

## Announcements

Reference for today: Chapters 19, 20

Design Class Diagrams

Implementations

## DCDs

Recall domain models and their class diagrams  
They are closely related to Design Class Diagrams (DCDs)

See Figure 17.5

- Design class diagrams have similarities, but are different too
- See 19.2 (modified), then 19.2 (unmodified)

## Things that can be in a Design Class Diagram

- Classes
- Methods
- Attributes
- Navigability:
  - "attribute visibility ...exists when B is an attribute of A" – p. 281
  - See Figure 19.8 (why 1-to-1?)
- Associations: iff navigability
- Interfaces: screens are done with classes
  - Can interface classes be in a DCD? Domain model?
- Dependency: said to exist when any type of visibility exists
  - (Except attribute visibility) -- see Figure 19.11

## Design Class Diagrams in Context

- The point:
  - While with domain Class Diagrams we deal with **conceptual classes**...
  - in Design Class Diagrams we deal with **software classes**
- Both are drawn using UML Class Diagrams
- Design Class Diagrams have more details than domain Class Diagrams
- So naturally, Design Class Diagrams use more parts of the UML class diagram sublanguage
- *Let's build a DCD on the board, step by step*

## Creating Design Class Diagrams

- We will account for:
  - Classes, Attributes, Methods, Associations, Interfaces, Navigability, Dependencies...
- Start by identifying the classes
  - Simply use the classes found in the interaction diagrams
    - Recall:
      - interaction diagrams include sequence diagrams & collaboration diagrams
    - See Figures 17.17, 17.20, etc.
      - Just make a list! Let's do it...
  - Then, draw a very basic Class Diagram (rectangles, no arcs)
    - Add in attribute names that might be found in the domain model (Figure 12.7)
      - Let's try it! Also consider adding attribute types (e.g., quantity : Int)

## Creating DCDs II

- Method names are missing – let's add
- Get most of them from the interaction diagrams
  - Which of the figures we've seen are interaction diagrams?
- Let's scan the interaction diagrams for methods
- Some might not be in the interaction diagrams
  - Accessor and mutator methods are examples
  - Java developers often use them
    - Declare the member variables private
    - Work with them only via accessor & mutator methods
    - Why?

## Creating DCDs III

- There is no need for a method called "create()" in Java or C++
  - Objects in these languages are created with the `new` operator and a `constructor` call
  - "...it is common to omit [these] from a DCD" – p. 290
    - Are there any? If so, let's delete

## Creating DCDs IV

- Methods for getting or changing attribute values...
  - A convention (e.g. common in Java) is:
    - Declare all attributes (member variables) `private`
    - Define for each
      - A method to get the value
      - A method to set the value
  - Example: attribute is `trafficLightColor`
    - Define **accessing** methods
      - `getTrafficLightColor()`, `setTrafficLightColor()`
- No need to have them in DCDs since convention already accounts for them
  - Let's check if there are any, and delete if so

## Creating DCDs V

- **Collections** are an interesting programming construct
- See Figure 19.6
- In that figure the `find()` message makes no sense as a member of any particular `ProductSpec` object
  - (it is implemented in the collection class, not shown in the DCD)
  - What might the collection class be like?
  - ...so get rid of any such methods

## Creating DCDs VI

- Add associations ("navigability")
  - Put an arrow from class A to class B if:
    - A sends a message to B, or
    - A creates or contains an object of class B
      - (i.e. B is an attribute of A, i.e., B is a member of A)
- Associations are found in the interaction diagrams
  - Let's add some to the DCD now...
  - Label the arrows with the kind of association
    - ...usually not the same as the interaction diagram message

## Creating DCDs VII

- Dependency relationships
  - Solid arrows show attribute visibility
    - (Recall previous chapter on visibility types)
  - Use dotted arrows to show other kinds of visibility
    - Parameter (passed in as a method argument)
    - Local (e.g. returned by a method)
    - Global (global variables)
  - See Figure 19.11
    - Do any of these apply to ours?

## Implementation Notes

- Reference: chapter 20
- Going from design to implementation should be fairly mechanical, usually
- Implementation can uncover design defects
  - Fix the design and then implement
  - The new design is an input to the next iteration (see Figure 20.1)



## Implementation II

- Let's see how to go about implementing representative bits of our DCD...
- Also, note that exception handling is more of a focus in implementation...
- 20.8 recommends implementing...
  - ...least coupled classes first, and
  - ...most coupled last