

## Introduction to GRASP: Assigning Responsibilities to Objects

---

CS 4354  
Summer II 2016

Jill Seaman

Much of the material in this lecture is from the Larman book listed on the Readings page (accessible from the class website). There are also links to other (free) presentations and You Tube videos there.

1

## Object Analysis & Design in the textbook

---

- Chapter 2 Object Oriented Design Process
  - ◆ Gives good guidelines for identifying and assigning the following:
    - Classes
    - attributes
    - associations, aggregations, inheritance relationships
    - Good start to a class diagram representing the problem
  - ◆ But what about operations?
    - Sequence diagrams are good tools to explore interactions and operations
    - But little advice is given on how to decide who does what.

2

## The design of behavior

---

- What methods go in what classes? How should objects interact?
  - ◆ These are critical questions in the design of behavior.
  - ◆ Poor answers lead to abysmal, fragile systems with low reuse and high maintenance.

3

## Responsibility-Driven Design

---

- Assigns responsibilities to classes
- Methods are implemented to fulfill responsibilities of the given class.
- Methods may act alone or in collaboration to fulfill their obligations.
  
- Responsibilities of classes:
  - ◆ Knowing: about attributes, related classes, computed values
  - ◆ Doing: Calculating, coordinating, creating, controlling
- Responsibilities come from the use cases: If “the system does X”, then what class is responsible for carrying out X?

4

## GRASP Patterns

---

### GRASP

- General Responsibility Assignment Software Patterns.
- These are well-known best principles for assigning responsibilities.
- Nine core principles that object-oriented designers apply when assigning responsibilities to classes and designing message interactions.
  - ◆ We will look at 5 of these 9 principles
- Can be applied during the creation of sequence diagrams, or even during implementation.
- After or in tandem with developing the class diagrams.

5

## Patterns

---

- Named description of a problem/solution pair that can be applied in new contexts, with advice on how to apply it in novel situations, and discussion of its trade-offs.
- Notable benefits of patterns:
  - ◆ Simplifying: provides a named, generally understood building block
    - Facilitates communication
    - Aids thinking about the design
  - ◆ Accelerates learning to not have to develop concepts from scratch

6

## Pattern: **Information Expert**

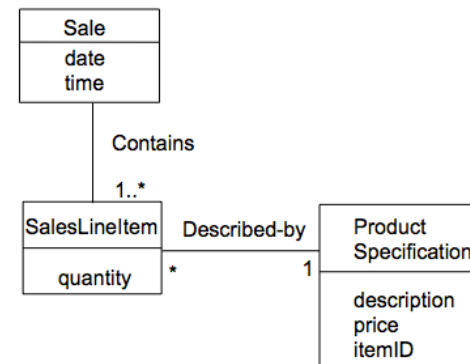
---

- Problem: What is most basic, general principle of responsibility assignment?
- Solution: Assign a responsibility to the object that has the information necessary to fulfill it.
  - ◆ “That which has the information, does the work.”
- In a “Point of Sale” (think: cash register) application, who should be responsible for knowing the grand total of a sale?
- By Information Expert we should look for that class that has the information needed to determine the total.

7

## POS class diagram

---

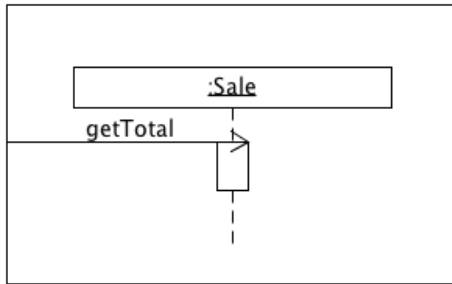


- It is necessary to know about **all** the SalesLineItem instances of a sale and the sum of the subtotals.
- A Sale instance contains these, i.e. it is an information expert for this responsibility.

8

## POS Information Expert

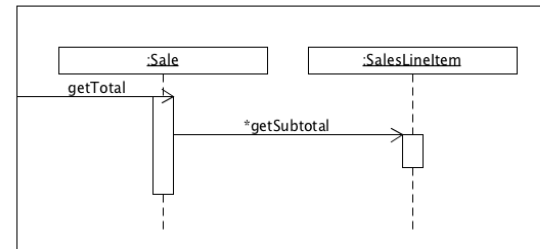
- So getTotal will be a method of the Sale class.



9

## POS Information Expert

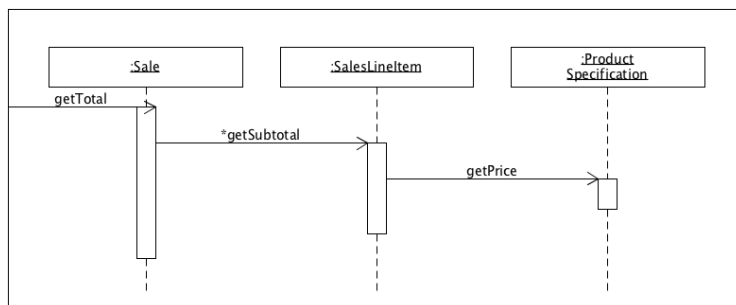
- What information is needed to determine the line item subtotal?
  - quantity and price.
- SalesLineItem should determine the subtotal.
- This means that Sale needs to send getTotal() messages to each of the SalesLineItems and sum the results.



10

## POS Information Expert

- To fulfill the responsibility of knowing and answering its subtotal, a SalesLineItem needs to know the product price.
- The ProductSpecification is the information expert on answering its price.



11

## POS Information Expert

Class	Responsibility
Sale	Knows Sale total
SalesLineItem	Knows line item total
ProductSpecification	Knows product price

- To fulfill the responsibility of knowing and answering the sale's total, three responsibilities were assigned to three design classes
- The fulfillment of a responsibility often requires information that is spread across different classes of objects. This implies that there are many "partial experts" who will collaborate in the task.

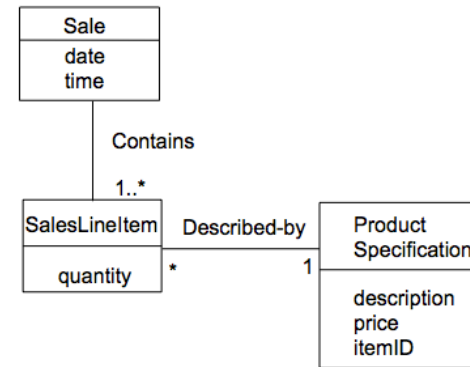
12

## Pattern: **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 or more of the following is true:
  - ◆B aggregates A objects.
  - ◆B records instances of A objects.
  - ◆B has the initializing data that will be passed to A when it is created (thus B is an Expert with respect to creating A).
- The more that are true, the better.

13

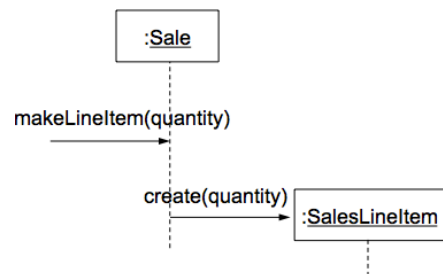
## POS class diagram



- In the POS application, who should be responsible for creating a SalesLineItem instance?
- Since a Sale contains many SalesLineItem objects, the Creator pattern suggests that Sale is a good candidate.

14

## POS Creator



- This assignment of responsibilities requires that a makeLineItem method be defined in Sale.

15

## Pattern: **Low Coupling**

- **Coupling** (in a class diagram) is a measure of how strongly one class is connected to, has knowledge of, or relies on other classes.
- A class with high coupling depends on many other classes (libraries, tools).
- Problems because of a design with high coupling:
  - ◆Changes in related classes force local changes.
  - ◆Harder to understand in isolation; need to understand other classes.
  - ◆Harder to reuse because it requires additional presence of other classes.
- Problem: How to support low dependency, low change impact and increased reuse?
- Solution: Assign a responsibility so that coupling remains low.

16

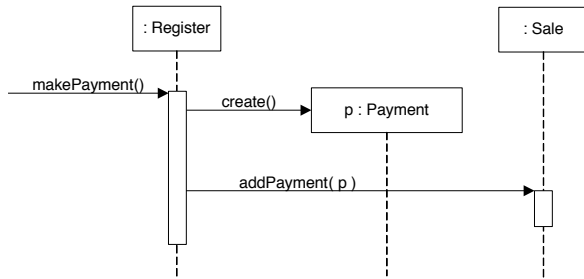
## POS: Low Coupling

- Which class should be responsible for creating a Payment and associating it with a sale?

◆ Since Register records a payment (in real life), it could be Register, by the Creator pattern

◆ Register could then send an addPayment message to Sale, passing along the new Payment as a parameter.

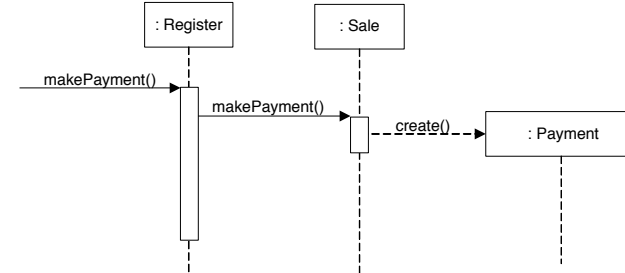
◆ This assignment of responsibilities couples the Register class to knowledge of the Payment class.



17

## POS: Low Coupling

- An alternative solution is to have the Sale object create the Payment and associate it with the Sale.
- No coupling between Register and Payment.



18

## Pattern: High Cohesion

- **Cohesion** (in a class diagram) is a measure of how strongly related and focused the responsibilities of a class are.
- A class with low cohesion does many unrelated activities or does too much work.
- Problems because of a design with low cohesion:
  - ◆ Hard to understand.
  - ◆ Hard to reuse.
  - ◆ Hard to maintain.
  - ◆ Delicate, affected by change.
- Problem: How to keep complexity manageable?
- Solution: Assign a responsibility so that cohesion remains high.

19

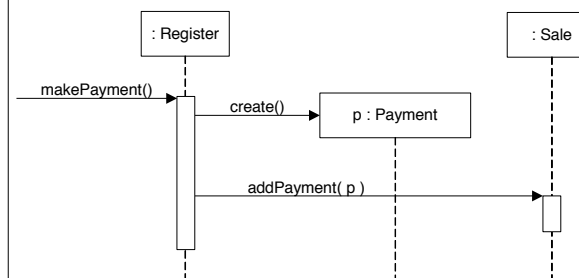
## POS High Cohesion

- Let's compare the same two examples as before with respect to cohesion:

◆ Since Register records a payment (in real life), it could be Register, by the Creator pattern

◆ Register could then send an addPayment message to Sale, passing along the new Payment as a parameter.

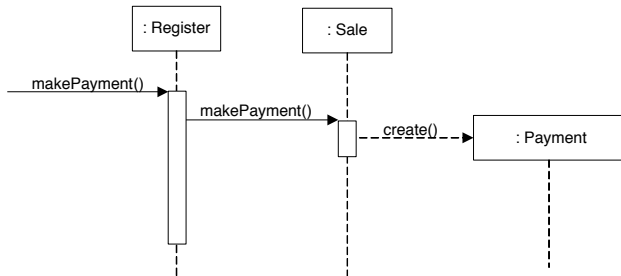
◆ Register may become bloated if it is assigned more and more system operations.



20

## POS: High Cohesion

- An alternative design delegates the Payment creation responsibility to the Sale, which supports higher cohesion in the Register.
- No class has too much work (good delegation).
- This design supports high cohesion and low coupling.



21

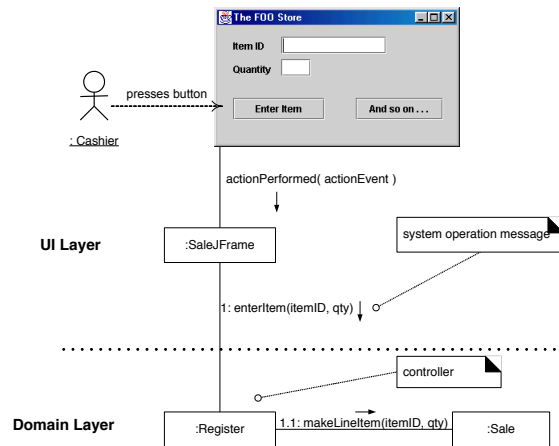
## Pattern: Controller

- What class should handle system event messages (such as input from the user/user interface)?
- Solution: Choose a class whose name/job suggests:
  - ◆The overall “system,” device, or subsystem
  - ◆OR, represents the use case scenario or session
- Recall: MVC architectural pattern: the Controller component

22

## POS: Controller

- In this example, the Register object (a controller) handles the input event.

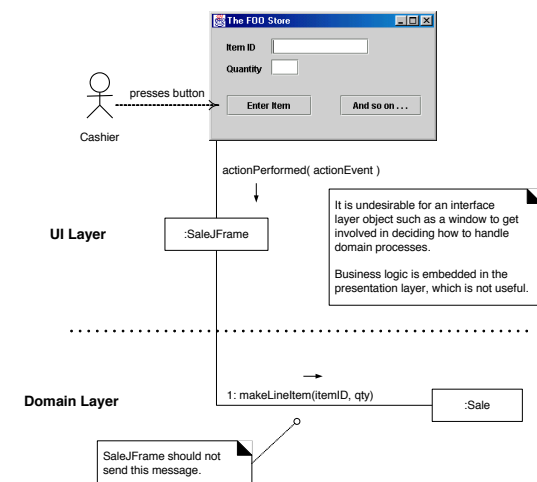


23

## POS: Controller

- In this example, SaleJFrame, a UI object handles the input event

Don't want the UI objects tightly coupled with the internal objects (Sale)



24

## Summary of Introduction to GRASP

- 5 principles for deciding how to assign responsibility (behavior) to classes:
  - ◆Information Expert
  - ◆Creator
  - ◆Low Coupling
  - ◆High Cohesion
  - ◆Controller
- These decisions can be made
  - ◆When designing the CRC cards
  - ◆When designing the sequence diagrams from the use cases.

25

## Example: Object-Oriented Analysis & GRASP

- This example is based on the Inventory system described in Assignment 2.
- I will treat “Process Sale” as a use case for the Fulfillment Specialist actor. (I’m not going to address any other use cases).

26

## Use Case: Process Sale

1. The Fulfillment Specialist enters the following values: the sku of the sold item, the quantity that were sold, and the cost to ship all of the items.
2. The System finds the Product with the given sku in the Inventory.
3. The System decreases the quantity of the Product by the given quantity that were sold.
4. The System computes the total price, shipping credit, commission, and profit, and outputs these values to the FulfillmentSpecialist.

**Variation #1.** No product with given sku

- 1.1. Start after Step 1.
- 1.2. System cannot find Product, System outputs an error message and aborts.

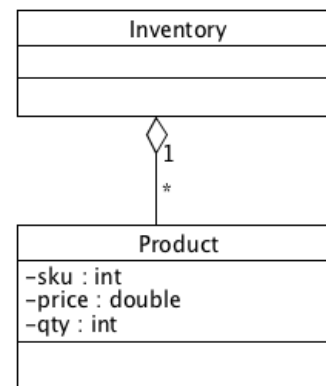
**Variation #2.** Not enough quantity for the sale

- 2.1. Start after Step 2.
- 2.2. If the quantity of the Product in the inventory is not greater than the quantity sold, System outputs an error message and aborts.

27

## Class diagram with attributes, associations

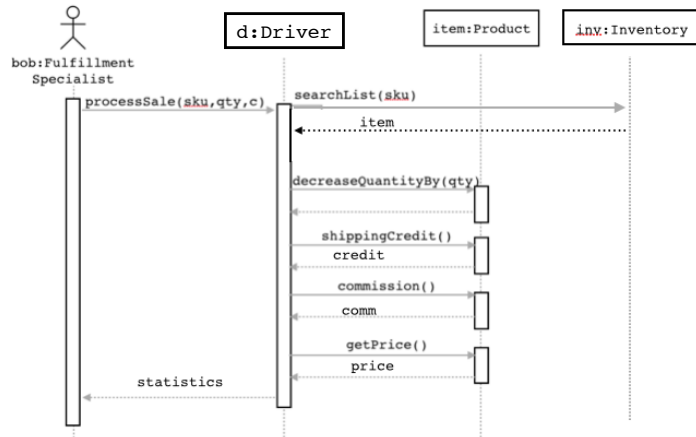
- The Product price is needed to compute the desired statistics
- The Product qty will be updated by the use case.



28

## Sequence diagram for process sale, v.1

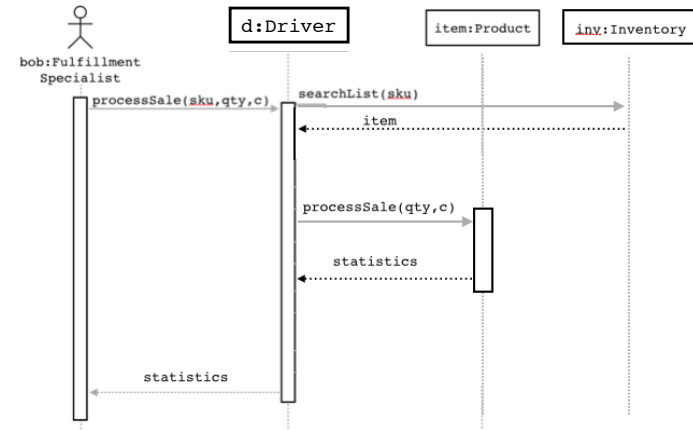
- Forwards engineering (this is not like the code I wrote).
- Driver does all the work (low coherence, bad)



29

## Sequence diagram for process sale, v.2

- Now the Driver dispatches the work to Inventory and Product.
- The Product has all the information to compute the statistics, so now it calculates the statistics (by Information Expert)

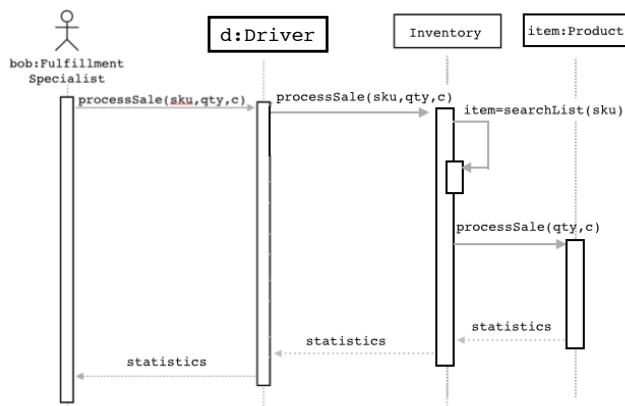


But what about coupling?

30

## Sequence diagram for process sale, v.3

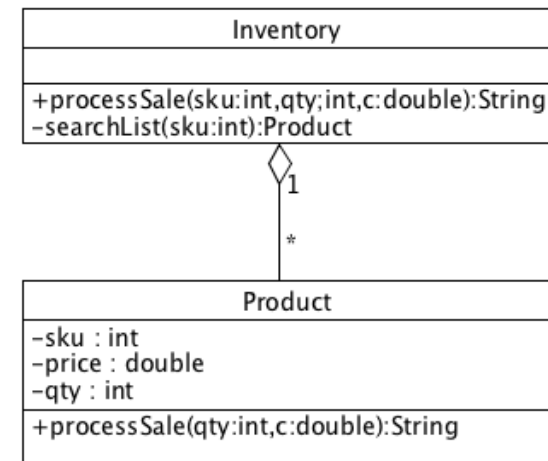
- Now the Driver dispatches the work to Inventory, who delegates to the Product.
- Low coupling, high cohesion, and information expert is applied



31

## Add operations to class diagram

- searchList can be private now (and returns Product)



32



## Explain GRASPatterns used

---

- `Inventory.processSale(sku,qty,c)`: By high cohesion, Inventory should be responsible to process the sale (so the Driver doesn't have to do all of it).
- `Product.processSale(qty,c)`: By information expert, Product has the information needed to compute the statistics (and update the quantity) so it will do that work.
- `Inventory.processSale(sku,qty,c)`: By low coupling, Inventory should find the product AND send the `processSale` message to it (so the Driver doesn't have to talk to the Product).