Fundamentals of Object-Oriented Design

Kevin Burleigh

A Brief History of OO

Based on the Nygaard Classification

(In the beginning, there was nothing...)

Procedural (Imperative)

FORTRAN	first “high level” language optimizing compiler subroutines
COBOL	records (structs) macros (pre-compiler directives) comments
ALGOL	blocks and scope explicit typing references user-defined data types operator overloading

Constraint (Declarative)

SQL	(modern-day example)
Prolog	(post-dates Simula 67)
ALICE ML	(1978 - not the more recent Alice educational software)

Functional (Declarative)

Lisp	garbage collection dynamic typing closures

Object Oriented

Simula 67

considered the first
true OO language

created by Kristen Nygaard
and Ole-Johan Dahl

classes

objects

inheritance

virtual methods

abstract data types

Object Oriented (cont)

Alan Kay

inventor of Smalltalk

coined the phrase “object-oriented” c.1967...

...but forgot to write down a useful definition

proliferation of mechanisms:

- class-based inheritance
- modules / mixins
- duck-typing
- prototype-based inheritance
- compile- vs run-time

Object Oriented (cont)

Lieberman

Stein

Treaty of Orlando

Ungar

OOPSLA '87 in Orlando, FL

somewhat tongue-in-cheek:

WHEREAS among the purport
dynamicism, the vast majority of ex
conformance to strictly hierarchica

WHEREAS these issues have a b
was a reaction to these problem
mechanism based on the idea of d

exact mechanism is not critical for OO

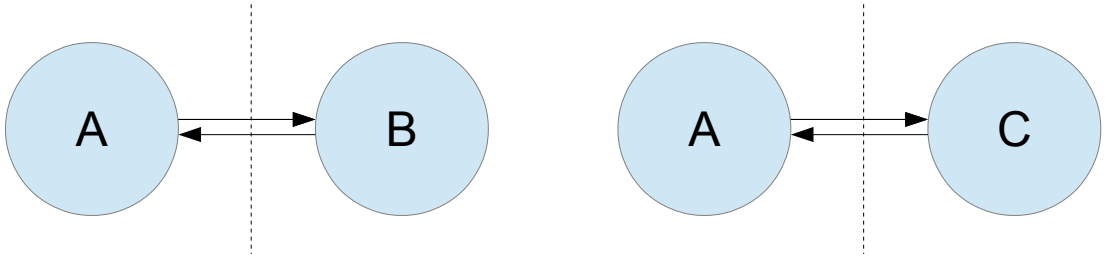
"...the intent of object-oriented programming is to provide a natural and straight-forward way to describe real-world concepts, allowing the flexibility of expression necessary to **capture the variable nature of the world being modeled**, and the **dynamic ability to represent changing situations**"

Object Oriented (cont)

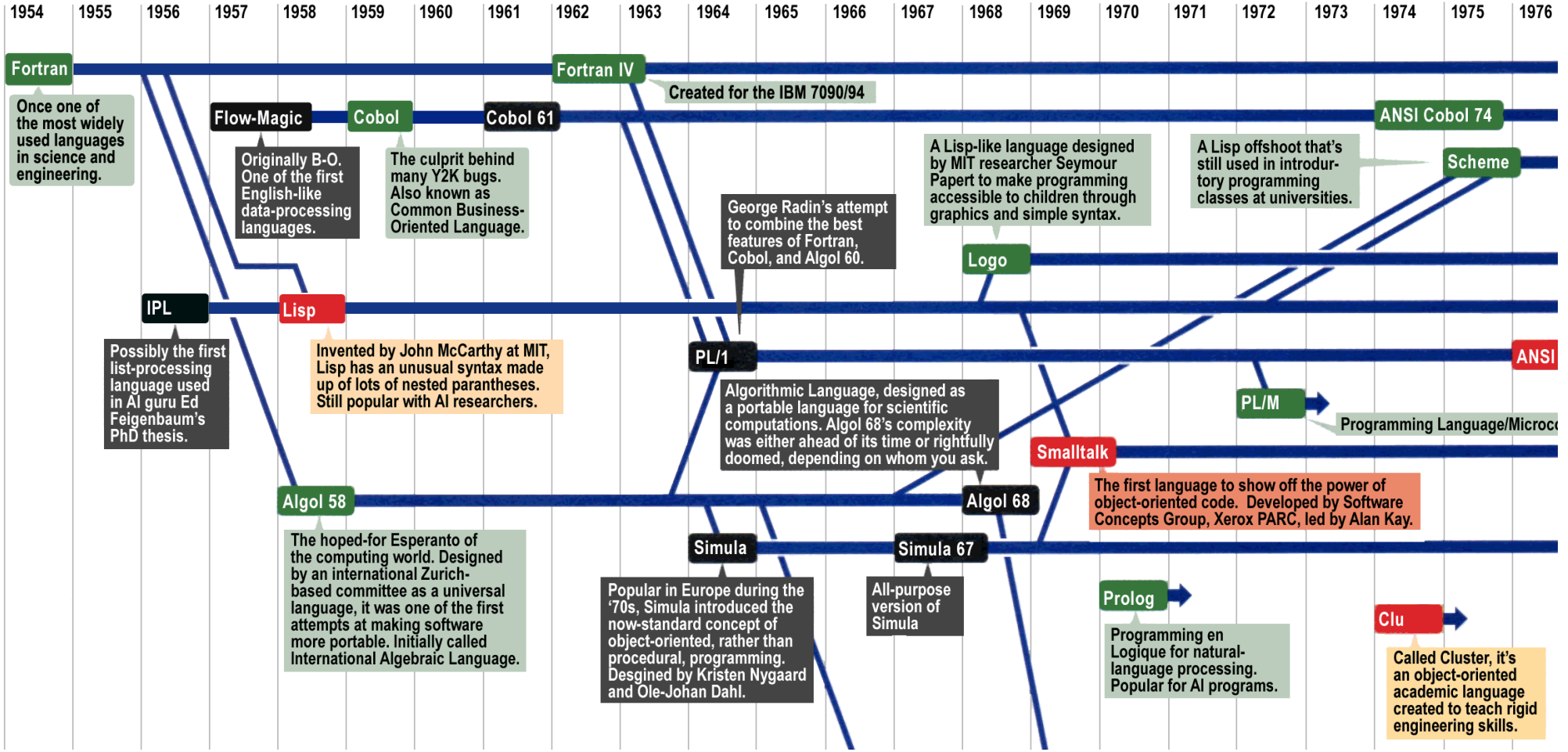
Alan Kay

"OOP to me means **only messaging**, local retention and protection and hiding of state-process, and **extreme late-binding** of all things."

(email c.2003)



Object Oriented (cont)



DigiBarn Computer Museum

My Working Definition

A Design is OO if...

...it models a system or domain
(real or imagined)

AND

...it utilizes late-binding polymorphism for extensibility.

Domain Modeling

We want code that is easy to maintain:

understand
reason about
change (in certain ways)
test

It helps a LOT to lean on the domain being modeled!

Domain experts understand the key concepts, their relationships, and likely areas for change.

Domain Modeling (cont)

Metz

"Reasonable" Code

"Small changes in requirements require correspondingly small changes in code."

"The cost of any change should be proportional to the benefits the change achieves."

Open-Closed Principle

We want code that we only need to understand/write/test/read once, but whose behavior can still be modified as our system changes.

This code would be:

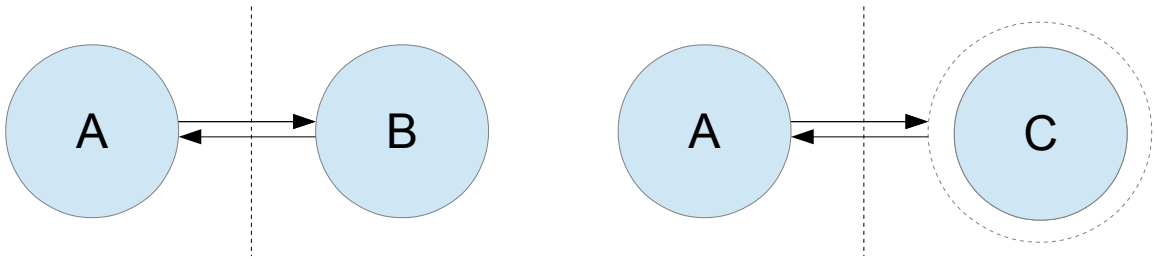
- open for extension
- closed for modification

But how can this possibly be achieved?

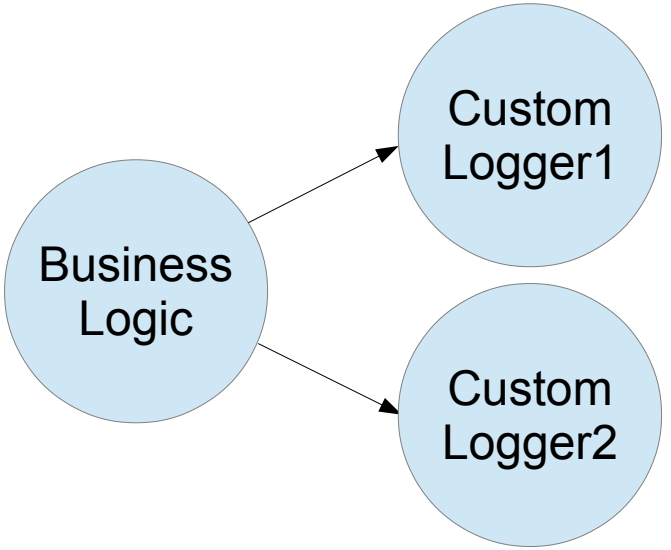
Dependency Inversion Principle

Instead of depending on specific concrete classes, create stable abstractions (types) and depend on those.

To extend behavior, pass different concrete implementations of those types into the system *polymorphically*.



DIP Example



DIP Example (cont)

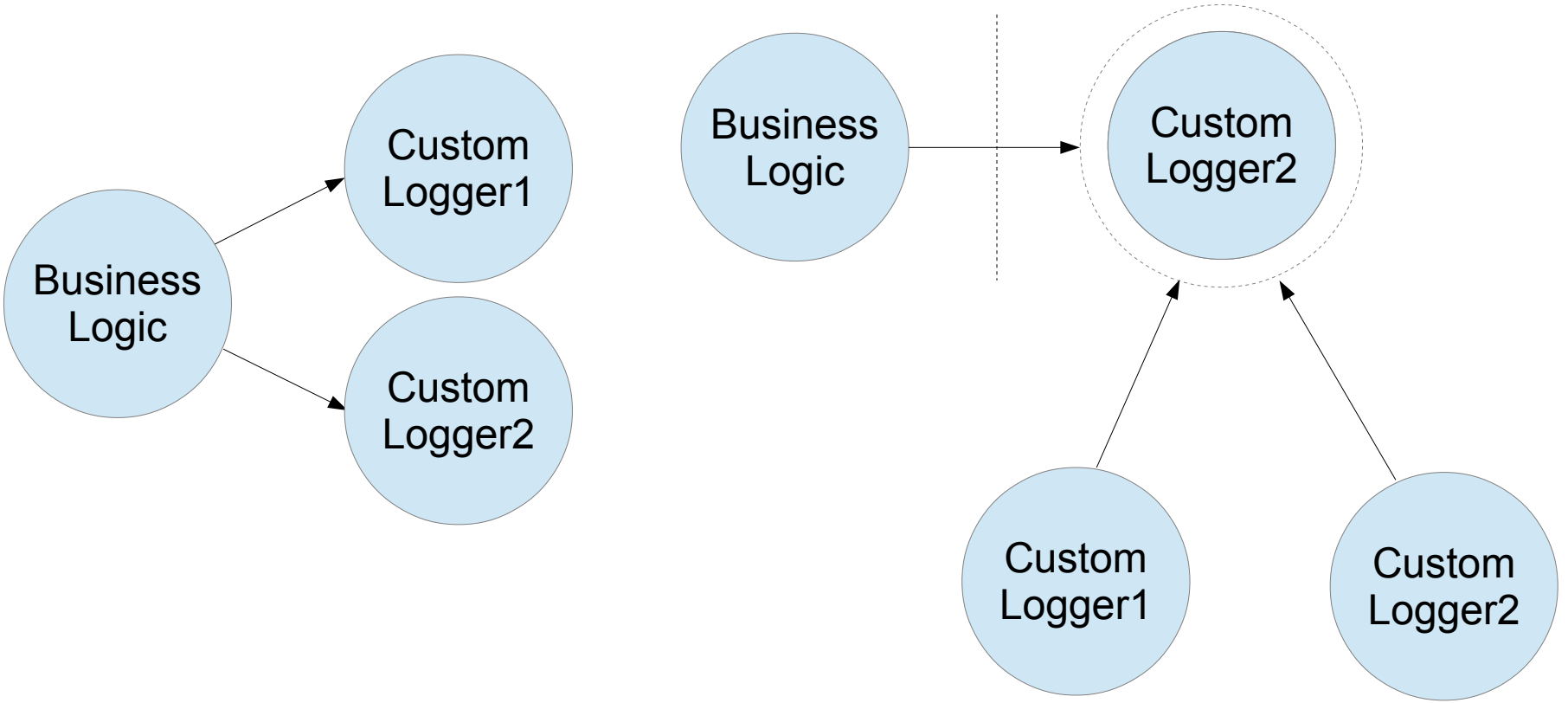
```
public class BusinessLogic {
    public void do_work() {
        CustomLogger1 logger1 = new CustomLogger1();
        CustomLogger2 logger2 = new CustomLogger2();

        // ... complicated business logic ...

        logger1.info("Zarf successful");
        logger2.record("Zarf successful");

        // ... complicated business logic ...

        logger1.info("Blarg detected");
        logger2.record("Blarg detected");

        // ... complicated business logic ...

        logger1.info("Frump completed");
        logger2.record("Frump completed");
    }
}
```

`BusinessLogic` depends on both `CustomLogger1` and `CustomLogger2`.

We know from our domain (and/or past experience) that the logging details are very likely to change more often than the business logic.

We would like to protect `BusinessLogic` from these changes.

DIP Example (cont)



DIP Example (cont)

```
public interface BusinessLogger {
    void log(String str);
}
```

```
public class BusinessLogic {
    BusinessLogger _logger;

    BusinessLogic(BusinessLogger logger) {
        _logger = logger;
    }

    public void do_work() {
        // ... complicated business logic ...

        _logger.log("Zarf successful");

        // ... complicated business logic ...

        _logger.log("Blarg detected");

        // ... complicated business logic ...

        _logger.log("Frump completed");
    }
}
```

First, we create a new `BusinessLogger` type.

Next, we redesign `BusinessLogic` to depend on the new abstract type.

Now `BusinessLogic` is open-closed with respect to logging.

Technically, the hard-coded strings could violate both OCP (if volatile) and SRP (if not driven by the needs of `BusinessLogic`).

DIP Example (cont)

```java
public class CustomLogger1 implements BusinessLogger
{
    @Override
    public void log(String str) {
        info(str);
    }

    public void info(String str) {
        System.out.println("str = [" + str + "]");
    }
}
```

```java
public class CustomLogger2 implements BusinessLogger
{
    @Override
    public void log(String str) {
        record(str);
    }

    public void record(String str) {
        System.out.println("STR = {" + str + "}");
    }
}
```

We then adjust our logger classes to implement the newly-created `BusinessLogger` type.

DIP Example (cont)

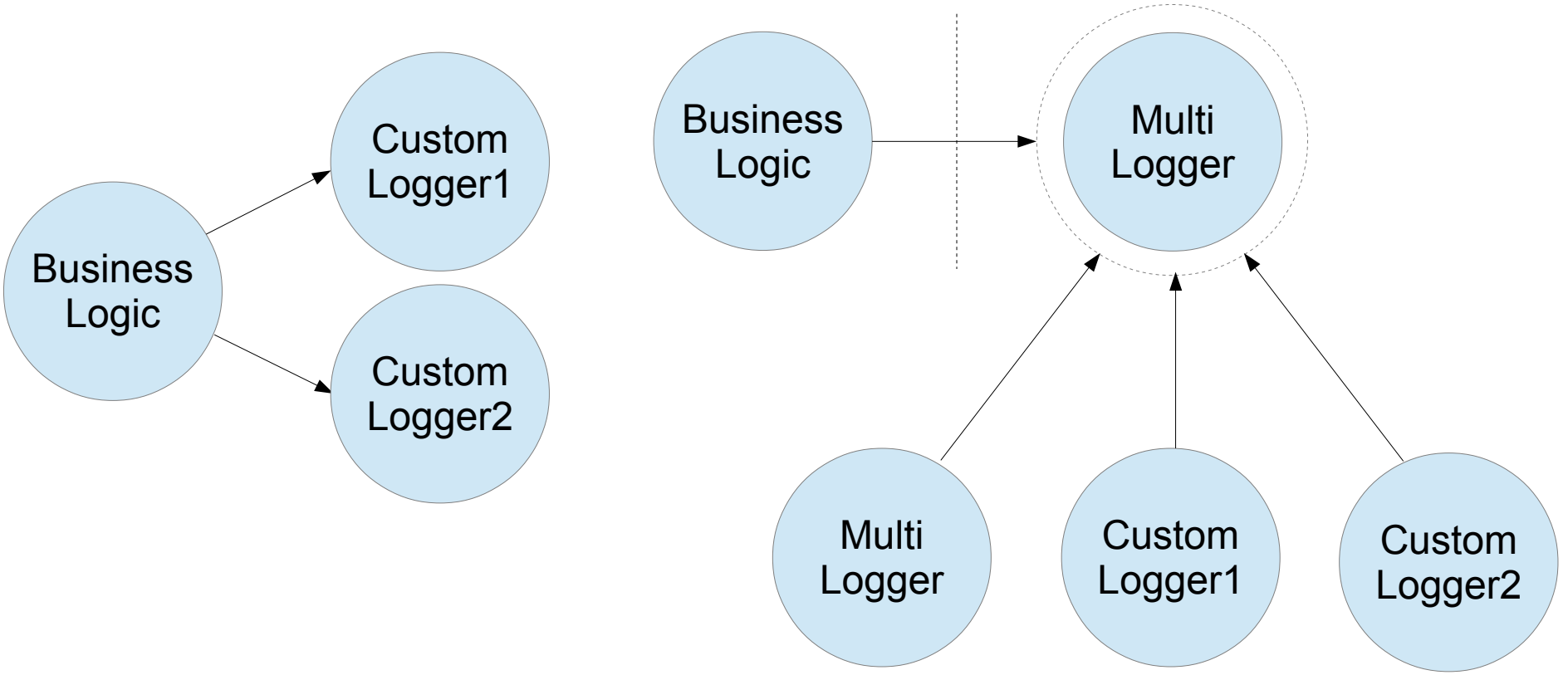
```
public class Main {
    public static void main(String[] args) {
        BusinessLogger logger = new CustomLogger2();
        BusinessLogic bl = new BusinessLogic(logger);
        bl.do_work();
    }
}
```

In this case, we'll have `main` create the dependency and inject it into `BusinessLogic` upon creation.

This is called “dependency injection”, and we just implemented the Strategy design pattern.

But what about the other logger?

DIP Example (cont)



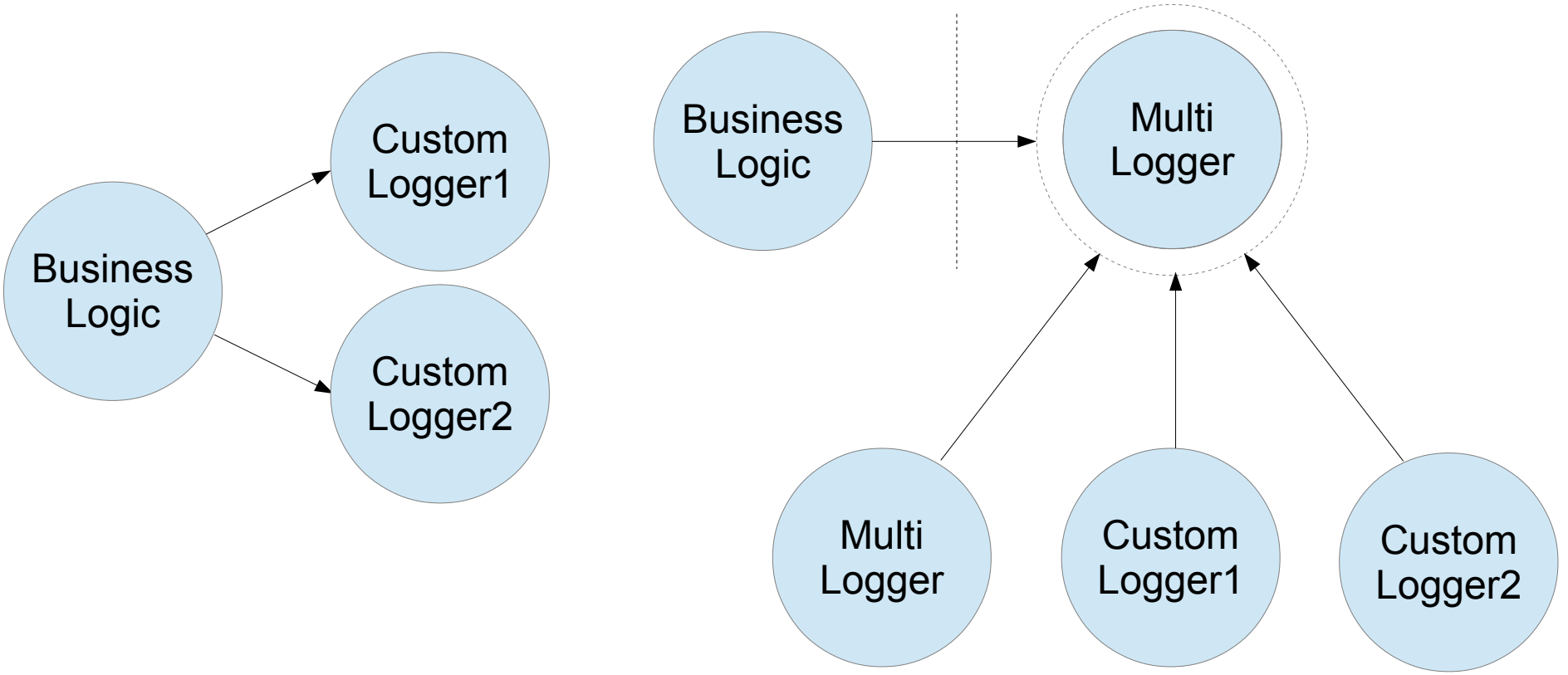
DIP Example (cont)

```
public class Main {
    public static void main(String[] args) {
        BusinessLogger logger1 = new CustomLogger1();
        BusinessLogger logger2 = new CustomLogger2();

        MultiLogger multilogger = new MultiLogger();
        multilogger.register(logger1);
        multilogger.register(logger2);

        BusinessLogic bl = new BusinessLogic(multilogger);
        bl.do_work();
    }
}
```

This is the Observer design pattern.

```
public class MultiLogger implements BusinessLogger {
    List<BusinessLogger> _loggers = new ArrayList<BusinessLogger>();

    public void register(BusinessLogger logger) {
        _loggers.add(logger);
    }

    @Override
    public void log(String str) {
        for(BusinessLogger logger : _loggers) {
            logger.log(str);
        }
    }
}
```

DIP Example (cont)



Liskov Substitution Principle

How do we know that our extensions really *are* polymorphic, and that they won't result in unexpected problems?

Liskov

Wing

Subtype Requirement:

If S is a subtype of type T, then any property provable about T must be true for S.

S must accept all requests that are valid for T, and must limit its responses and behaviors to those that are valid for T.

LSP: Violations

```
public void log(String str) {
    throw new NotImplementedException("SUCKA!!");
}
```

```
public void log(String str) {
    while(true) {
        // HA HA HA!!
    }
}
```

```
public void log(String str) {
    _logfile.delete();
}
```

Sometimes the type definition is well-documented, but even in those cases there are often many implicit expectations.

LSP: Duck Typing

The abstract types described by the LSP exist in any OO system, whether or not those types are explicit in the design...

...I'm looking at you, ruby and python developers!

Late-Binding

What, exactly, constitutes “late” binding?

For our purposes, “late-binding” means that the concrete implementations of types are **created** and **chosen after implementation** of the system we are designing is complete.

Our system achieves the promise of OCP.

A Design is OO if...

...it models a system or domain
(real or imagined)

AND

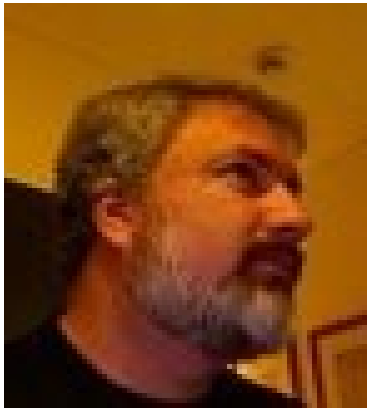
...it utilizes late-binding polymorphism for extensibility.

OO Principles

SOLID


Martin


Coplien

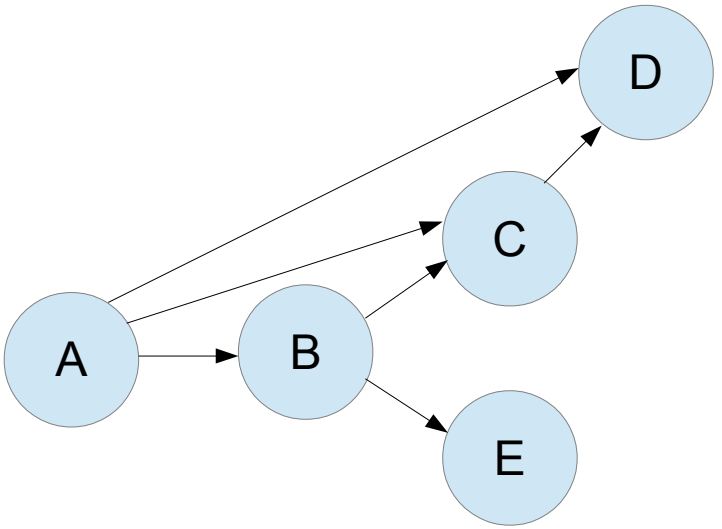

Feathers

SRP	Single Responsibility Principle
OCP	Open-Closed Principle
LSP	Liskov Substitution Principle
ISP	Interface Segregation Principle
DIP	Dependency Inversion Principle

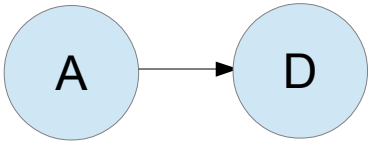
Don't be fooled: there are six more principles!

Law of Demeter

```
class A {
    void do_work(B b) {
        b.c().d().do_something();
    }
}
```




```
class A {
    void do_work(D d) {
        d.do_something();
    }
}
```



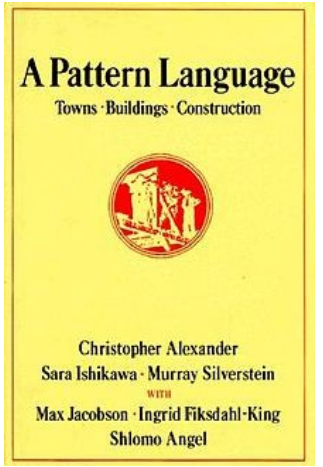
Don't go looking for things!

A related concept: Tell, Don't Ask.

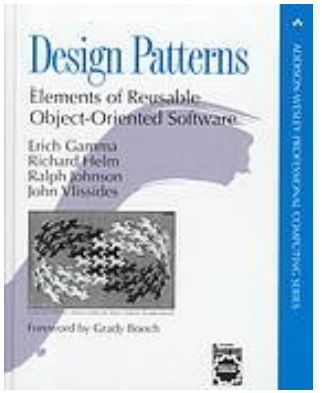
Design Patterns

Patterns are effective solutions to recurring design problems.

Originally applied to buildings and towns in 1977.



The "Gang of Four" (GoF) created a book in 1994 describing 23 software design patterns.



This led to the term "anti-pattern", which describes a seemingly good solution that has significant drawbacks.

Clarity of code comes from clarity of thought.

Thank you for coming!

Attributions

Image of Robert Martin by Tim-bezhashvyly:
https://commons.wikimedia.org/wiki/File:Robert_Cecil_Martin.png
Image of Barbara Liskov by Kenneth C. Zirkel:
https://commons.wikimedia.org/wiki/File:Barbara_Liskov_MIT_computer_scientist_2010.jpg
Image of Jeannette Wing by World Economic Forum:
https://commons.wikimedia.org/wiki/File:Jeannette_Wing,_Davos_2013.jpg
Image of Michael Feathers (blog):
https://www.goodreads.com/author/show/25201.Michael_C_Feathers/blog
Image of Jim Coplien from YouTube:
https://www.youtube.com/watch?v=KtHQGs3zFAM
Image of Sandi Metz by Npostolovski:
https://commons.wikimedia.org/wiki/File:Sandi_Metz.jpg
Image of Alan Kay by Marcin Wichary:
https://commons.wikimedia.org/wiki/File:Alan_Kay_(3097597186).jpg
Image of Kristen Nygaard and Ole-Johan Dahl:
http://history-computer.com/ModernComputer/Software/Simula.html
Image of Lynn Andrea Stein on Olin College of Engineering:
http://www.olin.edu/faculty/profile/lynn-andrea-stein/
Image of Henry Lieberman from MIT:
http://web.media.mit.edu/~lieber/
Image of David Ungar by Dcoetzee:
https://commons.wikimedia.org/wiki/File:David_Ungar.jpg
Image of Homer Simpson (20th Century Fox):
http://www.quoteslike.com/doh-free-images-at-clker-com-vector-clip-art-online-royalty-free-Uz2q1H-quote/