

Announcements

Today: Applying **GRASP Patterns** to the **Design Problem** (Chapter 17)

Use-Case Realizations

- Reference: Chapter 17
- Our goal: create **use-case realizations**
 - A use-case realization is a **design** that meets the **requirements** of a **use-case scenario**
- Recall **use-case scenario**
 - A (generic) path through the use case
 - POS system: credit card and cash are different scenarios
 - 10 items vs. 15 items are not different scenarios
- Note that cash vs. credit card scenarios have design **differences**
- Questions:
 - Can a use-case scenario be designed more than 1 way?
 - Can a design for a full use-case consist of more than one use-case realization?
 - Is a use-case realization a part of the overall system design?

Use-Case Realizations and GRASP Patterns

- GRASP Patterns are principles of design
- Use-case realizations are pieces of the overall system design
- Therefore...
 - **...to make good use-case realizations, use GRASP patterns**

Use-Case Realizations and Interaction Diagrams

- **Use-case realizations** are pieces of the overall system **design**
- Interaction diagrams...
 - ...have 2 varieties
 - Sequence diagrams and collaboration diagrams
 - ...they are a way to describe designs
 - So,
 - **...to make use-case realizations, use (GRASP patterns and) interaction diagrams**

Some Artifacts Related to Design

(See Figure 17.20)

Another Design Artifact

- Recall class diagrams can be used for both
 - Domain models
 - Design models
 - See Figure 17.5
- Is every class in the design model also in the domain model?
 - 1) No, domain can have entities outside system
 - 2) No, design can have entities outside reality
 - (Like what?)

How to Design

- Start with an idea of what classes are useful
 - (Via domain and design models)
- Look at various use-case scenarios
- For each, make interaction (sequence or collaboration) diagrams
- These diagrams assign responsibilities to the classes
- To do this well, use GRASP patterns
- The GRASP patterns are principles for good assignment of responsibilities
- Let's look at the POS system as an example...

Designing the POS System

- Let's start with creating a new sale
- Recall main success scenario (p. 50):
 - 1. Customer arrives at checkout with items to purchase
 - 2. Cashier starts a new sale
 - 3. Cashier enters an item identifier
 - 4. System records line item and displays item's description, cost, and current total
 - 5... (more stuff)
- As the book does (section 17.4)...
 - let's build a design just for the first few steps

Creating a New Sale

- Creator pattern says, give class A responsibility for creating a new object
 - if A: aggregates, contains, records instances of, closely uses, or has the initializing data...
 - ...for the new object
- So, what class should create a new sale? (See e.g. Fig. 10.1)

Receiving the makeNewSale() Event Message

More on making a new sale:

- Apply the **controller pattern** to determine a class suitable for receiving a makeNewSale() message
- Controller pattern says:
 - Use the overall system class (e.g. containing main()), or
 - Use the class with primary responsibility for initiating the scenario's activities
- Consider some potential classes for this part of the design (Figure 10.1)
- What would be a good candidate?
 - (Actually, this is a bit on the subtle side! See Fig. 17.7. Think about what the creator pattern suggested)

Creating a New Sale - Conclusion

- The Register class would be a good choice for the following responsibilities:
 - Creating a new sale object
 - Receiving a makeNewSale() message

Entering Items in a Sale

- ...this happens after a new sale has been started
- Let's do this one from a contract instead of directly from the use case (p. 255 – see transparency)
- Build an interaction diagram satisfying the postconditions, using patterns as guides
 - Use objects of class Sale, Register, SalesLineItem, & ProductCatalog
 - We'll do it together in the following slides...
 - ...then we'll try it alone

Entering Items in a Sale II

- Using the controller pattern, what class should handle the scanItem() message?

Entering Items in a Sale III

- Using the controller pattern, what class should handle the enterItem() message?
 - Recall controller is the class that gets the message initiating some sequence of activity
 - Book suggests register class
 - No explanation is given
 - Would Sale class be a good choice also?
 - Why or why not?

Entering Items in a Sale IV

- Using the creator pattern, what class should create each new SalesLineItem?
- Recall **creator pattern** says, give a class A responsibility for creating a new object
 - if A: aggregates, contains, records instances of, closely uses, or has the initializing data...
 - ...for the new object
- So, what class should create a new SalesLineItem? (See e.g. Fig. 10.1)

Entering Items in a Sale V

- Must associate a ProductSpecification with each SalesLineItem
 - (See contract)
- Here, let's apply the **information expert pattern** (p. 257)
- Input to contract is an ItemID
- Information expert pattern asks,
 - what class should be responsible for having the information about product spec, given an itemID?
- See domain model in Fig. 10.11 for hints
- Hence, a getProductSpec() method should be present in what class definition?

Entering Items in a Sale VI

- ...
- See domain model in Fig. 10.11 for hints
- getProductSpec() should be in _____
- What class should **request** the product specs (i.e. what class issues getProductSpec() call?)

Entering Items in a Sale VII

- Which is better, Register or SalesLineItem?



Entering Items in a Sale IIX

- Book tosses out a new principle to use here, **visibility**
 - ...to be explained in a later chapter, but what is your best guess? (see p. 258)
- But my design differed from the book's, slightly...



Exercising Our Neurons...

- See contract, page 255
- Build a design to meet this contract...



The endSale() Operation

- Controller pattern: who should contain this activity-initiating method?
- Information Expert pattern: "who should be responsible for setting the isComplete attribute of the Sale object to *true*?" (Hint – see the contract, next)



endSale() Contract

- Precondition: sale is under way
- Postconditions: Sale.isComplete has the value *true*



Potential Answers

- Controller pattern: who should contain this activity-initiating method?
 - Register (no reason given on p. 260)
 - Any other answers?
- Information Expert pattern: "who should be responsible for setting the isComplete attribute of the Sale object to *true*?"
 - Sale, because it has the information needed to store that information (it has the isComplete attribute - see the contract)