CSSE 374: More GRASP'ing for Object Responsibilities

Shawn Bohner

Office: Moench Room F212

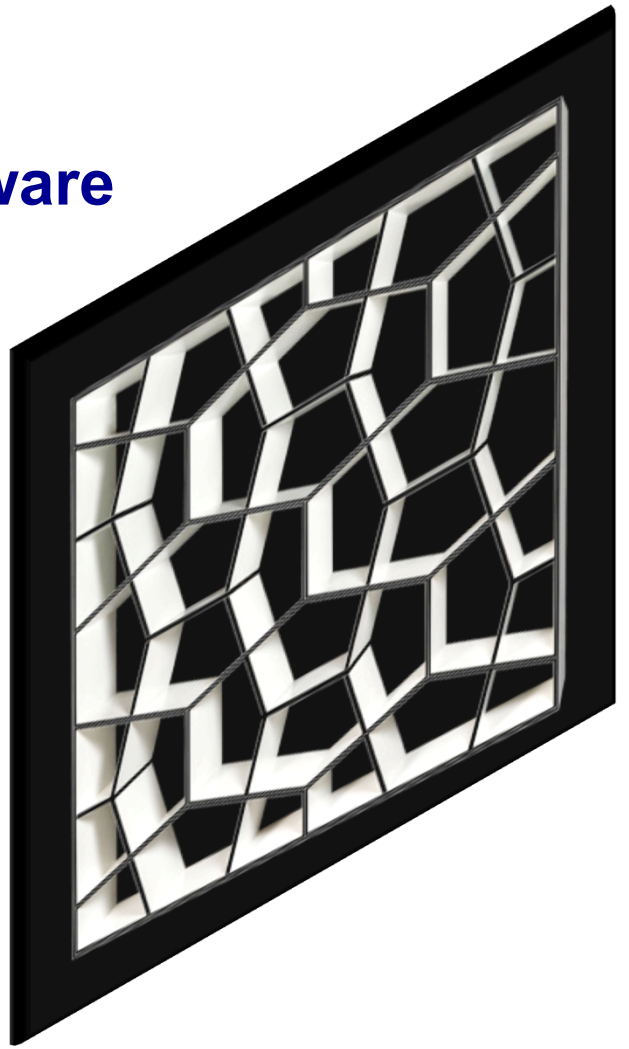
Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Learning Outcomes: Patterns, Tradeoffs

Identify criteria for the design of a software system and select patterns, create frameworks, and partition software to satisfy the inherent trade-offs.



- Two more Design Studios
- Four more GRASP Patterns:
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variation

Design Studios

Objective is to share your design with others to communicate the approach or to leverage more eyes on a problem.

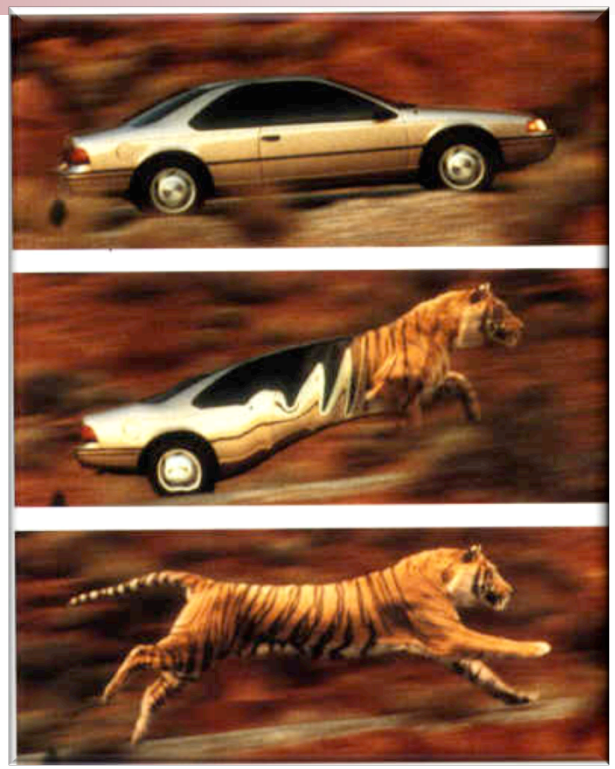
- Minute or so to set up…
- 5-6 minute discussion
- 1-2 minute answering questions

1. Team 2.4 – Rovio
2. Team 2.1 – GUI Evaluation Tool

GRASP II – And Furthermore…

- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

Polymorphism



Problem:

- How do we **handle alternatives based on type**?
- How do we **create pluggable software components**?

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 1/6

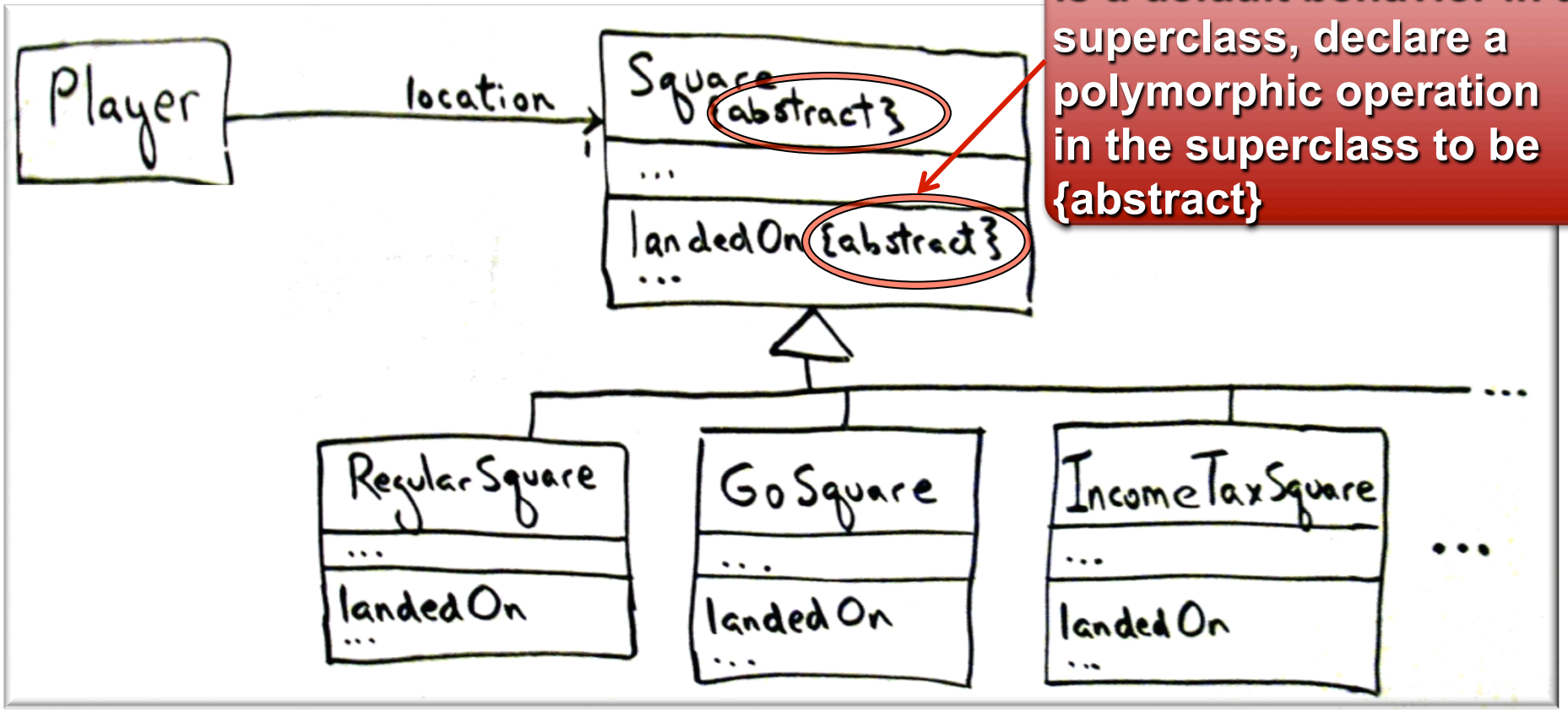
Bad:

```
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?

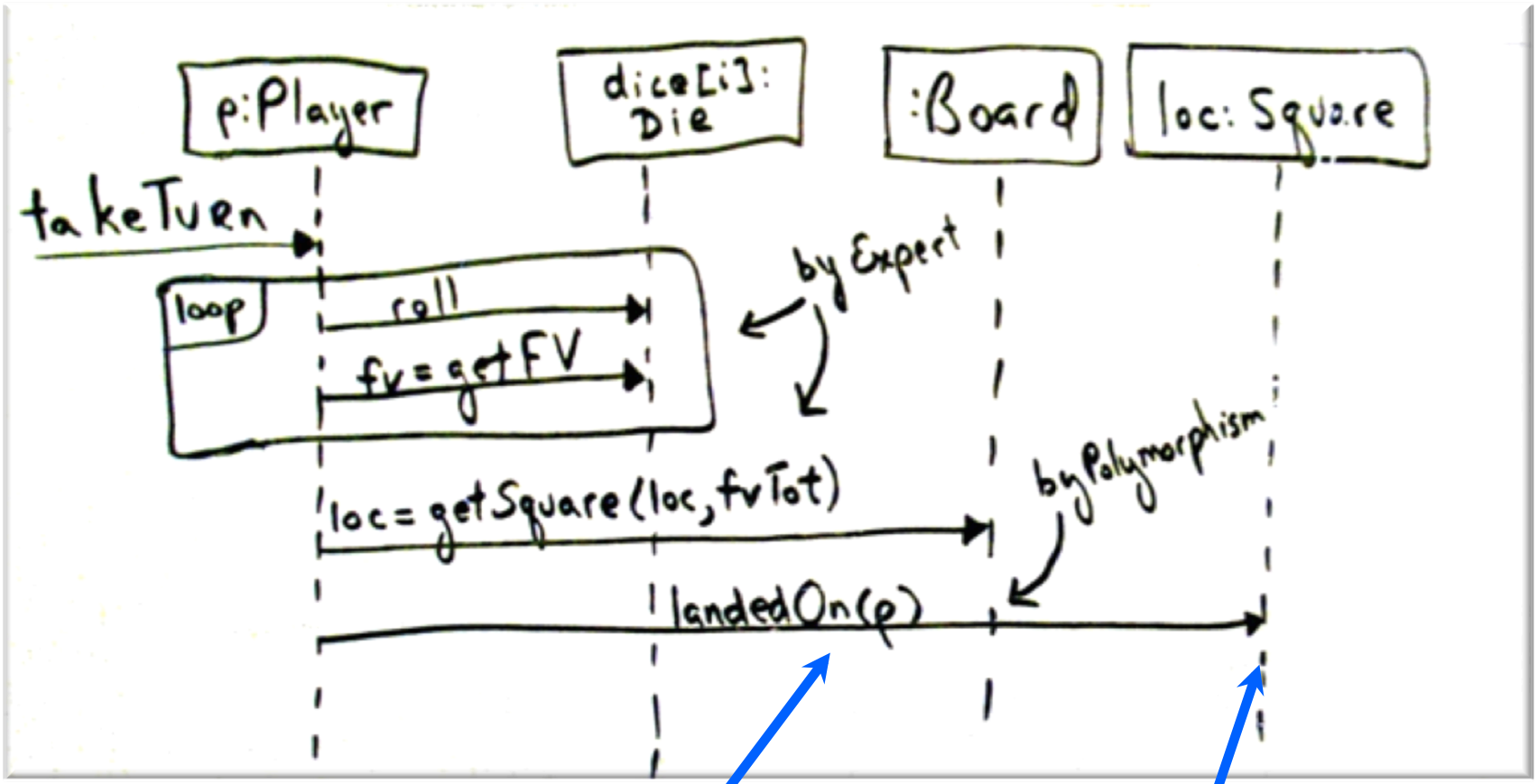
Solution: Replace switch with polymorphic method call

Polymorphism Example 2/6



Guideline: Unless there is a default behavior in a superclass, declare a polymorphic operation in the superclass to be {abstract}

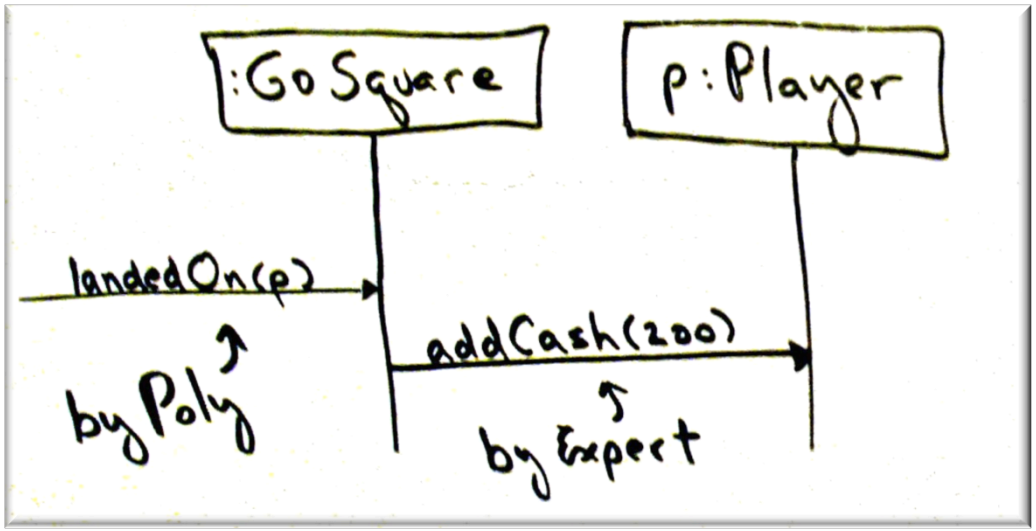
Polymorphism Example 3/6

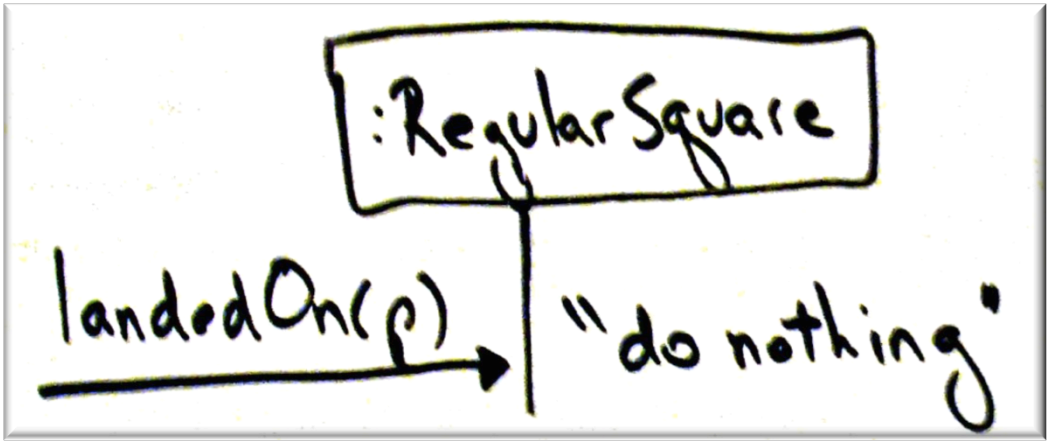


Make abstract unless clear default behavior

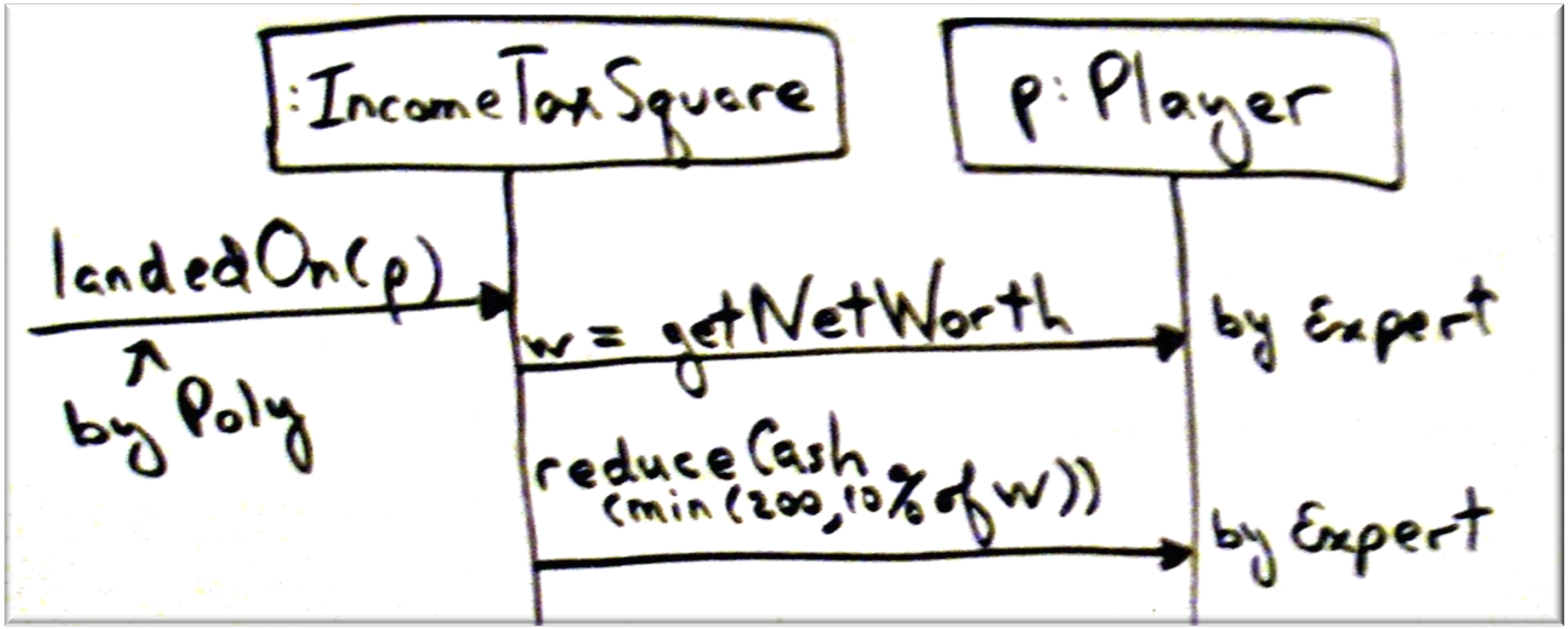
Details of polymorphic method drawn separately

Polymorphism Example 4/6

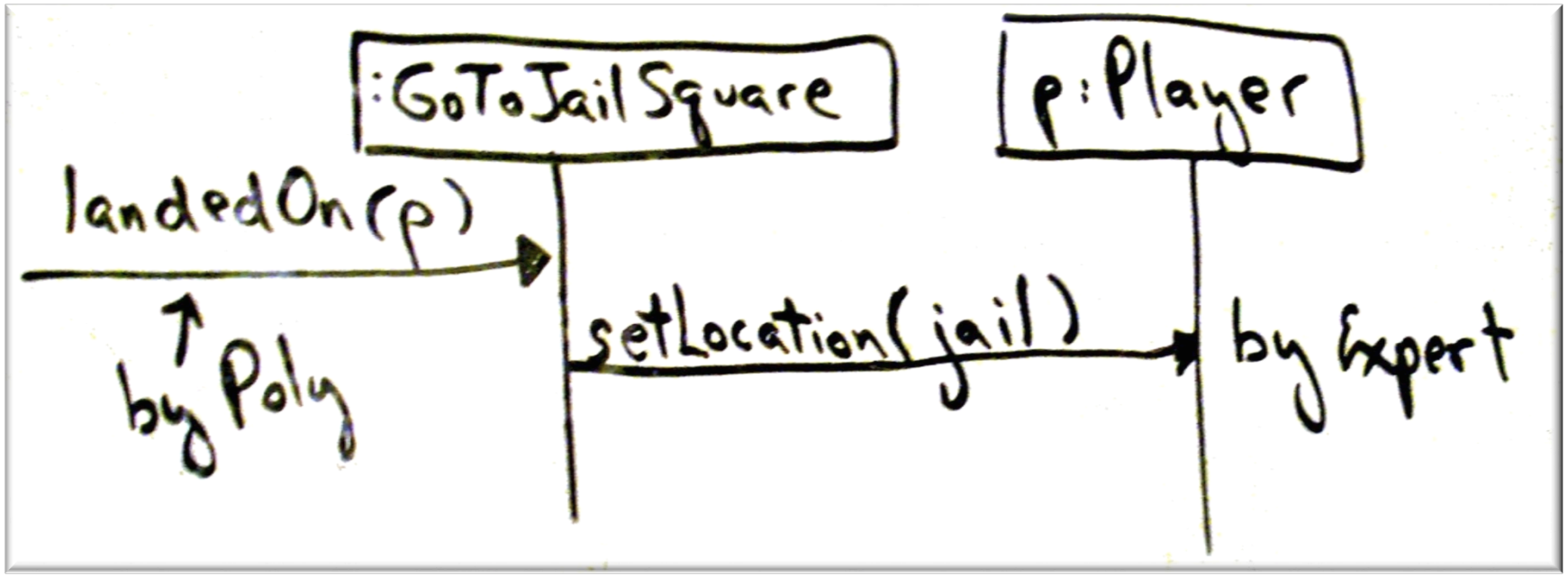




Polymorphism Example 5/6



Polymorphism Example 6/6



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"

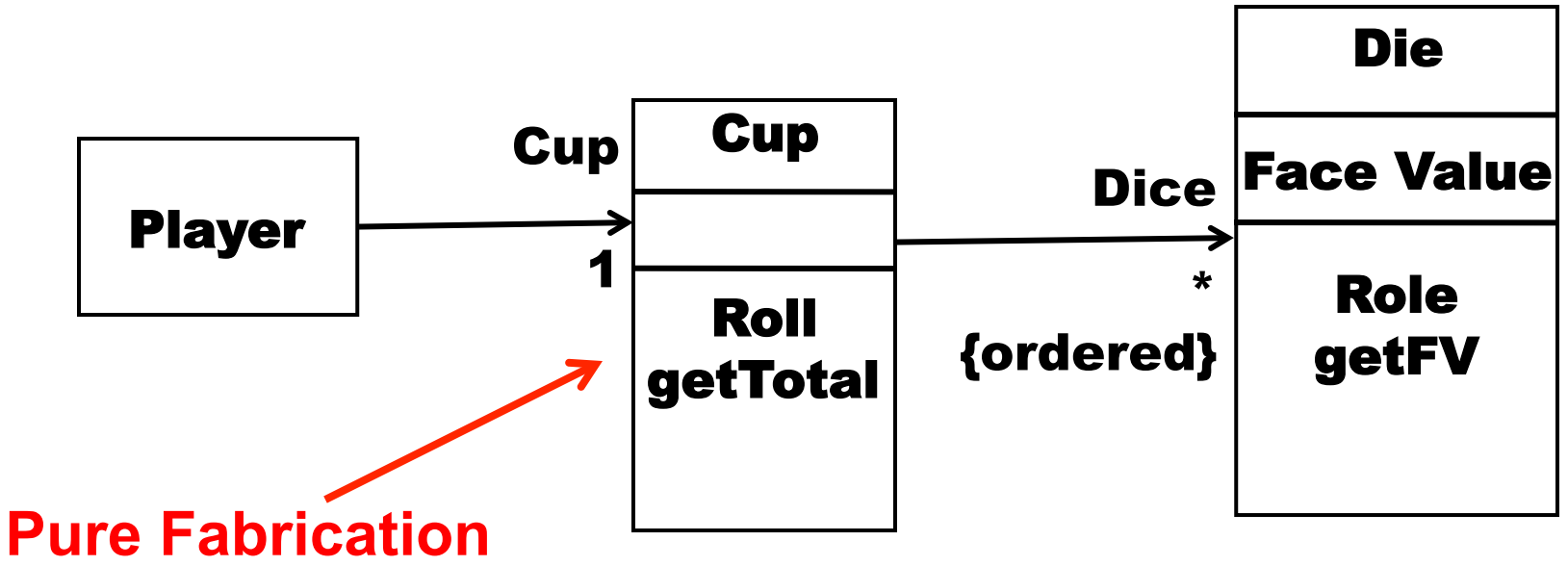
Don't be too clever!

Pure Fabrication

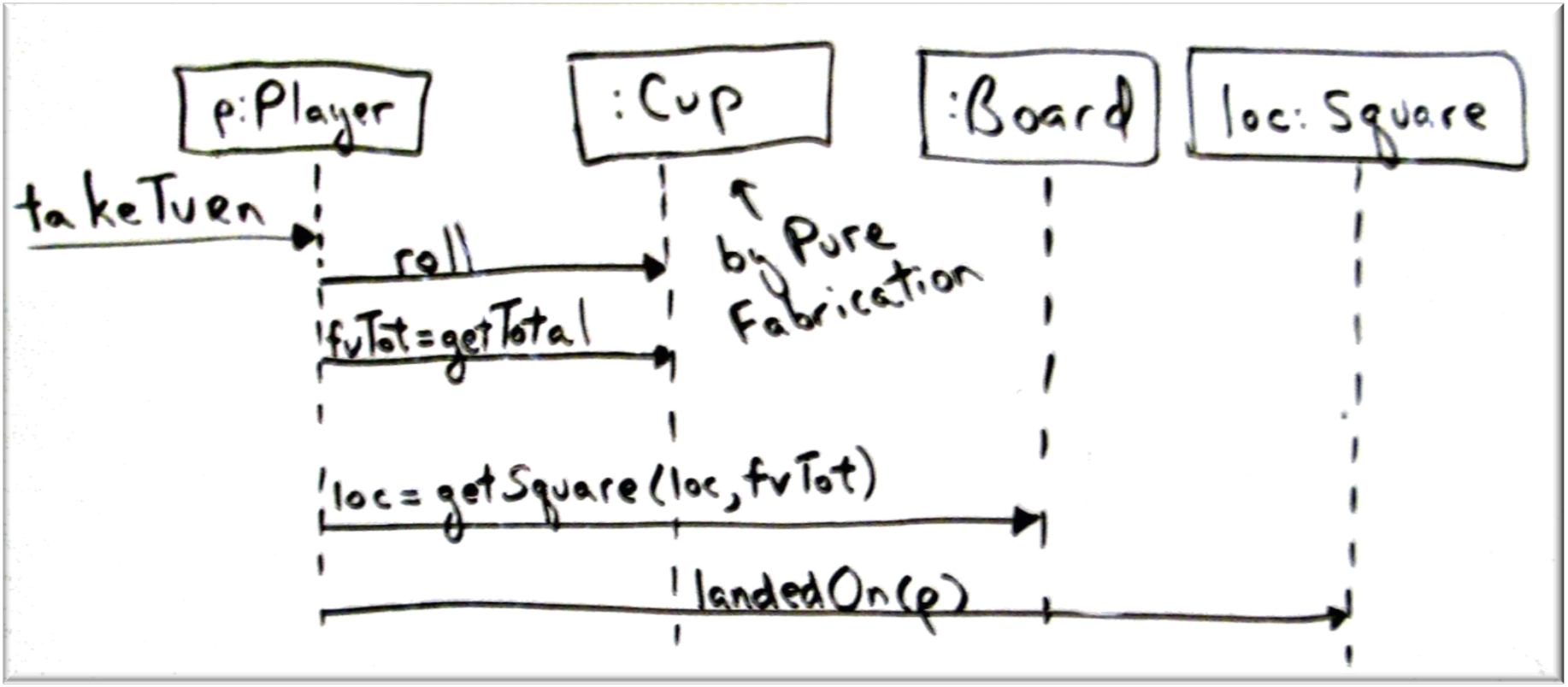
- **Problem**:
  What object should have responsibility when solutions for low representation gap (like Info. Expert) lead us astray (i.e., into high coupling and low cohesion)

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

Pure Fabrication Example 1/2



Pure Fabrication Example 2/2



Common Design Strategies

- Representational decomposition
  - Based on what they represent in domain
  - Lowering the representation gap (noun-based)
- Behavioral decomposition
  - Based on what they do!
  - 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

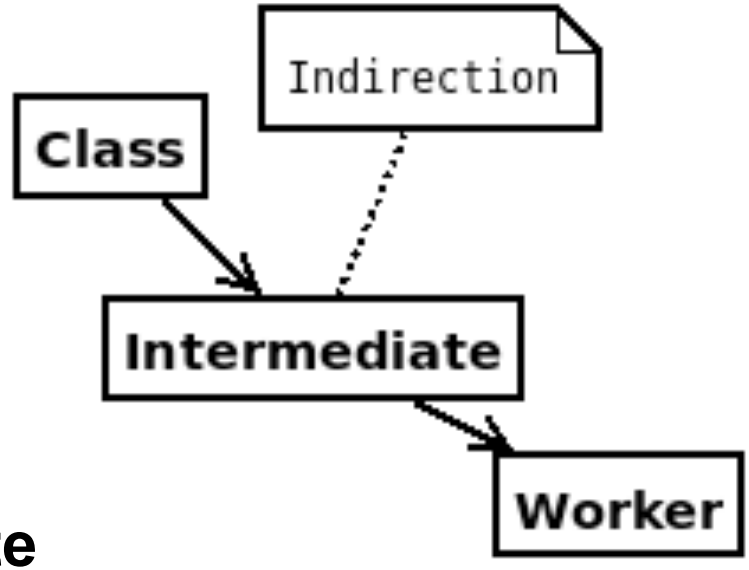
Keep operations with data unless you have a good reason not to

Cartoon of the Day

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

Indirection

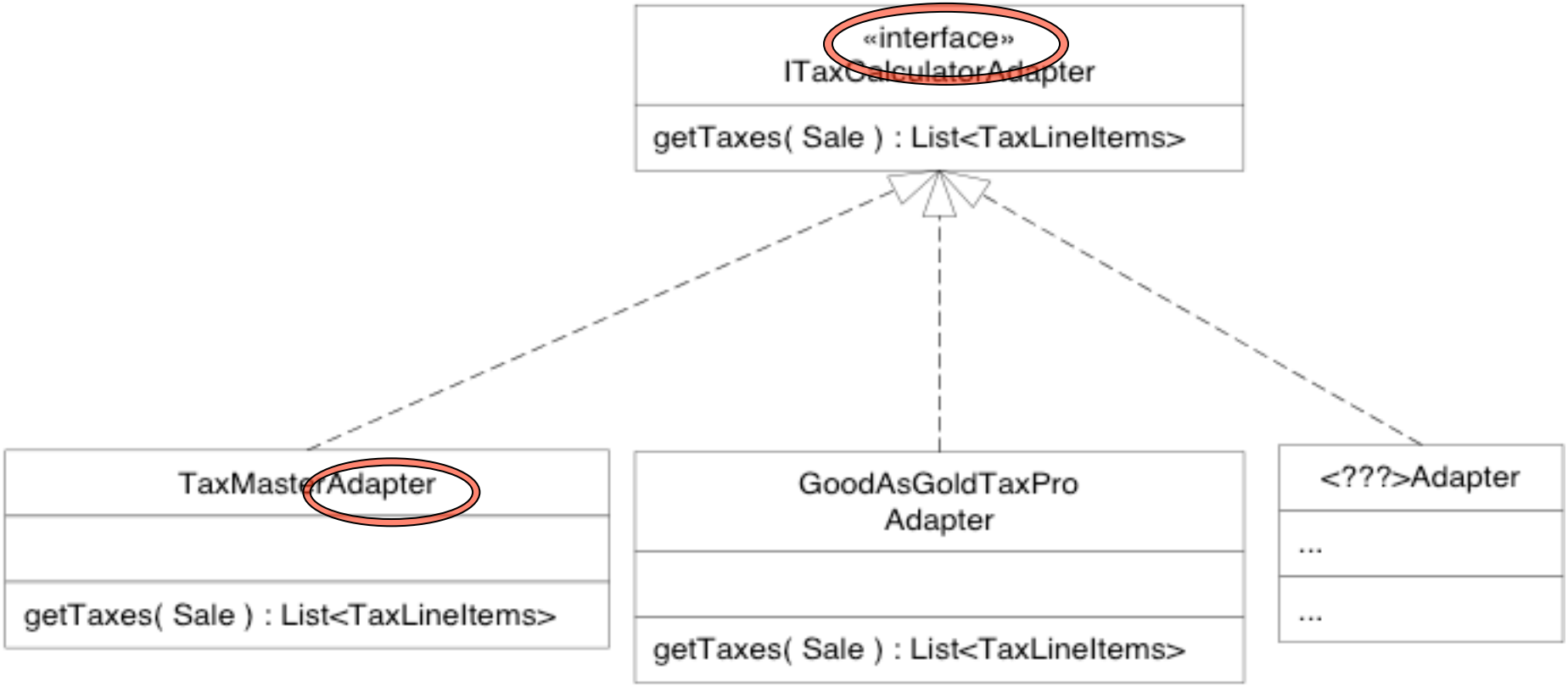
- **Problem**:
  - How to assign responsibility in order to avoid direct coupling that is undesirable?
- **Solution**:
  - Assign responsibility to an intermediate object to mediate between the other components



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

— David Wheeler

Indirection & Polymorphism Example



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

Protected Variations: Observations

When to use it?

- Variation point – a known area where clients need to be protected from variable servers
- Evolution point – 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

Protected Variations by Other Names

Information hiding [David Parnas '72]

- "… a list of difficult design decisions which are likely to change. Each module is then designed to hide such a decision from the others."

Liskov Substitution Principle [Barbara Liskov '87]

- Methods of a subtype must have (at least) the expected behavior of overridden methods

Open-Closed Principle [Bertrand Meyer '88]

- Modules should be both open for extension and closed to modification[s] that affect clients

Law of Demeter

Don’t talk to strangers who seem <u>unstable</u>

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

Homework and Milestone Reminders

- Read Chapter 26 on Gang of Four Design Patterns
- Milestone 4 – Junior Project Design with More GRASP'ing
  - Due by 11:59pm on Friday, January 28th, 2011
- Coming Homework 5 – BBVS Design using more GRASP Principles
  - Due by 11:59pm Tuesday, January 25th, 2011