GRASP: Designing Objects with Responsibilites

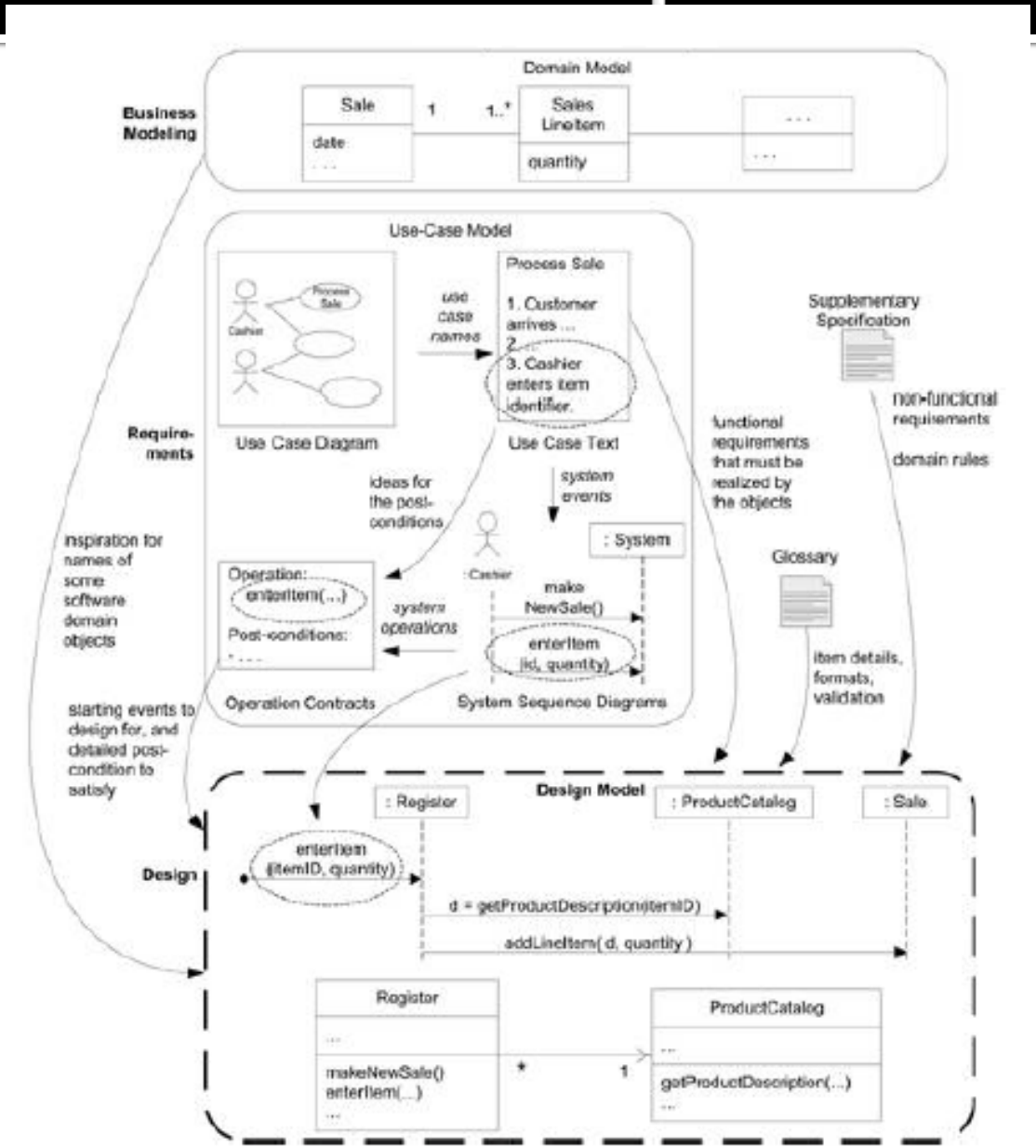
Software Design and Analysis

CSCI 2040

Objectives

- Define patterns.
- Learn to apply five of the GRASP patterns.

Artifact Relationships



Introduction

- Deciding what methods belong where, and how the objects should interact, is
  - terribly important and
  - anything but trivial!
- It takes careful explanation,
  - applicable while diagramming and programming.
- This is at the heart of what it means to develop an object-oriented system.

GRASP

- **The GRASP patterns:**
  - help one understand essential object design, and
  - apply design reasoning in a methodical, rational, explainable way.
- **This approach to using design principles is based on *patterns* of assigning responsibilities.**

Responsibilities

- **Responsibilities are related to the *obligations* of an object in terms of its *behavior*.**
- Responsibilities are assigned to classes of objects during object design.
- These responsibilities are of the following two types:
  - **Knowing**
  - **Doing**

Doing Responsibilities

- **Doing responsibilities of an object include:**
  - **doing something itself, such as creating an object or doing a calculation**
  - **initiating action in other objects**
  - **controlling and coordinating activities in other objects**

Knowing Responsibilities

- **Knowing responsibilities of an object include:**
  - **knowing about private encapsulated data**
  - **knowing about related objects**
  - **knowing about things it can derive or calculate**

Doing and Knowing Examples

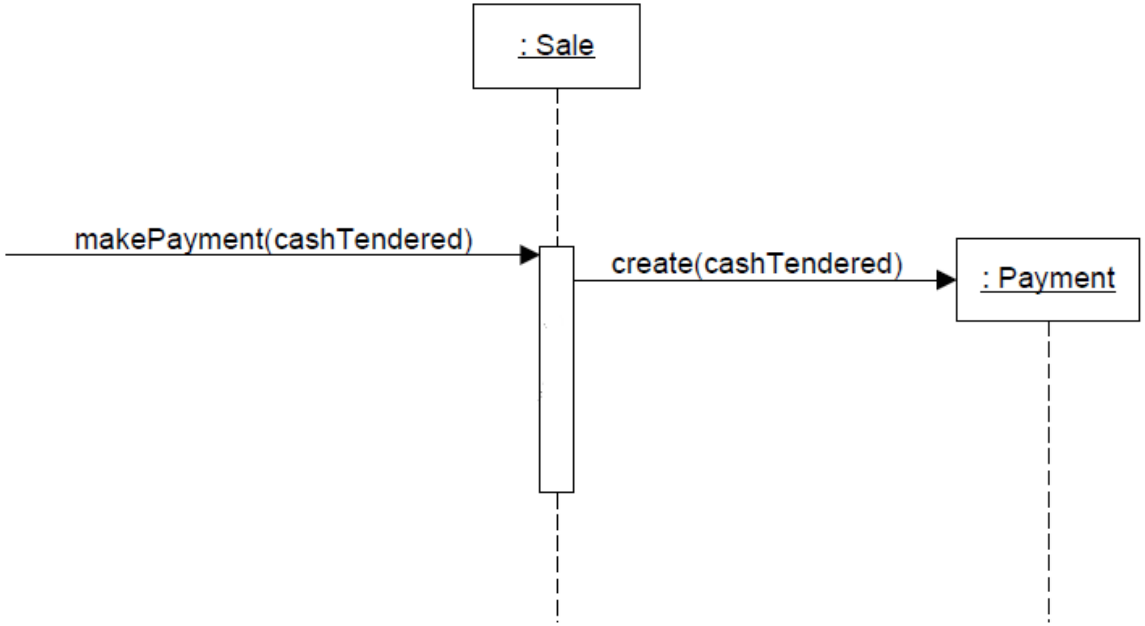
- Doing Example
  - **e.g., declare that a *Sale* is responsible for creating *SalesLineItems.***
- Knowing Example
  - **e.g., a *Sale* is responsible for knowing its total.**

Methods vs Responsibilities

- **A responsibility is NOT the same as a method,**
  - **but methods are implemented to fulfill responsibilities.**
- **Responsibilities are implemented using methods that either act alone or collaborate with other methods and objects.**
  - **For example, the *Sale* class might define one or more methods such as *getTotal* to know its total**
    - **the *Sale* may collaborate with other objects, such as sending *getSubtotal* message to each *SalesLineItem* object asking for its subtotal.**

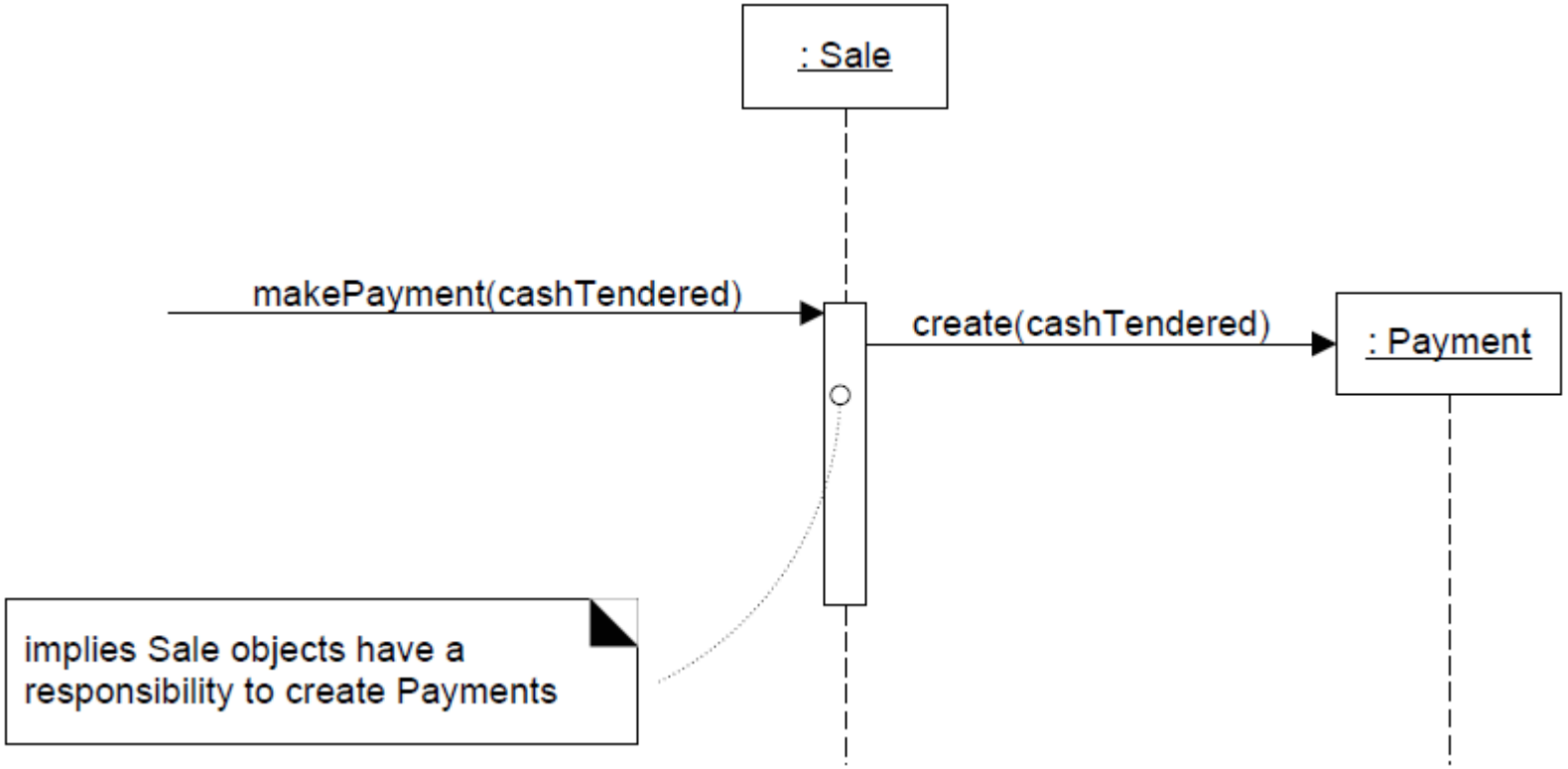
Responsibilities & Sequence Diagrams

- **Responsibilities and methods are related.**
  - **A common context where these responsibilities (implemented as methods) are considered is during the creation of Sequence Diagrams**
    - What is the responsibility of Sale objects in the Sequence Diagram below?



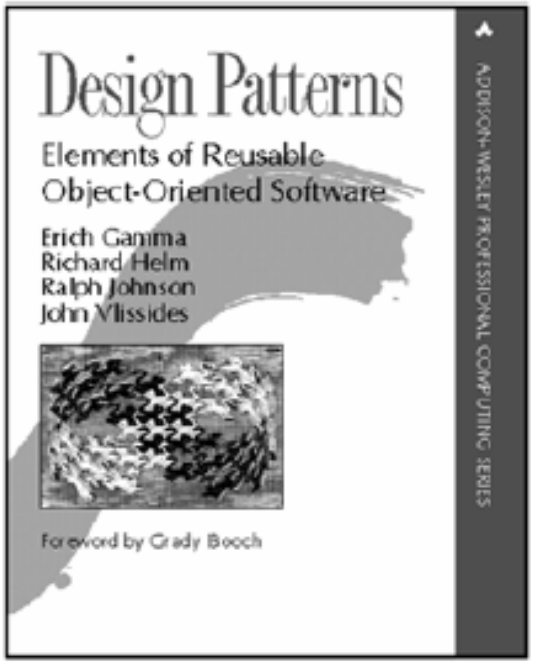
Responsibilities & Interaction Diagrams

- Responsibilities and methods are related.
  - A common context where these responsibilities (implemented as methods) are considered is during the creation of interaction diagrams



How to Apply GRASP Patterns

- GRASP patterns guide choices in where to assign responsibilities.
- First five GRASP patterns are:
  - Information Expert
  - Creator
  - High Cohesion
  - Low Coupling
  - Controller



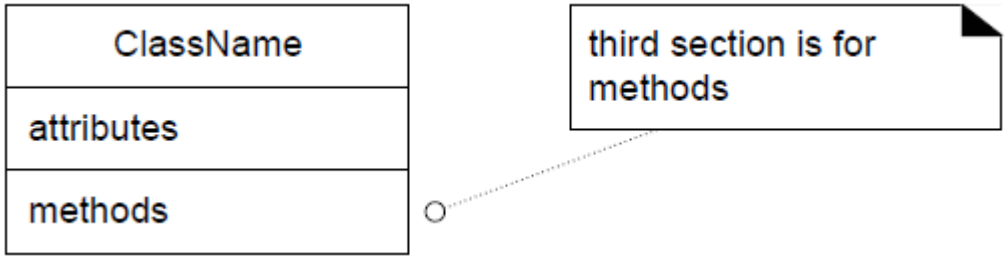
Patterns

- A pattern is a named problem/solution pair that can be applied in new context, with:
  - advice on how to apply it in new situations and/or
  - discussion of its trade-offs.
- GRASP patterns describe fundamental principles of object design and responsibility assignment, expressed as patterns.

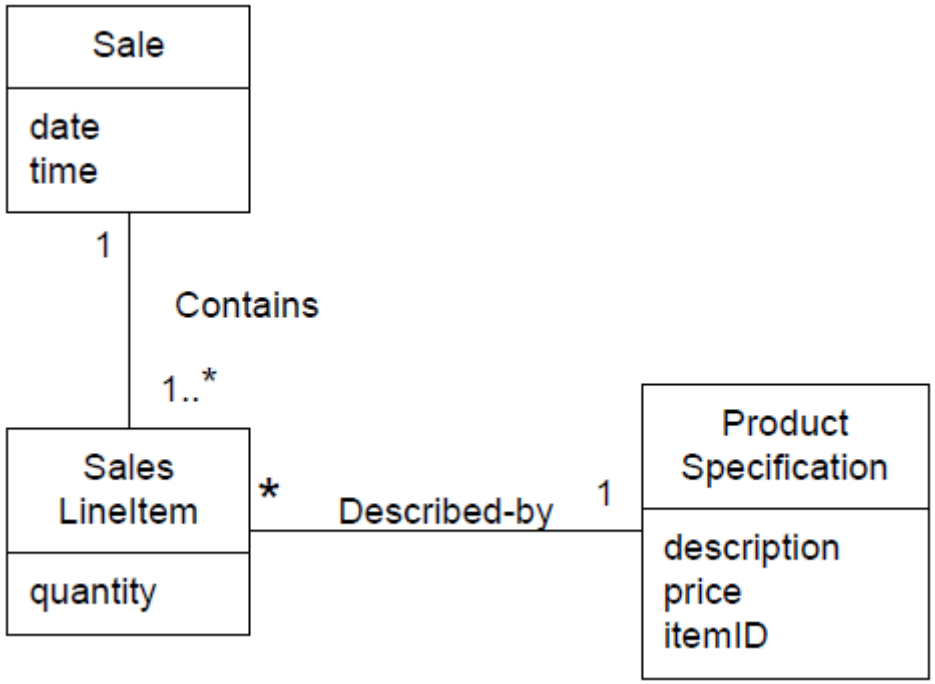
Information Expert Pattern

- **Solution:**
  - **Assign a responsibility to the class that has the information needed to fulfill it.**
- **Problem it solves:**
  - **What is a basic principle by which to assign responsibilities to objects.**
- **Benefits:**
  - **Information encapsulation is maintained.**
    - **This usually supports *low coupling*, which leads to more robust and maintainable systems.**

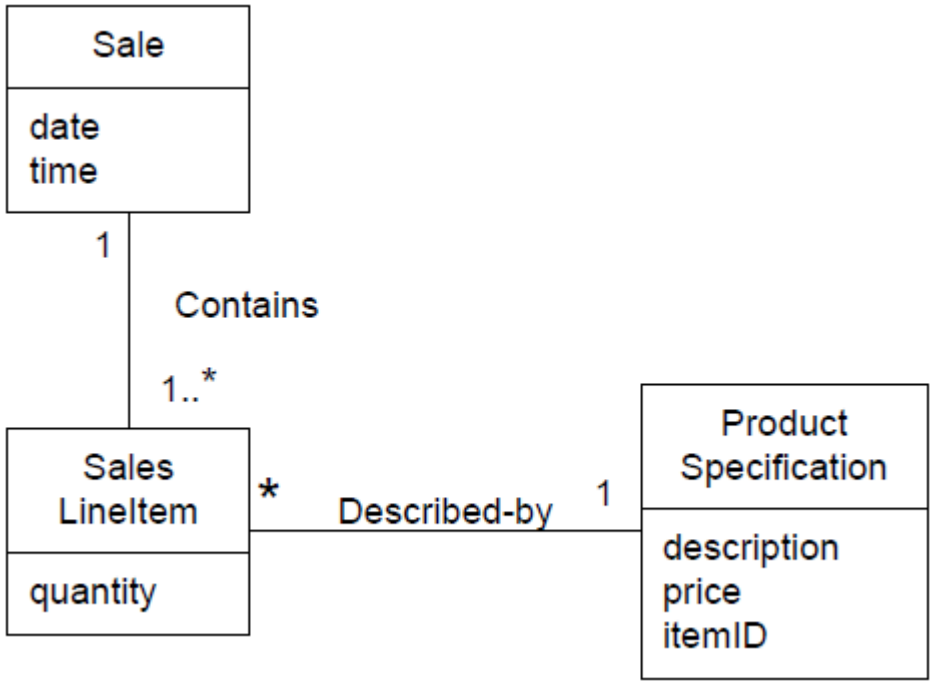
Software Classes Methods



Associations of POS System

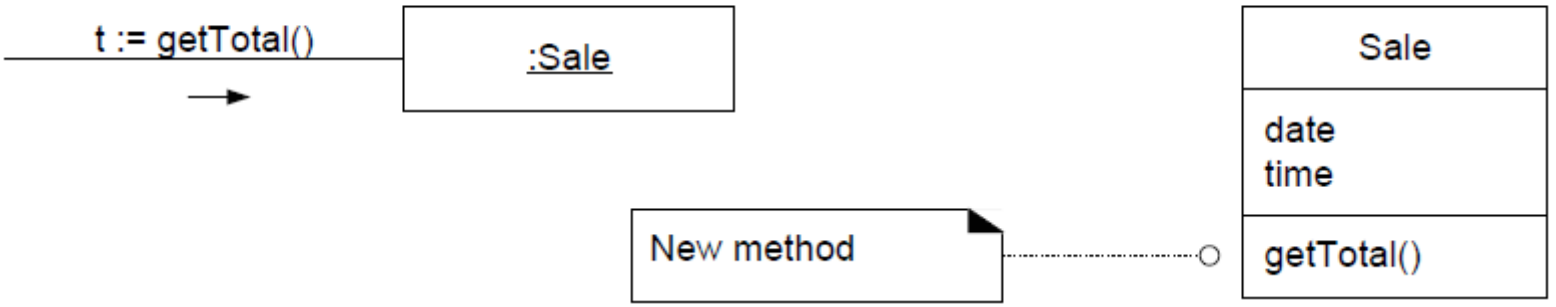


Associations of POS System



- What are the potential responsibilities of Sale, SalesLineItem and Product Specification objects?

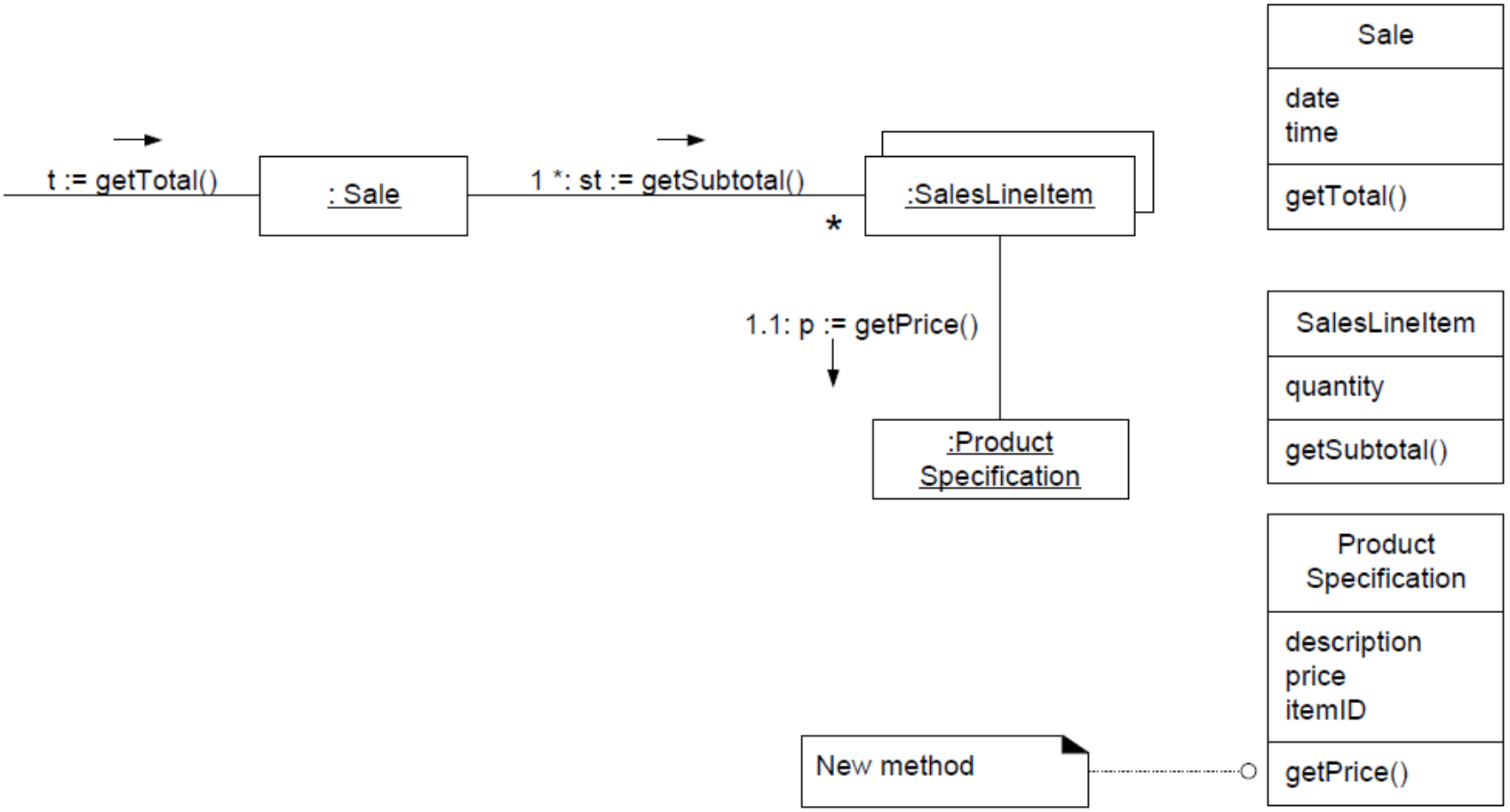
Partial Sequence and Class Diagrams



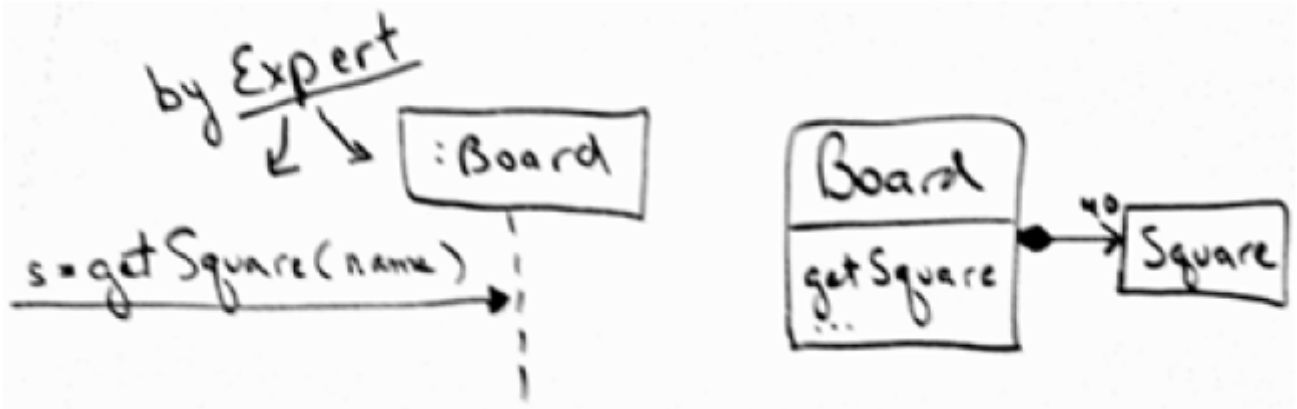
Class Responsibilities

Design Class	Responsibility
Sale	knows sale total
SalesLineItem	knows line item subtotal
ProductSpecification	knows product price

Calculating Sale Total



Applying Expert to Monopoly

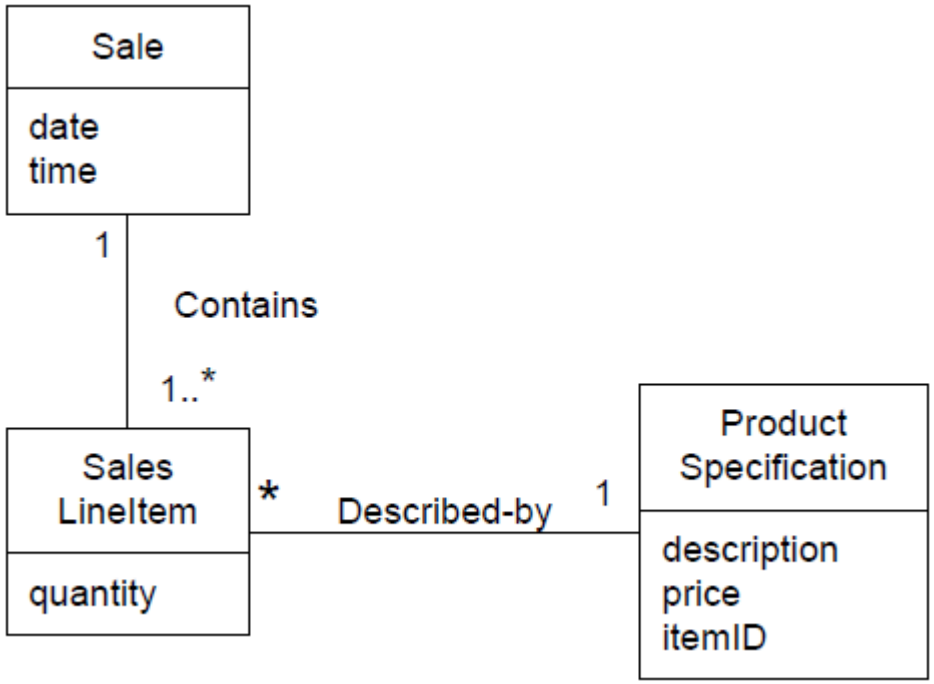


Creator

- **Solution:**
  - **Assign class B the responsibility to create an instance of class A if one or more of the following is true:**
    - **B *aggregates* A objects.**
    - **B *contains* A objects.**
    - **B *closely uses* A objects.**
    - **B *has the initializing data* that will be passed to A when it is created.**
- **Problem it solves:**
  - **Who should be responsible for creating a new instance of some class?**

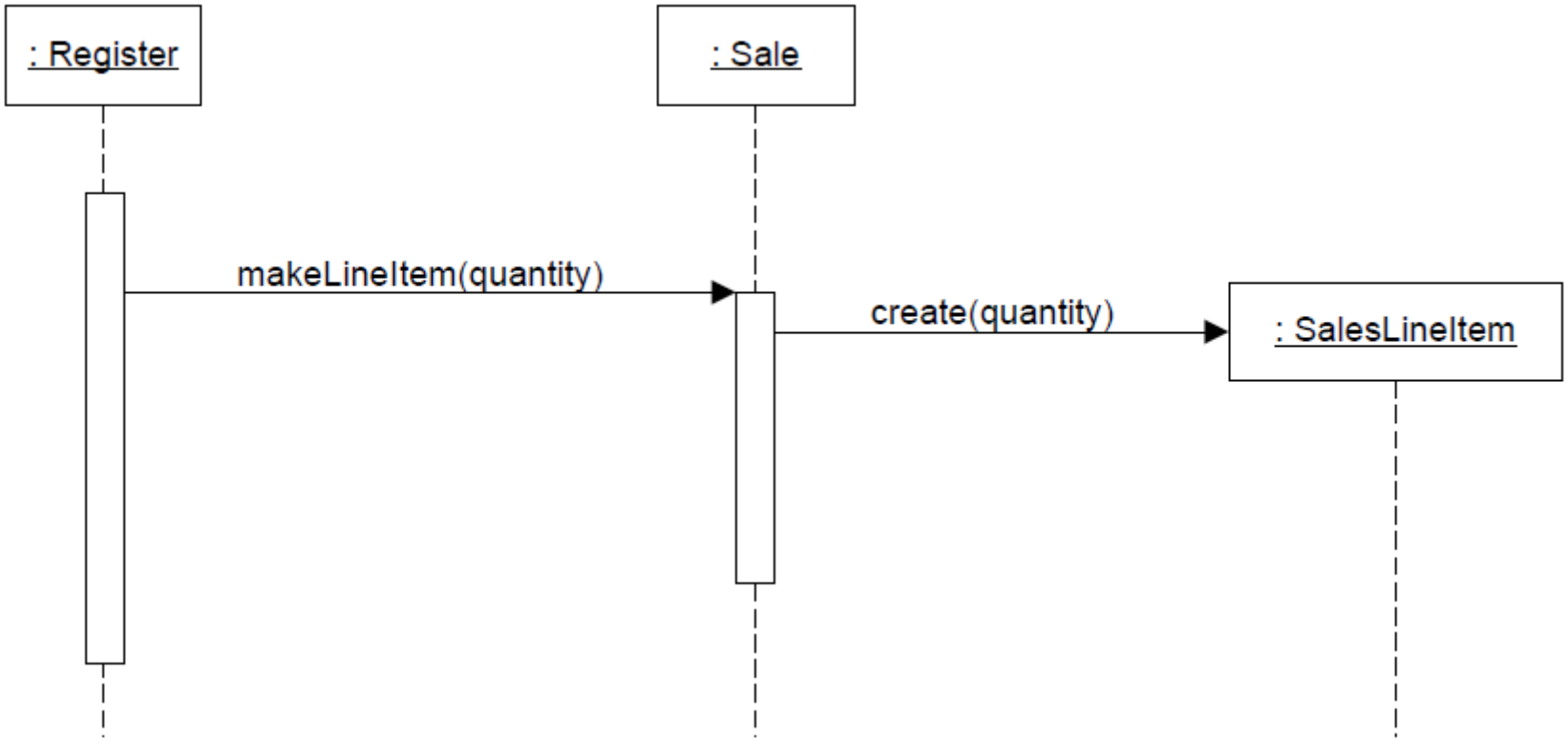
Partial Domain Model

- Who should create a *SalesLineItem* instance?

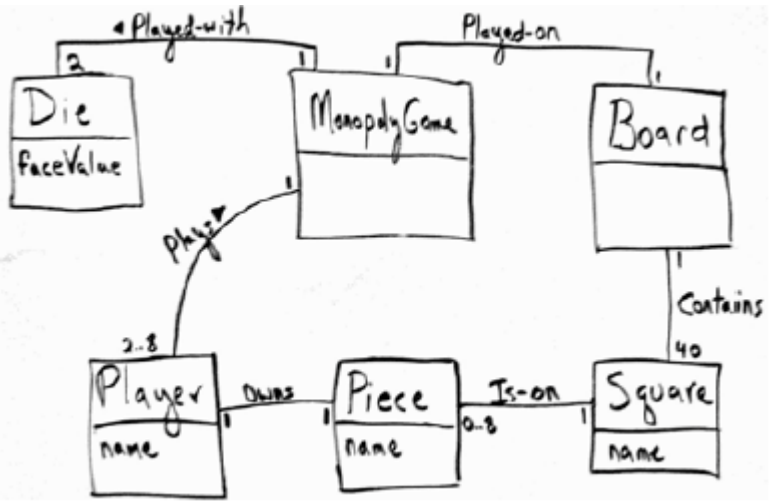


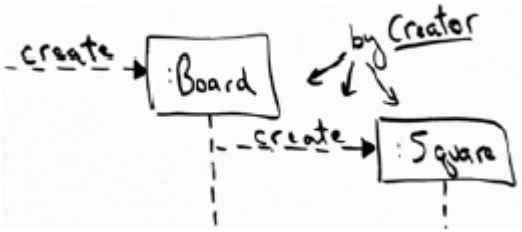
Creating Sales Line Item

- Sales should create, since it contains (aggregates) many *SalesLineItem* objects.



Monopoly Game (Creator)





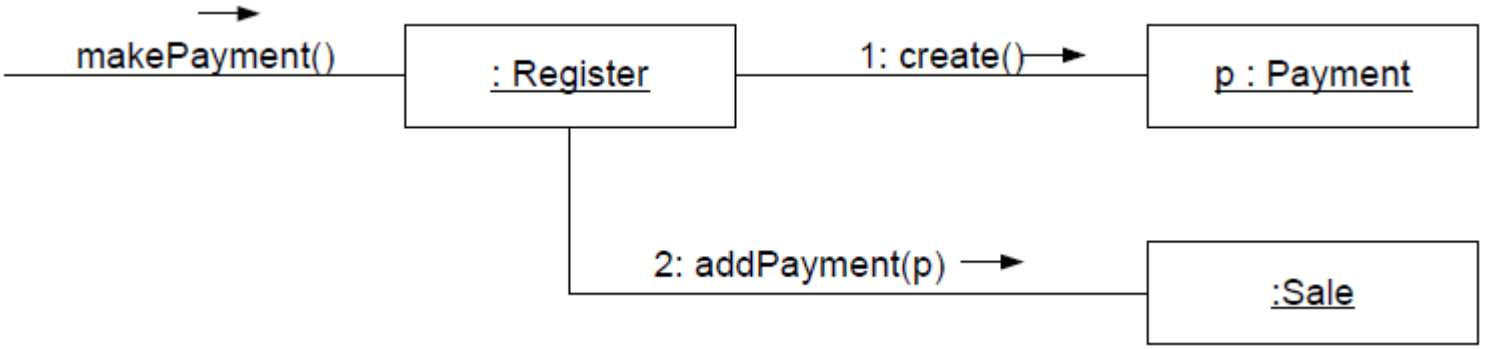
Low Coupling

- **Coupling is a measure of how strongly one element is connected to or relies on other elements.**
- **Solution:**
  - **Assign a responsibility so that coupling remains low.**
- **Problem it solves:**
  - **How to support low dependency, low change impact, and increased reuse?**

NextGen Case Study

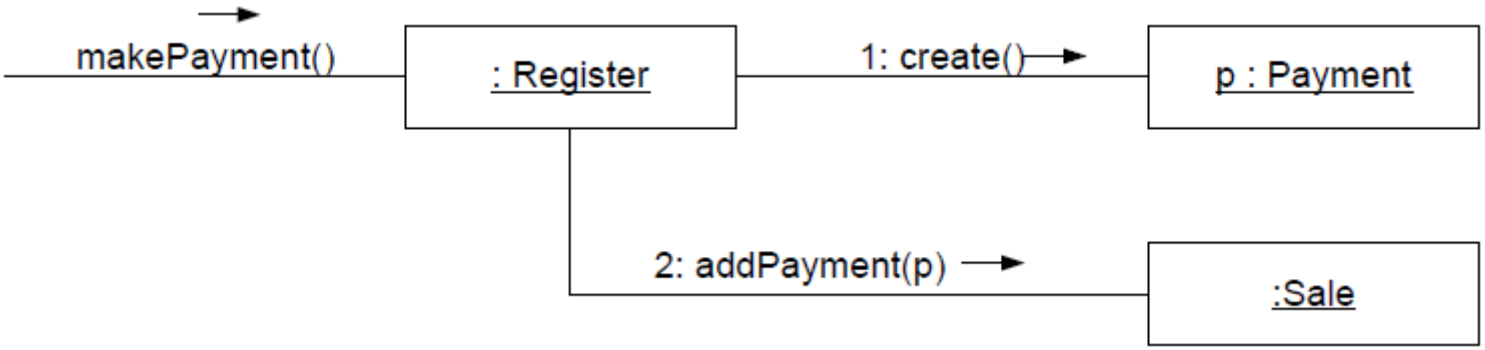
Register Creates Payment

- Register creates and sends payment object *p*.
- This assignment of responsibilities couples the *Register* class to knowledge of the *Payment* class.



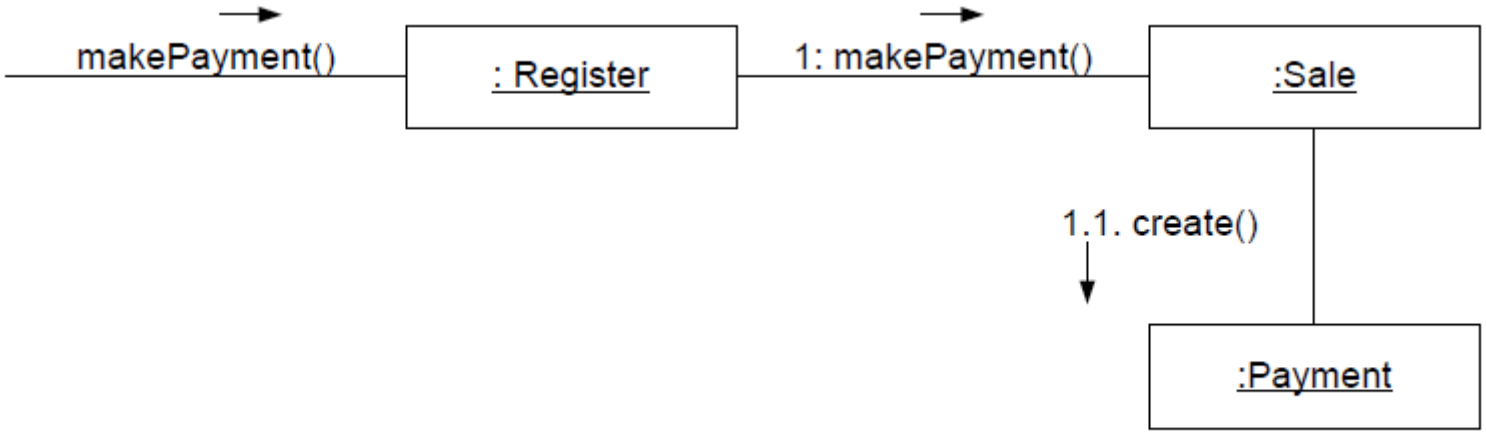
Register Creates Payment

- Register creates and sends payment object *p*.
- This assignment of responsibilities couples the *Register* class to knowledge of the *Payment* class.
  - Is it a good design in terms of coupling?



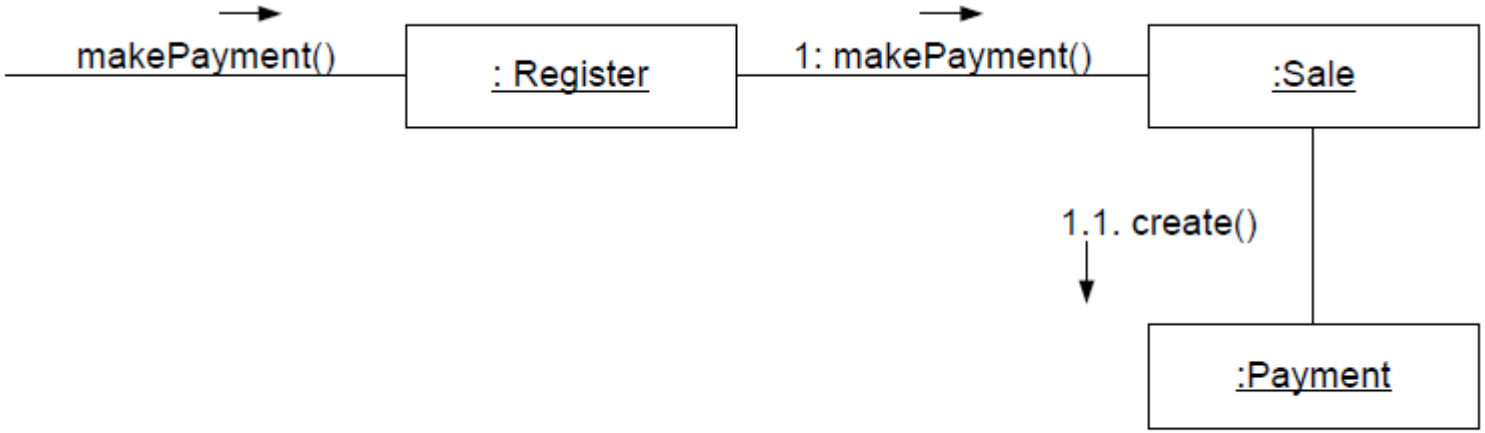
Sales Creates Payment

- An alternative solution to creating the *Payment* and associating it with the *Sale* is:



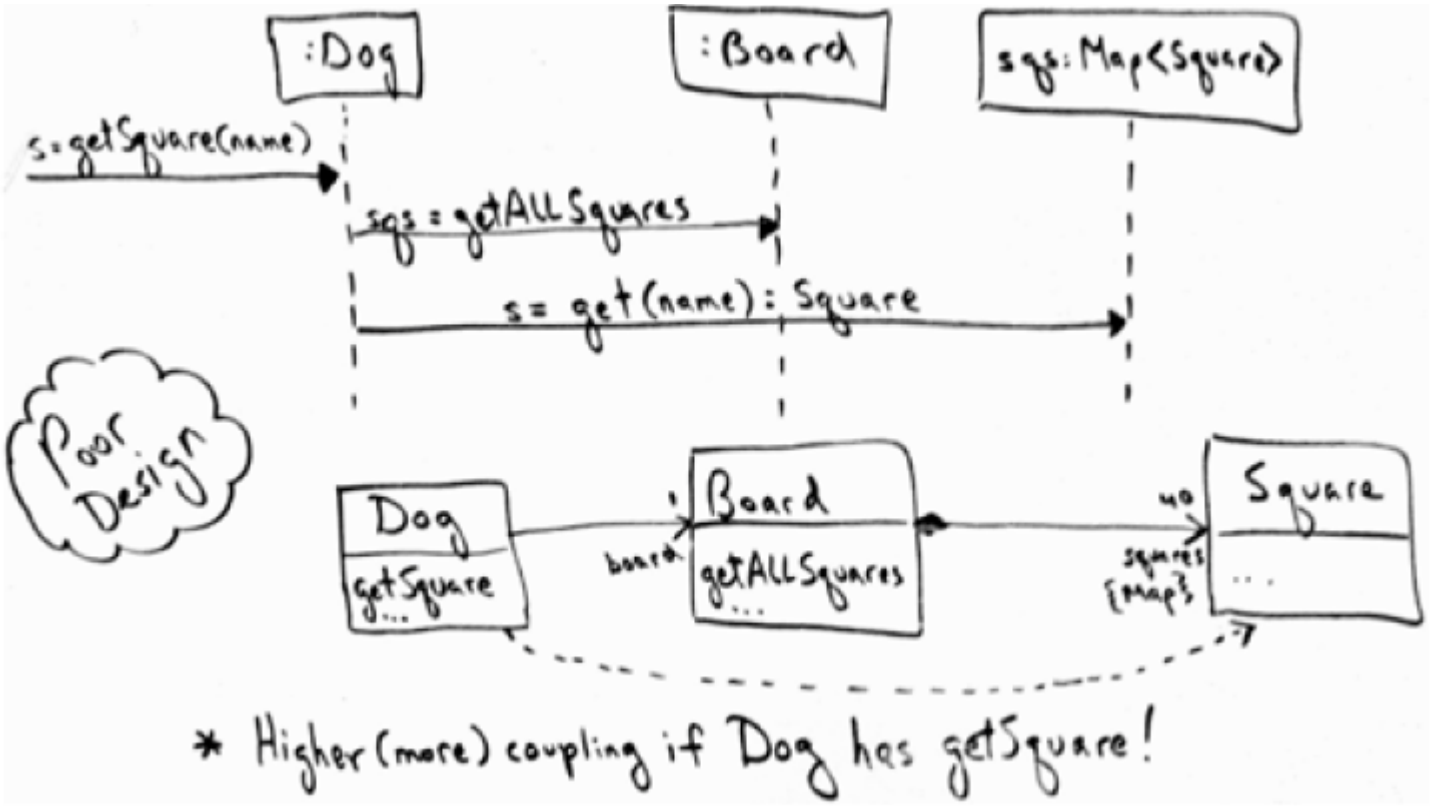
- Which design is better?

Sales Creates Payment



- Design Two is preferable because overall **lower coupling is maintained**, i.e., no coupling from Register to Payment.

Coupling in Monopoly

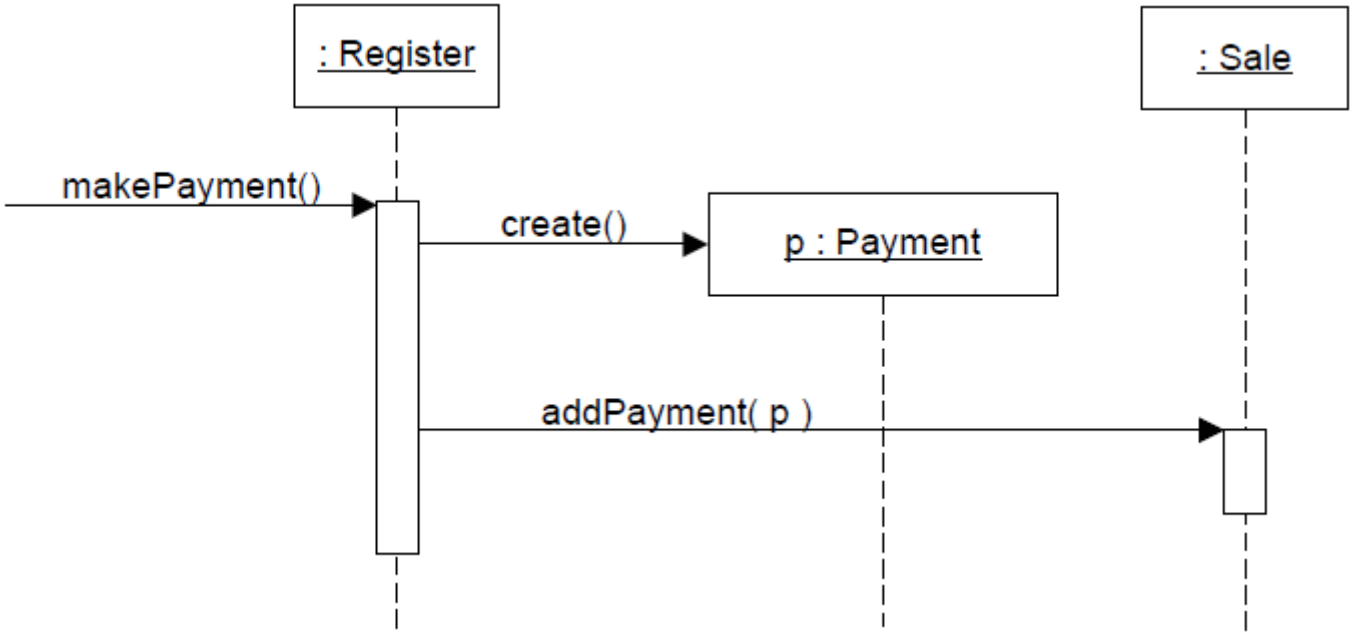


- The design with Board having *getSquare* method has lower coupling

High Cohesion

- **Cohesion is a measure of how strongly related the responsibilities of an element are.**
  - An element with highly related responsibilities has high cohesion.
- **Solution:**
  - **Assign a responsibility so that cohesion remains high.**
- **Problem it solves:**
  - **How to keep complexity manageable?**

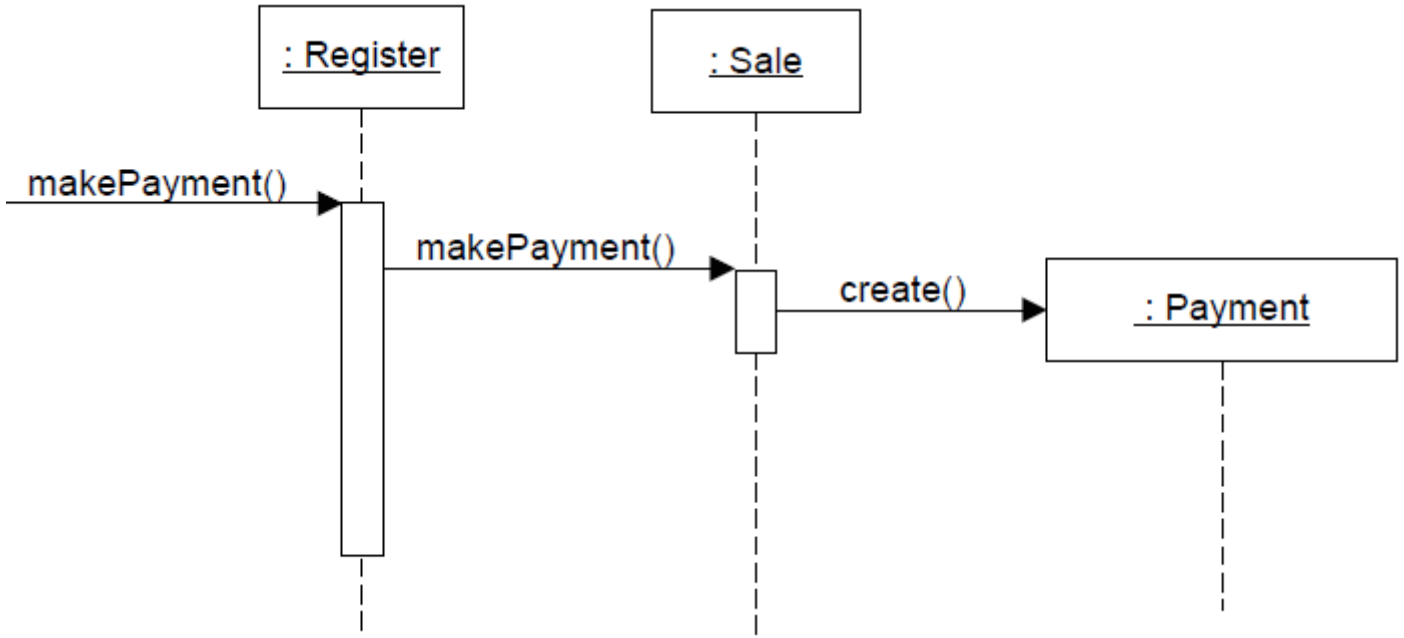
Register Creates Payment



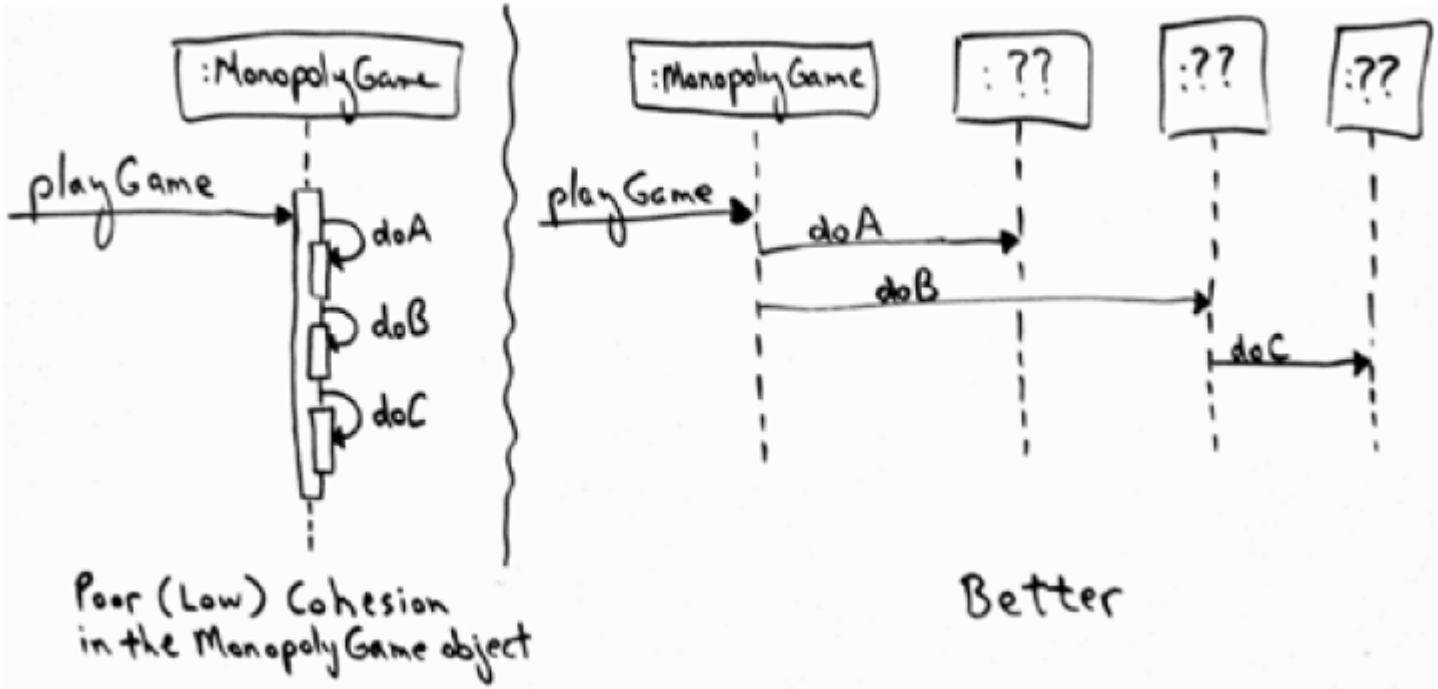
- Is it a good design in terms of high cohesion?

Sale Creates Payment

- **The second design delegates the payment creation responsibility to the *Sale*, which supports higher cohesion.**



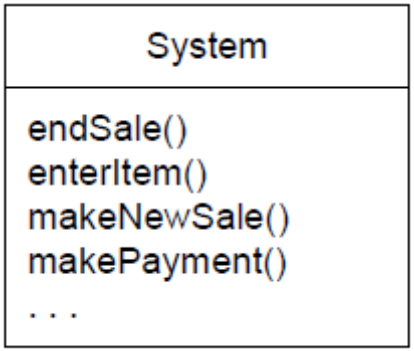
Monopoly Cohesion



Controller

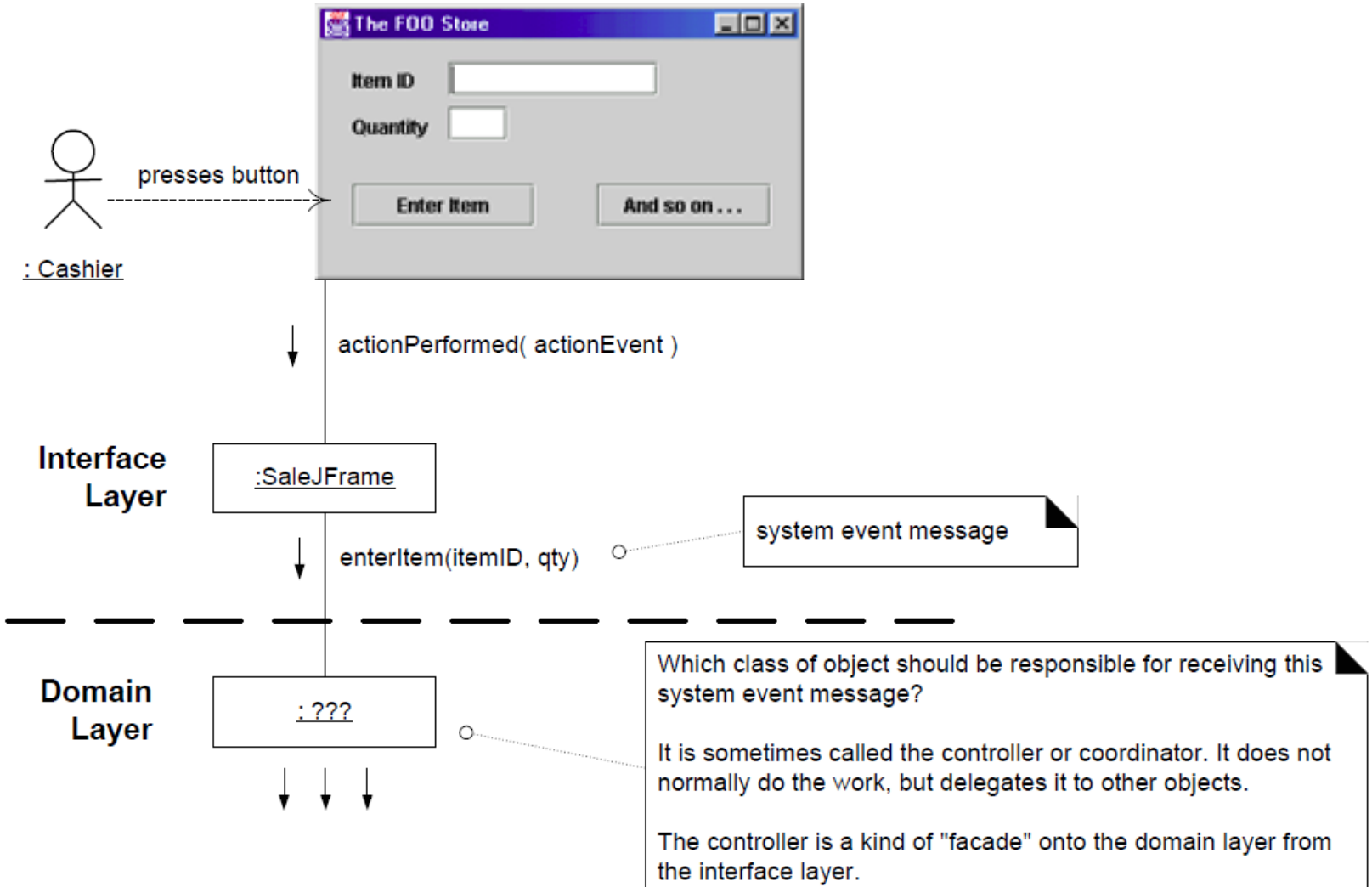
- A Controller is a non-user interface object responsible for receiving or handling a system event.
- Solution:
  - Assign the responsibility for receiving or handling a system event message for an object(s) that delegate it.
- Problem it solves:
  - Who should be responsible for handling an input system event?

System Operations Associated with System Events



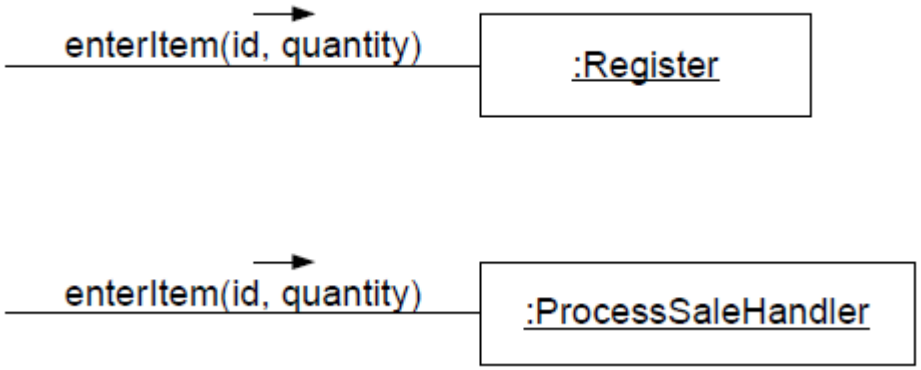
Controller for enterItem?

- Who should be the controller for system events such as *enterItem* and *endSale*

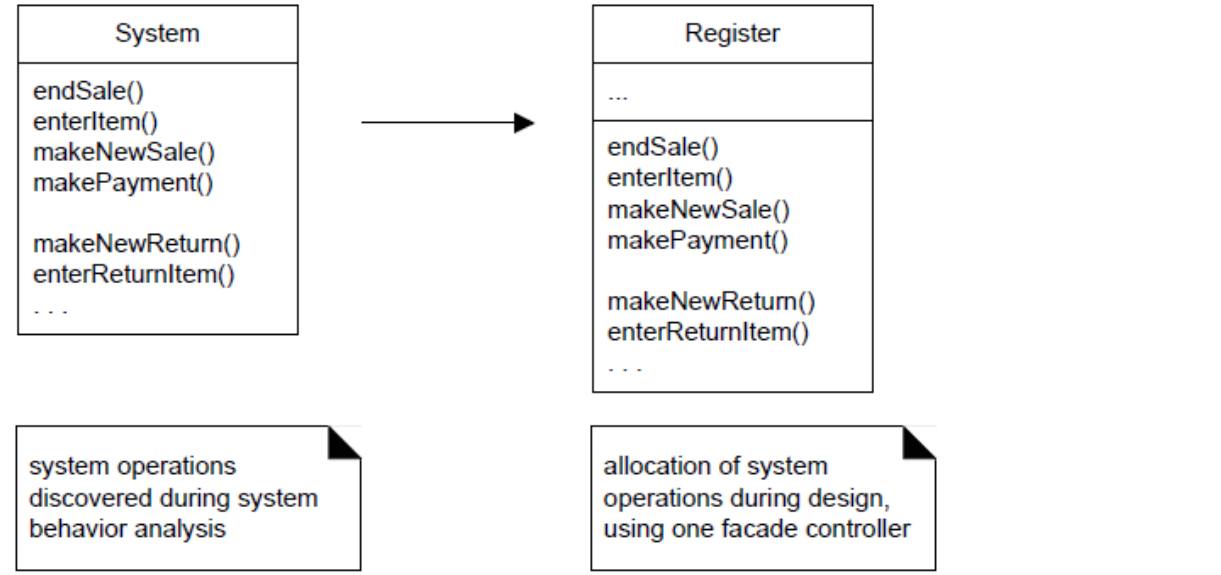


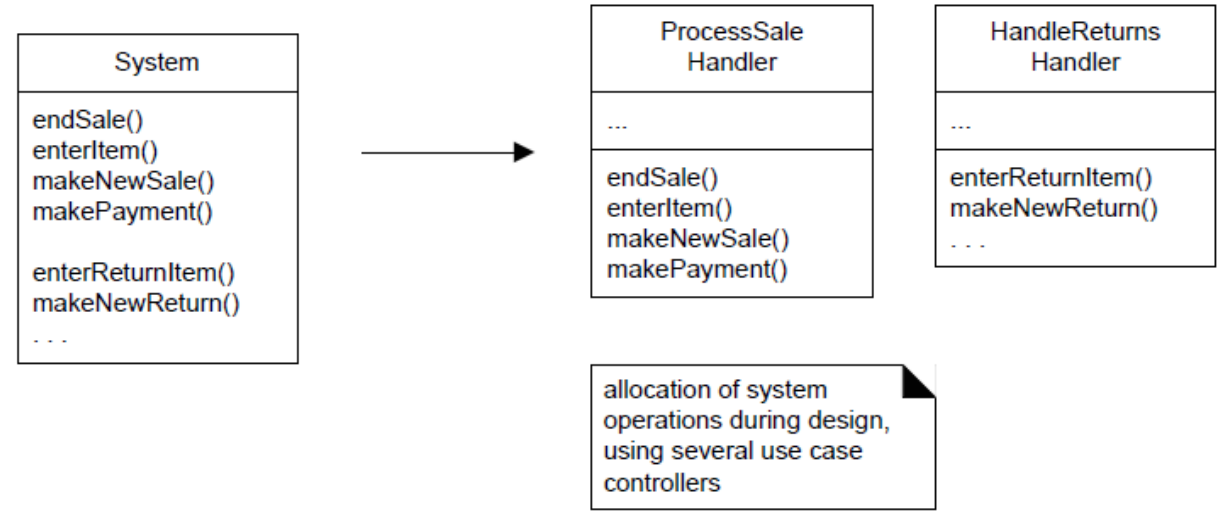
Controller Choices

- Normally, a controller should *delegate* to other objects the work that needs to be done;
  - it coordinates or controls the activity. It does not do much work itself.



Allocation of System Operations



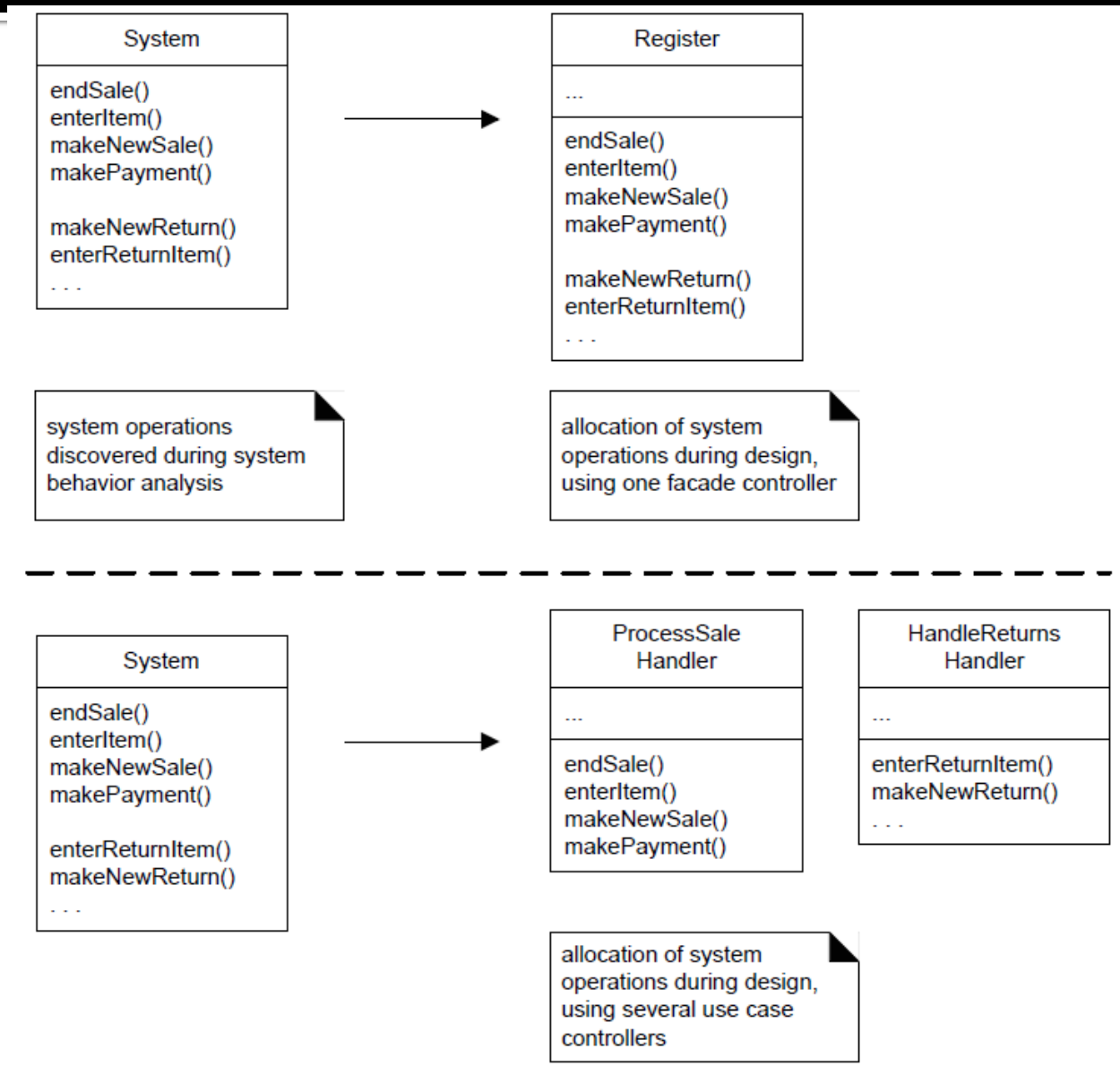


Bloated Controllers

- **Poorly designed, a controller class will have low cohesion**
  - There is only a *single* controller class receiving *all* system events in a complex system.
  - The controller itself performs many of the tasks necessary to fulfill the system event without delegating the work.
  - A controller has many attributes, and maintains significant information about the system, which should have been distributed to other objects.

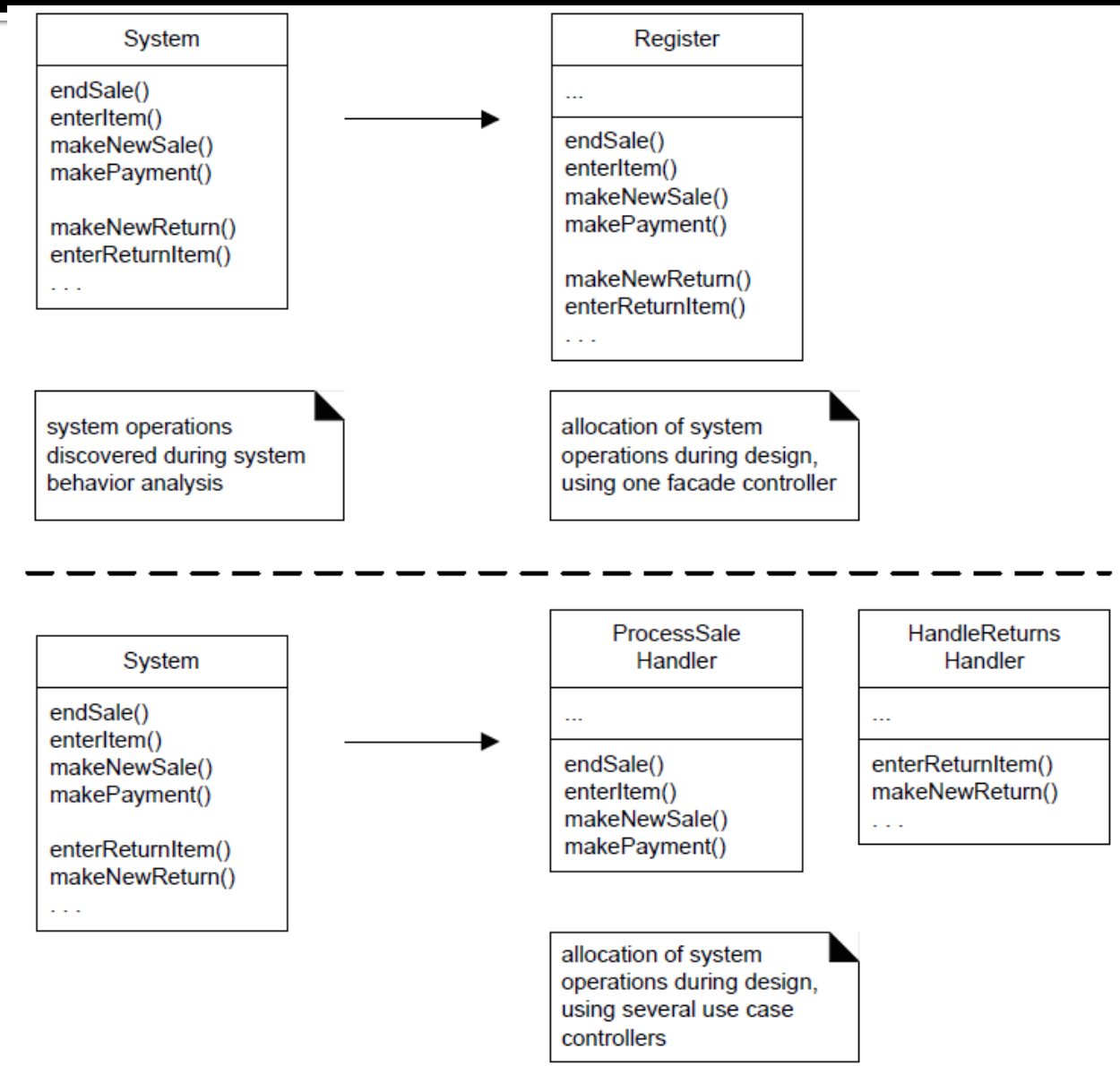
Allocation of System Operations

- Which design is more recommended in this case?

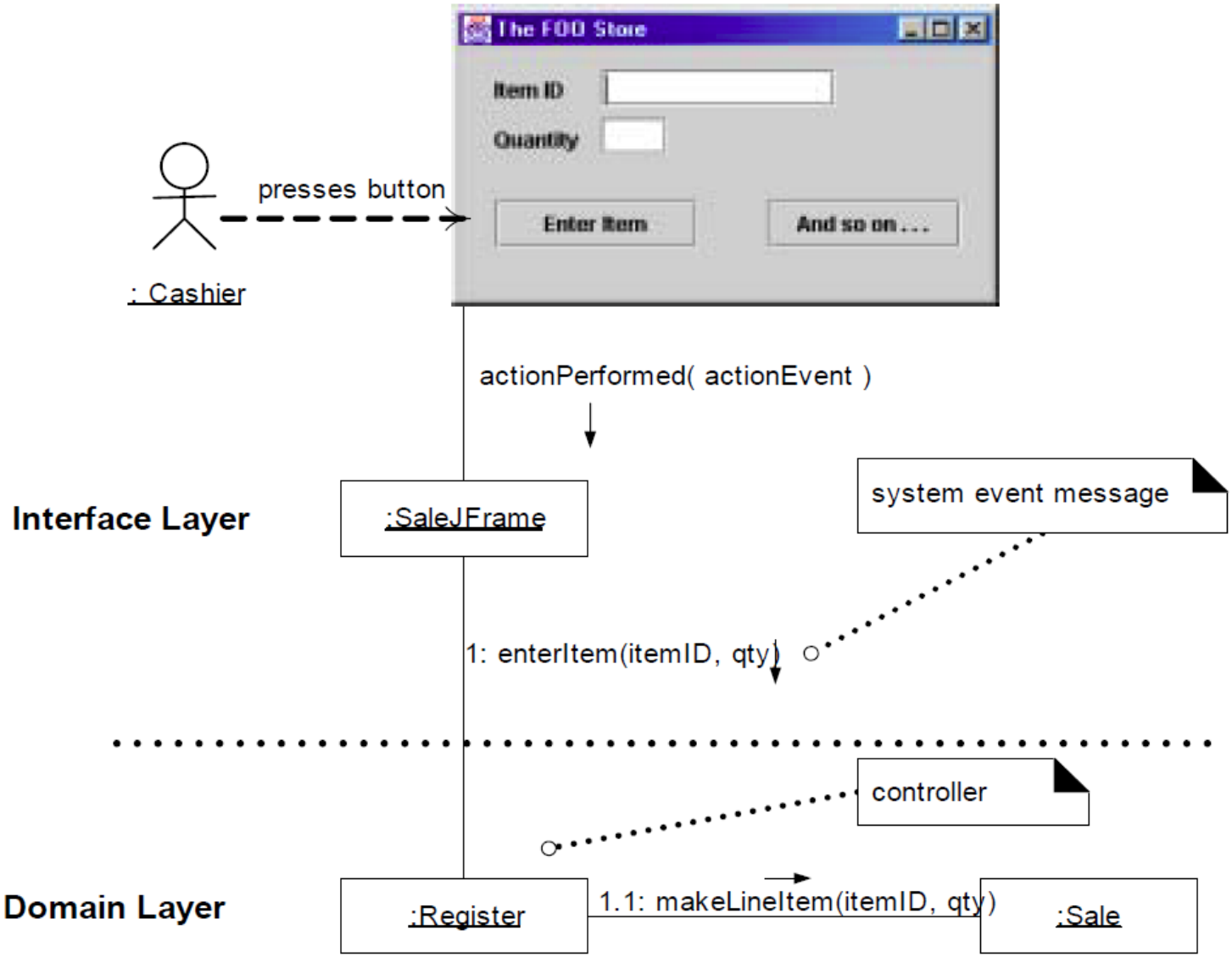


Allocation of System Operations

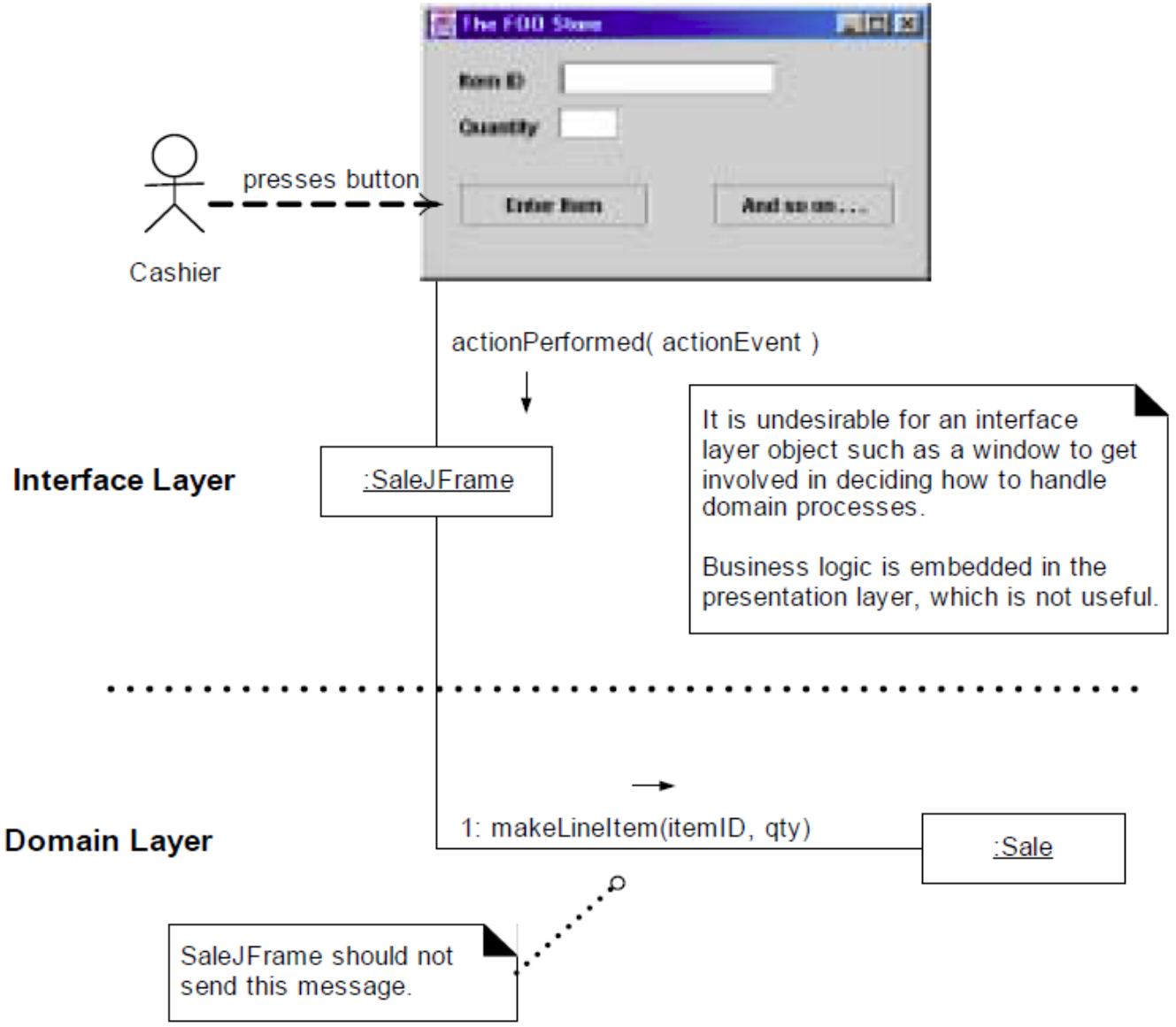
- 2nd design has higher cohesion.



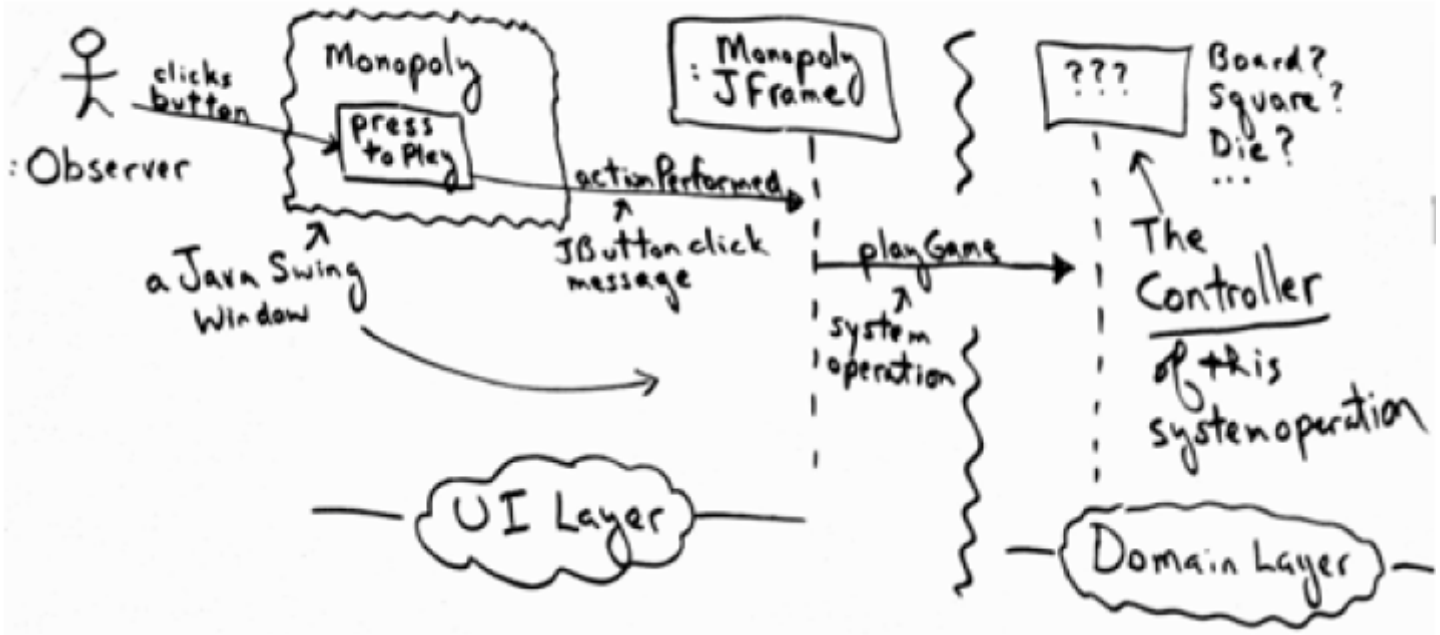
Coupling of Interface to Domain Layer



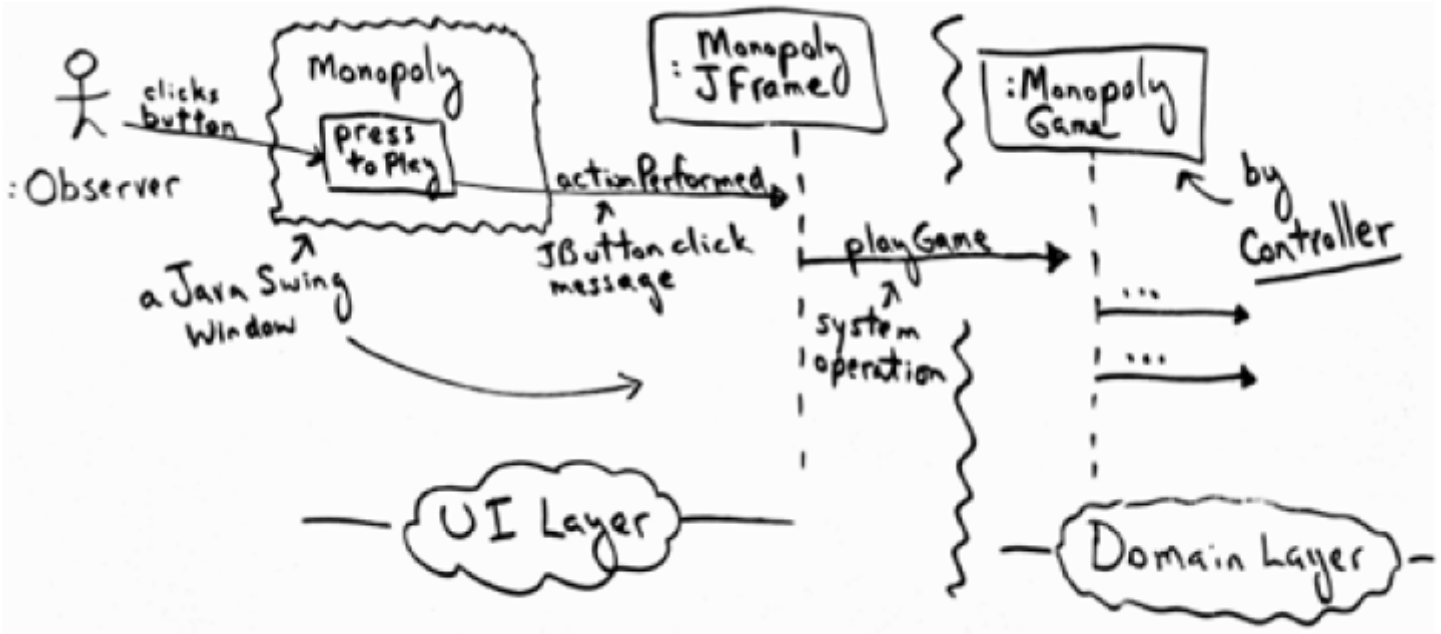
Less Desirable Coupling



Monopoly Controller



Monopoly Controller?



Quiz

- What are the two types of responsibilities?
- What is GRASP?
- What are the five basic types of patterns?
- What is the problem and solution for controller pattern?

Actions

- Review Slides.
- Read Chapter 16 and 17
  - *Applying UML and Patterns*, Craig Larman