

Today

- Begin Elaboration Iteration #3 section of text
- Reference: chapters 25-26, actually through 34
- Chapter 25 gives more information on use cases
- Chapter 26 discusses inheritance in OO
- Let's start with use cases...

Use Cases: Repeating Chunks

- Different use cases/use case scenarios
 - ...can have the same sequence of steps
- In source code also
 - ... a sequence of statements can reoccur
 - What is the proper way to handle this?

Repeating Chunks II

- As with code, with use cases we want...
 - just one copy of the repeating stuff...
 - but somehow use it many times
- Coding techniques that can meet this need:
 - Procedure/function/method calls
 - Macros
 - Loops
- Use case writing techniques that can do it:

Repeating Chunks III

- Use case writing techniques that can do it:
 - "include" "extend" "generalizes"
- "Include" is best
 - "...the most common and important" – p. 386
 - "...always use the *include* relationship.... People...and their readers have less confusion than...people who mix *include* with *extend* and *generalizes*."
 - You'd think the text wouldn't even discuss those
 - (Why say don't use it, then say how to use it)
 - The reality: a few paragraphs on "extend" plus a boxed comment not to use it
 - One paragraph on "generalizes" that says don't use it

Repeating Chunks: Include

- POS example:
 - Credit card authorization steps are needed
 - For the main success scenario
 - For a rental (not purchase) scenario
 - For installment purchase plan scenario
 - For lay-away plan scenario
 - We wish to write those steps down just once
 - ...and then refer to them in multiple places

More on Include

- How to use it is best shown by example
 - See 2 illustrations, pp. 386-387
- There are other reasons to use it
 - (Same as for source code)
- Reason 1:
 - Have one copy of stuff used in several places
- Reason 2:
 - Break up a long use case (into comprehensible pieces)
- Reason 3:
 - Describing asynchronous event handling

Reasons for Include

- Reason 1:
 - Have one copy of stuff used in several places
- Reason 2:
 - Break up a long use case (into comprehensible pieces)
 - Makes the use cases more understandable
 - ...just like it makes code more understandable
- Reason 3:
 - Describe asynchronous event handling elsewhere
 - See illustration, p. 388
 - Recall Use Case notation for asynchrony
 - "Include" is especially suited for asynchronous events because asynchronous stuff is more weakly coupled

More on Use Cases

- Some technical vocabulary/concepts (ref. 25.2)
- **Concrete** use case (e.g. **process sale**)
 - An full sequence of actions that are actually done
 - **Abstract** use case (e.g. **credit payment**)
 - A part of other use cases (not a use case itself)
 - **Base** use case (e.g. which of the above?)
 - A use case that includes, is extended, or specialized by another
 - **Addition** use case (e.g. which of the above?)
 - A use case that is included, extends, or specializes another
 - Base and addition use cases are base or addition...
 - ...with respect to other use cases

Include in Use Case Diagrams

- See Figure 25.1

Another Topic: Generalization

Reference: chapter 26

- Generalization and specialization are a big part of OO
- They are done through inheritance
- They need to be designed properly (of course)
- How would you match (generalization, specialization) with (subclass, superclass)?

Generalization: Getting Started

- To find classes, we earlier
 - Identified nouns in use cases
 - See another example, p. 395
- Once identified, we can arrange
 - conceptual (i.e. potential) classes, into
 - generalization/specialization heirarchies
- Let's try it with the illustration...

Generalization: Getting Started

- To find classes, we earlier
 - Identified nouns in use cases
 - See another example, p. 395
- Once identified, we can arrange
 - conceptual (i.e. potential) classes, into
 - generalization/specialization heirarchies
- Let's try it with the illustration...
- UML: point to superclass with solid arrow, empty triangle for arrowhead

Generalization and Superclasses

- A superclass is more general than a subclass
 - It is more *encompassing*
- Figure 26.4 shows an example, but let's extend our previous work on the board

Generalization, Superclasses, and the "Is-a" Relationship

- "Is-a" refers to the concept
 - "this **is a** that"
 - A mouse is-a computer peripheral
 - A rat is-a rodent
 - A check is-a payment
- If x is-a member of a subclass, then x is-a member of its superclass

"Is-a" II

- If x is-a member of a subclass, then x is-a member of its superclass

Generalization, Superclasses, and the "Is-a" Relationship

- If x is-a member of a subclass, then x is-a member of its superclass
- If x is a check, then x is a payment
- Etc. (see our own example that we've been developing)

Generalization, Superclasses, and Inheritance

- "100% of the conceptual superclass's definition should [apply] to the subclass." (p. 399)
- This includes
 - Attributes
 - Associations
- See Figure 26.5

When to Have Subclasses of a Superclass

- See Table 26.2
- Consider this example (ref. p. 401)
- Would it be useful in the POS system to -
 - have the following subclasses of **Customer**:
 - **CustomerMale**
 - **CustomerFemale**
- Why or why not?