COLLABORATIONS AND HIERARCHIES


Outline

- Collaborations
  - Identifying collaborations
  - Recording collaborations
- Hierarchies
  - Hierarchy graphs
  - Venn diagrams
Motivation for Collaborations

- Two ways to perform responsibilities
  - ?
  - ?

- Collaboration is
  - Request from one object to another in order to fulfill a responsibility.

Motivation (Cont.)

- Why identify collaborations?
  - Collaborations represents flow of control and information through the system,
  - May identify misplaced responsibilities, and
  - May identify missing responsibilities.
  - In sum, shows dynamics of the system.
Finding Collaborations

- Look at each responsibility of each class:
  - Is the class capable of fulfilling this responsibility by itself?
  - If not, where can it get what it needs?

- Look at each class:
  - What other classes need what this class does or knows?

Finding More Collaborations

- Examine relationships between classes, especially:
  - The “is-part-of” relationship, and
  - The “has-knowledge-of” relationship.
"Is-part-of" Relationship

- May imply responsibilities for maintaining information.
- May fulfill responsibilities by delegating responsibilities.
- Two of relationships:
  - Composite
  - Aggregate

"Has-knowledge-of" Relationship

- May know other classes that are not in part-of relationships (i.e., associations in UML).
- May imply responsibilities to know information, and thus collaborations.

Q: Can you think of an example?
Recording Collaborations

- Write the name of the server (or helper) class on the CRC card of the client.
- Write the name directly to the right of the responsibility the collaboration fulfills.

Collaboration Graph

- Graphical representation of collaborations
- Arrow from client to a “contract” of the server, denoted by a semicircle
  - Contract: group of responsibilities (more on this later in the semester)
Other Tools - UML

- UML interaction diagrams
  - Sequence diagram
  - Communication diagram (a.k.a. collaboration diagram)

Sequence Diagram

- Customer
  - Place an order
- Order
  - Process
- Payment
  - Validate
  - If (payment ok)
  - Deliver
  - If (not in stock)
  - Back order
- Product
  - Get address
- Supplier
  - Mail to address

- Object
- Control
- Message
- Lifetime
Sequence Diagram

Collaboration

Communication Diagram

object
link
message
Communication Diagram

Collaboration

1.1: ok := validate()
1.2 [ok]: deliver(c)
1.2.1 [not in stock]: back order(p)
1.2.2: get address()

Outline

- Collaborations
- Hierarchies
  - Hierarchy graph
  - Venn diagram
1. Exploratory phase
   - Identify a preliminary list of classes, responsibilities and collaborations.

2. Analysis phase
   - Obtain a more global understanding of the design, e.g., by using tools such as:
     - Hierarchy graphs,
     - Venn diagrams, and
     - Contracts.

Hierarchy Graph

- A graphical representation of inheritance between related classes.
- A hierarchy graph is a general tree.
- The nodes are classes and the arcs represent inheritance.
- Ancestor nodes are superclasses of descendent nodes, which are subclasses.
- Notation of [Wirfs-Brock-etal90]
- Leaf nodes are often concrete classes.
- Non-leaf nodes are often abstract classes.
Abstract Classes

- Classes that cannot be instantiated.
- Designed only to be inherited from, thus allowing reuse and avoiding duplication.
- Used to factor common behaviors among classes.

Finding Abstract Classes

- At the exploratory stage, all identified classes are probably concrete and few may have been identified as abstract.

- But, do you have all your abstract classes? That is, have you used the power of abstraction (factoring behavior)?
Another Example of Hierarchy Graph

Venn Diagram

- Another tool to understand inheritance relationships.
- If a Venn diagram views a class as a set of responsibilities, then
  - what does an intersection mean?
    - Common responsibilities
  - What might an intersection lead to?
    - Abstract classes
Example

```
set of responsibilities
PartTime  Employee  FullTime

set of objects
Employee
  PartTime  FullTime
```

Exercise

```
Draw a Venn diagram for the following hierarchy graph.

Ordered Collection
  Indexable Collection

Array
  Magnitude

Matrix  String  Date
```
In Sum, Hierarchies …

- Are for reviewing the inheritance relationships, and
- Use hierarchy graphs and Venn diagrams as a notation and an analysis tool
Building Good Hierarchies

- Model "is-a" relationships.
- Factor common responsibilities as high as possible.
- Make sure abstract classes do not inherit from concrete classes.
- Eliminate classes that do not add functionality.

Example – Using Venn Diagram To Check Hierarchy

If B supports only a part of responsibilities defined for A, what does the Venn diagram look like? How to fix this?
Q1. How many responsibilities do you need in order to create an abstract class?

Answer: 1, but there’s a catch!!!
Suppose that you are designing a new Java compiler and you identified the following concrete classes for various kinds of declarations allowed in Java.

- Class Declaration
- Interface Declaration
- Field Declaration
- Method Declaration
- Local Variable Declaration
- Method Parameter Declaration

Organize the above classes into a class hierarchy.

- Use both a hierarchy graph and a Venn Diagram notation
- Feel free to (and you should) introduce new classes