The University of Iowa

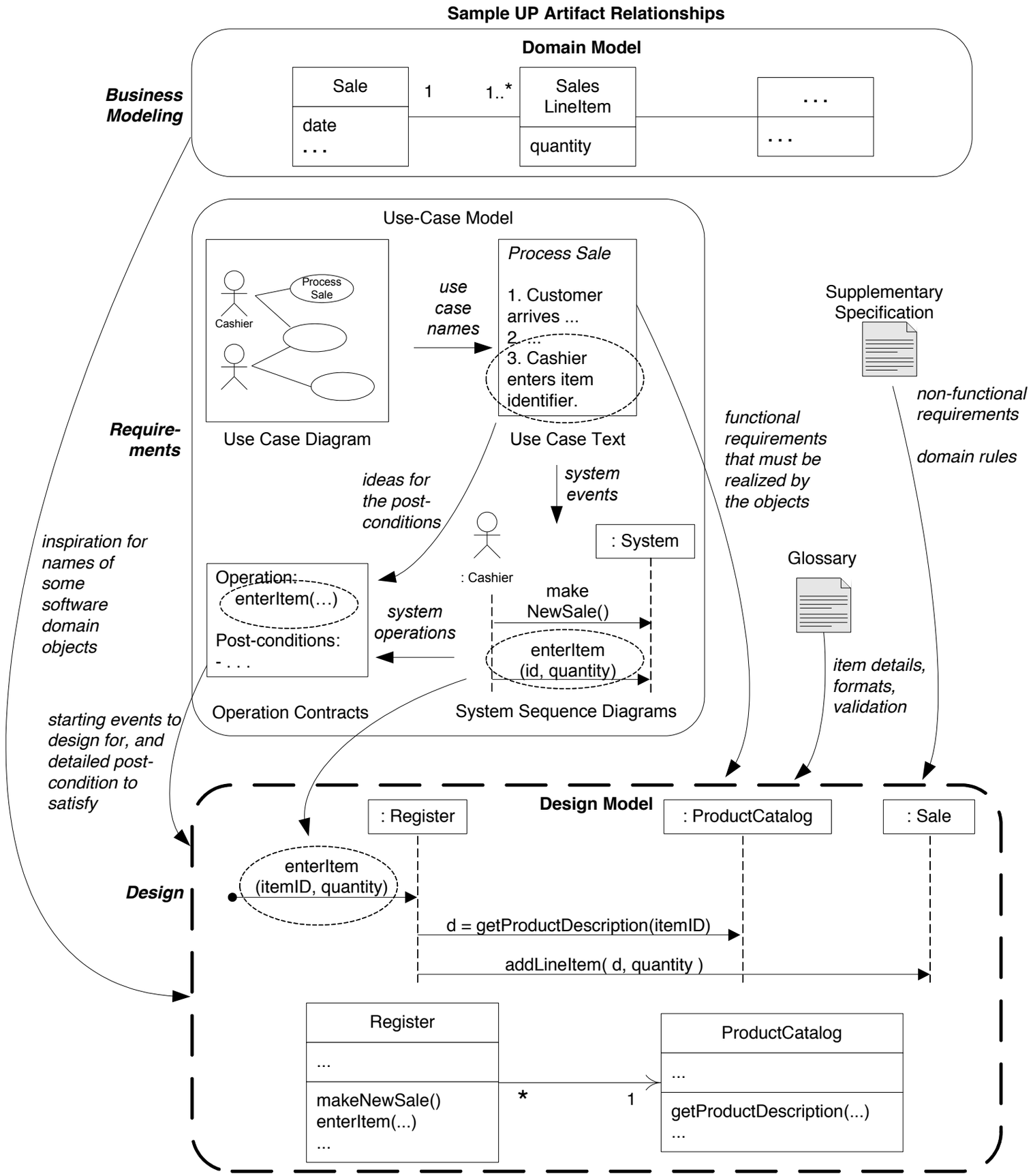
CS:2820 (22C:22)
Object-Oriented Software Development

Spring 2015

Design Patterns

by

Cesare Tinelli


Sample UP Artifact Relationships
Domain Model
Business Modeling
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
Use-Case Model
Require-ments
Process Sale
Cashier
Use Case Diagram
use case names
Process Sale
1. Customer arrives ...
2. ...
3. Cashier enters item identifier.
Use Case Text
Supplementary Specification
non-functional requirements
domain rules
functional requirements that must be realized by the objects
ideas for the post-conditions
system events
: System
: Cashier
make NewSale()
enterItem (id, quantity)
Operation:
enterItem(…)
Post-conditions:
- . . .
system operations
Operation Contracts
System Sequence Diagrams
Glossary
item details, formats, validation
inspiration for names of some software domain objects
starting events to design for, and detailed post-condition to satisfy
Design Model
Design
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
d = getProductDescription(itemID)
addLineItem(d, quantity)
Register
...
makeNewSale()
enterItem(...)
...
*
1
ProductCatalog
...
getProductDescription(...)
...

Responsibility-Driven Design

Designing systems in terms of object responsibilities, roles, and collaborations

- Action responsibilities
- Knowledge responsibilities

Design Patterns

Named and well-known problem/solution pairs that can be applied in new contexts, with advice on how to apply them in novel situations

Design patterns help during responsibility assignment in RDD

Advantages of Patterns

- They support chunking and incorporating a concept into our understanding and memory
- They facilitate communication

Established Design Patterns

- Creator
- Information Expert
- Low Coupling
- Controller
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variation
- …

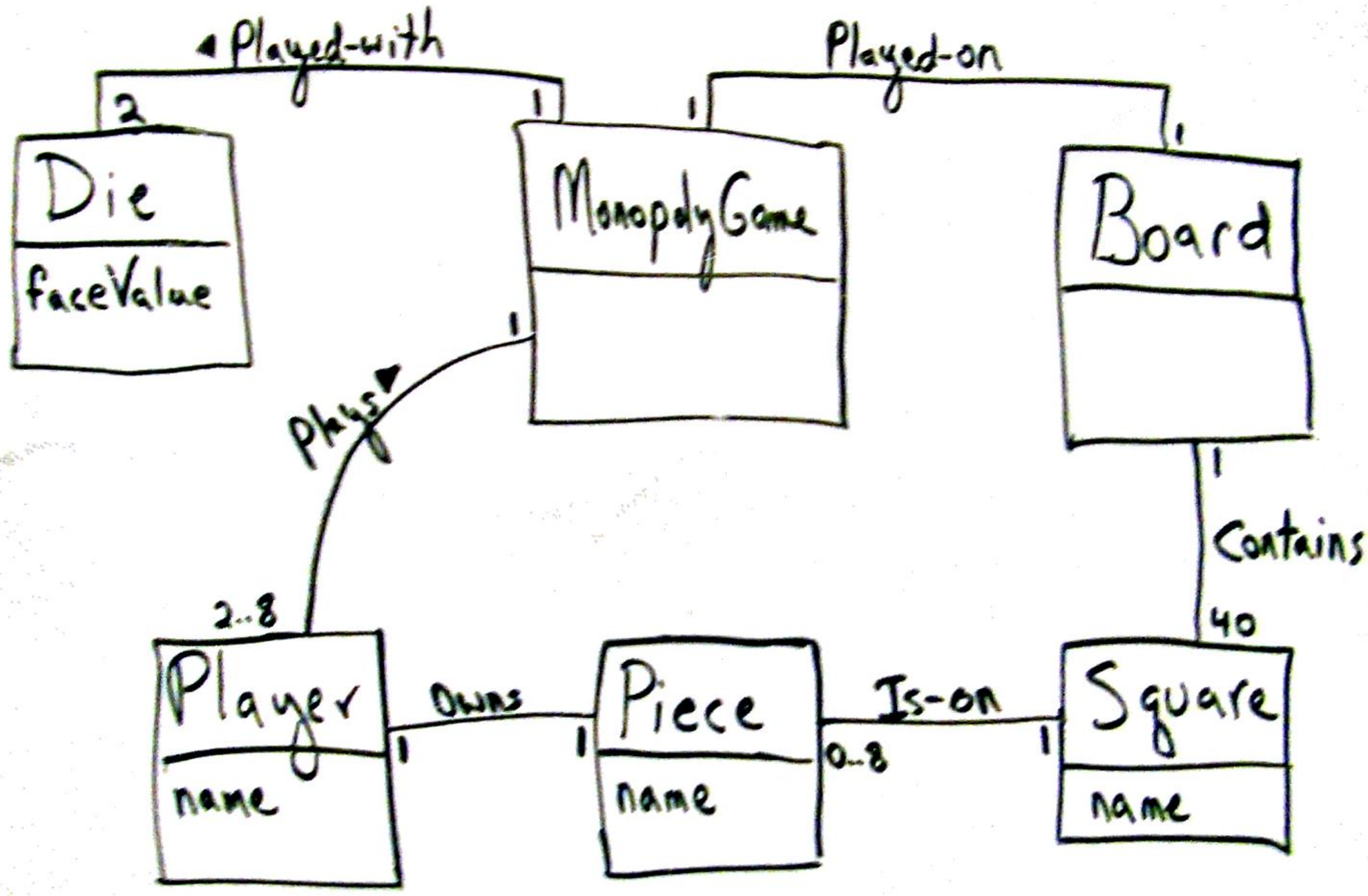
Creator Pattern

P: Who creates a new instance of some class A?

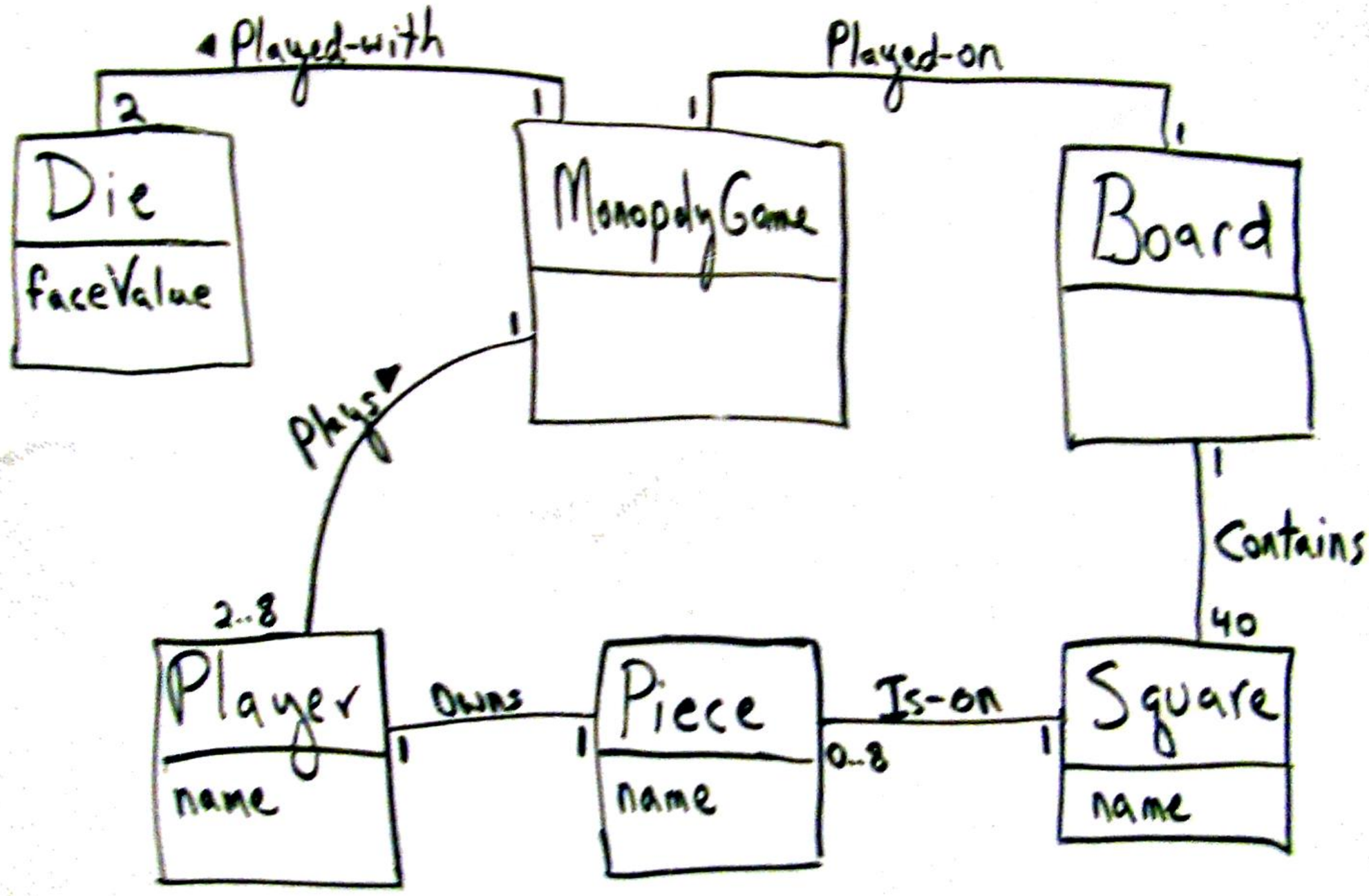
S: Class B get the responsibility if:

- B "contains" or compositely aggregates A,
- B closely uses A, or
- B has the initializing data for A instances

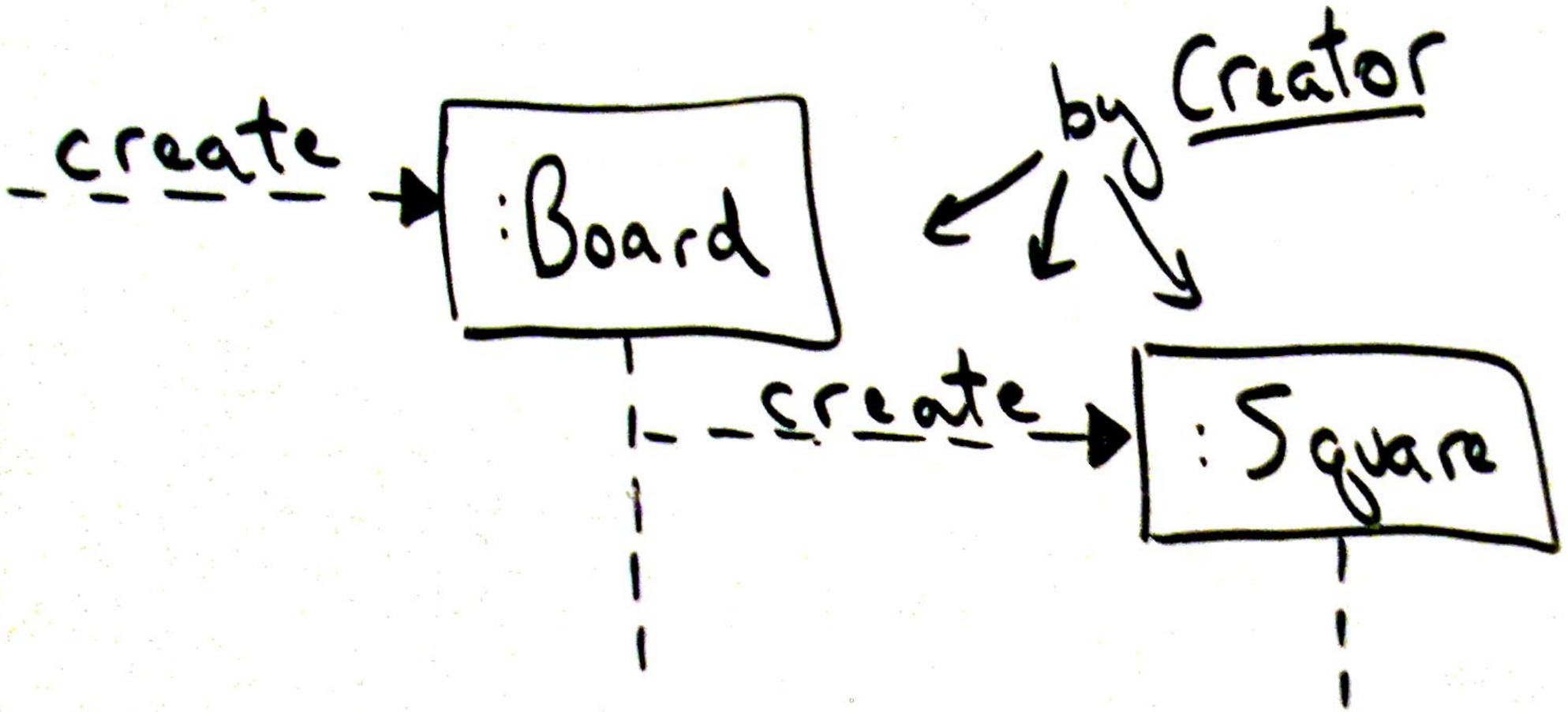
Monopoly Game



Who should create the squares?



Who should create the squares?

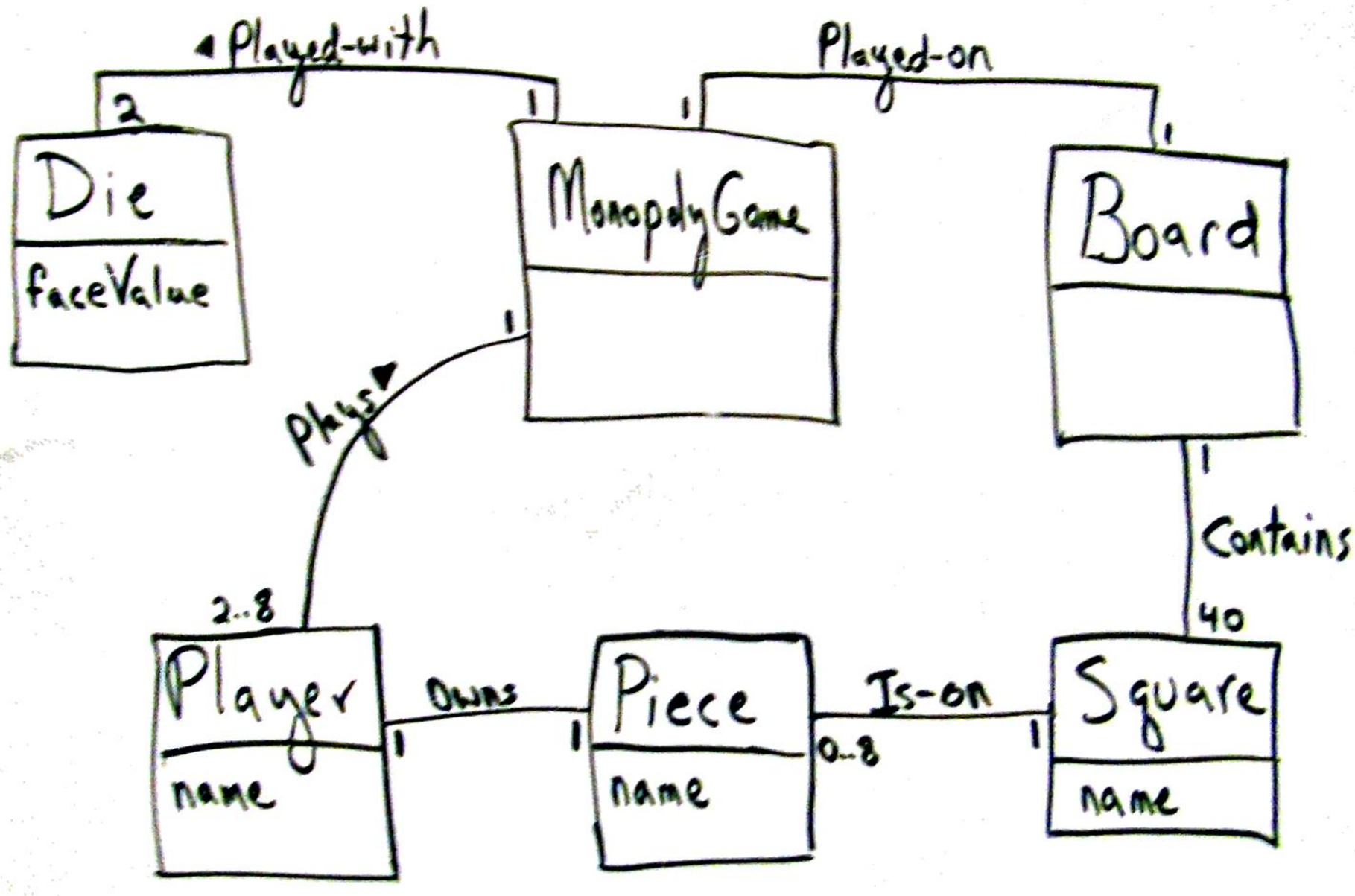


Information Expert Pattern

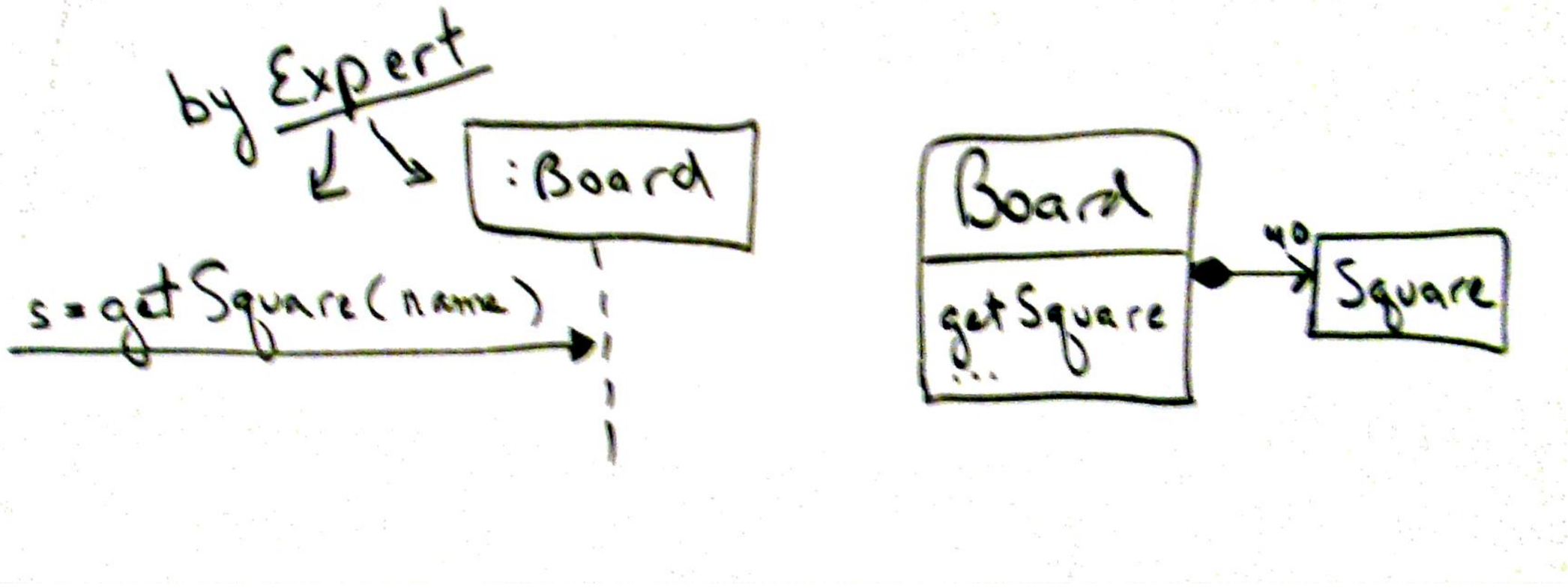
P: How to assign responsibilities to objects?

S: Assign a responsibility to the class that has the information needed to fulfill it

Who should return a square, given its name?



Applying the IE Pattern



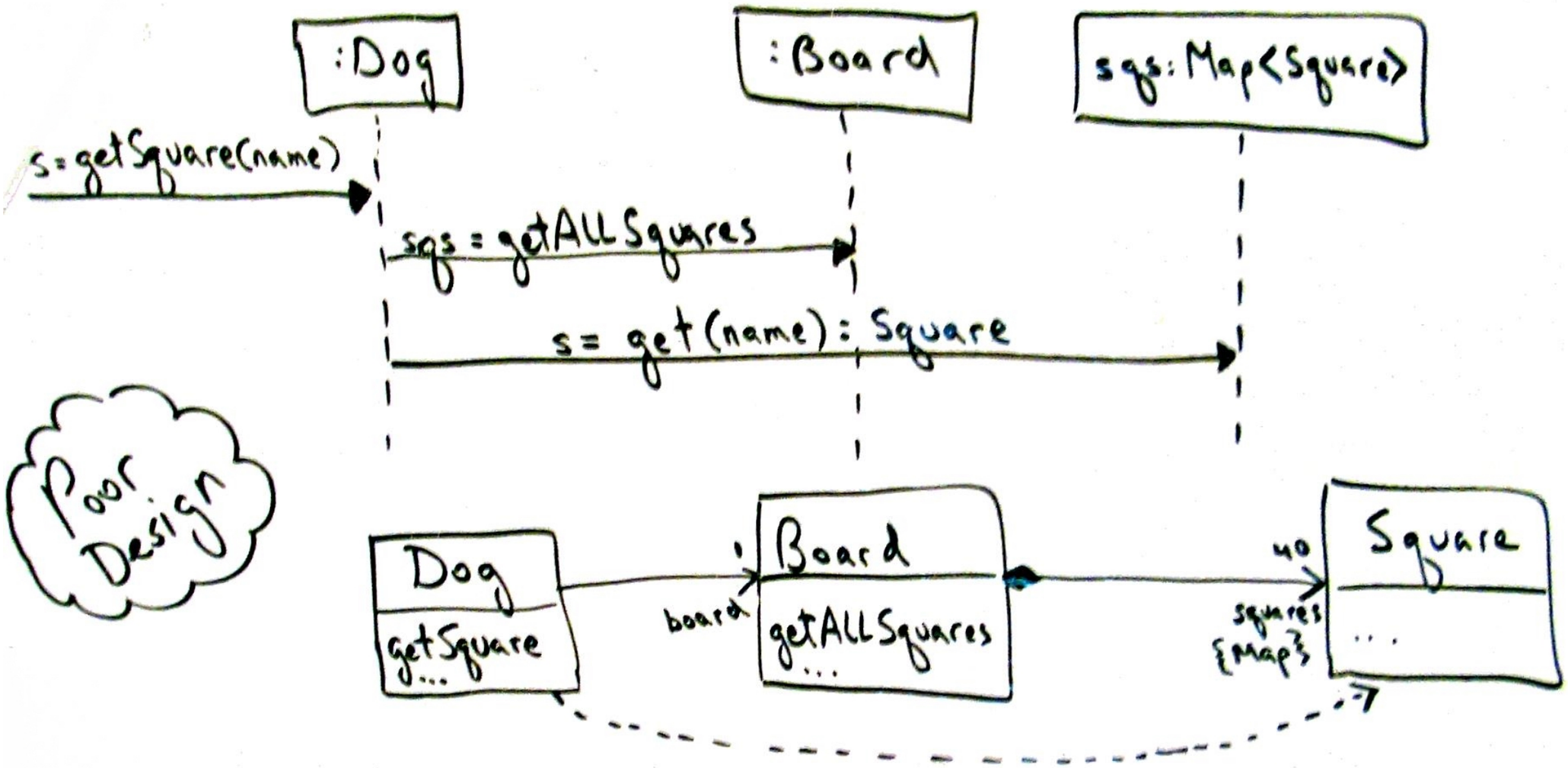
Low Coupling Pattern

P: How to reduce the impact of change?

S:

- Assign responsibilities so that (unnecessary) coupling remains low
- Use this principle to evaluate alternatives

Not Applying Low Coupling



* Higher (more) coupling if Dog has getSquare!

Observation

- Low Coupling **is one of the most important goals in design**
- **It tends to reduce time, effort and defects in software evolution**
- **It is supported by** Information Expert

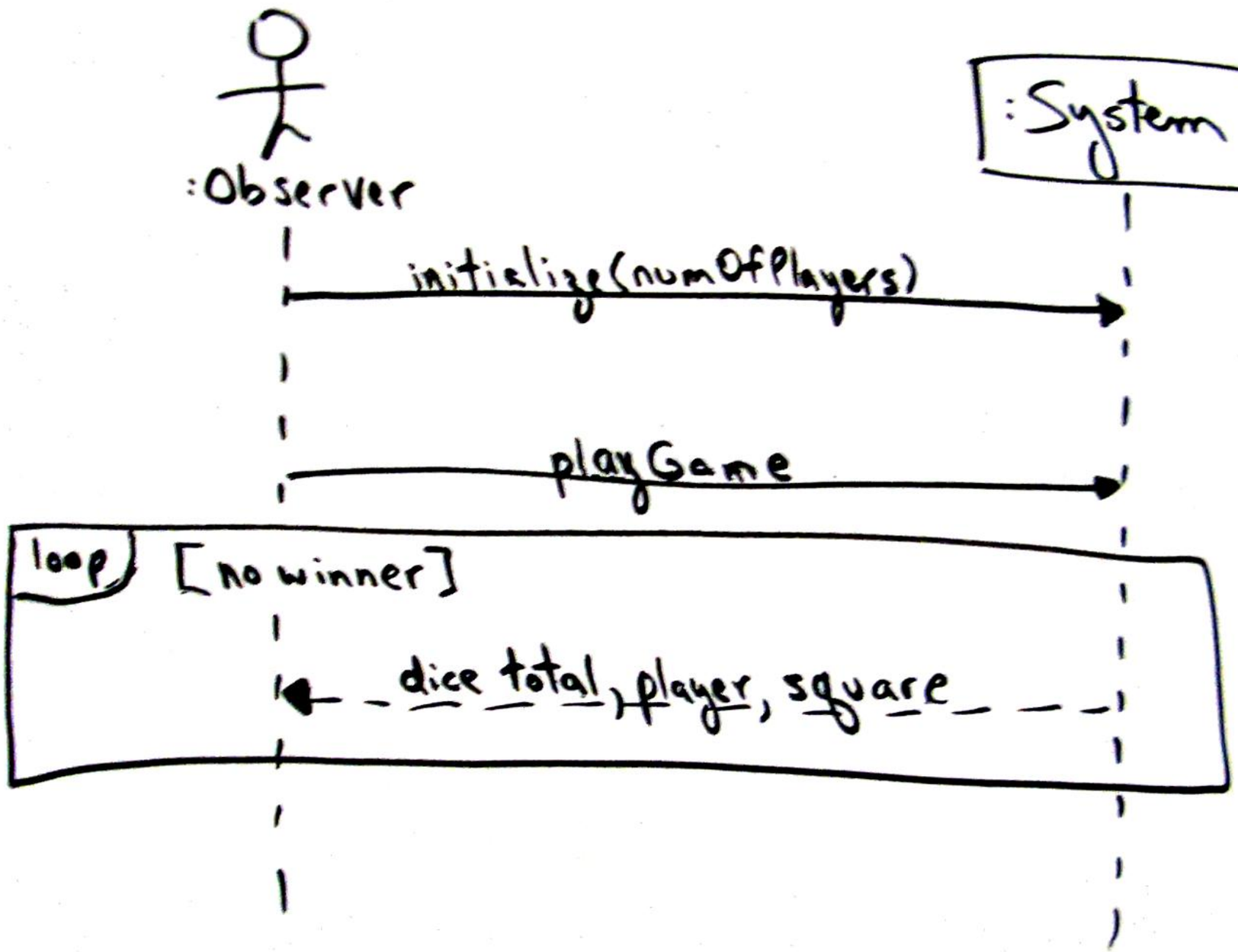
Controller Pattern

P: Which object beyond the UI layer first receives and coordinates a system operation?

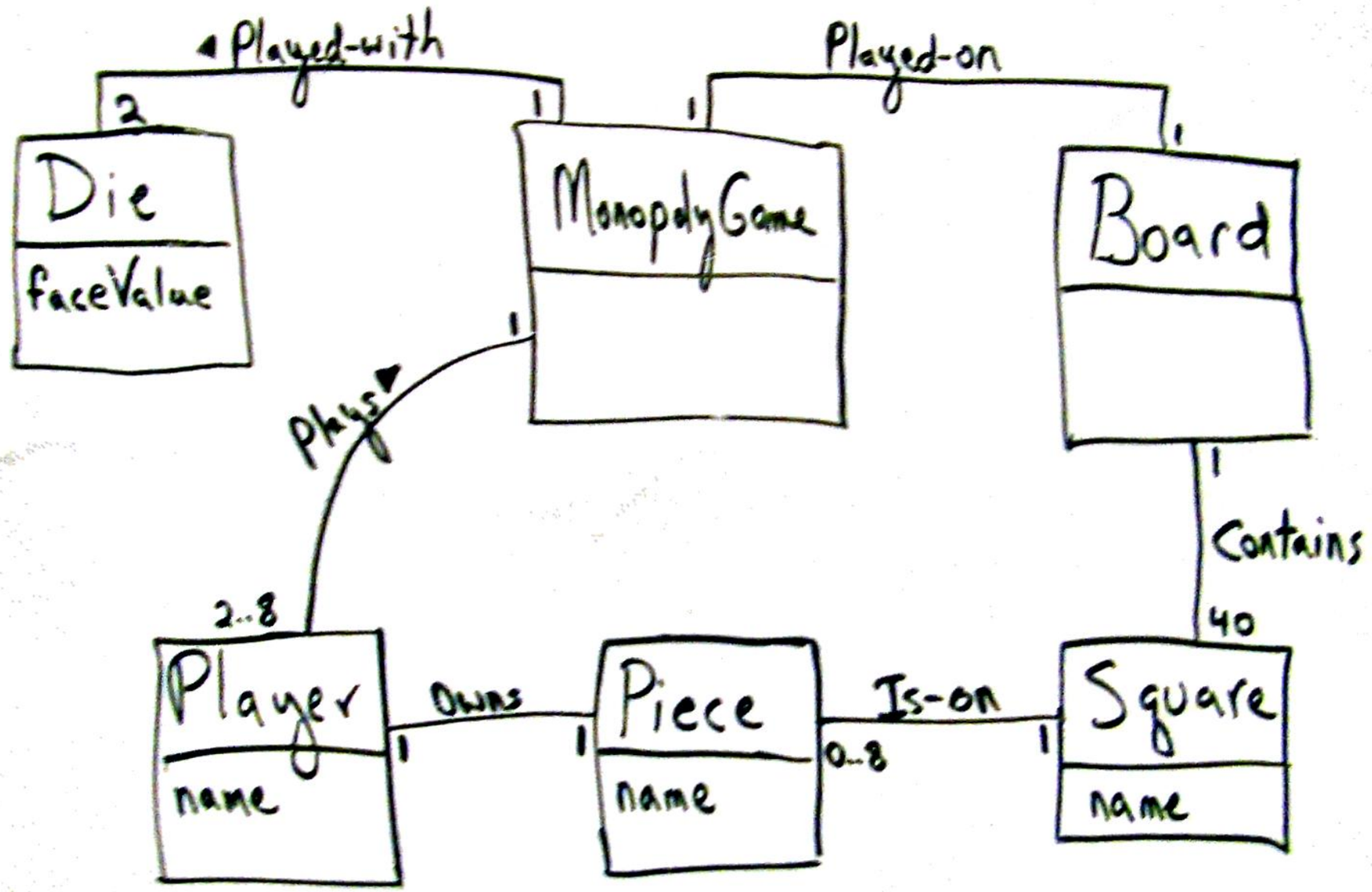
S: Assign the responsibility to an object representing:

- **the overall system, a *root object*,** (facade controller)
- **a use case scenario within which the system operation occurs** (session controller)

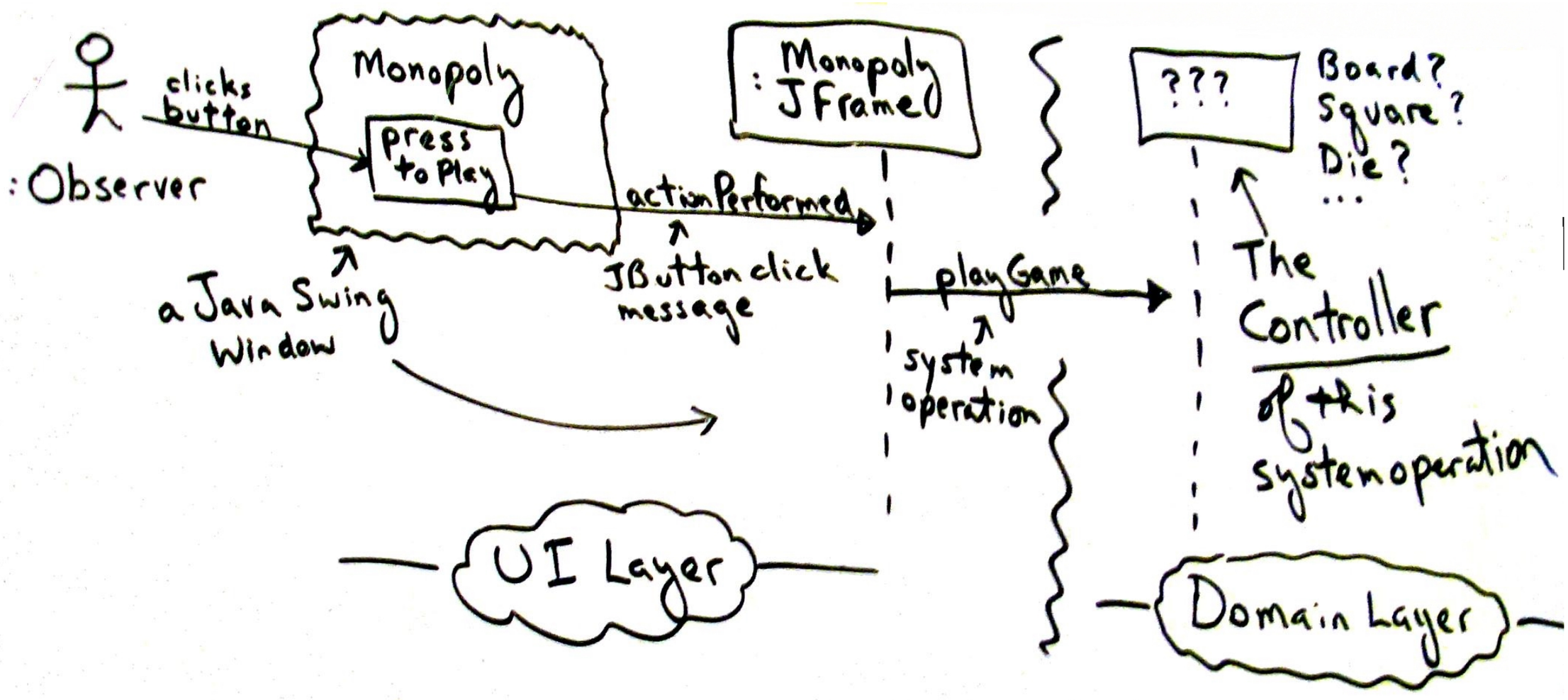
System Sequence Diagram



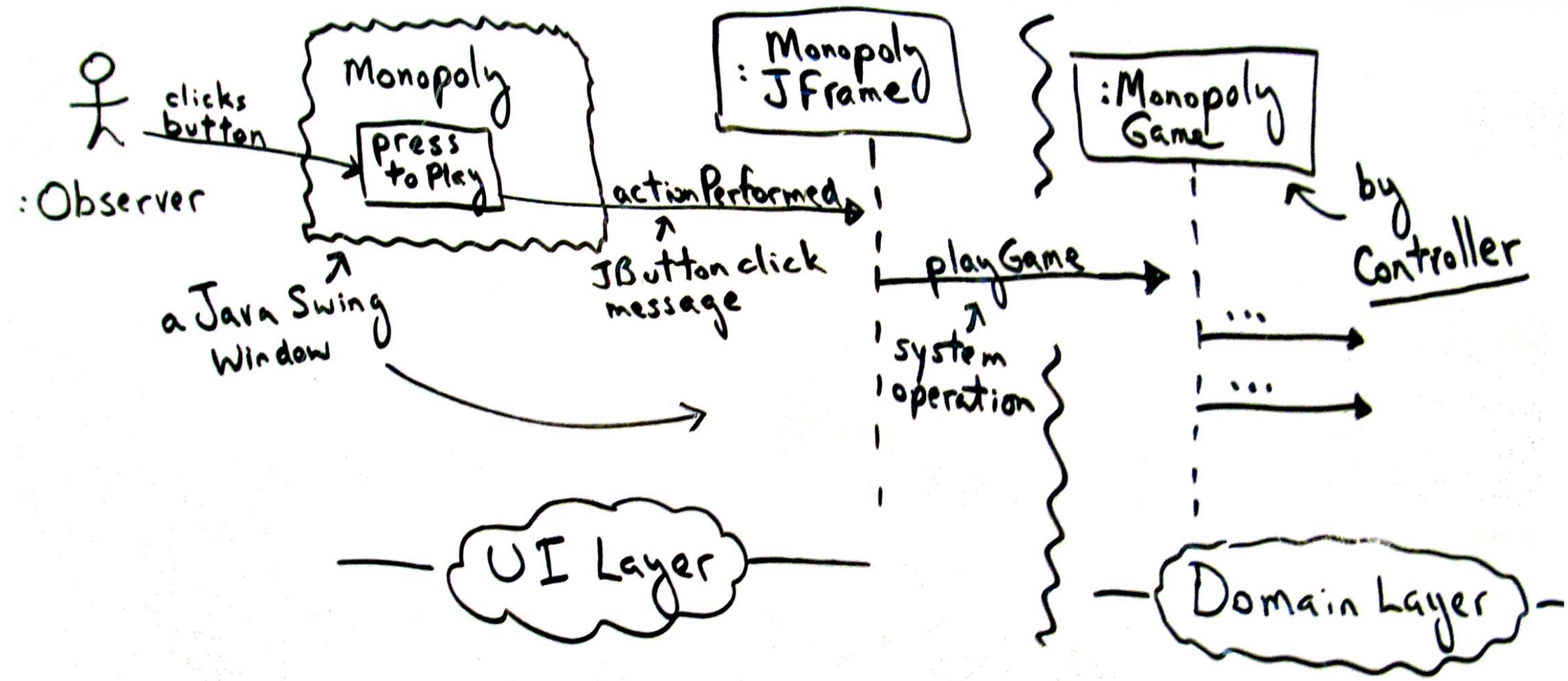
Who should be the controller of playGame?



Who should be the controller of playGame?



Who should be the controller of playGame?



High Cohesion Pattern

P: How to keep objects focused, manageable, and understandable?

S:

- Assign responsibilities so that the cohesion of an object remains high
- Use this principle to evaluate alternatives

Cohesion

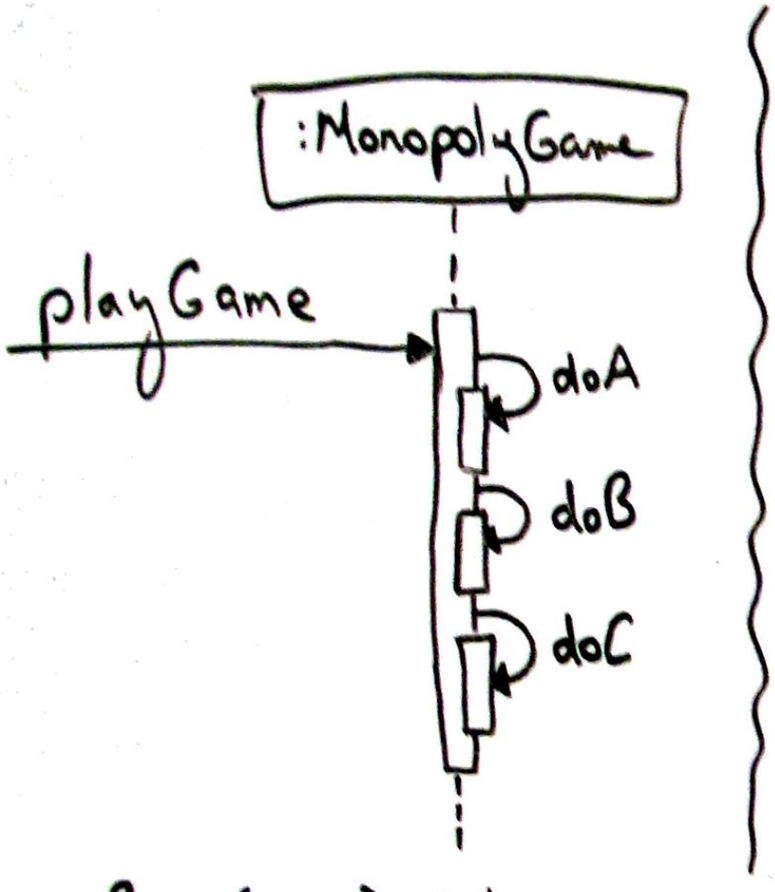
Informally, a measure of

- how functionally related the operations of a software component are
- how much work a software component is doing

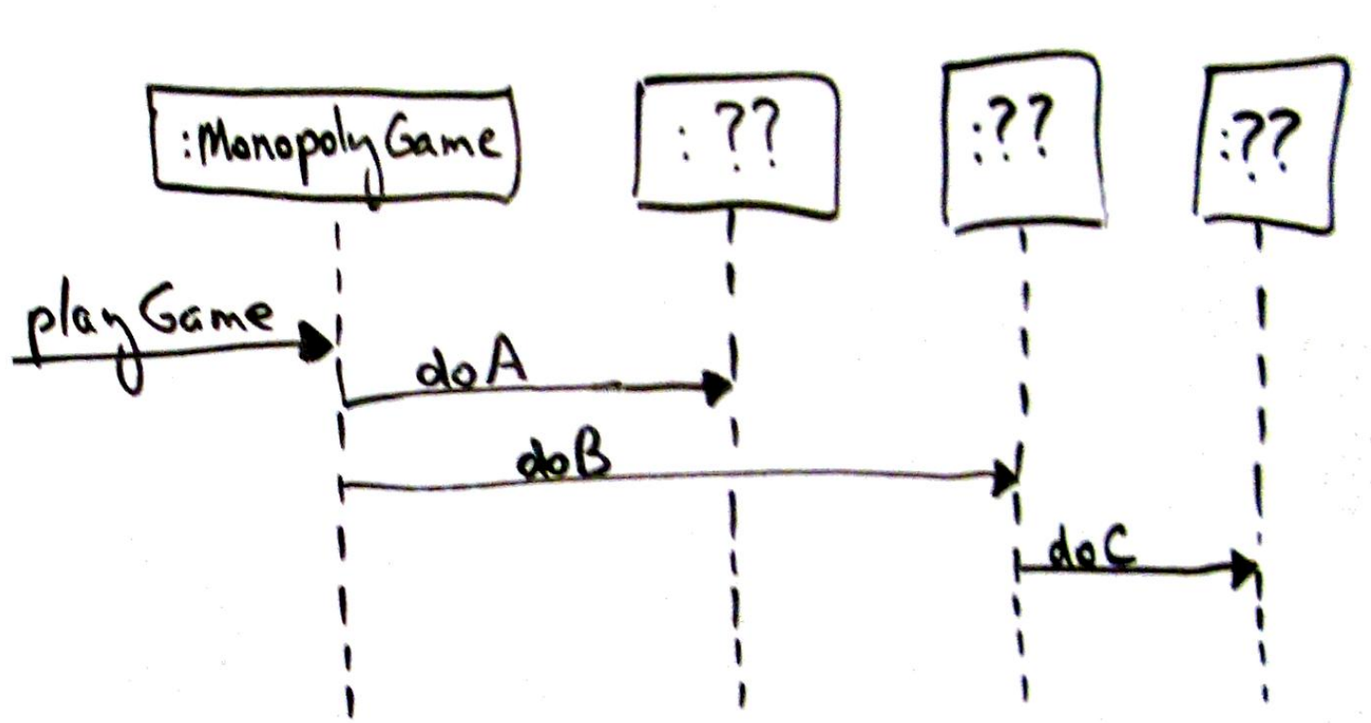
Cohesion

- Bad cohesion and high coupling are positively correlated
- All other things being equal, a design with higher cohesion is preferable

Who has higher cohesion?



Poor (Low) Cohesion
in the MonopolyGame object



Better

Credits

Notes and figures adapted from

Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development by C. Larman. 3rd edition. Prentice Hall/Pearson, 2005.