficient testing
  - Patches
  - Accumulating technical debt

CAUSES. ANTI-PATTRENS

- **Apathy**
  - Unwilling to find the proper solution
  - General lack of concern or care about solving a problem

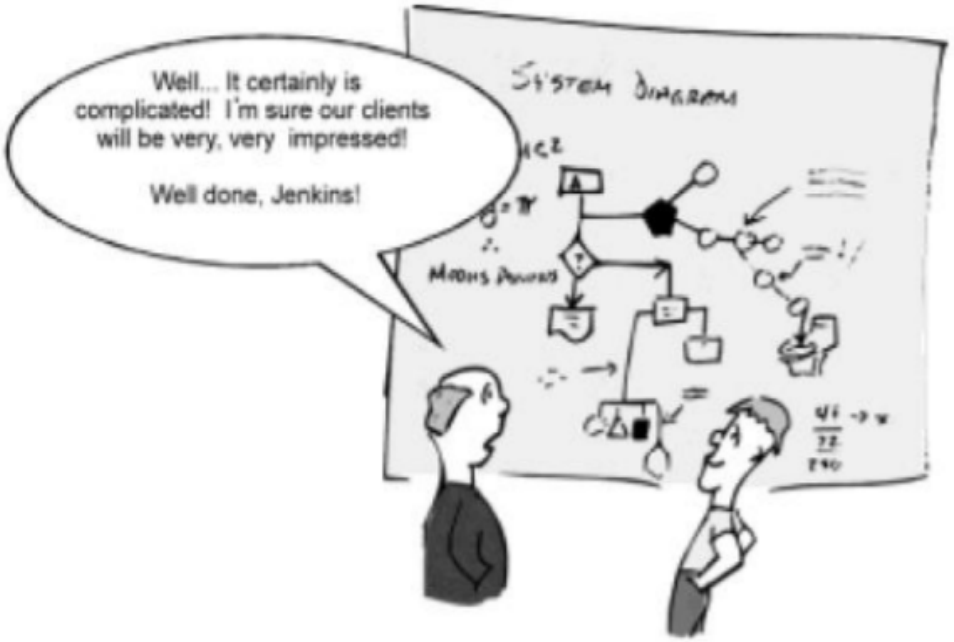
CAUSES. ANTI-PATTRENS

- **Narrow mindedness**
  - Refusal to practice solutions that are otherwise wildly known to be effective

CAUSES. ANTI-PATTRENS

- **Sloth**
  - Poor decisions based upon an “easy answer”

CAUSES. ANTI-PATTRENS



- **Avarice**
  - Modeling of excessive/insufficient abstraction adding accidental complexity

CAUSES. ANTI-PATTRENS

- **Ignorance**
  - Failure to seek a clear understanding of a problem or solution space (both intentional ad non-intentional)

CAUSES. ANTI-PATTRENS

- **Pride**
  - The sin of pride is the Not-Invented-Here syndrome

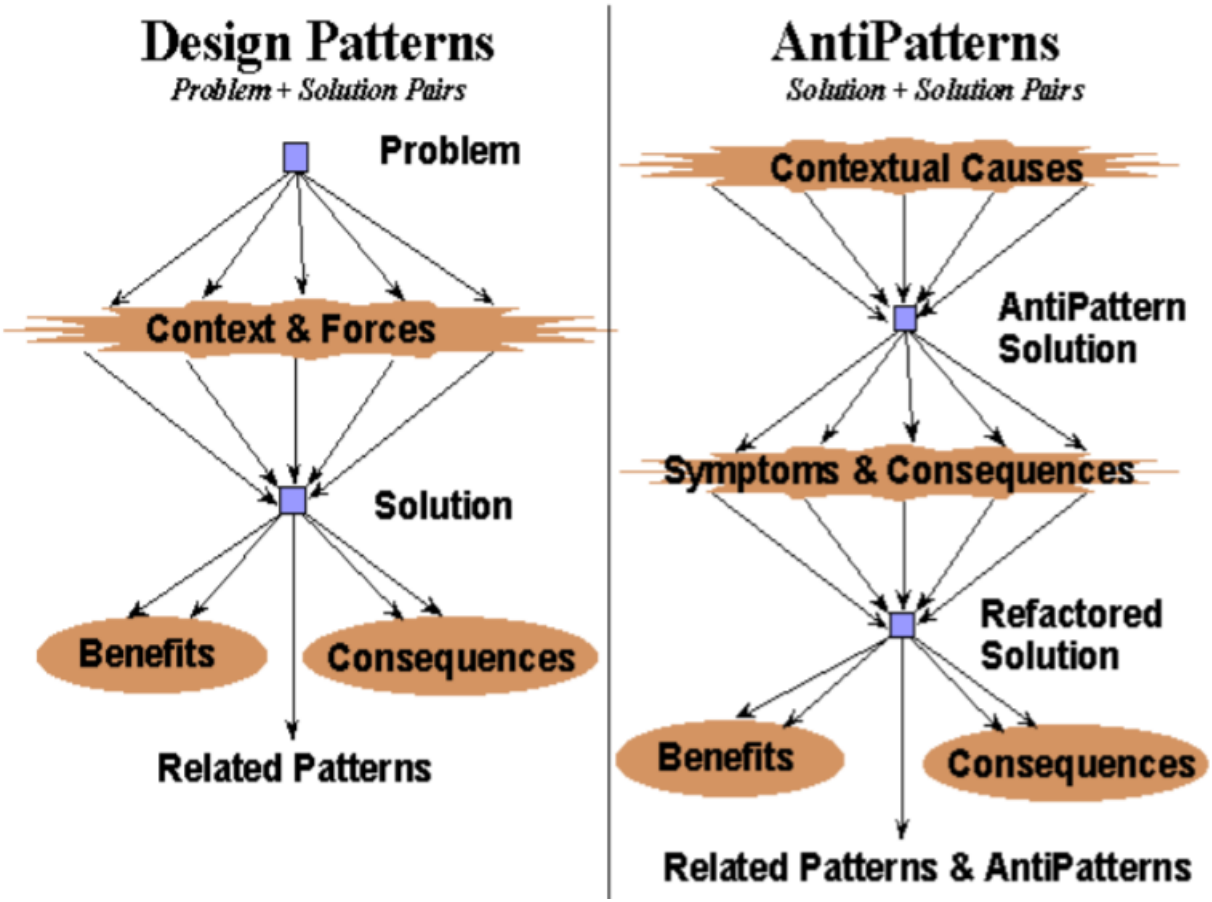
SYMPTOMS. ANTI-PATTRENS

- **Quick demonstration code integrated in the running system**
- **Obsolete or scanty documentation**
- **50% time spent learning what the code does**
- **“Hesitant programmer syndrome”**
- **Perhaps easier to rewrite this code**
- **More likely to break it then extend it**
- **Cannot be reused**
  - Cannot change the used library/components
  - Cannot optimize performance
- **Duplication**
  - “I don’t know what that piece of code was doing, so I rewrote what I thought should happen, but I cannot remove the redundant code because it breaks the system.”

SYMPTOMS IN OO PROGRAMMING

- **Many OO method with no parameters**
- **Suspicious class or global variable**
- **Strange relationships between classes**
- **Process-oriented methods**
  - Objects with process-oriented names
  - OO advantage lost
- **Inheritance cannot be used to extend**
  - Polymorphism cannot be used

DESIGN PATTERNS AND ANTI-PATTERNS



ANTI-PATTERNS. TYPES

- **Software development**
  - **Technical problems and solutions encountered by programmers**

- **Architectural**
  - Identify and resolve common problems in how systems are structured.

- **Software project management**
  - Address common problems in software processes and development organizations.

SOFTWARE BLOAT

- **Successive versions of a system demand more and more resources**
- **Reason**
  - Increase proportion of unnecessary features
- **Results**
  - Program use more system resources than necessary, while offering little or no benefit to its users
- **Solution**
  - Use plug-ins, extensions or add-ons
  - Use Unix philosophy: “write programs that do one thing and do it well

PATTERNS FETISH

- **Unreasonable and excessive use of design patterns**
- **Designers looks for places to use patterns**
- **Solution**
  - Look at the design problem
  - Favor simple solutions

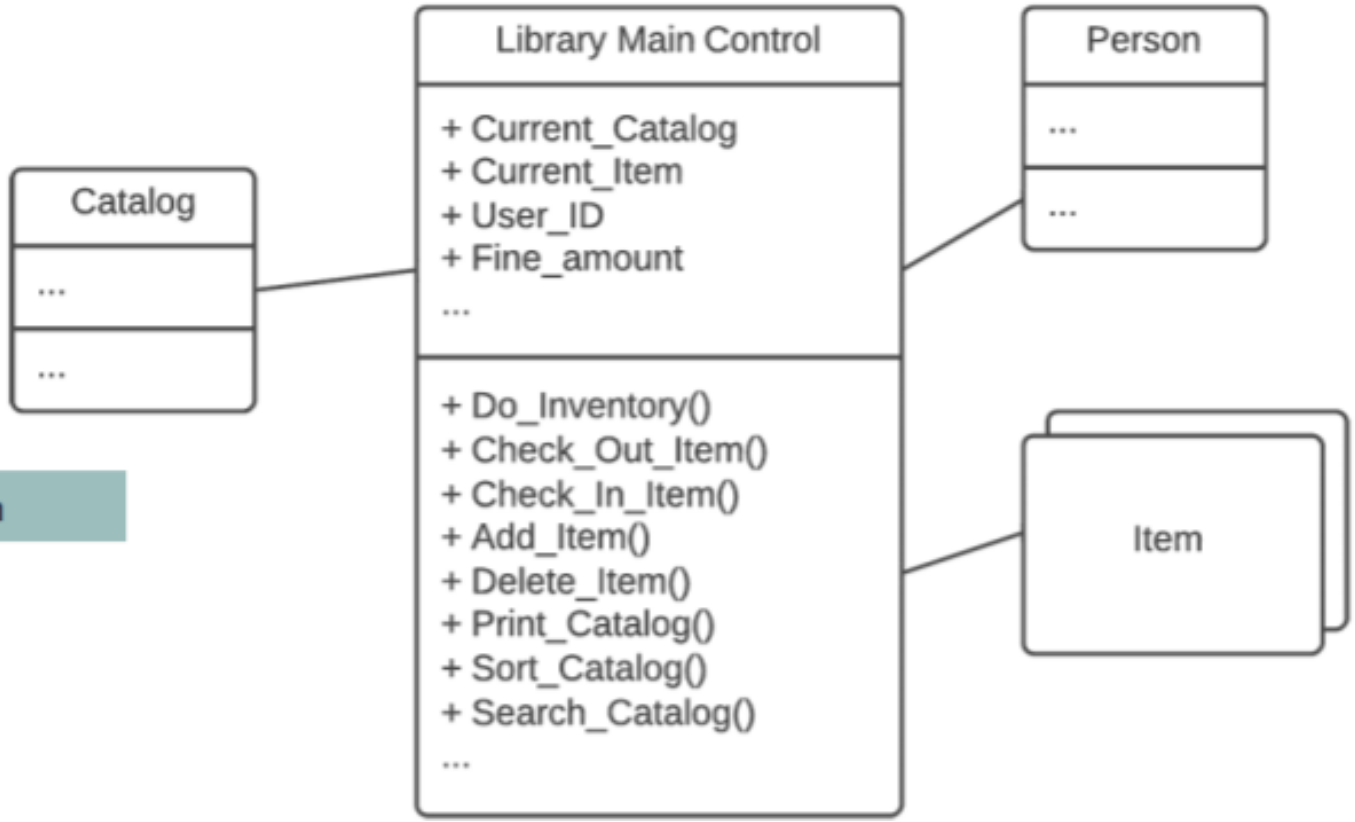
THE BLOB

- **Symptoms**
  - Single class with many attributes and operations
  - Controller class with simple, data-object classes
  - Lack of OO design
  - A migrated legacy design
- **Consequences**
  - Lost of OO advantages
  - Too complex to reuse or test
  - Expensive to load in memory
    - Way?

THE BLOB

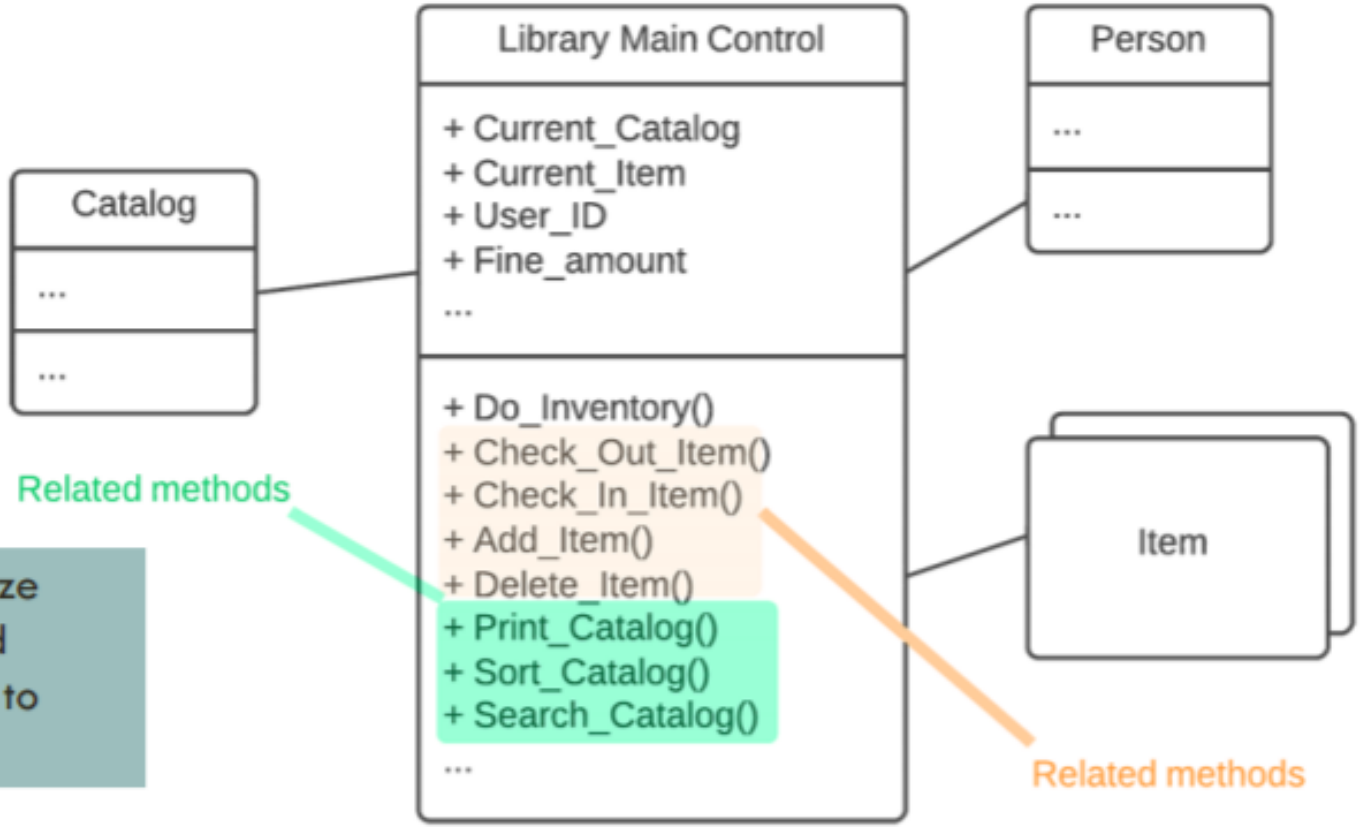
- **Solution**
  - Identify or categorize related things
    - Attributes, Operations
  - Where do these categories naturally belong?
    - Apply move method, move field refactorings
  - Remove redundant associations

THE BLOB



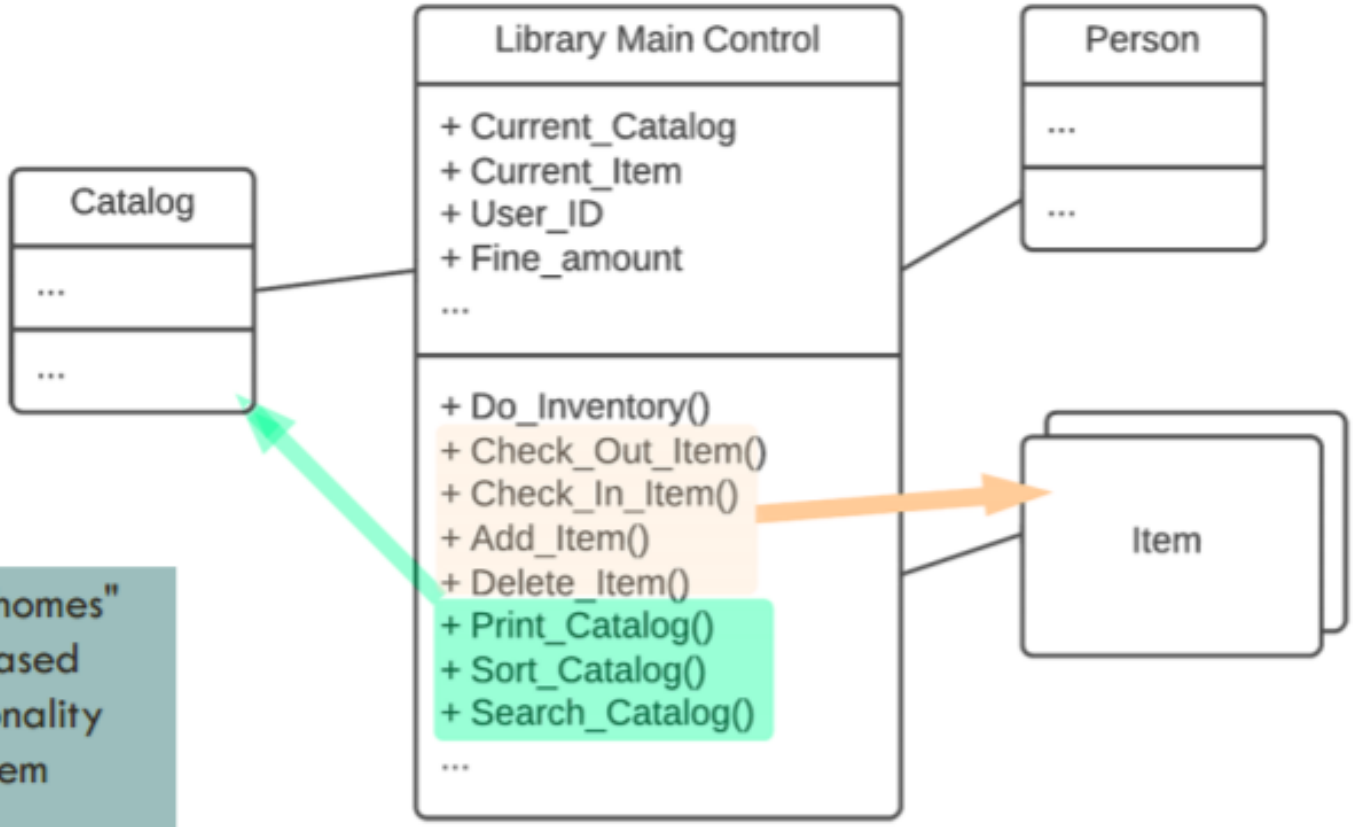
The library application

THE BLOB



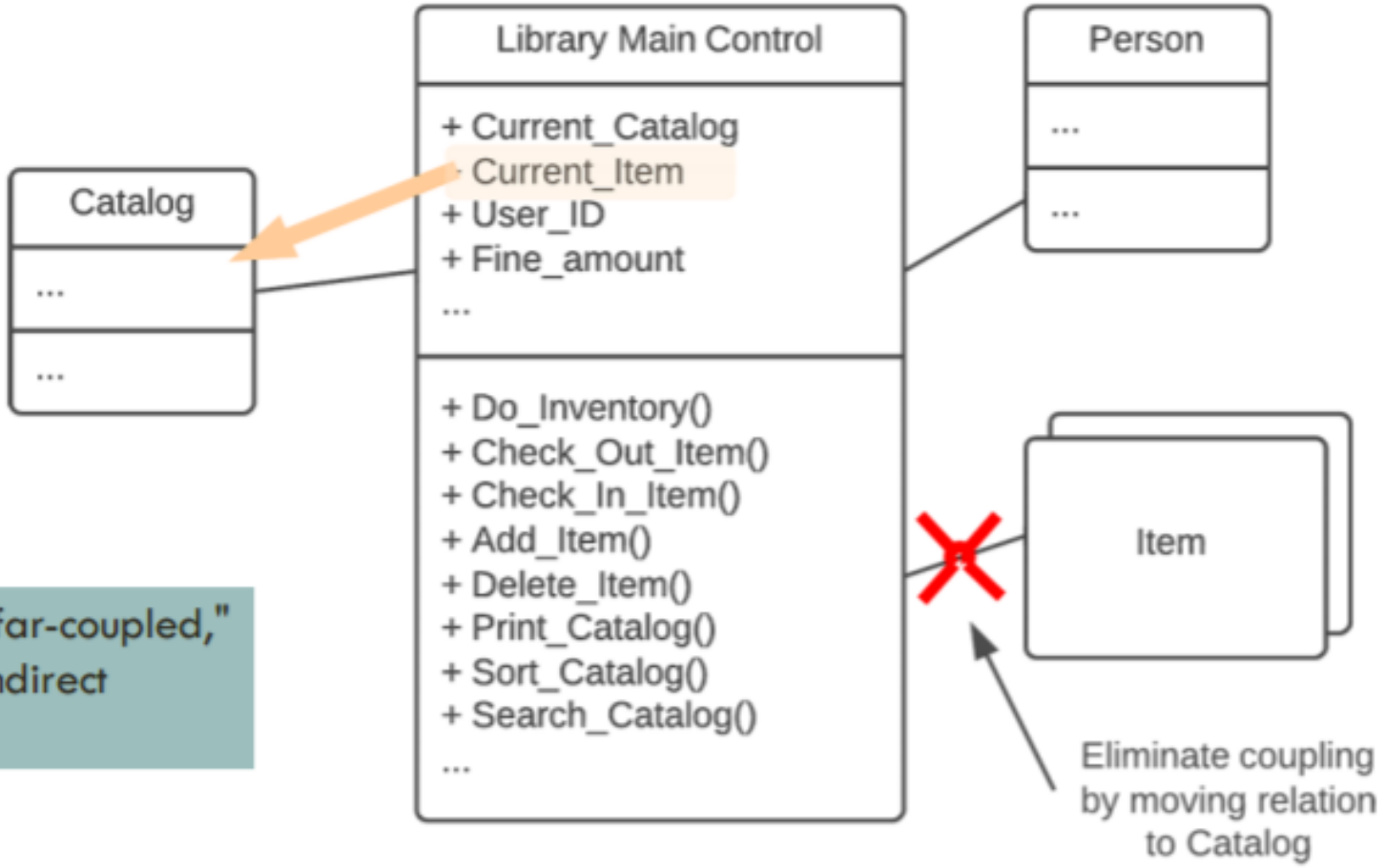
1.Identify or categorize related attributes and operations according to contracts.

THE BLOB



2.Look for "natural homes" for these contract-based collections of functionality and then migrate them there.

THE BLOB



3. Remove all "far-coupled," or redundant, indirect associations

POLTERGEISTS

- **Also Known As: Gypsy, Proliferation of Classes, Big Dolt Controller Class**

- **Symptoms**
  - Small Classes with very limited responsibilities and short life cycles
  - Redundant navigation paths.
  - Classes with few responsibilities
  - Classes with "control-like" operation names such as start_process_alpha

- **Consequences**
  - Excessive complexity
  - Unstable analysis and design models
  - Divergent design and implementation
  - Lack of system extensibility

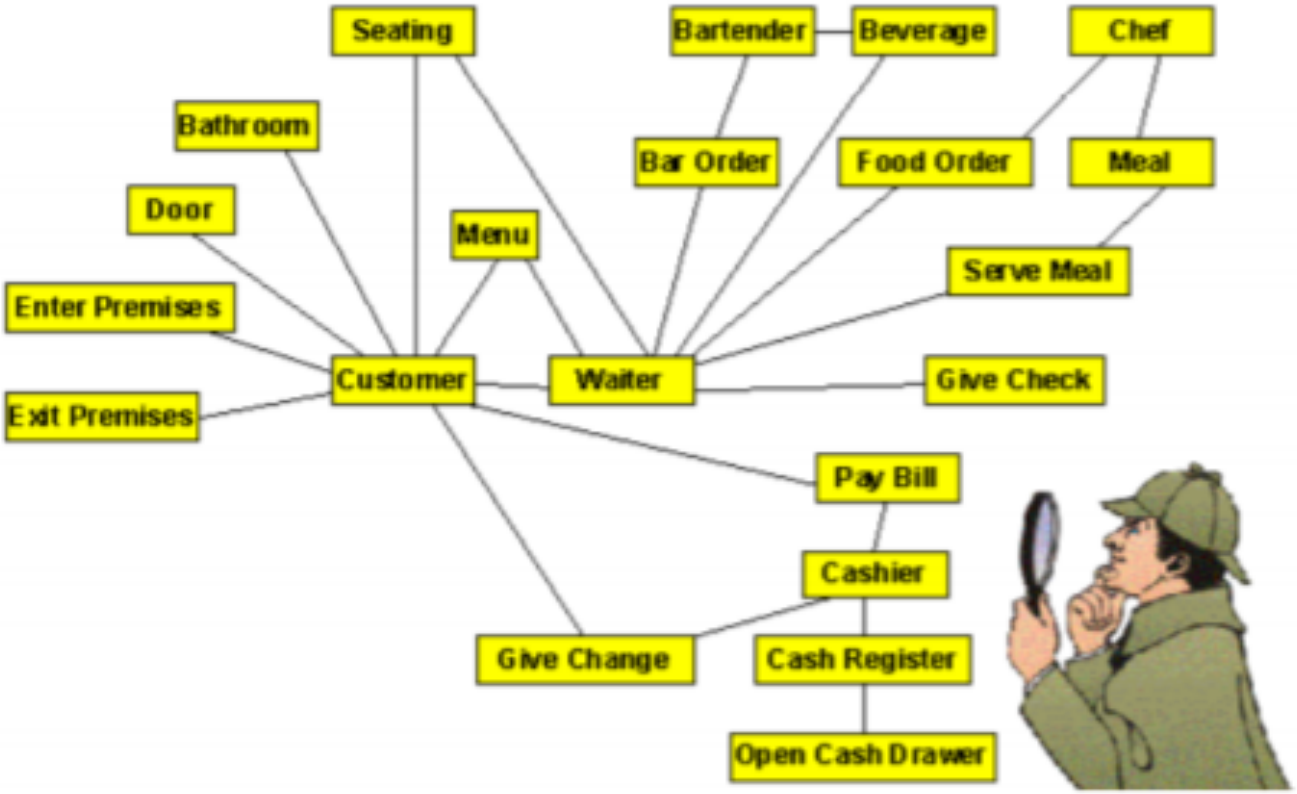
POLTERGEISTS

Example: Teach students stack class

- Rewrites all functions already existing in list class

```
public class LabStack<T> {
    private LinkedList<T> list;
    public LabStack() { list = new LinkedList<T>(); }
    public boolean empty() { return list.isEmpty(); }
    public T peek() throws EmptyStackException {
        if (list.isEmpty()) { throw new EmptyStackException(); }
        return list.peek();
    }
    public T pop() throws EmptyStackException {
        if (list.isEmpty()) { throw new EmptyStackException(); }
        return list.pop();
    }
    public void push(T element) { list.push(element); }
    public int size() { return list.size(); }
    public void makeEmpty() { list.clear(); }
    public String toString() { return list.toString(); }
}
```

POLTERGEISTS



ANTI-PATTERNS. TYPES

- **Software development**
  - **Technical problems and solutions encountered by programmers**
- **Architectural**
  - Identify and resolve common problems in how systems are structured.
- **Software project management**
  - Address common problems in software processes and development organizations.

ARCHITECTURAL ANTI-PATTERNS

- **Architectural AntiPatterns focus on some common problems and mistakes in the creation, implementation, and management of architecture.**
- **Types**
  - Architecture by Implication
  - Auto generated Stovepipe
  - Cover Your Assets
  - Design by Committee
  - Intellectual Violence
  - Jumble
  - Reinvent the Wheel
  - Spaghetti Code

ARCHITECTURAL ANTI-PATTERNS

- **Reinvent the Wheel**
  - Synopsis
    - Legacy systems with overlapping functionality. Every system built in isolation.
  - Refactored solution
    - Take advantage of existing, tested, and available systems

ARCHITECTURAL ANTI-PATTERNS

- **Vendor Lock in**
  - Synopsis
    - Proprietary, product-dependent architectures do not manage complexity and lead to a loss of control of the architecture and maintenance costs.
  - Refactored Solution
    - Providing an isolation layer between product-dependent interfaces and the majority of application software enables management of complexity and architecture.

ARCHITECTURAL ANTI-PATTERNS

- **Cover Your Assets**
  - Synopsis
    - Document driven software processes often employ authors who list alternatives instead of making decisions.
  - Refactored Solution
    - Establish clear purposes and guidelines for documentation tasks; inspect the results for the value of documented decisions.

ARCHITECTURAL ANTI-PATTERNS

- **Stovepipe System**
  - Synopsis
    - Ad hoc integration solutions and lack of abstraction lead to brittle, un-maintainable architectures
  - Refactored solution
    - Proper use of abstraction, subsystem facades, and metadata leads to adaptable systems.

ANTI-PATTERNS. TYPES

- **Software development**
  - **Technical problems and solutions encountered by programmers**
- **Architectural**
  - Identify and resolve common problems in how systems are structured.
- **Software project management**
  - Address common problems in software processes and development organizations.

SOFTWARE PROJECT MANAGEMENT ANTIPATTERNS

- **Areas where human communication can be destructive to the software process**
- **The purpose of management AntiPatterns is to develop awareness that enables you to increase your success.**
  - Types
  - Analysis Paralysis
  - Blowhard Jamboree
  - Corncob
  - Death By Planning
  - Email is dangerous
  - Fear of Success
  - Irrational management

SOFTWARE PROJECT MANAGEMENT ANTIPATTERNS

- **Analysis Paralysis**
  - Synopsis
    - Striving for perfection and completeness in the analysis phase leads to project gridlock.
  - Refactored Solution
    - Use an Incremental, iterative development processes. Defer the detailed analysis until the knowledge is available.

SOFTWARE PROJECT MANAGEMENT ANTIPATTERNS

- **Corncob**
  - Synopsis
    - Frequently, difficult people obstruct and divert the software development process.
  - Refactored Solution
    - Address agendas of the individual through various tactical, operational, and strategic organizational actions.

SOFTWARE PROJECT MANAGEMENT ANTIPATTERNS

- **Fear of Success**
  - Synopsis
    - People (software developers included) do crazy things when a project is near successful completion.
  - Refactored Solution
    - When project completion is close-at-hand, a clear declaration of success is important for the project environment.

SOFTWARE PROJECT MANAGEMENT ANTIPATTERNS

- **Smoke and Mirrors**
  - Synopsis
    - End-users mistakenly assume that a brittle demonstration is a capability that is ready for operational use.
  - Refactored Solution
    - Practice of proper ethics is important to manage expectations, risk, liabilities, and consequences in computing sales and marketing situations.

NEXT COURSES

- PROJECTS PRESENTATIONS & EXAM