Four More GRASP Principles

Shawn Bohner
Office: Moench Room F212
Phone: (812) 877-8685
Email: bohner@rose-hulman.edu

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Q1

GRASP II – And Furthermore…

- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

Polymorphism

Problem:

- How do we handle alternatives based on type?
  - ✓ Chained *if*s and lots of *switch* statements are a bad code smell → new types require finding conditions and editing
- How do we create pluggable software components
  - ✓ Pluggable components require swapping one module for another without changing surrounding design

Solution:

- When related alternatives vary by type, assign responsibility to the types for which the behaviors varying.
  - ✓ Use subtypes and polymorphic methods
  - ✓ Eliminates lots of conditional logic based on type
  - ✓ Corollary: Avoid *instanceof* tests

Polymorphism Example

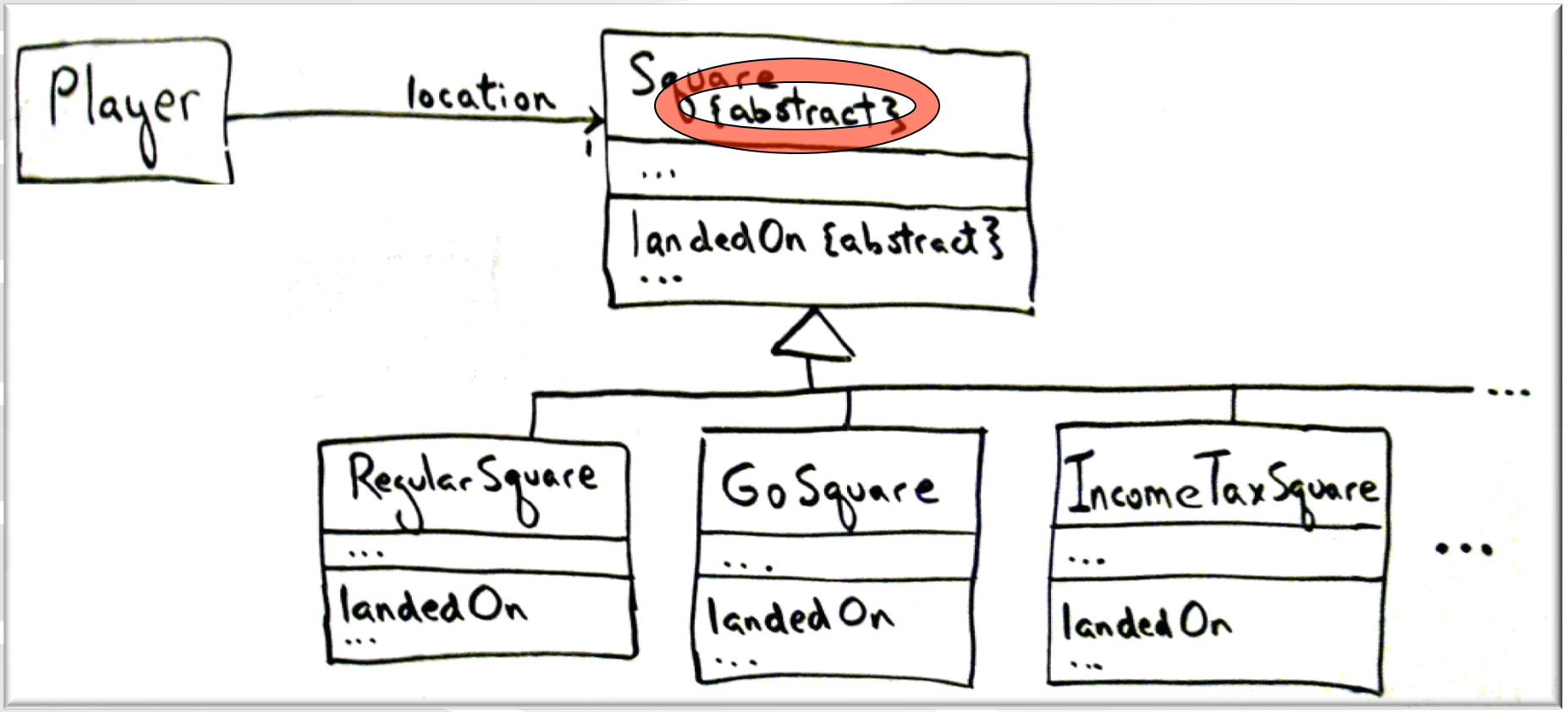
Bad:

```
switch (square.getType()) {
case GO:
        …
case INCOME_TAX:
        …
case GO_TO_JAIL:
        …
default:
        …
}
```

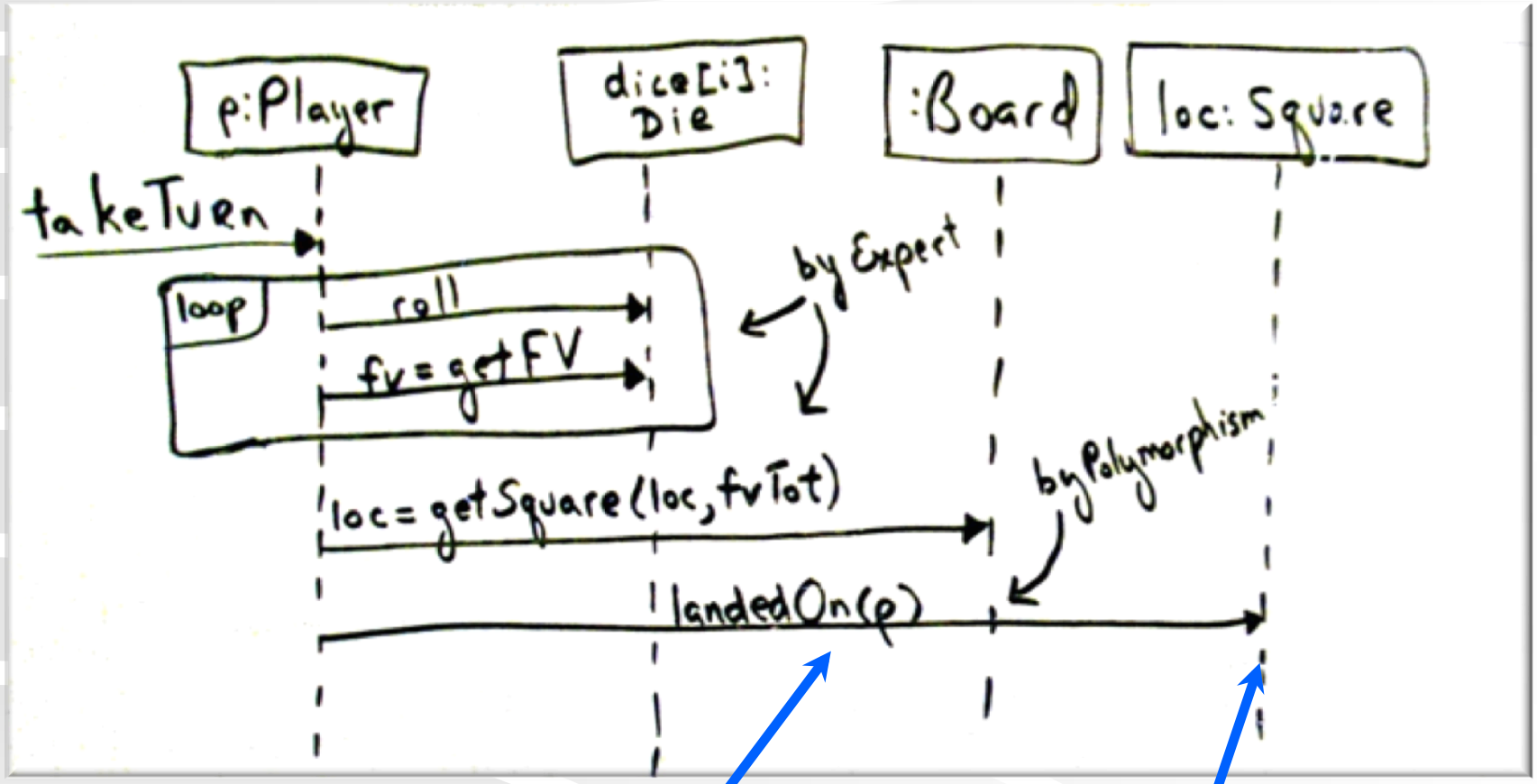
What happens when we need to add other sorts of squares in future iterations?

Solution: Replace switch with polymorphic method call

Monopoly Polymorphism Example



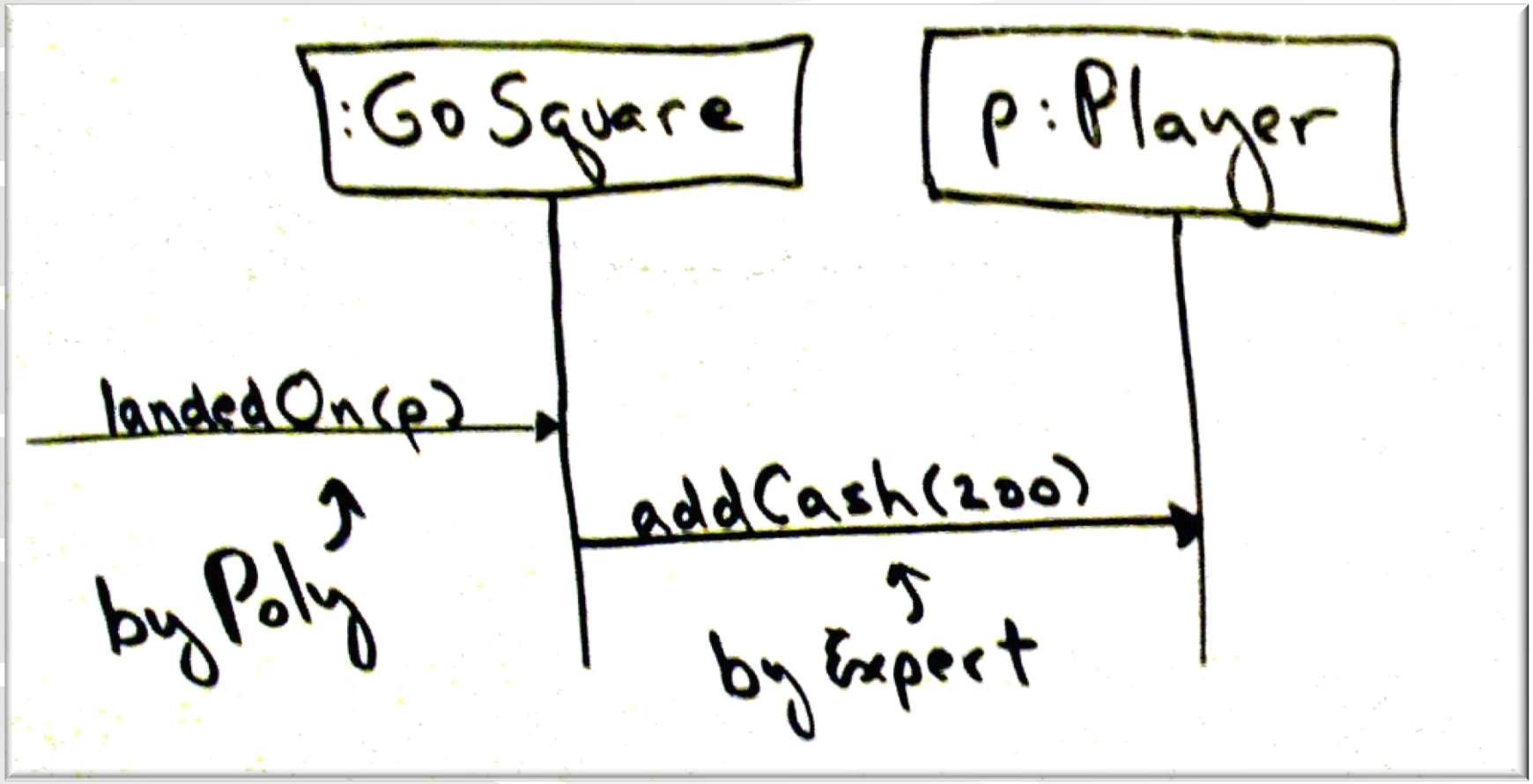
Polymorphism Example (continued)



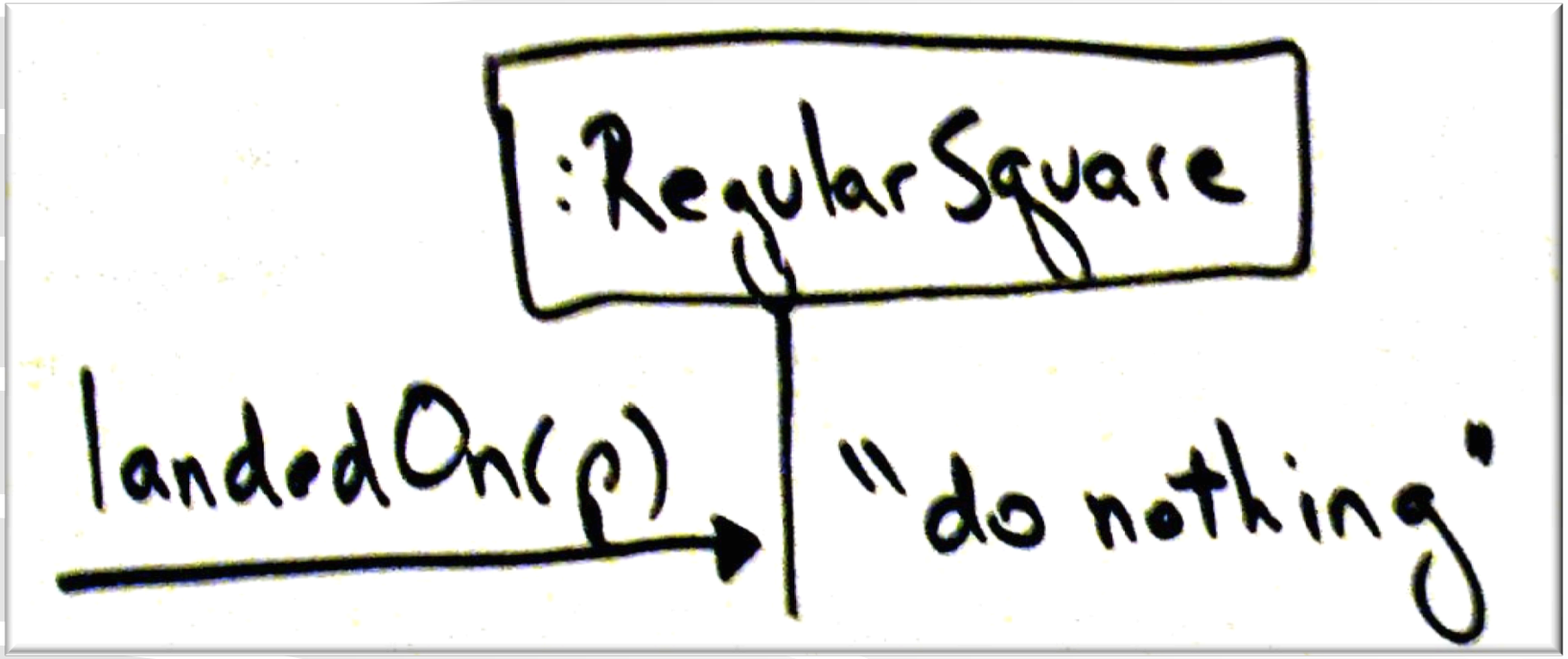
Make abstract unless clear default behavior

Details of polymorphic method drawn separately

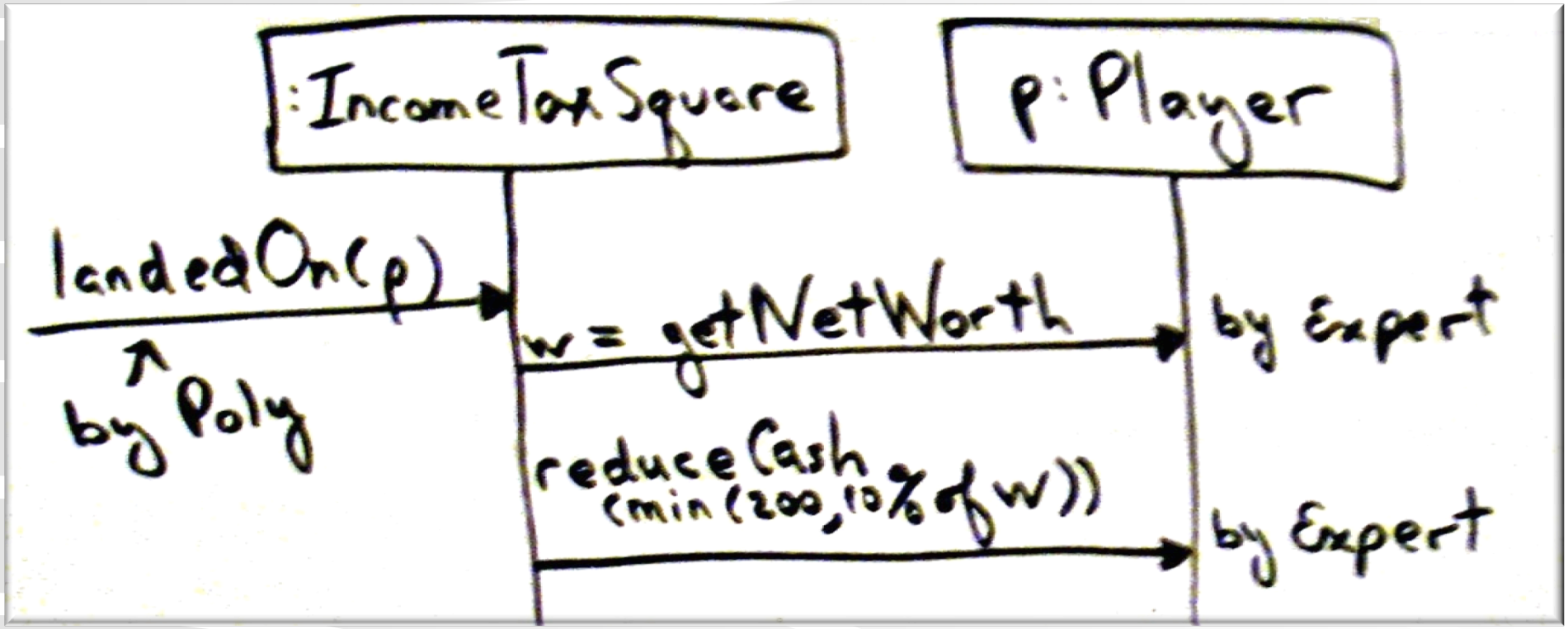
Monopoly Polymorphism Example



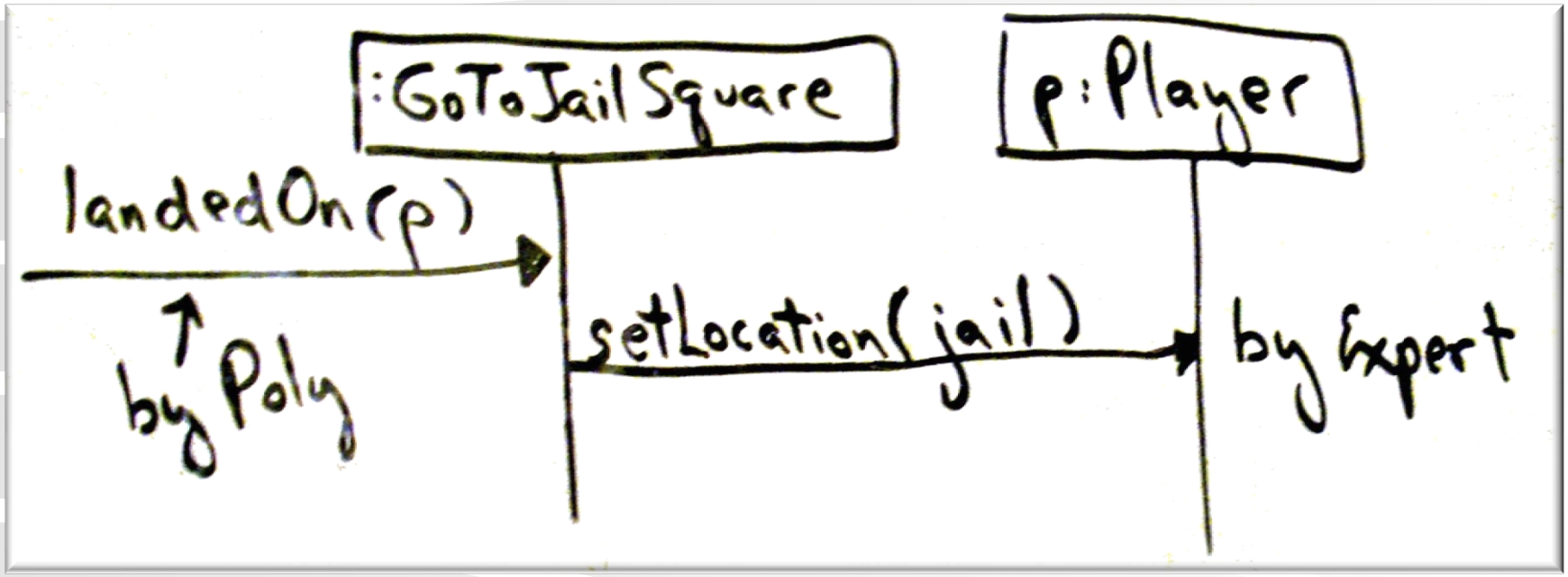
Monopoly Polymorphism Example



Monopoly Polymorphism Example

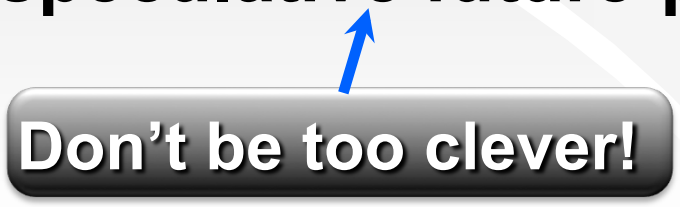


Monopoly Polymorphism Example



Polymorphism Observations

- Using polymorphism indicates that Piece class not needed since it's a proxy for the Player
- A design using Polymorphism can be easily extended for new variations
- When should supertype be an interface?
  - Don't want to commit to a class hierarchy
  - Need to reduce coupling
- Contraindication: Polymorphism can be over used – speculative future-proofing



Team Polymorphism

Q6 Working with your project team, **identify a situation** in your project **where Polymorphism might be applicable**.

If no such situation exists, try to come up with an extension to your system that might use Polymorphism.

What method(s) would behave differently for the different subtypes?

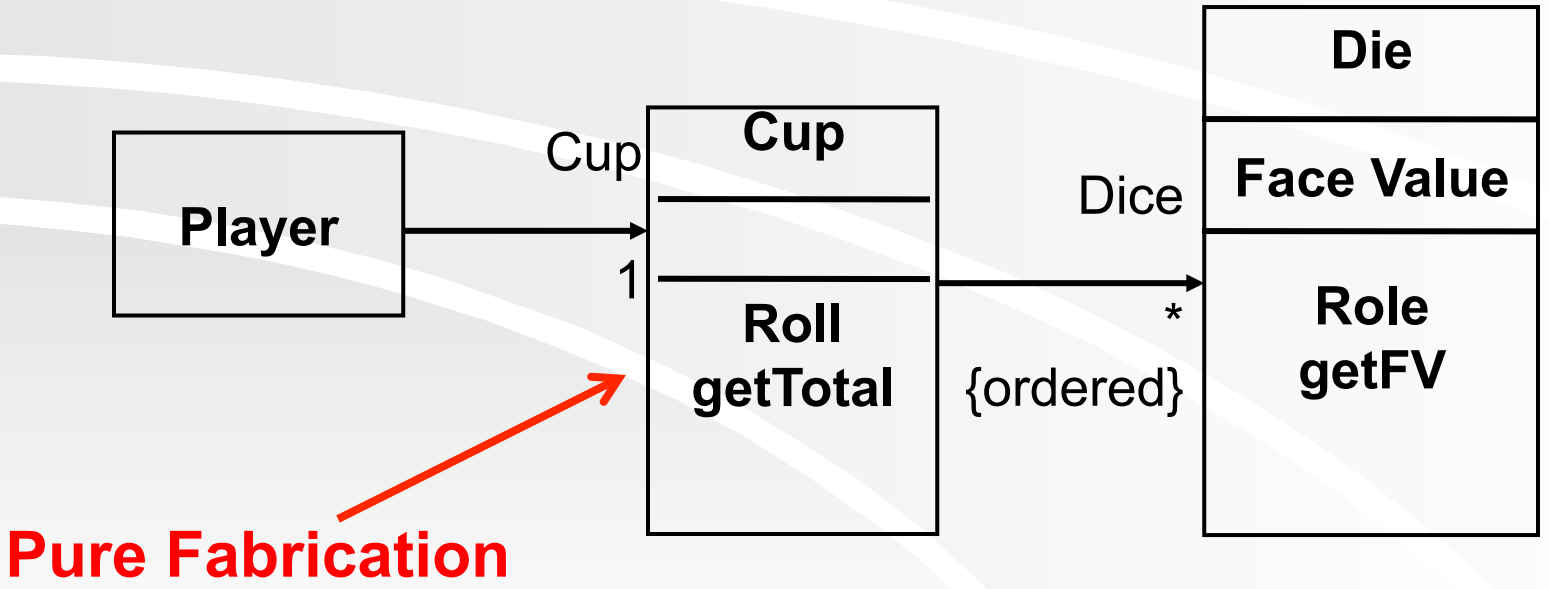
Pure Fabrication

- **Problem:**
  What object should have responsibility when solutions for low representation gap (like Info. Expert) lead us astray (i.e., into high coupling and low cohesion)

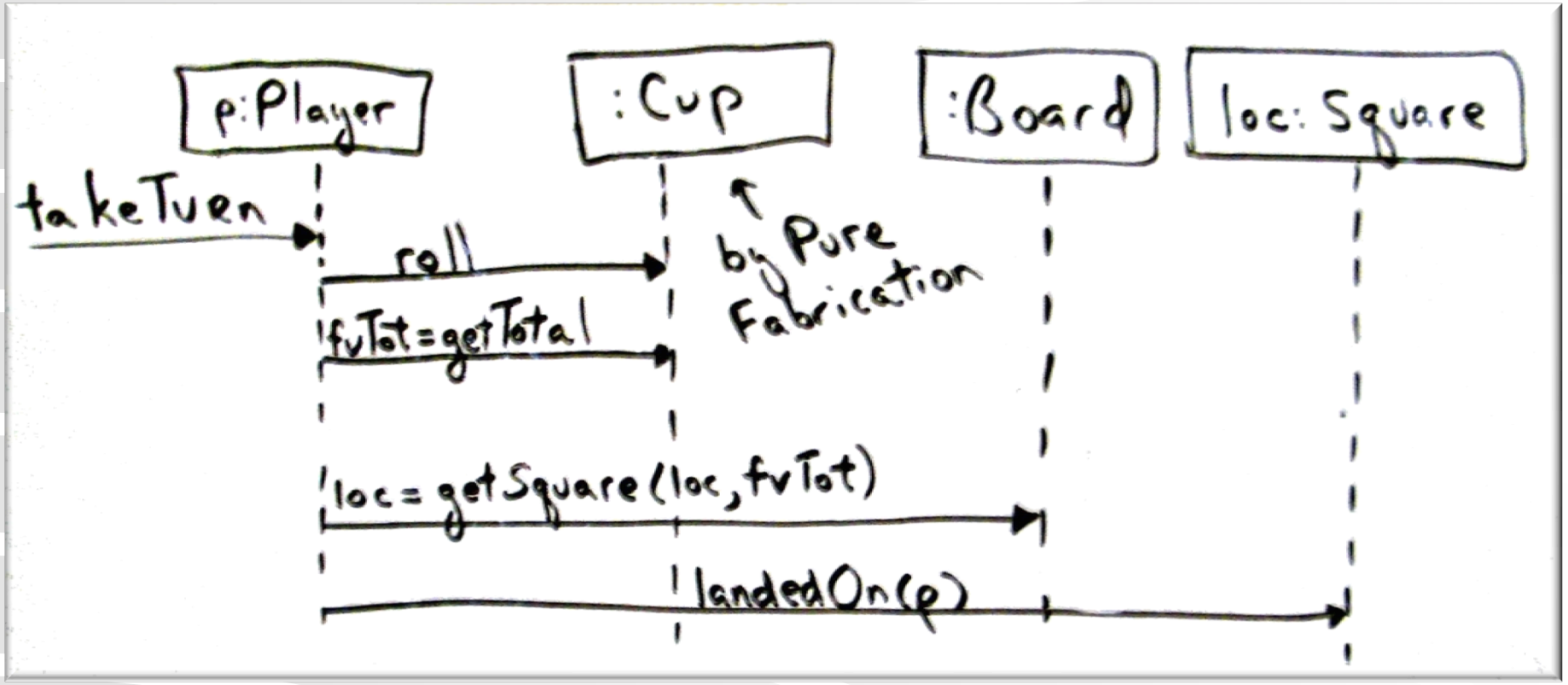
- **Solution:**
  Assign a cohesive set of responsibilities to an artificial (not in the domain model) class

Monopoly Pure Fabrication Example

- How do we model the player rolling the dice?
  - If Player rolls dice, then dice rolling behavior not very reusable
- How do we provide something that would be more reusable?



Monopoly Pure Fabrication Example



Common Design Strategies

- **Representational decomposition**
  - Lowering the representation gap (noun-based)
- **Behavioral decomposition**
  - Centered around behaviors (verb-based)

Pure Fabrications are often behavioral decompositions

Pure Fabrication Observations

- **Benefits:**
  - Higher cohesion
  - Greater potential for reuse
- **Contraindications:**
  - Can be abused to create too many behavior objects
  - Watch for data being passed to other objects for calculations

Keep operations with data unless you have a good reason not to

Cartoon of the Day

Used with permission. http://notinventedhe.re/on/2009-10-13

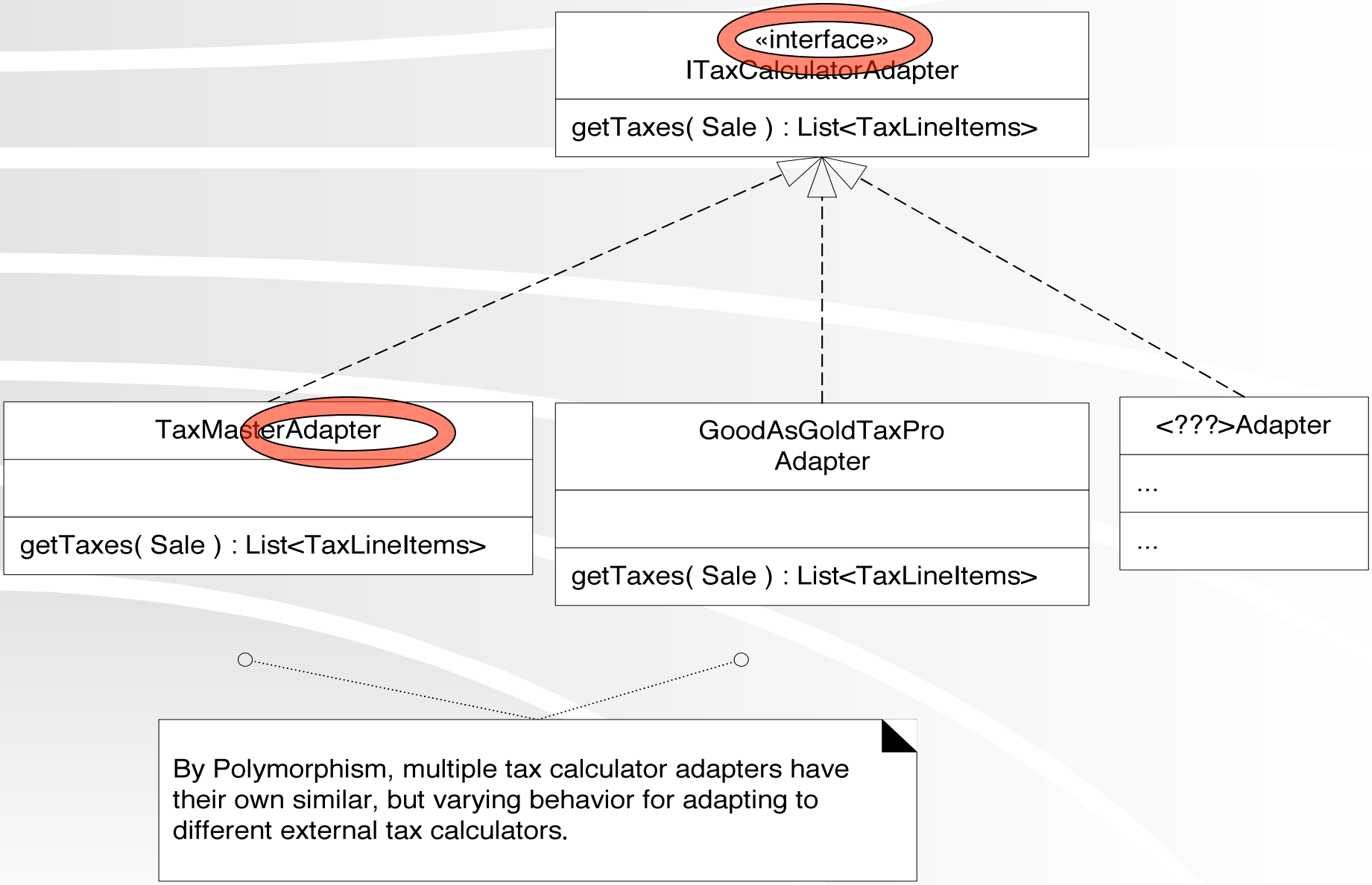
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Indirection

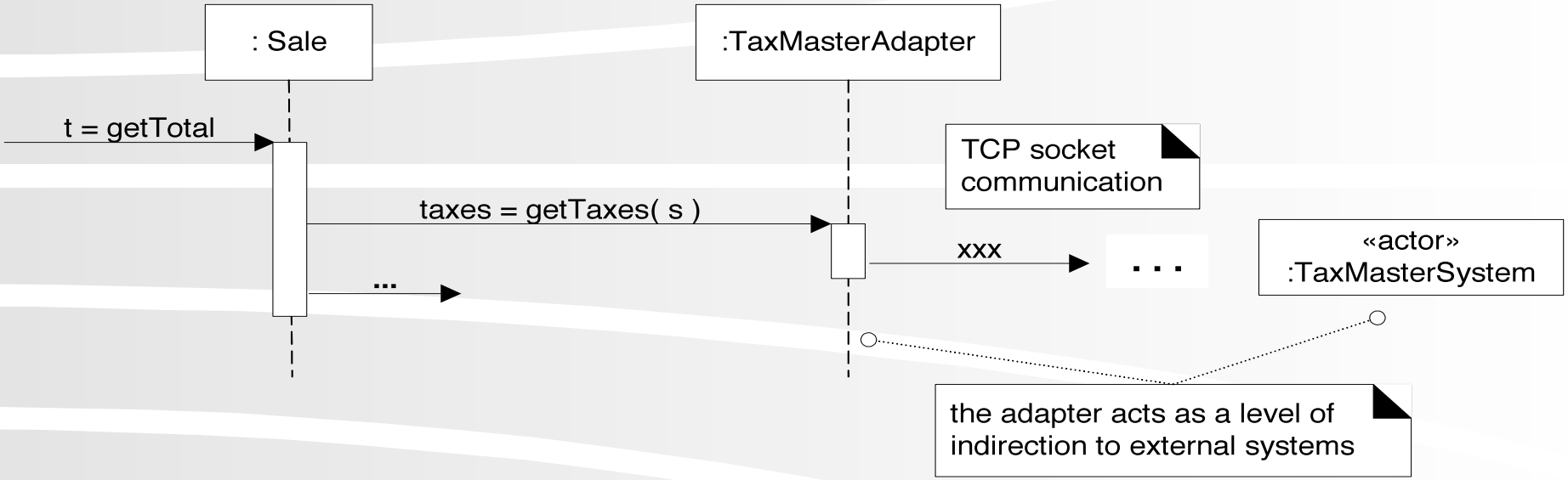
- **Problem:**
  - **Where do we assign responsibility if we want to avoid direct coupling between two or more objects?**
- **Solution:**
  - **Assign responsibility to an intermediate object to mediate between the other components**

There is no problem in computer science that cannot be solved by an extra level of indirection.
— David Wheeler

Indirection & Polymorphism Example



NexGen POS Indirection Example



- TaxMasterAdapter is a Pure Fabrication offering a level of Indirection
- Shields client (Sale) from variable server (proprietary tax calculator system)

Protected Variation

- **Problem:**
  How do we design objects and systems so that instability in them does not have undesirable effects on other elements?
- **Solution:**
  Identify points of predicted instability (variation) and assign responsibilities to create a stable interface around them

Q12,13

Protected Variation Pervasive in Computing

- Virtual machines and operating systems
- Data-driven designs (e.g., configuration files)
- Service lookup (URLs, DNS)
- Uniform access to methods/fields (Ada, Eiffel, C#, Objective-C, Ruby, …)
- Standard languages (SQL)
- Liskov Substitution Principle

Protected Variations: Observations

- When to use it?
  - Variation point is a known area where clients need to be protected from variable servers
  - Evolution point is an area where future variation may occur
- Should we invest in protecting against future variation?
  - How likely is it to occur? If it is, then should probably use PV now
  - If unlikely, then should probably defer using PV

Law of Demeter, or "Don't Talk to Strangers"

- Within a method, messages should only be sent to:
  - *this*
  - a parameter
  - field of *this*
  - element in collection of field of *this*
  - new objects

Better: Don't talk to strangers **who seem unstable**

This guideline warns against code like:
sale.getPayment().getAccount().getAccountHolder()

Protected Variations Observations

- Benefits (if we guessed variation points correctly):
  - Extensions easy to add
  - Can plug in new implementations
  - Lower coupling
  - Lower cost of change
- Risk: watch out for speculative future-proofing

Protected Variations by Other Names

- *Information hiding* [Parnas72]
  - "We propose instead that one begins with a list of difficult design decisions which are likely to change. Each module is then designed to hide such a decision from the others."
- *Open-Closed Principle* [Meyer88]
  - "Modules should be both open (for extension …) and closed (… to modification[s] that affect clients)"

Homework and Milestone Reminders

- Read Chapter 26
- Homework 6 – More GRASP on Video Store Design
  - Due by 5:00pm Tuesday, January 26th, 2010
- Milestone 4: Patterns and Detailed Design, with some Iteration 2 on the Side
  - Due by 11:59pm Friday, January 29th, 2010