GRASP-Case Study

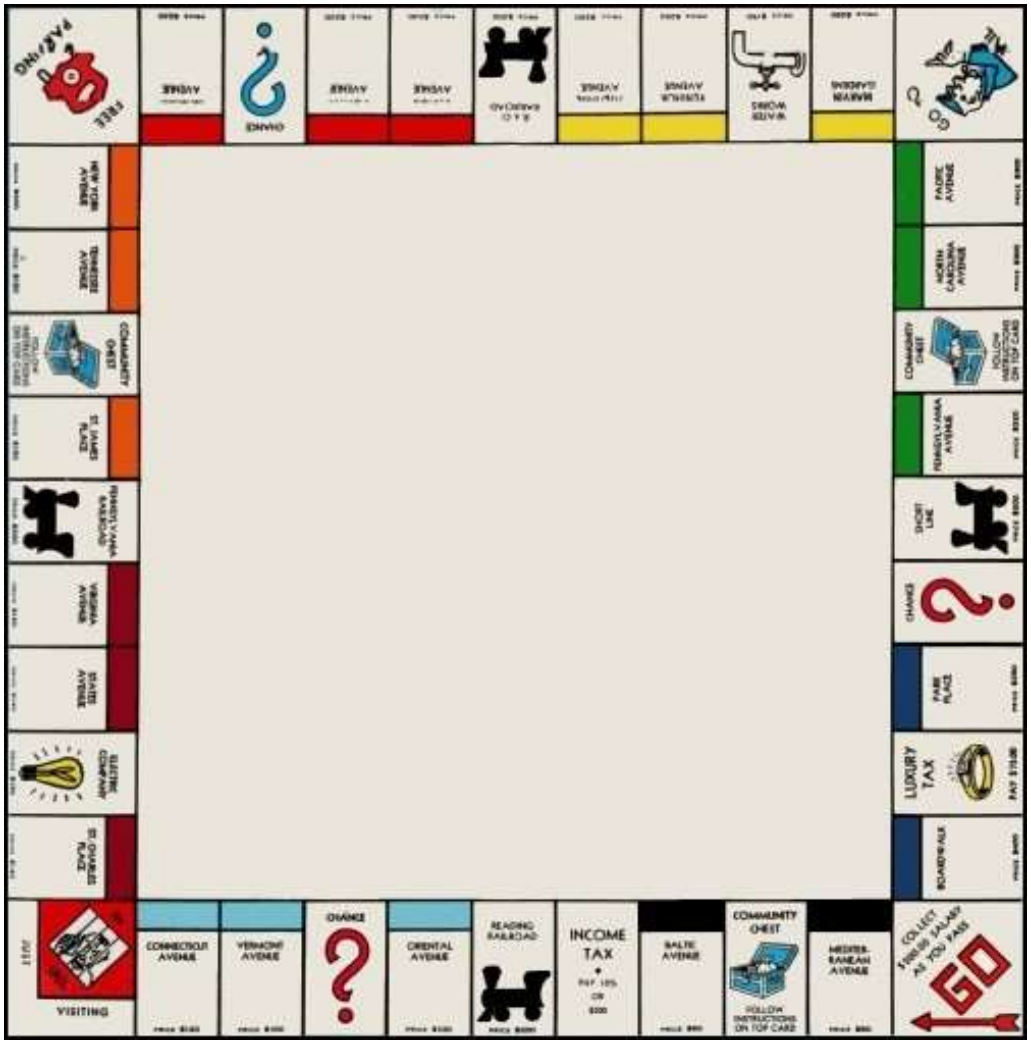
Using Design Patterns with GRASP

General Responsibility Assignment Software Patterns

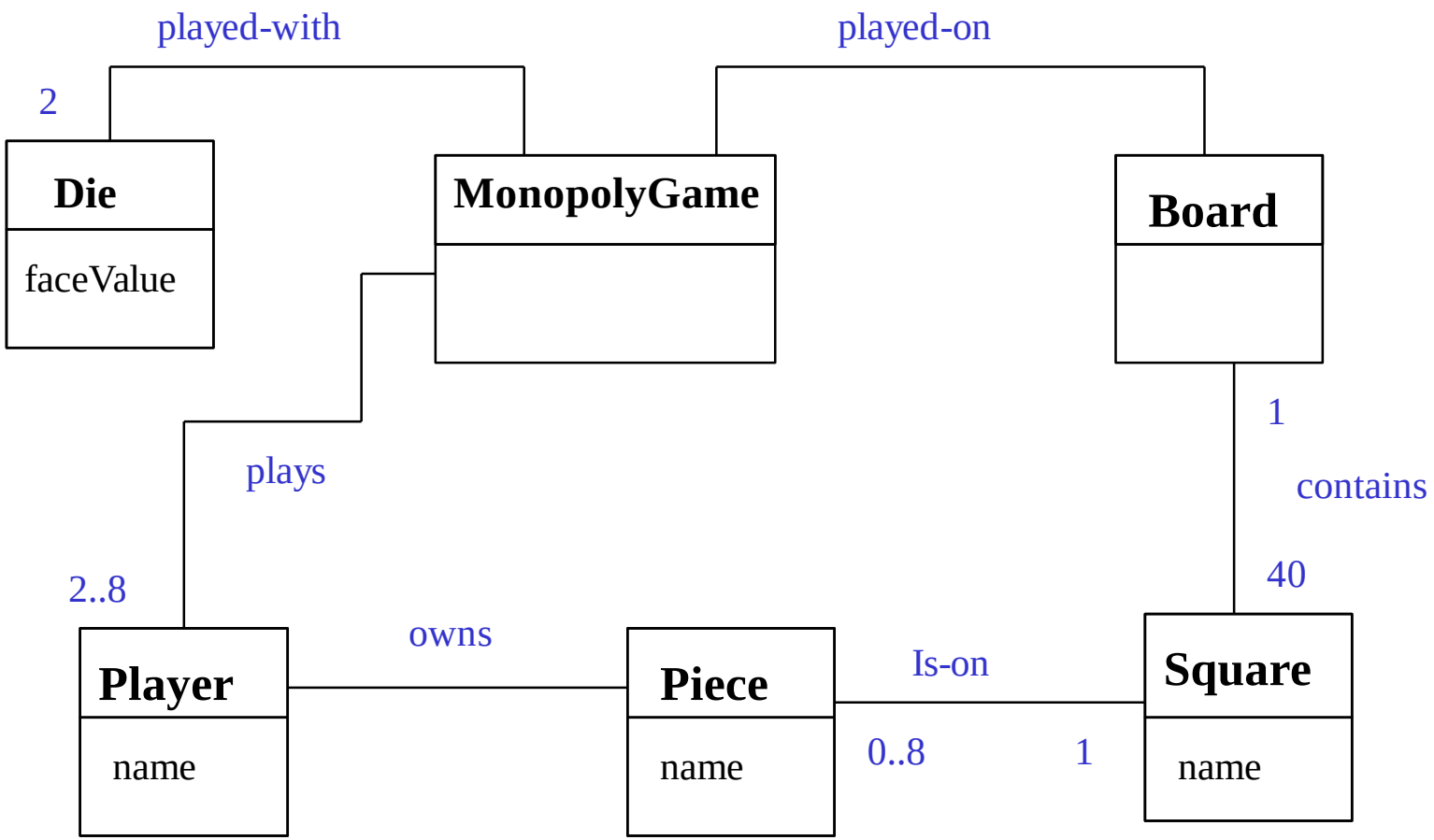
The "patterns" provide a representation of basic principles that form a foundation for designing object-oriented systems.

Creator	**Controller**	**Pure Fabrication**
Information Expert	**High Cohesion**	**Low Coupling**

Example of RDD – Monopoly Game

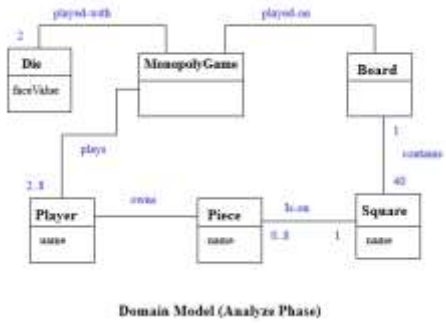


Example of RDD – Monopoly Game



Domain Model (Analyze Phase)

Monopoly Game Example



Domain Model (Analyze Phase)

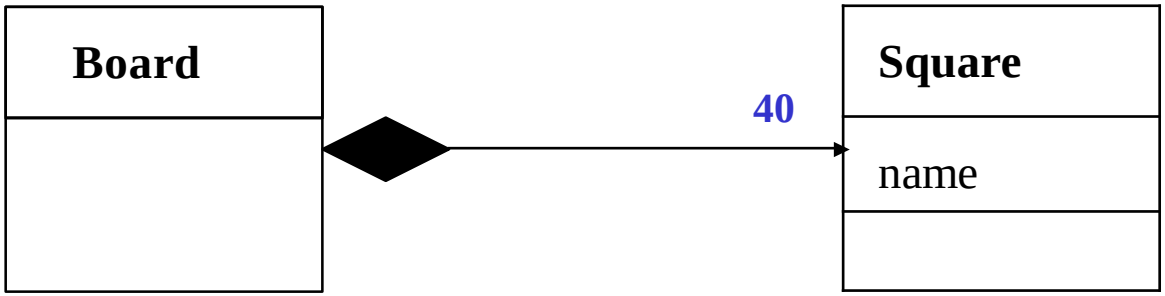
Who creates the Square object in a Monopoly game?

The **Creator** pattern

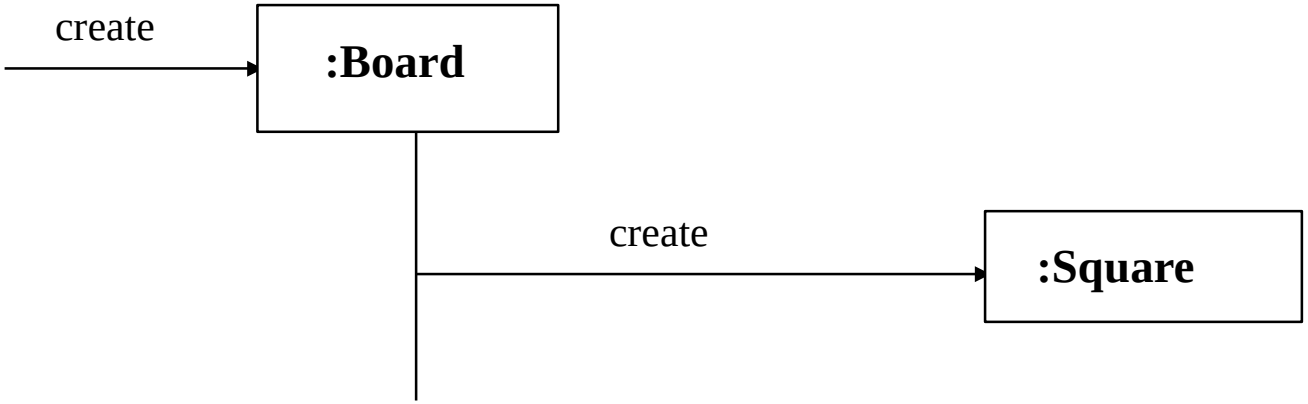
Java answer: Any object could
But what makes for good design?
Common answer: Make Board create Square objects
Intuition: Containers should create the things contained in them
Issue: Wait, but we don't have any classes yet, we only have the domain model!

We have to start somewhere, and we look at the domain model for inspiration to pick our first few important software classes

Implementation of the Creator Pattern in Monoply



Applying the Creator pattern in Static Model



Dynamic Model – illustrates Creator Pattern

Monopoly Game Example

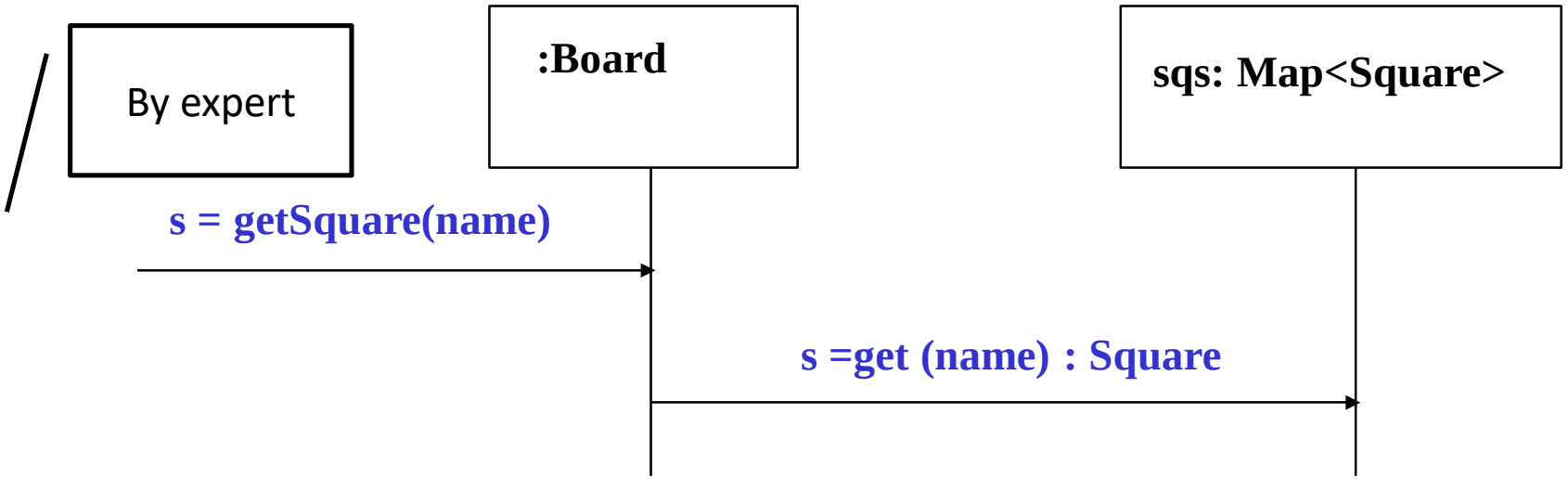
Who knows about a Square object, given a key?

Information Expert pattern

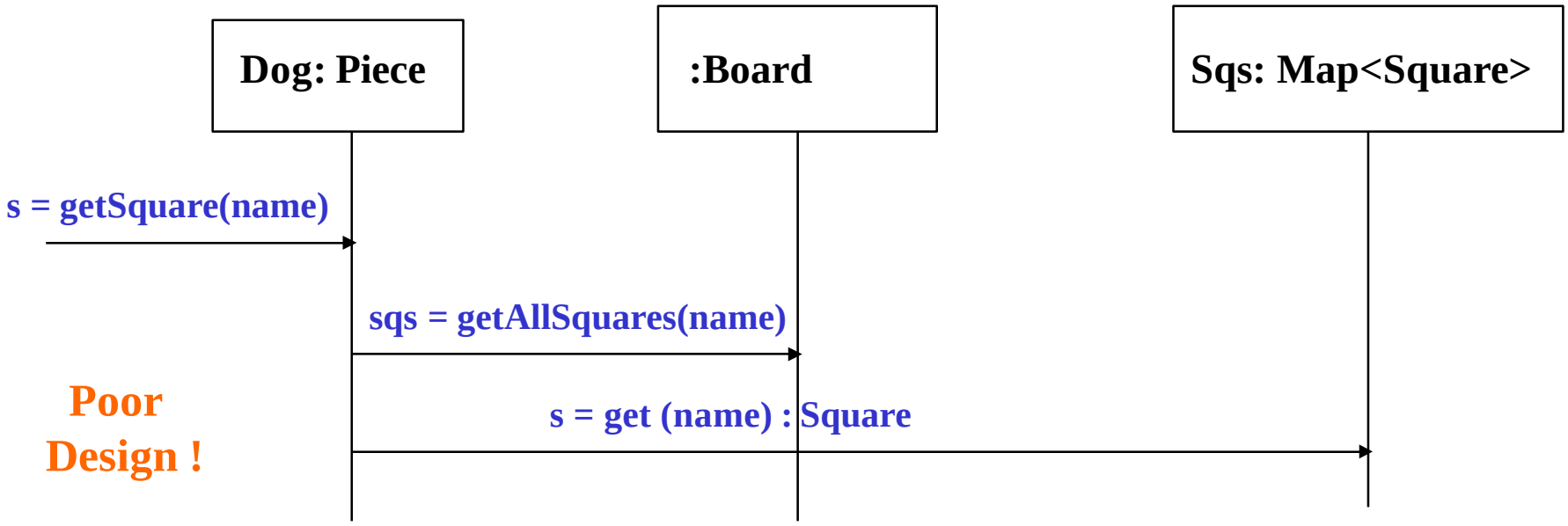
The player Marker needs to find the square to which it is to move and the options pertaining to that square.

The Board aggregates all of the Squares, so the Board has the Information needed to fulfill this responsibility.

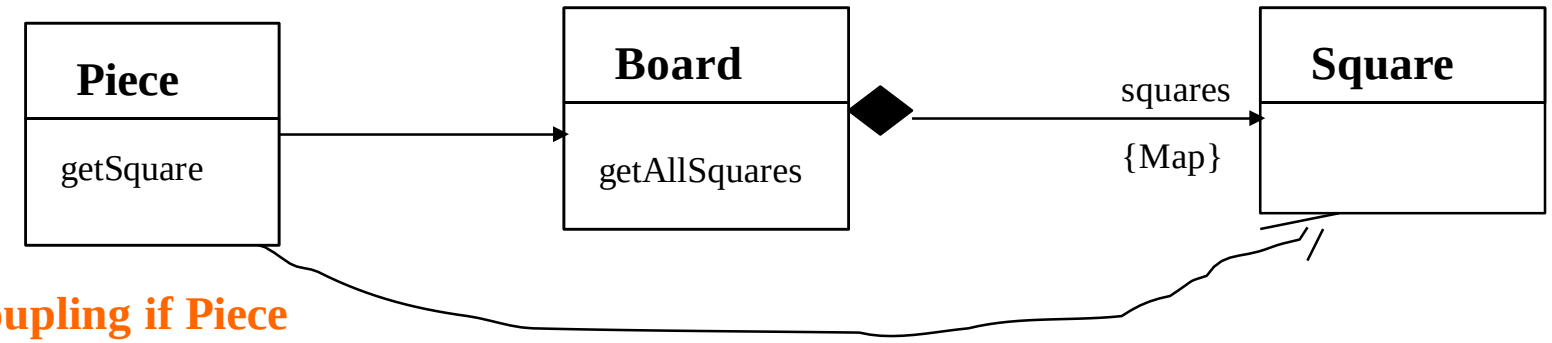
Make Board Information Expert



Alternative Design



Poor Design !



More coupling if Piece has getSquare ( )

Low Coupling

- **Key Point: Expert Supports Low Coupling**

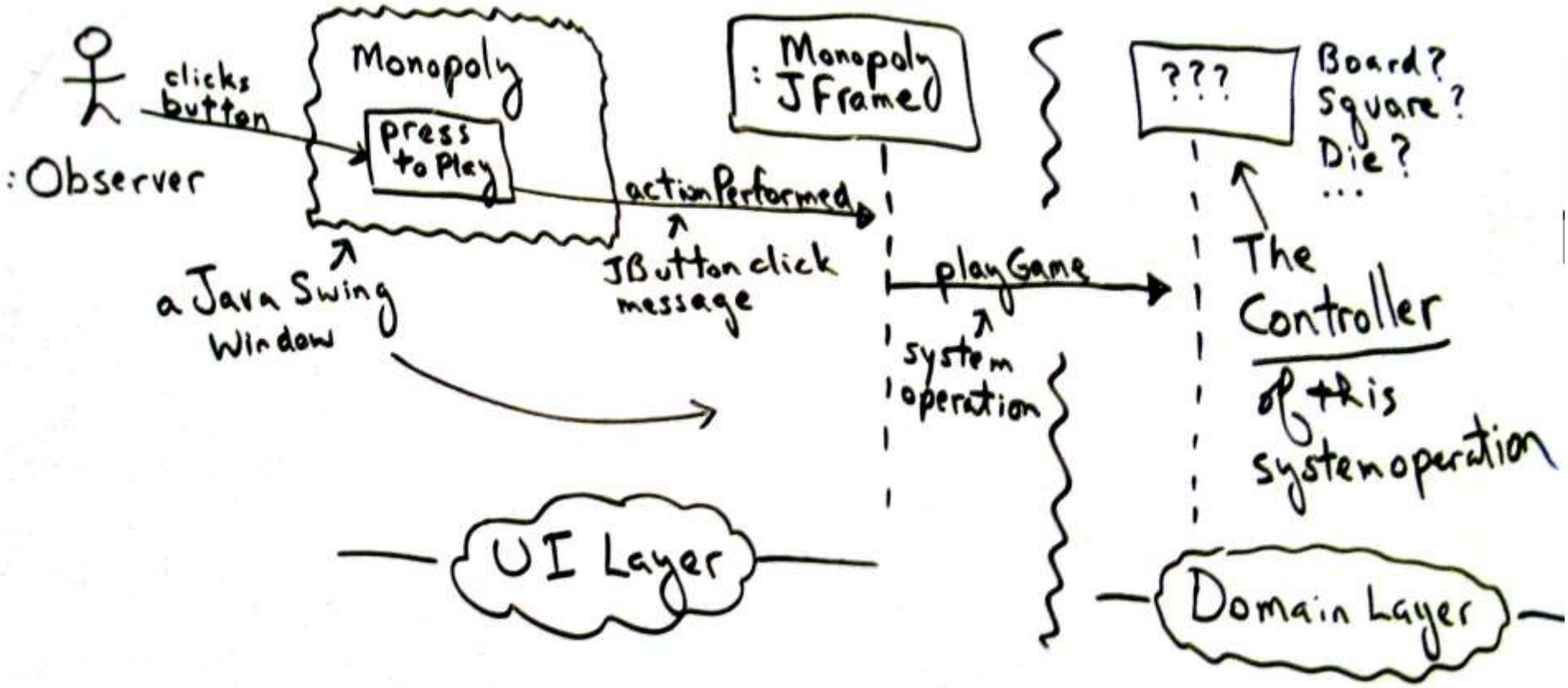
- To return to the motivation for Information Expert: it guides us to a choice that supports Low Coupling. Expert asks us to find the object that has most of the information required for the responsibility (e.g., Board) and assign responsibility there.

- If we put the responsibility anywhere else (e.g., Dog), the overall coupling will be higher because more information or objects must be shared away from their original source or home, as the squares in the Map collection had to be shared with the Dog, away from their home in the Board.

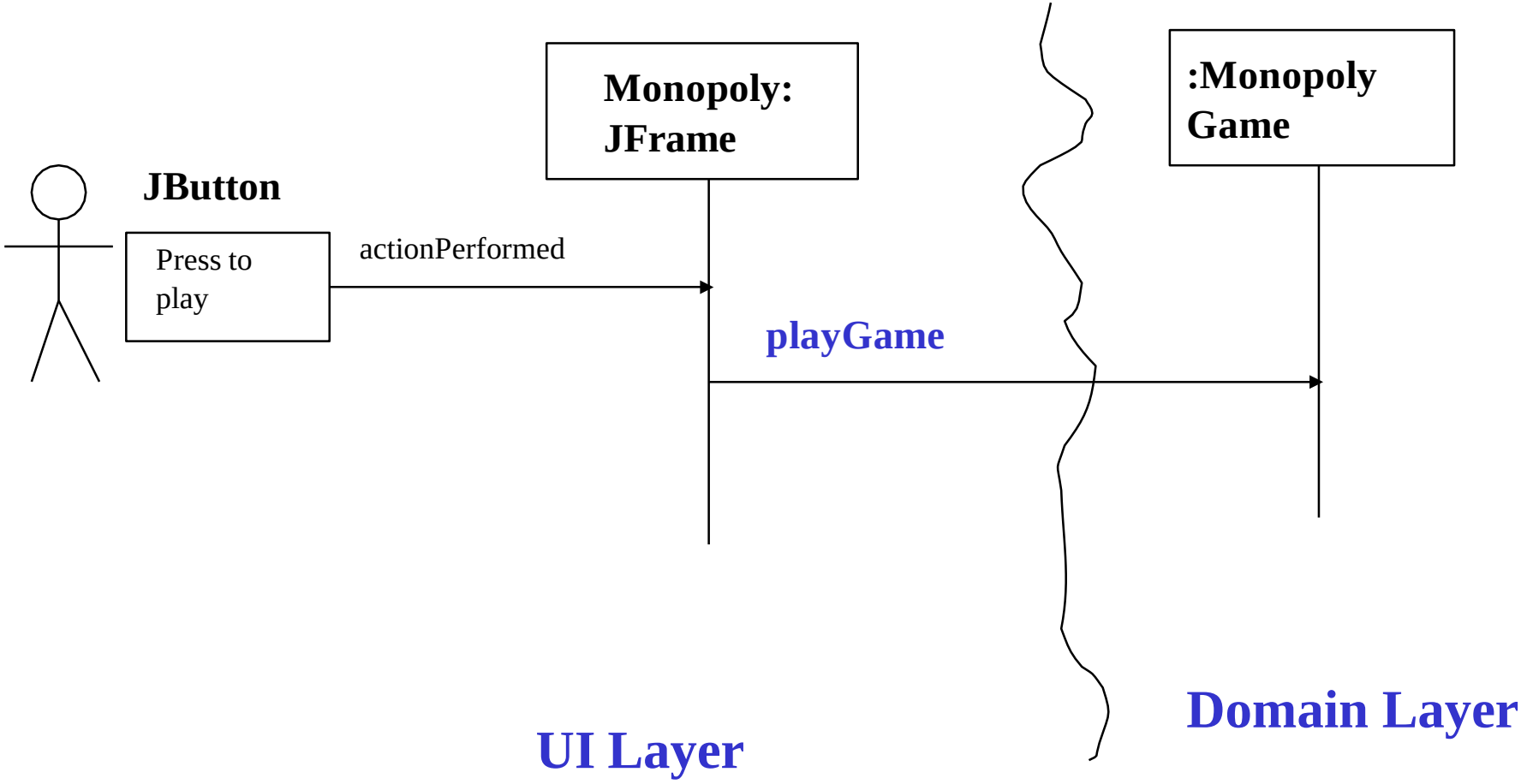
Controller

A simple layered architecture has a user interface layer (UI) and a domain layer. Actors, such as the human player in Monopoly, generate UI events (such as clicking a button with a mouse to play a game or make a move). The UI software objects (such as a JFrame window and a JButton) must process the event and cause the game to play. When objects in the UI layer pick up an event, they must delegate the request to an object in the domain layer.

What first object beyond the UI layer should receive the message from the UI layer?


clicks button
:Observer
Monopoly
press to Play
a Java Swing Window
actionPerformed
JButton click message
: Monopoly JFrame
playGame
system operation
???
Board?
Square?
Die?
...
The Controller of this system operation
UI Layer
Domain Layer

The Controller Pattern



Option 1: Represents the overall “system” or a “root object”

- An object called MonopolyGame

Option 2: Represents a device that the software is running within

- It doesn’t really apply here.
- It applies to designs with specialized hardware devices: Phone, BankCashMachine, …

Option 3: Represents the use case or session.

- The use case that the playGame system operation occurs in is called Play Monopoly Game
- A class such as PlayMonopolyGameHandler (or ...Session) might be used

The very first option looks good if there are only a few system operations

First Iteration of the Monopoly Game

In Iteration 1 – there is no winner. The rules of the game are not yet incorporated into the design. Iteration 1 is merely concerned with the mechanics of having a player move a piece around the Board, landing on one of the 40 Squares each turn.

Definition –

turn – a player rolling the dice and moving one piece

round – all players taking one turn

The game loop algorithm:

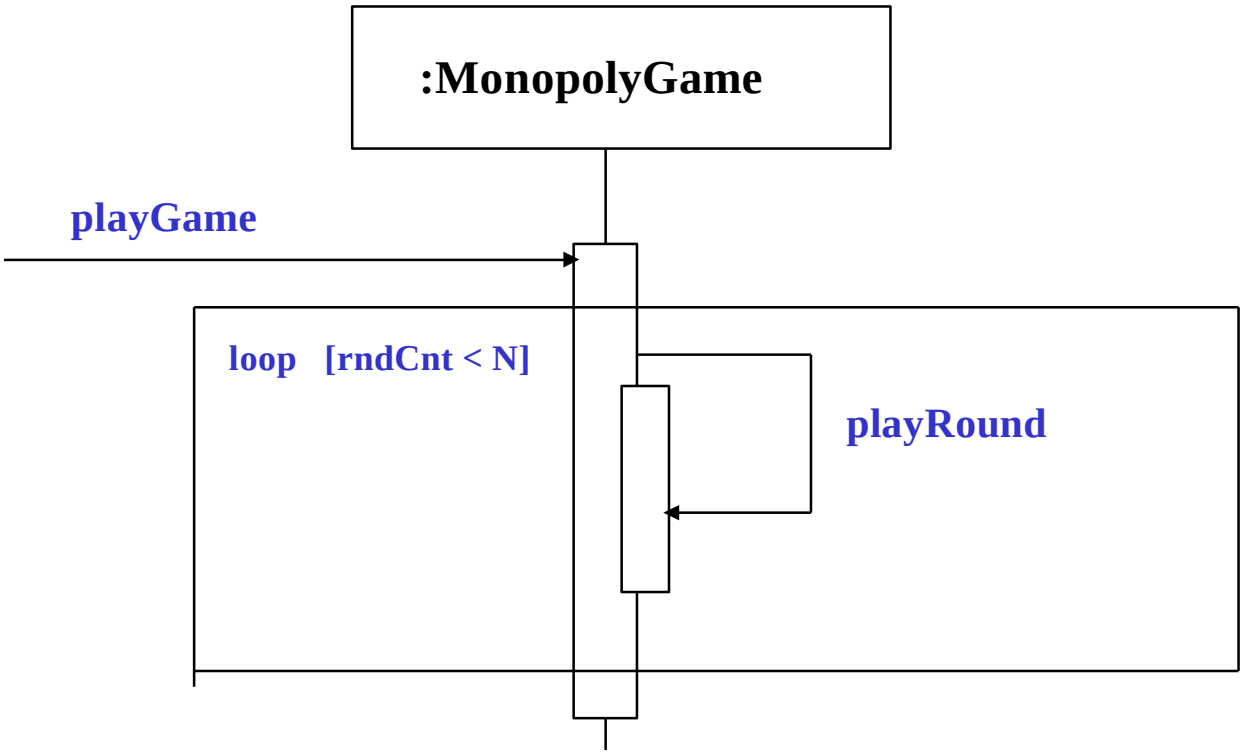
```
for N rounds
    for each player p
        p takes a turn
```

Assign Responsibility for Controlling the Game Loop

Information Needed	Who Has the Information?
The current round count	No object has it yet, but by LRG*, assigning this to the *MonopolyGame* object is justifiable
All the players (so that each can be used in taking a turn)	From examination of the domain model, *MonopolyGame* is a good candidate.

***LRG – low representational gap. Lower the gap between our mental and software models.**

Controlling the Game Loop



Who Takes a Turn?

Information Needed	Who Has the Information?
Current location of the player (to know the starting point of a move)	We observe from the domain model, a Piece knows its *Square* and a *Player* knows its *Piece*. Therefore, a *Player* software object could know its location by LRG.
The two Die objects (to roll them and calculate their total)	The domain model indicates that *MonopolyGame* is a candidate since we think of the dice as being part of the game.
All the squares – the square organization (to be able to move to the correct new square)	By LRG, *Board* is a good candidate.

Taking a Turn

Taking a turn means:

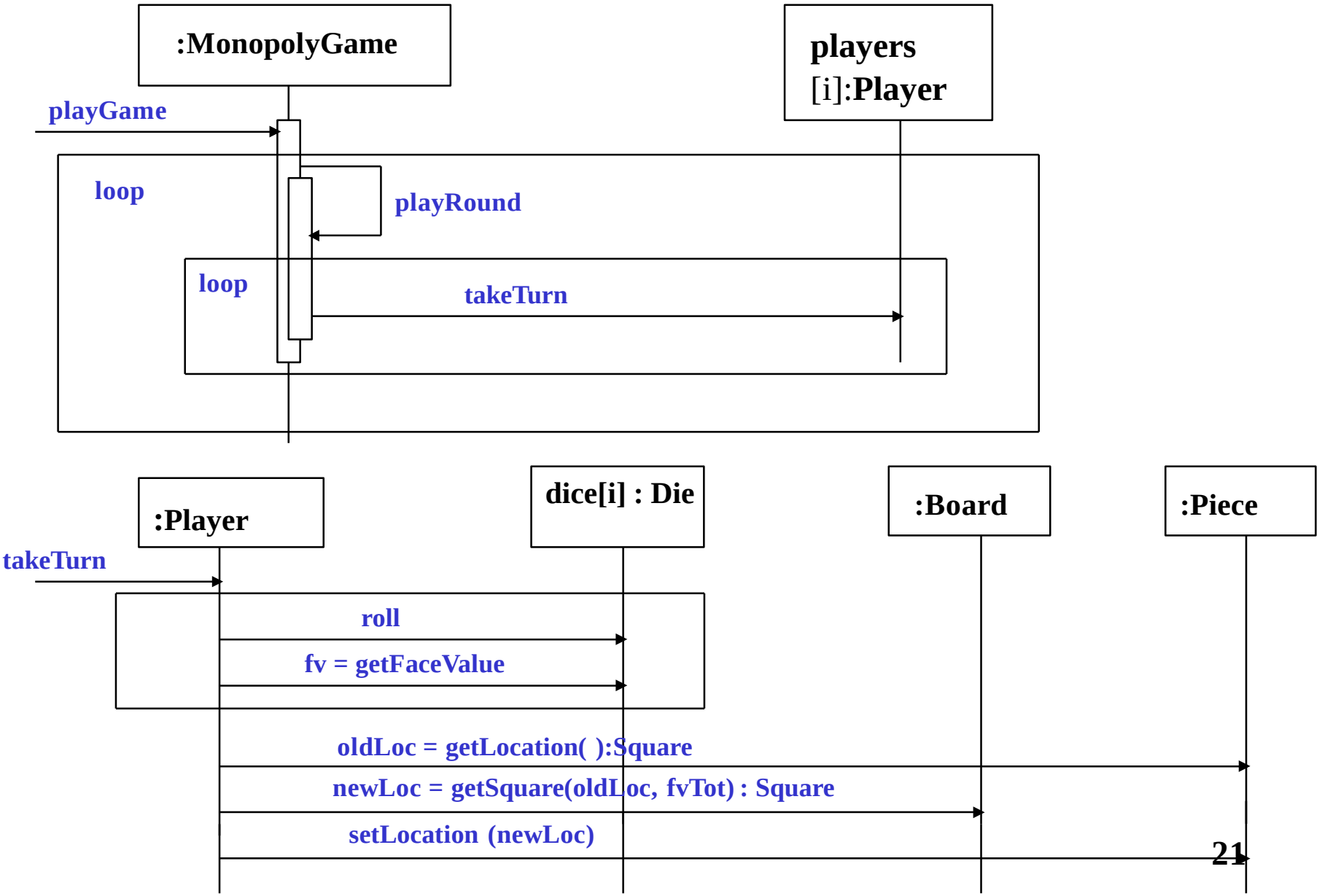
- **Calculating a random number between 2 and 12**
- **Determining the location of the new square**
- **Moving the player's piece from the old location to the new square.**

Calculating a new face value means changing information in *Die*, so by Expert, *Die* should be able to roll itself (generate a random number) and answer its face value.

The new square location problem: Since the *Board* knows all its *Squares*, it should be responsible for finding a new square location, given an old square location and some offset (the dice total)

The piece movement problem: By LRG it is reasonable for a *Player* to know its *Piece*, and a *Piece* its *Square* location (or even for a *Player* to directly know its *Square* location). By Expert, a *Piece* will set its new location, but may receive that new location from its *Player*.

Final Design of the System Operation playGame (Iter. 1)



Pure Fabrication

Name: **Pure Fabrication**

Problem: What object should have responsibilitywhen you do not want to violate High Cohesion and Low Coupling, or other goals, but solutions offered by Expert (for example) are not appropriate?

Sometimes assigning responsibilities only to domain layer software classes leads to problems like poor cohesion or coupling, or low reuse potential.

Solution: Assign a highly cohesive set of responsibilities to an artificial or convenience class that does not represent a domain concept.

Example: Rolling the dice in a Monopoly game – Dice are used in many games and putting the rolling and summing responsibilities in Player makes it impossible to generalize this service. Also, it is not now possible to simply ask for the current dice total without rolling again.

Pure Fabrication

Use a *Cup* to hold the dice, roll them, and know their total. It can be reused in many different applications where dice are involved.