DESIGN PATTERNS

Course 11

PREVIOUS COURSE CONTENT

- Applications split on levels
- J2EE Design Patterns
  - Business Delegate
  - Service Locator
  - Session Facade

COURSE CONTENT

Anti – patterns

- The blob
- Poltergeists
- Golden Hammer

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

- 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.

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

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.

SYMPTOMS

Haste

Look you --
Just "clean up" the code.
We ship tomorrow.

Apathy

Reuse? Reuse!!
Who's ever gonna reuse
this crappy code?
NO ONE! That's who.

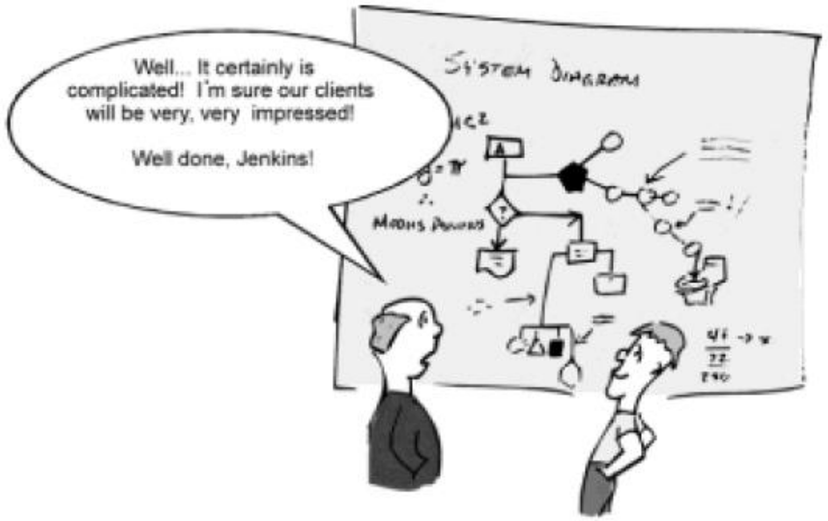
SYMPTOMS

Narrow-Mindedness

Sloth

SYMPTOMS

Avarice


Well... It certainly is complicated! I'm sure our clients will be very, very impressed!
Well done, Jenkins!
System Diagram

Ignorance

Look, Roy... I don't really have time to actually read this... Could you put together a one-page Executive Summary...
Dummy...
DCOM 101

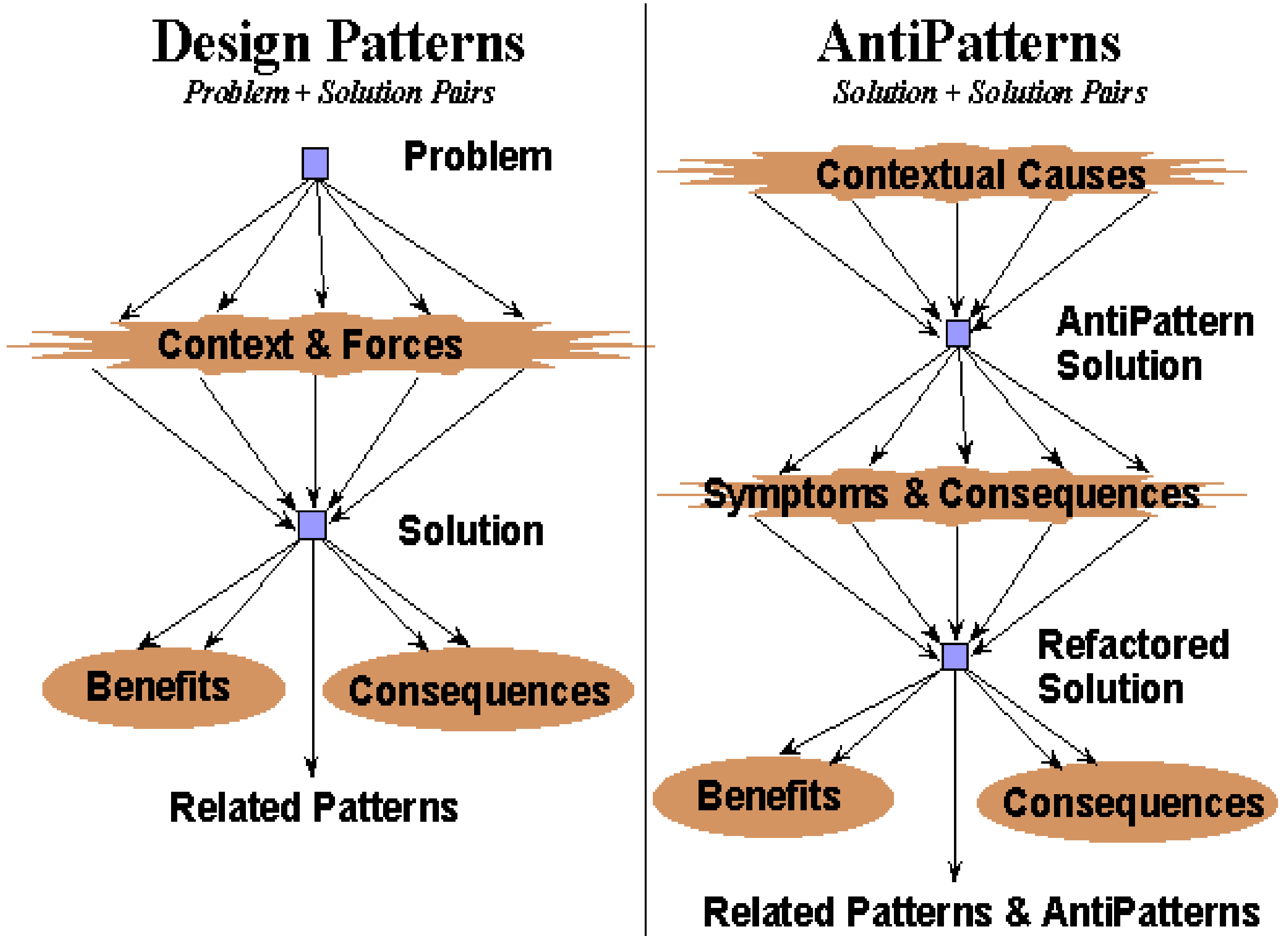
SYMPTOMS

- 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



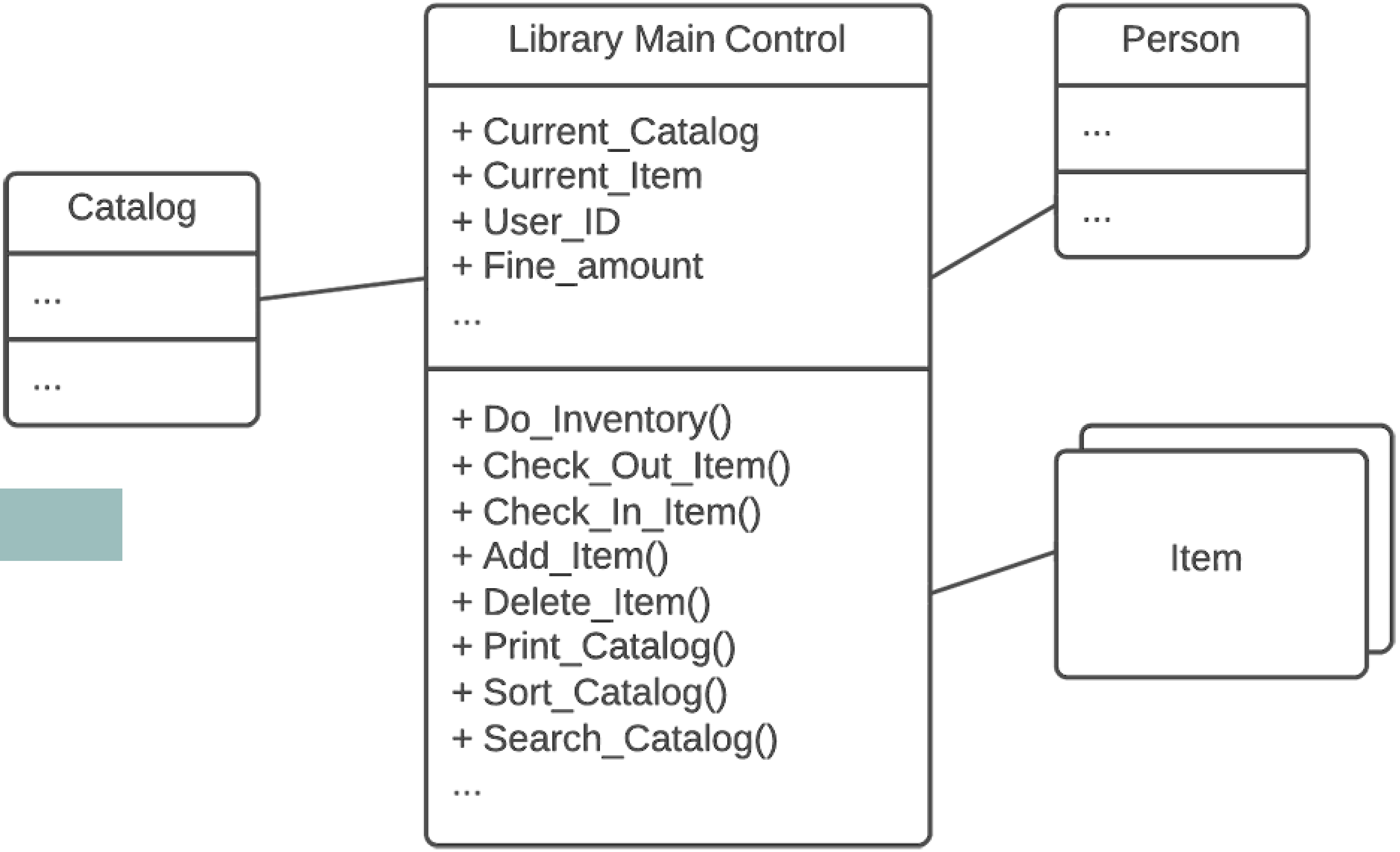
ANTI-PATTERNS

The Blob

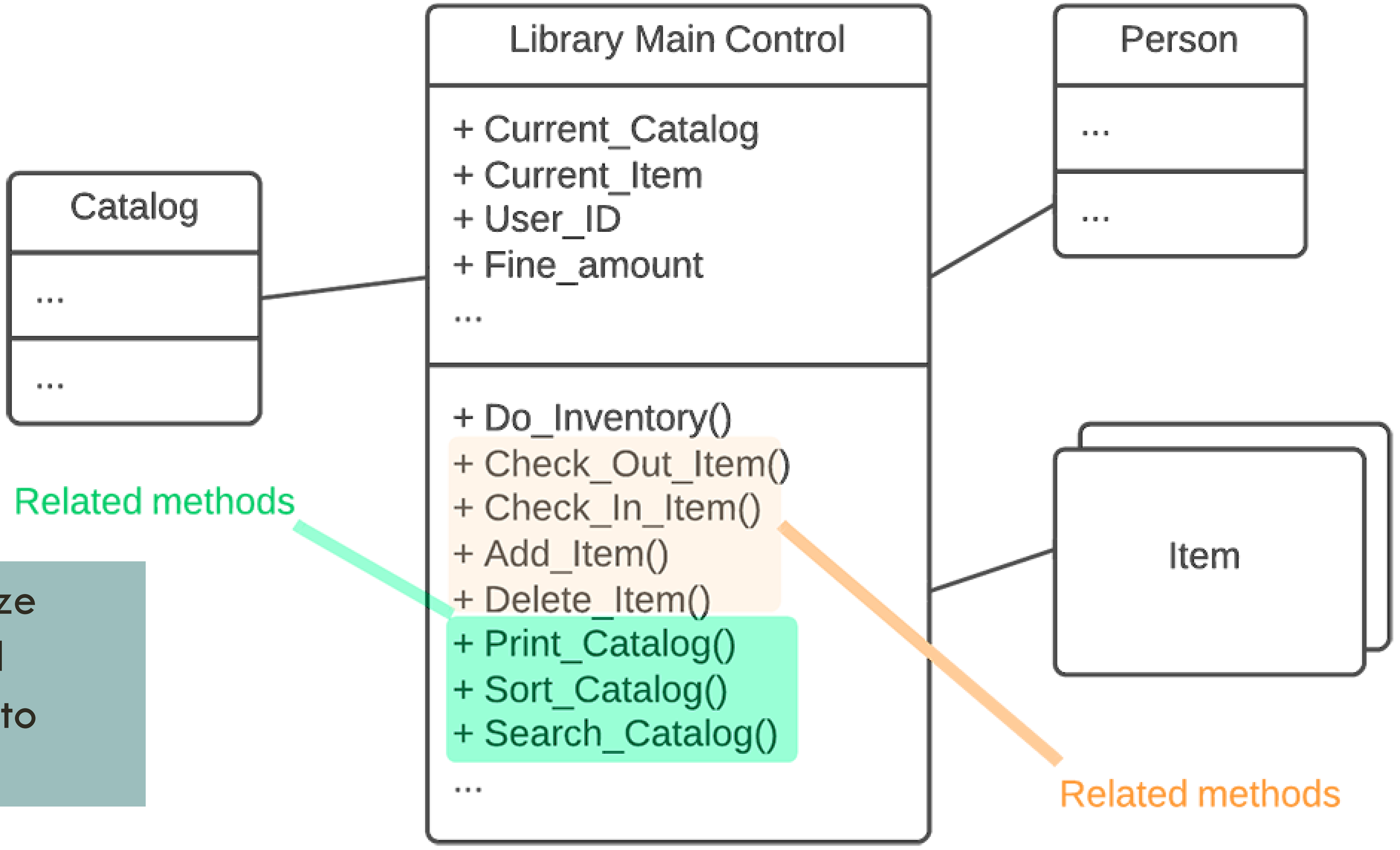
- ❑Symptoms
  - ❑Single class with many attributes and operations
  - ❑Controller class with simple, data-object classes
  - ❑Lack of OO design
  - ❑A migrated legacy design
- ❑Consequences
  - ❑Lost of OO advantages
  - ❑Too complex to reuse or test
  - ❑Expensive to load in memory
    - ❑Way?

THE BLOB

The library application

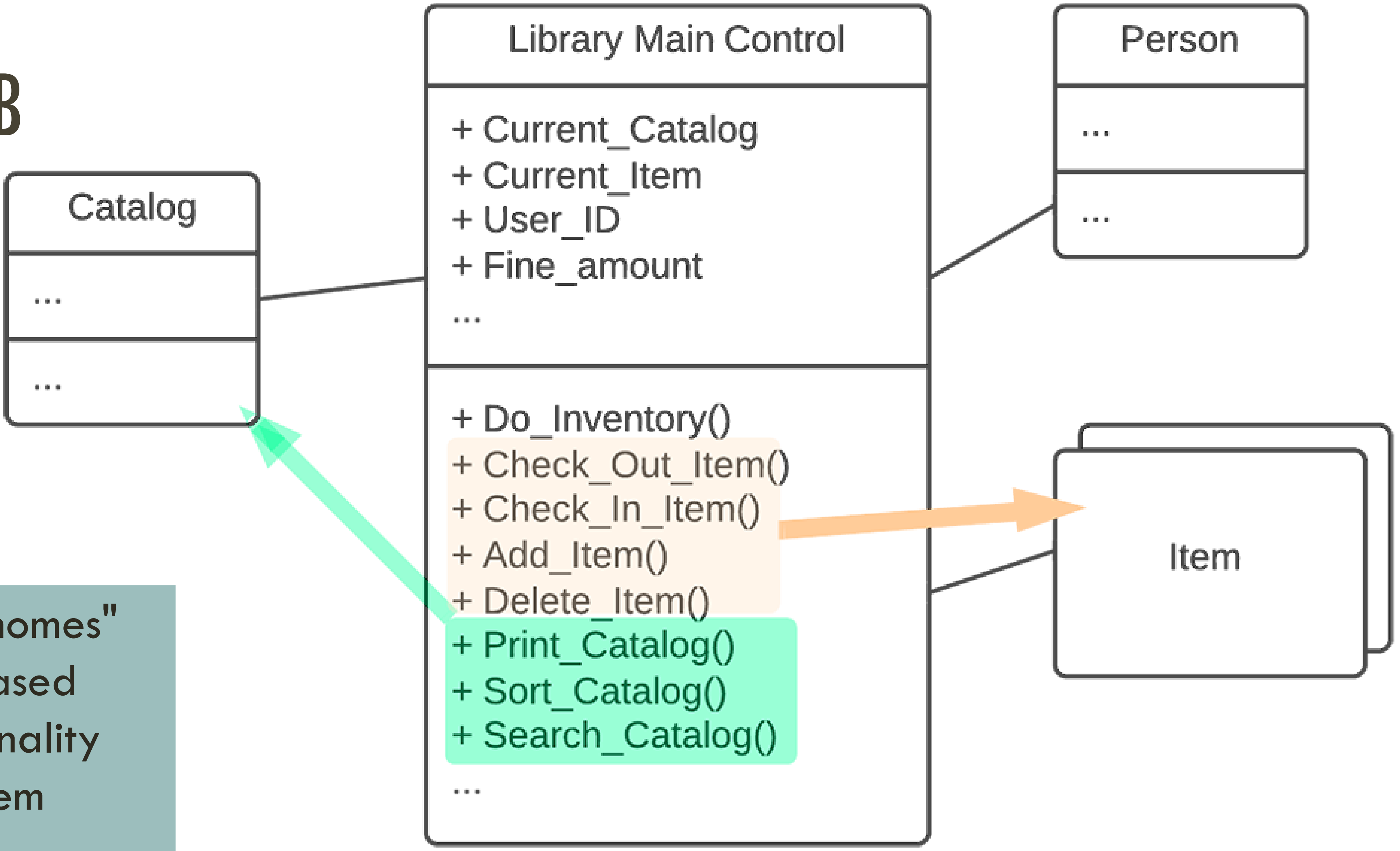


THE BLOB



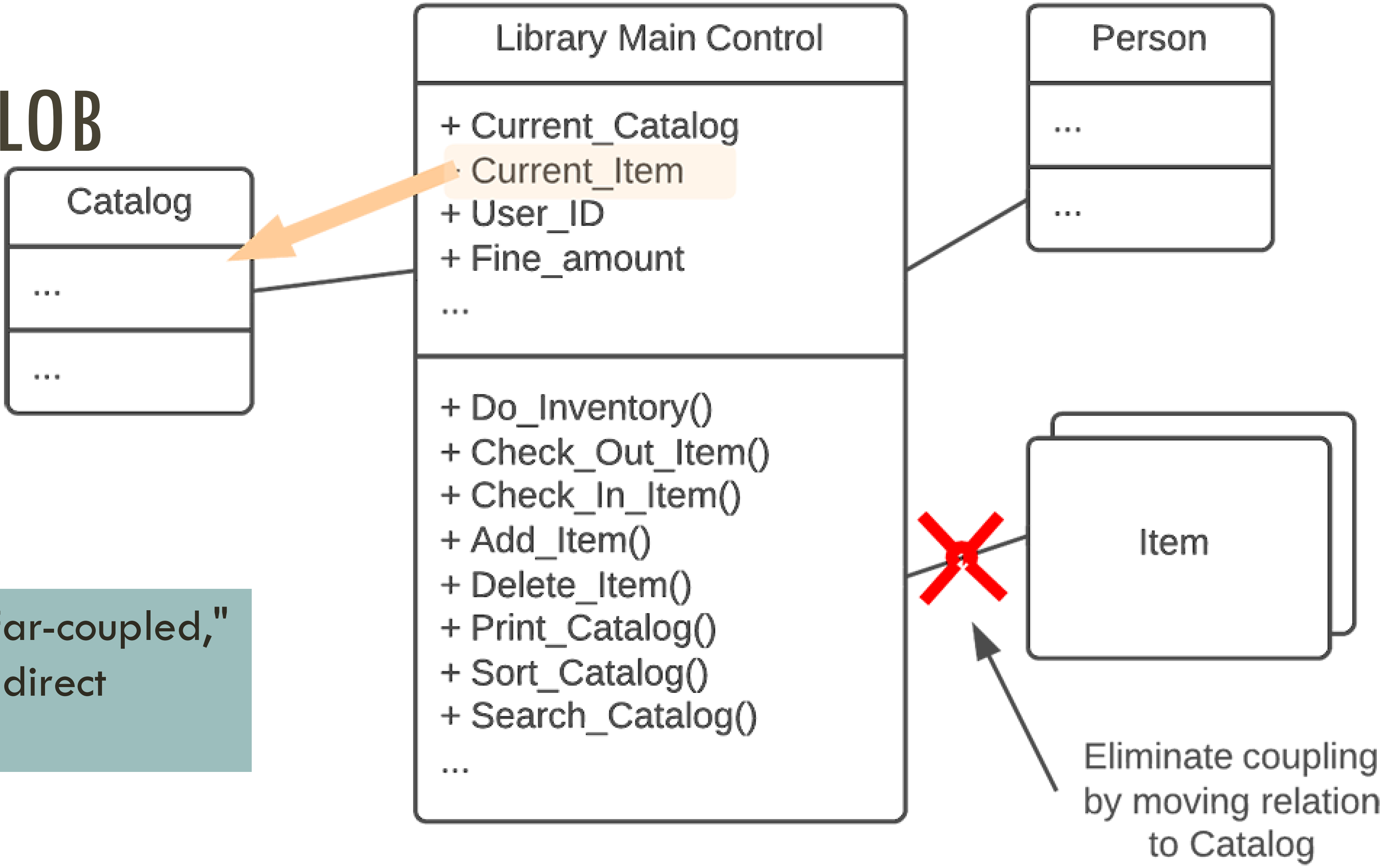
1.Identify or categorize related attributes and operations according to contracts.

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

THE BLOB

Refactiored solution

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

ANTI-PATTERNS

Poltergeists

- Also Known As: Gypsy, Proliferation of Classes, Big DoIt 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

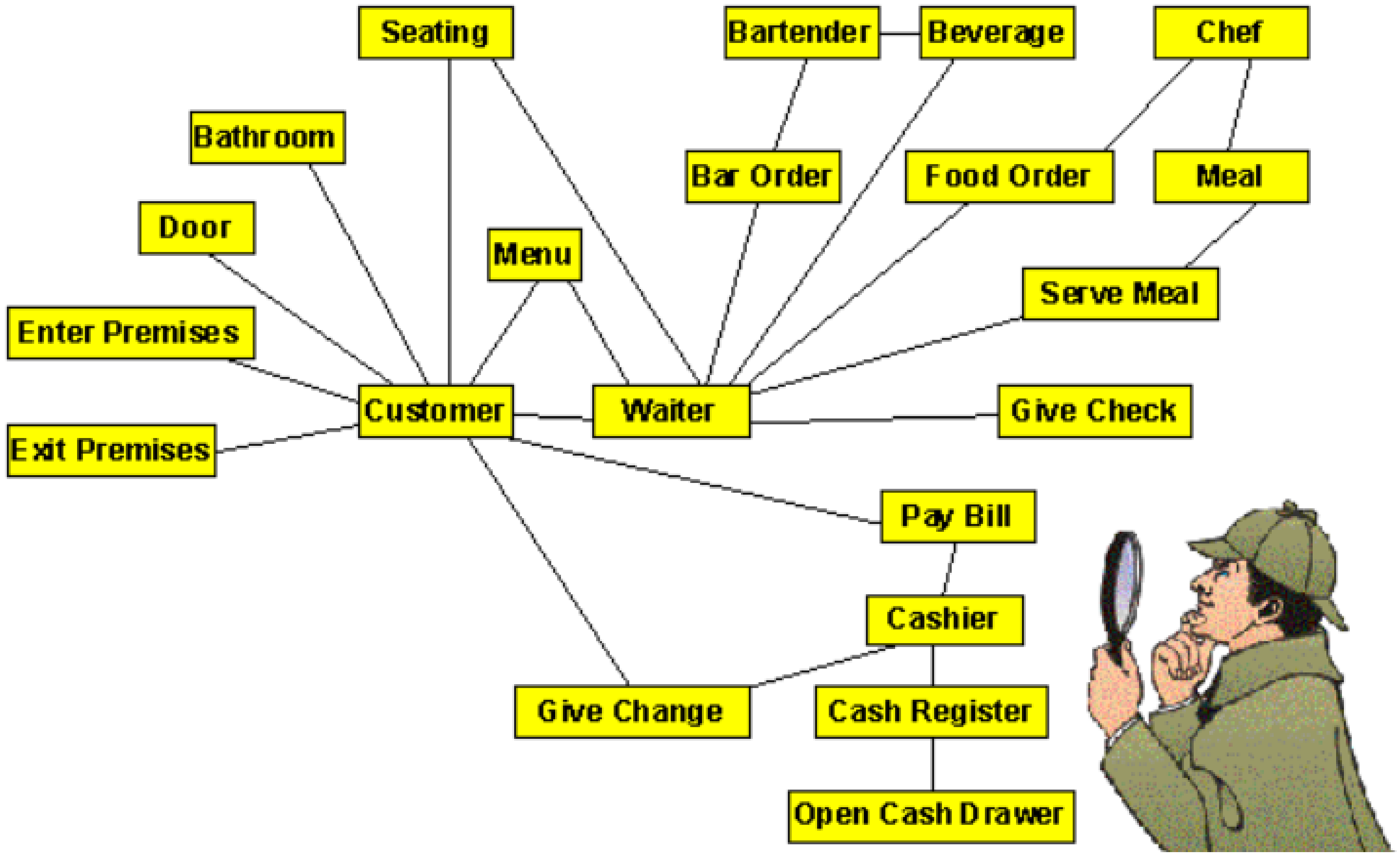
```
import java.util.EmptyStackException;
import java.util.LinkedList;

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



POLTERGEISTS

Solution

- ❑Refactor to eliminate irrelevant classes
  - ❑Delete external classes (outside the system)
  - ❑Delete classes with no domain relevance
- ❑Refactor to eliminate transient “data classes”
- ❑Refactor to eliminate transient “operation classes”
- ❑Refactor other classes with short lifecycles or few responsibilities
  - ❑Move into collaborating classes
  - ❑Regroup into cohesive larger classes

ANTI-PATTERNS

Golden Hammer

- Also Known As: Old Yeller, Head-in-the sand
- Symptoms
  - You need to choose technologies for your development, *and you are of the belief that yc exactly* ***one*** *technology to dominate the architecture*
- Consequences
  - The development team are committed to the technology they know
  - The development team are not familiar with other technologies
  - Other, unfamiliar, technologies are seen as risky
  - It is easy to plan and estimate for development in the familiar technology

GOLDEN HAMMER

Causes

- ❑Several successes have used a particular approach.
- ❑Large investment has been made in training and/or gaining experience in a product or technology.
- ❑Group is isolated from industry, other companies.
- ❑Reliance on proprietary product features that aren't readily available in other industry products.

ANTI-PATTERNS

Spaghetti Code

- Symptoms and Consequences
  - Methods are very process-oriented; frequently, in fact, objects are named as processes.
  - The flow of execution is dictated by object implementation, not by the clients of the objects.
  - Minimal relationships exist between objects.
  - Many object methods have no parameters, and utilize class or global variables for processing.
  - The pattern of use of objects is very predictable.
  - Code is difficult to reuse, and when it is, it is often through cloning. In many cases, however, code is never considered for reuse.
  - Object-oriented talent from industry is difficult to retain.
  - Benefits of object orientation are lost; inheritance is not used to extend the system; polymorphism is not used

ANTI-PATTERNS

Lava Flow

- Also Known As: Dead Code
- Symptoms and Consequences
  - Frequent unjustifiable variables and code fragments in the system.
  - Undocumented complex, important-looking functions, classes, or segments that don't clearly relate to the system architecture.
  - Whole blocks of commented-out code with no explanation or documentation.
  - Lots of "in flux" or "to be replaced" code areas.
  - Unused (dead) code, just left in.
  - Unused, inexplicable, or obsolete interfaces located in header files.
  - If existing Lava Flow code is not removed, it can continue to proliferate as code is reused in other areas.
  - If the process that leads to Lava Flow is not checked, there can be exponential growth as succeeding developers, too rushed or intimidated to analyze the original flows, continue to produce new, secondary flows as they try to work around the original ones, this compounds the problem.
  - As the flows compound and harden, it rapidly becomes impossible to document the code or understand its architecture enough to make improvements.

ANTI-PATTERNS

Cut-And-Paste Programming

- Also Known As: Clipboard Coding, Software Cloning, Software Propagation
- Symptoms and Consequences
  - The same software bug reoccurs throughout software despite many local fixes.
  - Lines of code increase without adding to overall productivity.
  - Code reviews and inspections are needlessly extended.
  - It becomes difficult to locate and fix all instances of a particular mistake.
  - Code is considered self-documenting.
  - Code can be reused with a minimum of effort.
  - This AntiPattern leads to excessive software maintenance costs.
  - Software defects are replicated through the system.
  - Reusable assets are not converted into an easily reusable and documented form.
  - Developers create multiple unique fixes for bugs with no method of resolving the variations into a standard fix.
  - Cut-and-Paste Programming form of reuse deceptively inflates the number of lines of code developed without the expected reduction in maintenance costs associated with other forms of reuse.

ANTI-PATTERNS

Magic Numbers and Strings

- Problem
  - Using unnamed numbers or string literals instead of named constants in code.
- Symptoms and Consequences
  - Semantics of the number or string literal is partially or completely hidden without a descriptive name or another form of annotation
  - Unnamed string literals in code makes internationalization harder
- Solution
  - Use named constants, resource retrieval methods, or annotations.

ANTY-PATTERNS

- Anti-patterns are a more compelling form of patterns

- Each anti pattern includes a solution
  - Anti-patterns solutions generate mostly negative consequences
  - Refactor solution generates mostly positive benefits

- Anti-patterns are useful for refactoring, migration, upgrade and reengineering