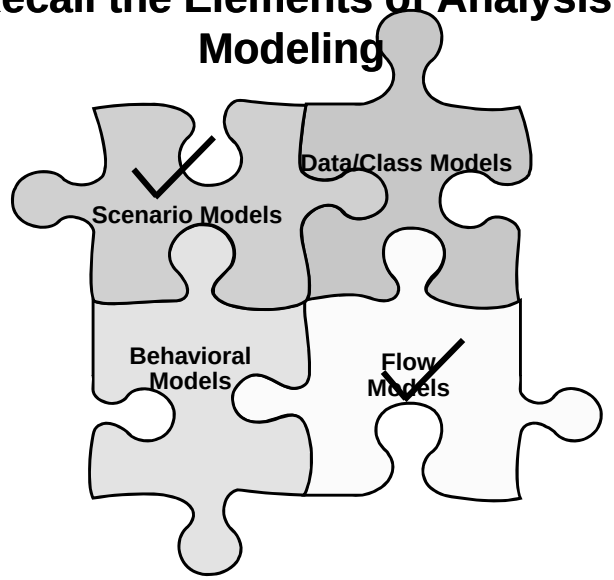
Software Engineering I CSC-382

Lecture 12

Software Engineering I CS-382

- Lecture 12
- What we will cover: (Details of Analysis Modeling)
  - Chapter 8 Sections 8.7, and 8.8 in Pressman
  - Goal is to continue to develop the methods and tools available from Object Oriented Analysis for analysis modeling

Recall the Elements of Analysis Modeling



Recall Our Class-Based Modeling Thru OOA

- Object Oriented Methods view a system as a collection of these objects that communicate to each other thru messages
  - These messages request the various other objects to perform some function or task
- Identify **classes** by examining the problem statement
  - We used our Stereotypes and then our CRC Cards for making this a little easier

Recall Our Class-Based Modeling Thru OOA II

- We then must model:
  - The **attributes** of each class
  - The **operations** that manipulate the attributes
  - Later define **inheritances** and **aggregations** of these classes.
  - Now lets define **associations** and **dependencies** of the classes based on how they collaborate to accomplish the required functionality

Relationships Among Classes

- Recall the complete object model defines the objects and how they interact with each other
  - These interactions are defined by the **Relationships** that these object have with each other.
- Relationships can of two types:
  1. **Persistent** – they exist for the duration of the application
     - E.g. a car has 4 wheels for an ABS system application
  2. **Transitory** – they exist only for a period of time during an application
     - E.g. a Student uses a course selection screen to pick a class
     - These relationships exist to allow two objects to collaborate

Relationships Among Classes II

- There are three categories of **Relationships** we need to define to complete our object model
  1. **Association**
     - A persistent relationship between objects
  2. **Aggregation** (and a related concept called Composition)
     - Another persistent relationship
  3. **Dependency**
     - A transient relationship between objects

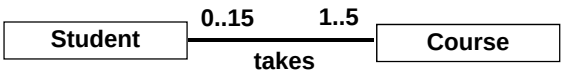
Using the CRC Card for Relationships

- CRC stands for: Class, Responsibility and Collaborators
  - **Class**: An object is a person, place, thing, event, or concept
  - **Responsibility**: Anything that a class knows or does
  - **Collaborator**: A class that another class needs to accomplish its purpose
    - In general, a collaboration implies either a request for information or a request for some action.
    - Therefore the collaborations will help us to define the **associations, aggregations, and dependencies** that exist between the various class candidates.

Associations

- They represent long-term relationships between classes
- While long term, they may not be permanent
- Associations can be refined by indicating:
  - **role** *(*how are the two related)
  - **multiplicity** *(*how many of each of the objects)
  - **directionality** (who holds a multiplicity of whom)

Associations II



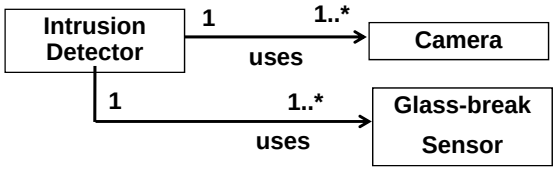
- This reads:
  - Between 0 and a maximum of 15 students may take a course and a student may take between 1 and 5 courses

Associations III

- Associations are managed in your class definitions through its **attributes** and its **methods**
  - For the student registration, methods such as add_course() and drop_course() would change the number of courses the student object has.

Associations IV

- For our SafeHome one example could be:



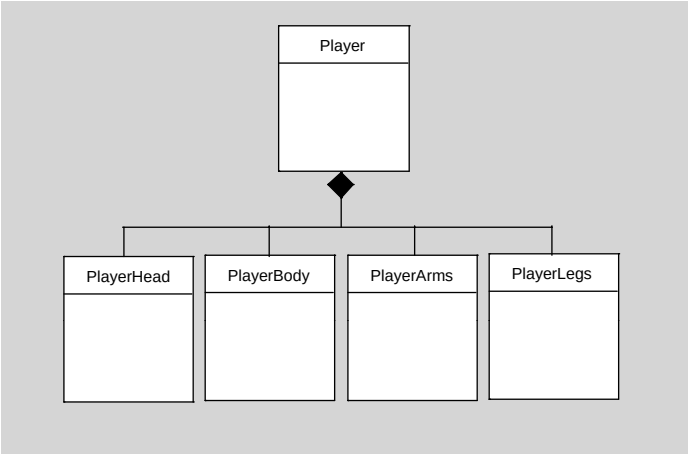
  - Through a Configuration Manager Object we could add or remove them (also the Self Tester could remove faulty sensors) from the list of available sensors suitable for intrusion detection

In-Class Association Diagrams

Aggregations and Composition

- Recall Associations are used for persistent, long term relationships between objects
  - E.g. a student takes classes, etc.
  - Recall these may not be permanent (a student can drop a course)
- **Aggregations** are a stronger form of Relationship where the relationship has a permanence to it.
  - E.g. A car has 4 wheels – this cannot change without dire results
  - Composition is even stronger, where an object can only belong to one other object
    - E.g. an engine can be related to one and only one car object

Example of an Aggregate Class

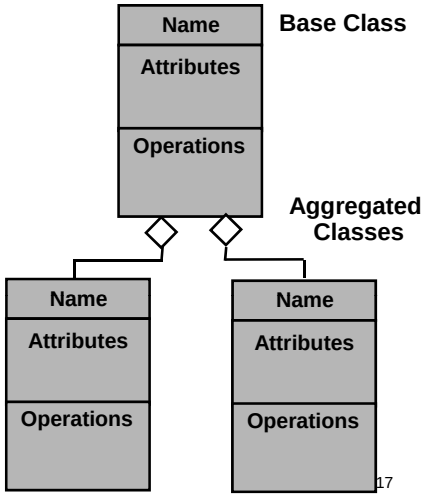


Tips for Finding Aggregations

1. Apply the rule: the object is ***a part of*** the whole
   - If this is true then the relationship is aggregation/composition
   - If this does not apply (e.g. student takes a class, but she is not part of the class) then try **Association**
2. Only the whole object should manage the component objects
3. The parts must be of interest
   - If you don't need to keep track of it then drop it
4. Don't forget to show the multiplicity and the roles

In-Class Aggregation Diagram

Can we think of any possible aggregations in our SafeHome ?

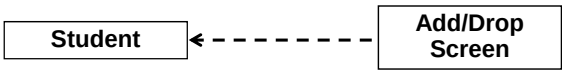


Dependencies

- Both Associations and Aggregations are persistent relationships
- We also need to define and represent the **transitory** relationships as well
  - We do this through **Dependencies**

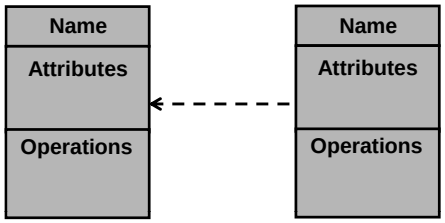
Dependencies II

- An example is the client-server relationship that exists between two classes.
  - The relationships exists only until the responsibility is satisfied and then it ends
  - Often one of the participants is a transitory object as well



In Class Dependencies Diagram

Can we think of any possible dependencies in our SafeHome ?



For Next Class

- Finish studying Chapter 8