<table>
<thead>
<tr>
<th>Design Pattern:</th>
<th>Singleton Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idiom: domain (the world custom cow rh.).</td>
<td>Global variables are bad</td>
</tr>
<tr>
<td>interface: opens.</td>
<td>access issue: public/private.</td>
</tr>
<tr>
<td>license:</td>
<td>weights to use global variables</td>
</tr>
<tr>
<td>Solution for what condition is related pattern consequence.</td>
<td>Weights to use global variables</td>
</tr>
<tr>
<td>part of design.</td>
<td>They're not: useless</td>
</tr>
</tbody>
</table>

**Static Factory methods:**

- B珞's Effective Java
- Reads: Chap 1
- Small: Do one thing, Strong cohesion
- reading materials: refactoring
- Side effect: modify the var don't pass into parameter.
- more calls, less efficiency
- Avoid output argument: E.g. String name = "" + name.toUpperCase()

**Design:**

- UML diagrams
- Documentations
- Data Structures
- Database layouts
- Efficiency
- Requirement

**Design Qualities:**
Coherence: focus on one thing.

Coupling: relationship between 2 modules.

Two connections: Static/dynamic, "fan" interacting with each other, "spoke" defined in one another's member or internal.

Most readability, flexibility, maintainence more important.

Encapsulating operations on data, why is this one of DRY principle, "preserve what".

Level of perspective in OO:

Conceptual:
- Domain-level, problem-level -- not yet considering solutions.

Specification:
- Solution-level, but an abstract view, interface, not internal implementation.
Levels of perspective in OO

Conceputal specification & Implementation.

Domain-level, Solution-level but on abstract view

problem-level, what considering solution, Interfaces not intend

- objects responsibility

```
Book
- title:
- author:

BankAccount
- check: boolean
- password: string
```

So what's an objection.

The graph depends on what level you're using, where you're at developer.

Conceputal:

- define in problem-domain
- not define in direction relationship

Specification:

- define in Specifying (mostly)
- not define in Conceptual level commonly

Implementation:

Designers work at specification (mostly).

DRY - don't repeat yourself.

Abstract Class vs Interface. (both are types).

- mixed with interface:
- unimplement

Can have static fields

- method:
- signature:
- Static, public, final
- one reference, can change.

Can have coded

- at least one
- must

- Shared behaviors - share other unique behaviors
Polymorphism: runtime. Type's method be invoked by high level abstract class.

Inheritance: Diamonds.

Separate Abstractions: Secure?

```
Account + 1..*Customer

| Credit | Indiv | Trust | Flexible |
```

1333 Mel 205 6pm

\[
\begin{align*}
A + f_{CD} &= f_t \\
A' &= f_{CD} = f_t
\end{align*}
\]
Composition / Aggregation.

HAS-A

PART-OF

On insistence = one class

advantage

owner

没有解决复杂间关系

1:1 account

* : 0...* accounts  diagram
capture relationship

3. perspectives not the 2

Readings chapter 2 in class textbook

class diagrams developed in

a series of detail

object domain level, high level structural things

class diagram

two classes association
dependency

multiplicity, it makes no sense that "book is a copy of copy"

Generalization and inheritance.
null
Stereotypes

guilemet = <<

[Diagram of a model with arrows and labels: Interface, Isingulf, etc.]

david matthew

Event Constraints

use sparingly

 stash - not implement, drop other relationship

 association class

 composite design pattern

[Diagram with labeled boxes: Switch, Line, Console, Phone, etc. with arrows indicating relationships and numbers like 1:* and 2:*]
observers
observer pattern