GRASP (cont.)

Cpt S 321

Washington State University

GRASP: 9 principles for assigning responsibilities

- We are *already familiar* with some of them:
  - **Information Expert**
  - **Low Coupling**
  - **High Cohesion**
  - **Polymorphism**

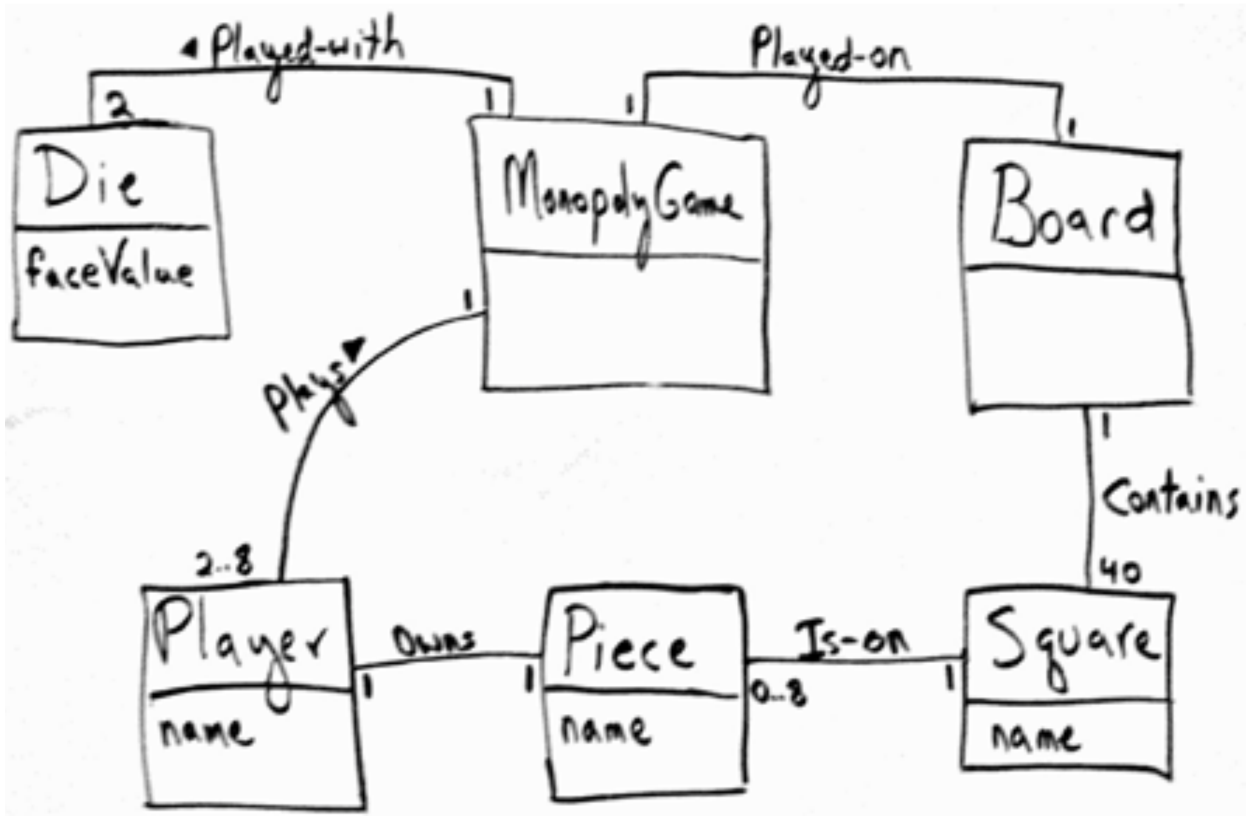
- Today we *will learn about*:
  - **Creator**
  - **Controller**
  - **Pure Fabrication**
  - **Indirection**
  - **Protected variations**

Domain model

- Definition: A conceptual model of the **problem domain**.
- Represents concepts and their relations
- Represents both state and behavior
- Resembles a class diagram visually but at a more high level – no implementation details are shown.

Example: Monopoly

- Domain model



Point-Of-Sale (POS) application

- Similar to the one used in retail stores: includes hardware, code bar scanner, and <u>software</u>
- Record sales
- Handle payments
- Customizable logic
    - When a new sale is instantiated
    - When a new line item is added

Example of Use Case: **UC1: Process Sale**

- **Primary Actor**: Cashier
- **Main Success Scenario**:

Actor Action (or Intention)	System Responsibility
1.Customer arrives at a POS checkout with goods and/or services to purchase.	
2.Cashier starts a new sale.	
3.Cashier enters item identifier.	4.Records each sale line item and presents item description and running total.
Cashier repeats steps 3-4 until indicates done.	5.Presents total with taxes calculated.
6.Cashier tells Customer the total, and asks for payment.	
7.Customer pays.	8.Handles payment.
	9.Logs the completed sale and sends information to the external accounting (for all accounting and commissions) and inventory systems (to update inventory). System presents receipt.

POS application

How would the domain model look like?

GRASP (cont.)

- **Creator**

  Problem: Who should be responsible for creating a new instance of some class?

  Solution: Assign class B the responsibility to create an instance of class A if one of these is true (the more the better):

  - B "contains" or **compositely aggregates*** A. (see next slide for definition)
  - B records A.
  - B closely uses A.
  - B has the initializing data for A that will be passed to A when it is created. Thus B is an Expert with respect to creating A.
  - B is a creator of A objects.

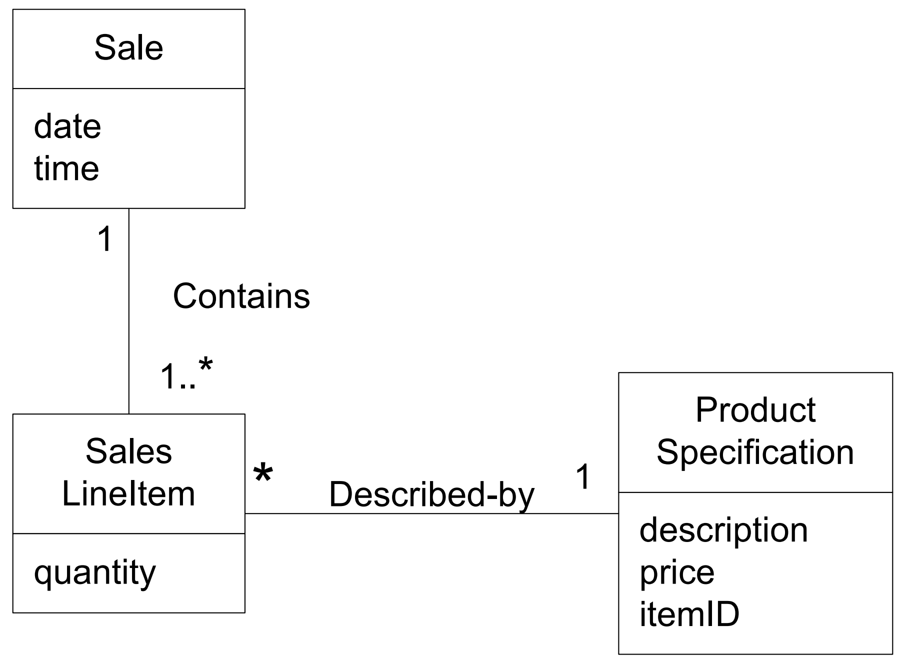
  If more than one option applies, usually prefer a class B which compositely aggregates or contains class A.

Composition (or composite aggregation)

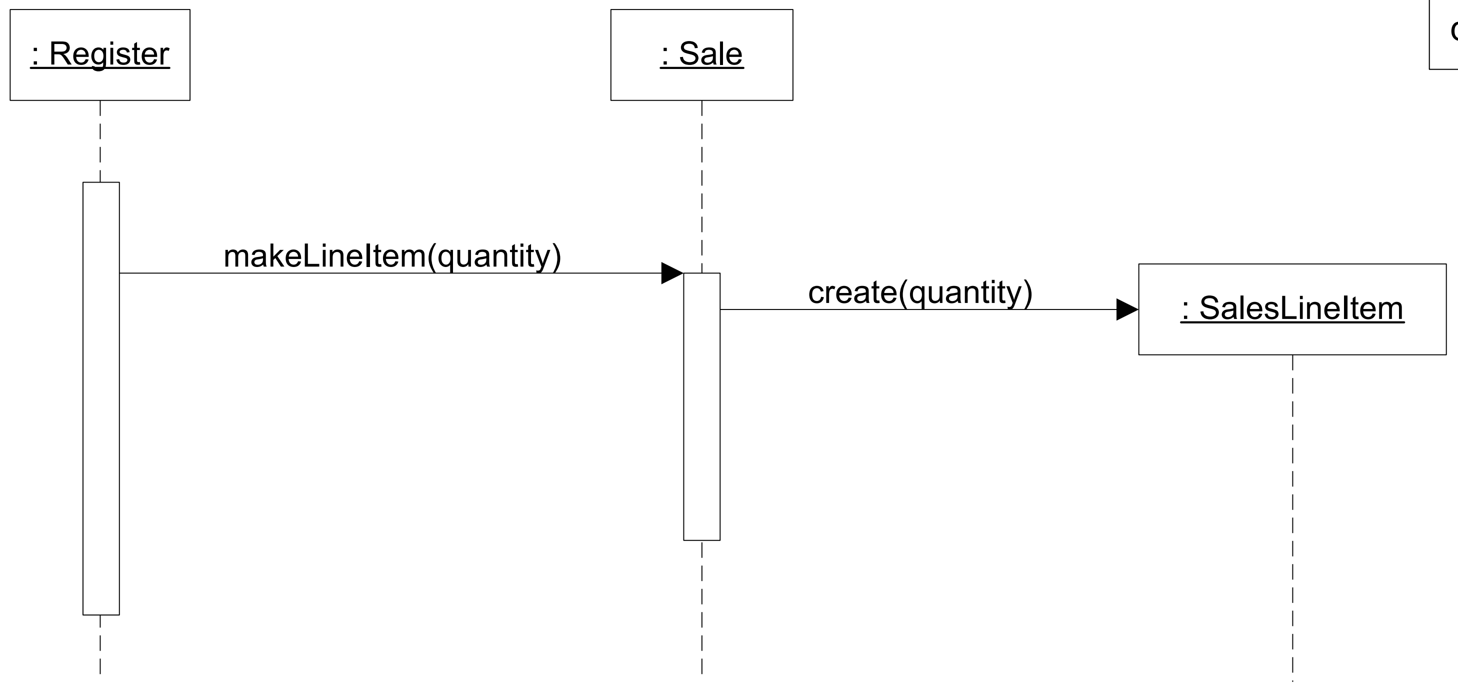
- A strong kind of whole-part aggregation
- A composition relationship implies that
  1) an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time,
  2) the part must always belong to a composite (no free-floating Fingers), and
  3) the composite is responsible for the creation and deletion of its parts either by itself creating/deleting the parts, or by collaborating with other objects. Related to this constraint is that if the composite is destroyed, its parts must either be destroyed, or attached to another composite.
     Again: no free-floating Fingers allowed!

GRASP (cont.)

- **Creator** – example: In a POS application, who should be responsible for creating a SalesLineItem instance?



Partial domain model



Interaction diagram

GRASP (cont.)

- **Controller**

  Problem: What first object beyond the UI layer receives and coordinates ("controls") a system operation?

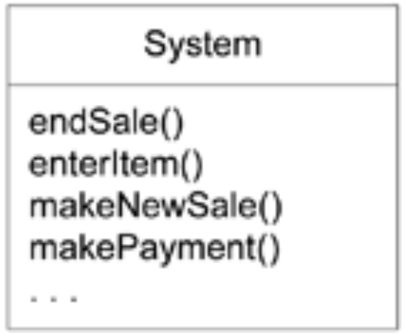
  Solution: Assign the responsibility to a class representing one of the following choices:

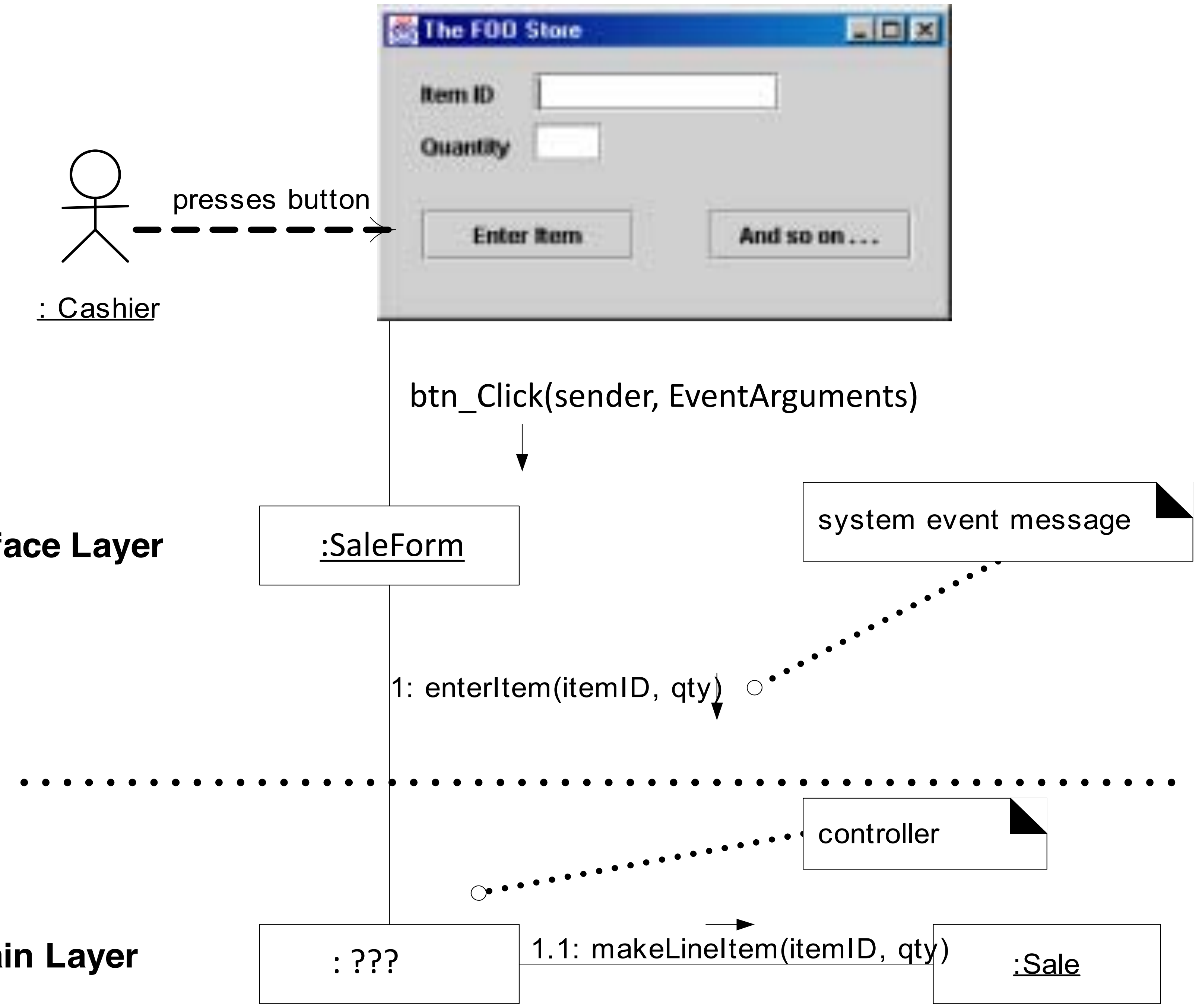
  - Represents the overall "system," a "root object," a device that the software is running within, or a major subsystem.
  - Represents a use case scenario within which the system event occurs, often named <UseCaseName>**Handler**, <UseCaseName>**Coordinator**, or <UseCaseName>**Session**. Use the same controller class for all system events in the same use case scenario.

  Note that ”form”, "window," "view," and "document" classes are not on this list. Such classes should not fulfill the tasks associated with system events; they typically receive these events and delegate them to a controller.

GRASP (cont.)

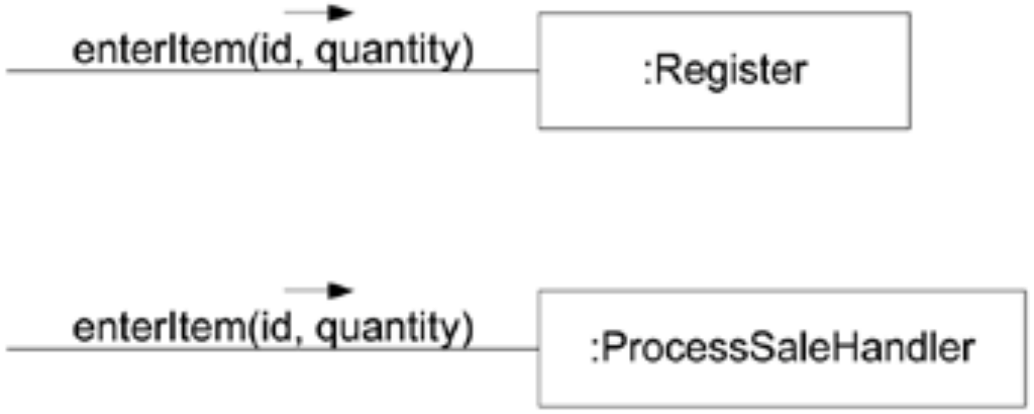
- **Controller** – example

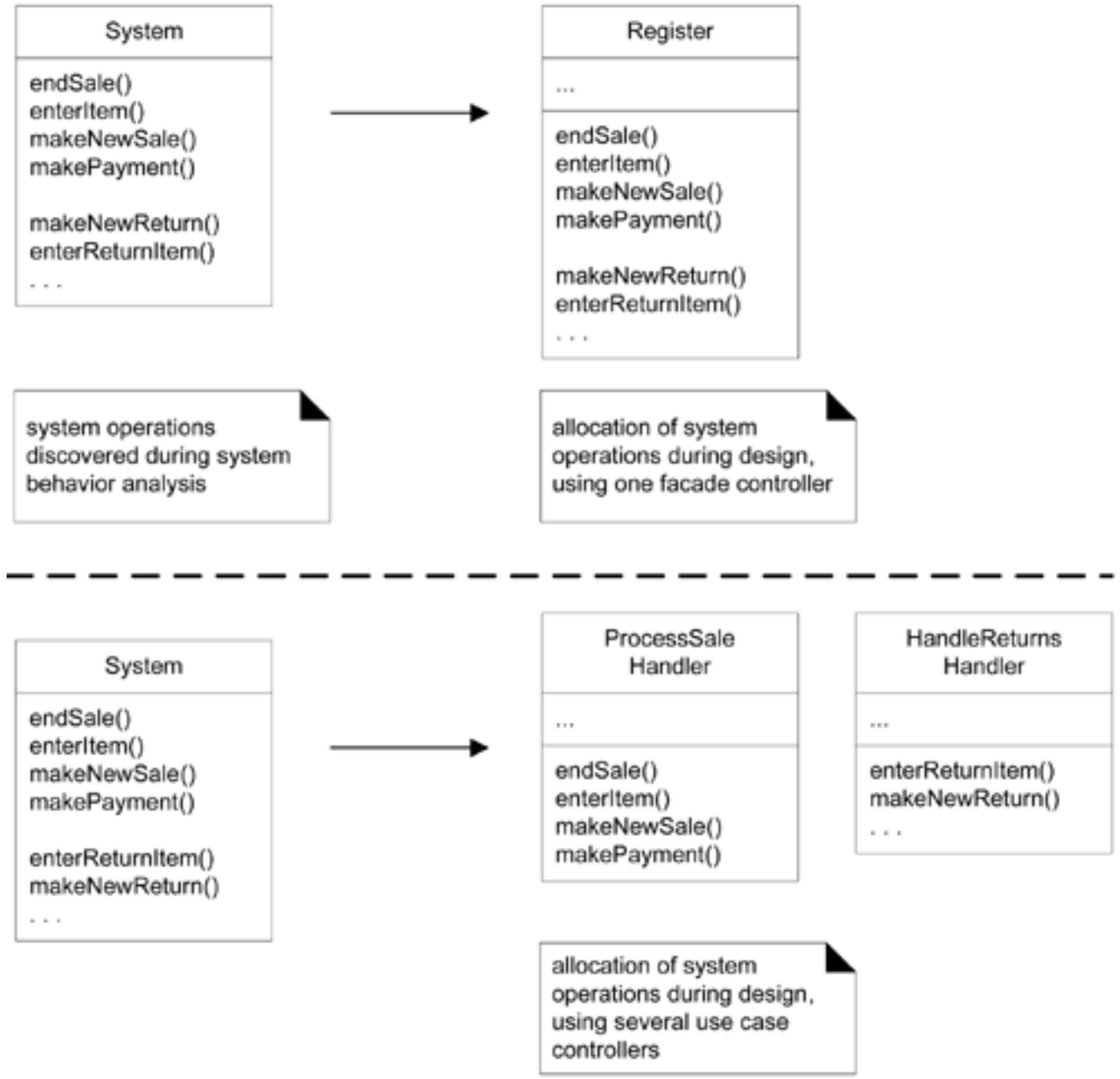




GRASP (cont.)

- **Controller** –
  example (cont.)





GRASP (cont.)

- **Pure Fabrication**

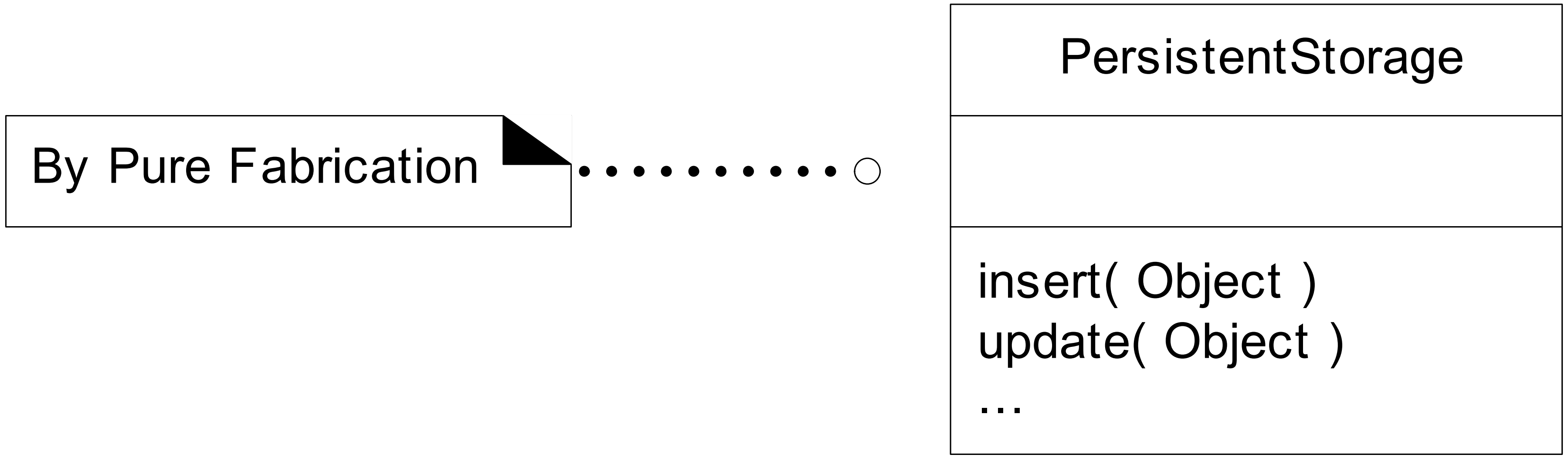
  Problem: What object should have the responsibility, when you do not want to violate High Cohesion and Low Coupling, or other goals, but solutions offered by Expert (for example) are not appropriate?

  Solution:

  - Assign a highly cohesive set of responsibilities to an artificial or convenience class that does not represent a problem domain concept something made up, to support high cohesion, low coupling, and reuse.
  - Such a class is a fabrication of the imagination. Ideally, the responsibilities assigned to this fabrication support high cohesion and low coupling, so that the design of the fabrication is very clean, or pure hence a pure fabrication.

GRASP (cont.)

- **Pure Fabrication** – example: Suppose that support is needed to save Sale instances in a relational database.
  - Option 1: Assign the responsibility to the Sale class (as per Information Expert)
  - Option 2: Create a new class PersistentStorage
  - Why is Option 2 a better design choice?



GRASP (cont.)

- **Indirection**

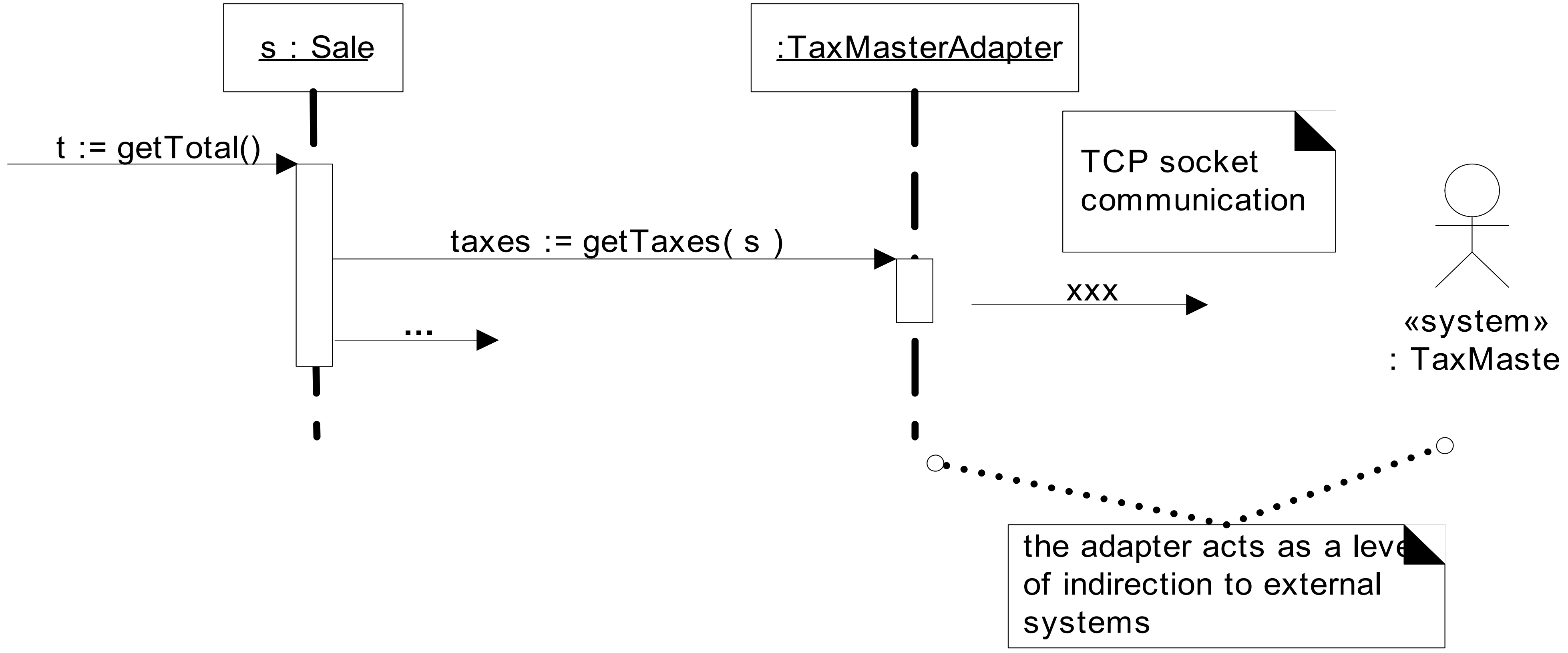
  Problem: Where to assign a responsibility, to avoid direct coupling between two (or more) things? How to de-couple objects so that low coupling is supported and reuse potential remains higher?

  Solution:

  - Assign the responsibility to an intermediate object to mediate between other components or services so that they are not directly coupled.
  - The intermediary creates an indirection between the other components.

GRASP (cont.)

- **Indirection** - example



GRASP (cont.)

- **Protected Variations**

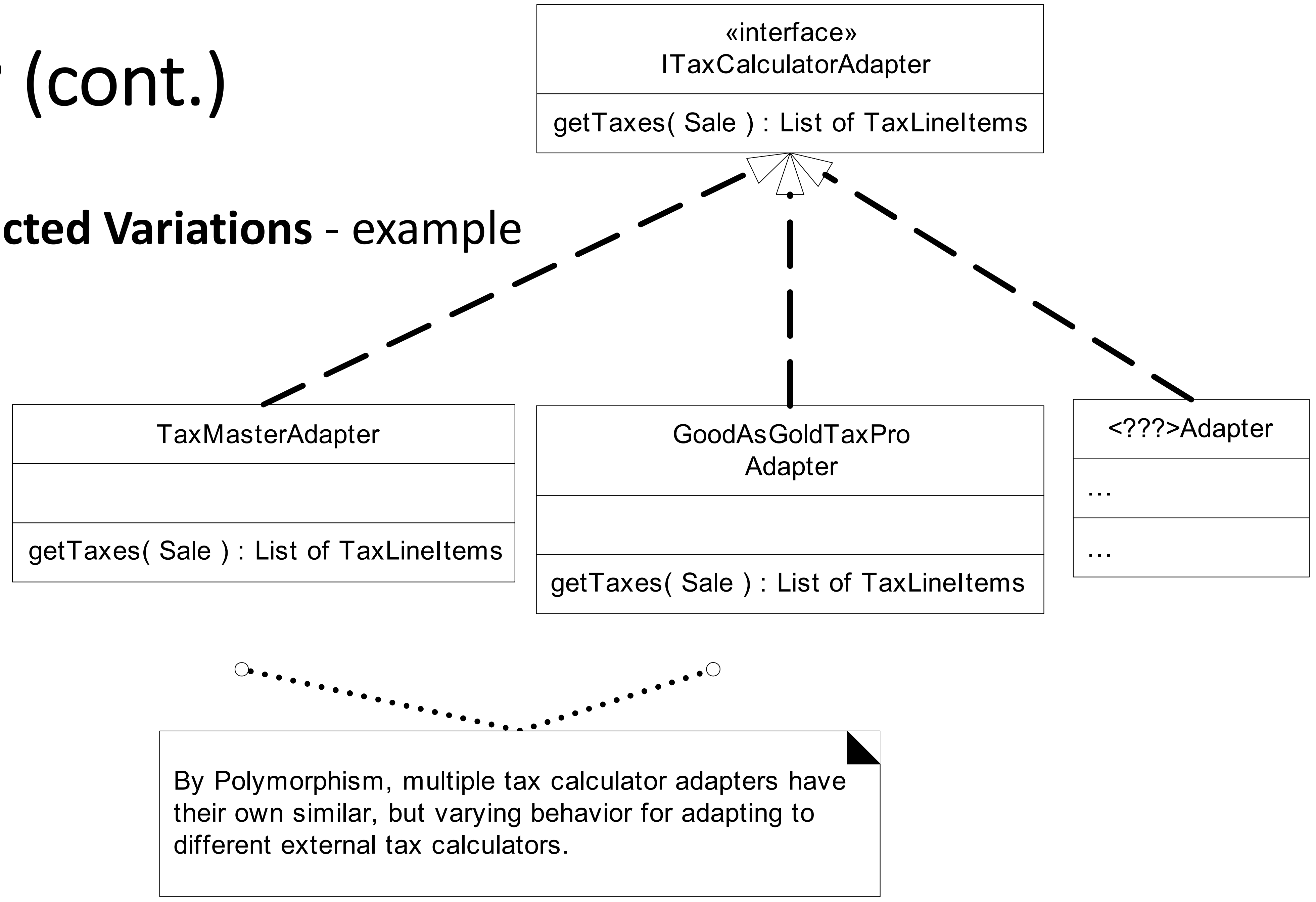
  Problem: How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

  Solution: Identify points of predicted variation or instability; assign responsibilities to create a stable interface around them.

  Note: The term "interface" is used in the broadest sense of an access view; it does not literally only mean a C# interface.

GRASP (cont.)

- **Protected Variations** - example

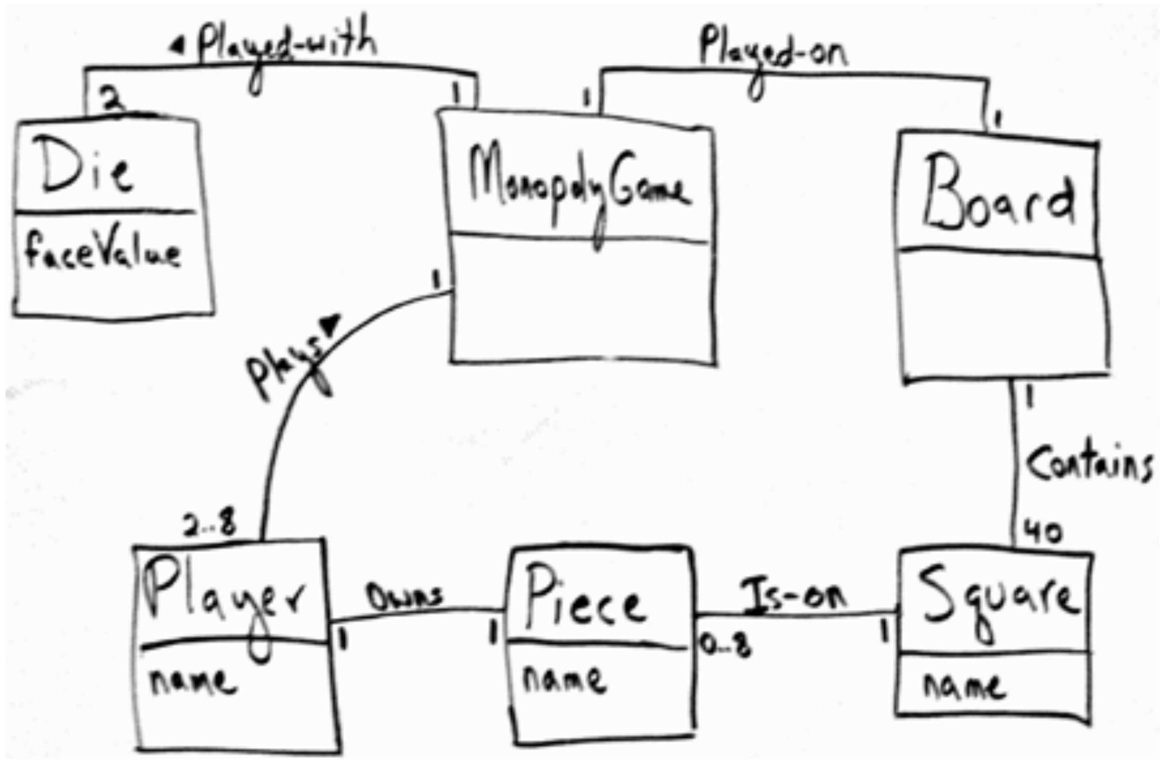


«interface»
ITaxCalculatorAdapter
getTaxes(Sale) : List of TaxLineItems
TaxMasterAdapter
getTaxes(Sale) : List of TaxLineItems
GoodAsGoldTaxPro Adapter
getTaxes(Sale) : List of TaxLineItems
<???>Adapter
...
...

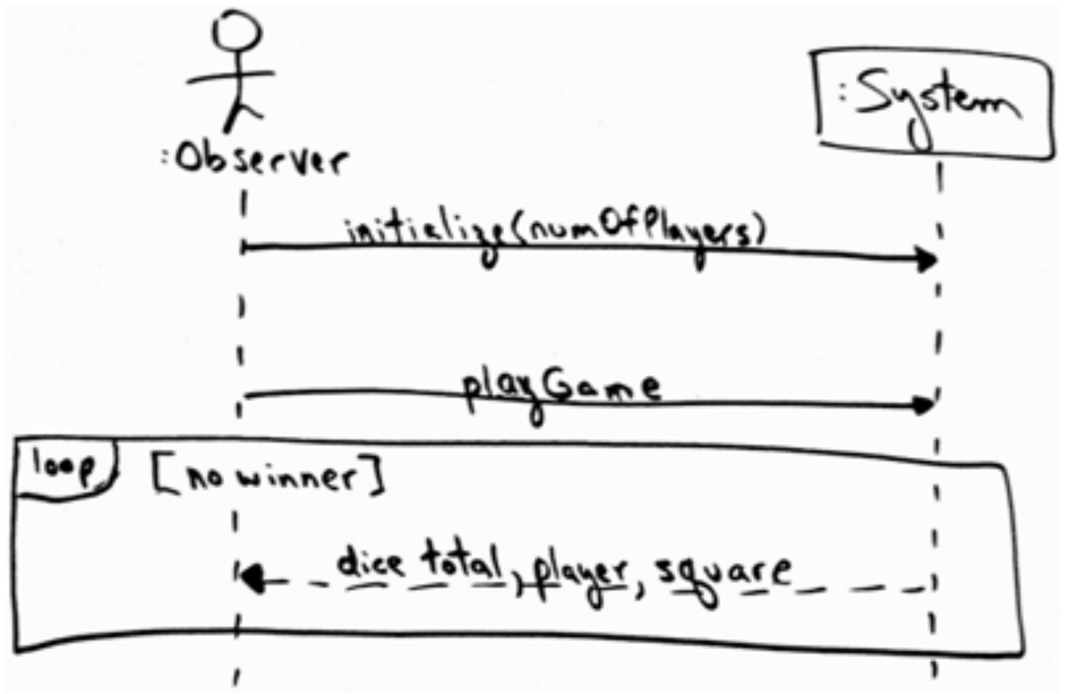
By Polymorphism, multiple tax calculator adapters have their own similar, but varying behavior for adapting to different external tax calculators.

Let’s practice

- Recall the Monopoly game



Domain model



SSD for the playGame operation

The Monopoly – questions

1. Who should create the Square object?
2. Who knows about a Square object, given a key?
3. Who is the Controller for the playGame system operation?
4. How should the controller interact with the rest of the classes in the domain layer?
5. Who should be responsible to handle the dice (rolling and summing the dice totals)?