Chapter 10

- **Requirements Modeling: Class-Based Methods**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman

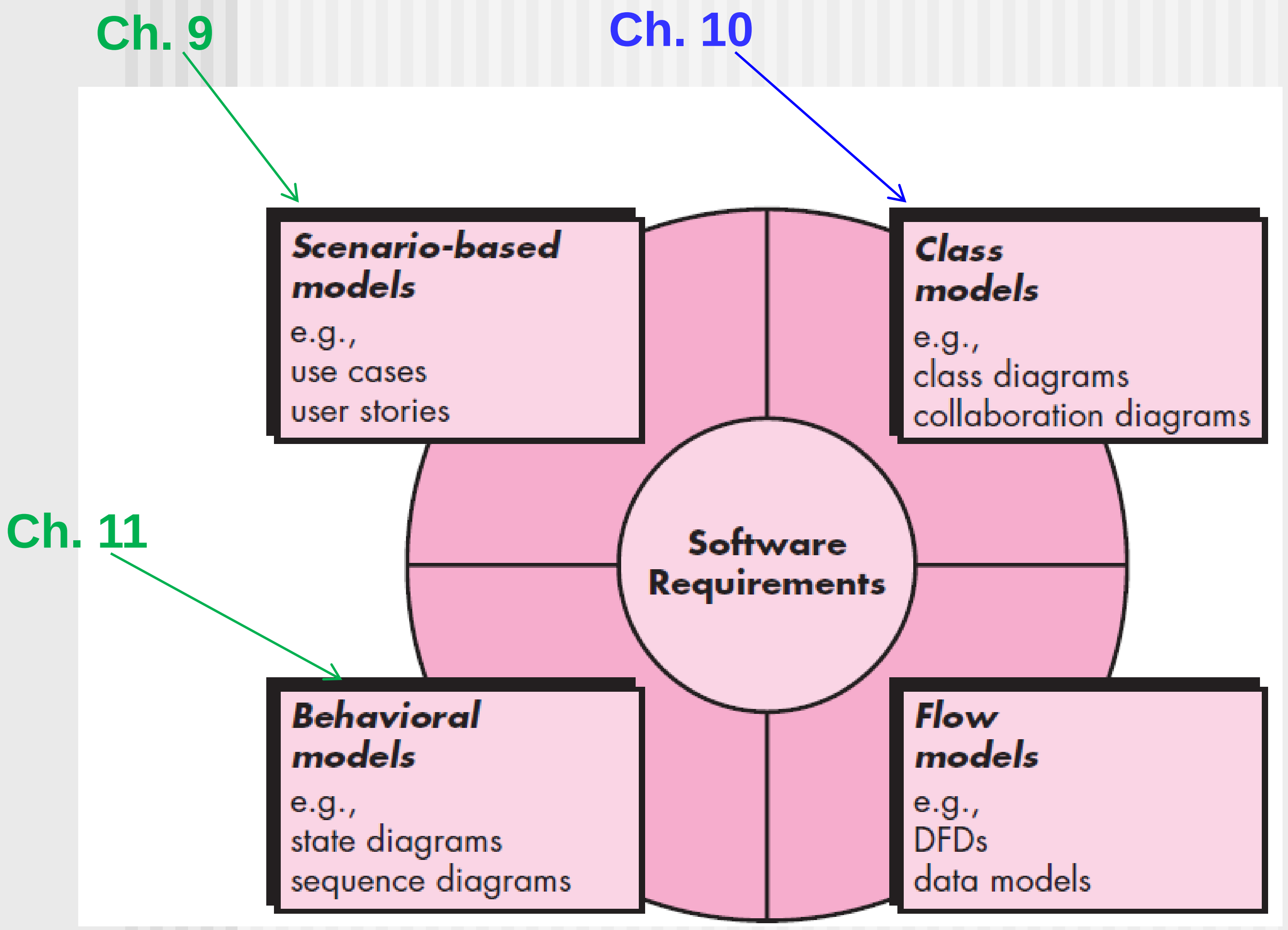
Table of Contents

Requirements Modeling Approaches

- *Structured analysis* considers:
  - Data objects: define their attributes and relationships.
  - Processes: transform data as data objects flow through the system.
- *Object-oriented analysis* focuses on
  - the definition of classes and
  - the manner in which they collaborate with one another to effect customer requirements.

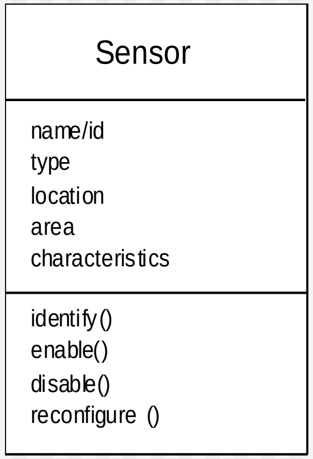
☛ Both approaches produce class models

Elements of the Analysis Model



Class-Based Modeling

- Class(-based) model represents:
  - objects that the system will manipulate
  - operations (also called methods or services) that will be applied to the objects to effect the manipulation
  - relationships (some hierarchical) between the objects
  - collaborations that occur between the classes that are defined.



10.1 Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse" [Abb83]
  - Underline each noun or noun phrase
  - Synonyms should be noted.
  - If the class (noun) is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.
- But what should we look for once all of the nouns have been isolated?

Manifestations of Analysis Classes

- External entities (e.g., other systems, devices, people) that produce or consume information
- Things (e.g, reports, displays, letters, signals) that are part of the information domain for the problem
- Occurrences or events (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation
- Roles (e.g., manager, engineer, salesperson) played by people who interact with the system
- Organizational units (e.g., division, group, team) that are relevant to an application
- Places (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function
- Structures (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects

Potential (=Candidate) Analysis Classes

- Retained information. The potential class will be useful during analysis only if information about it must be remembered so that the system can function.
- Needed services. The potential class must have a set of identifiable operations that can change the value of its attributes in some way.
- Multiple attributes. During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.
- Common attributes. A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.
- Common operations. A set of operations can be defined for the potential class and these operations apply to all instances of the class.
- Essential requirements. External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

10.2 Specifying Attributes

- *Attributes* describe a class

  Example (For professional baseball players)

  Can build two different classes:

  - **For Playing Statistics software:** name, position, batting average, fielding percentage, years played, and games played might be relevant
  - **For Pension Fund software:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.

10.3 Defining Operations

- Do a grammatical parse of a processing narrative and look at the verbs
- Operations:

  (1) manipulate data in some way (e.g., adding, deleting, reformatting, selecting)

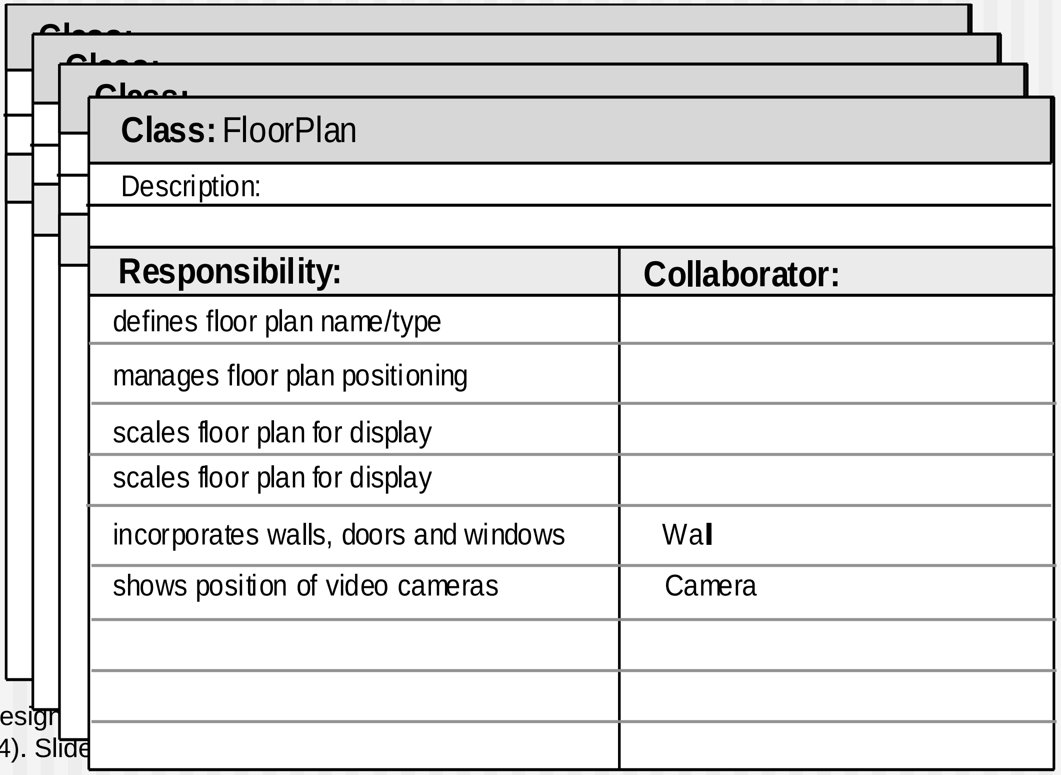
  (2) perform a computation

  (3) inquire about the state of an object, and

  (4) monitor an object for the occurrence of a controlling event.

10.4 CRC Models

- *Class-responsibility-collaborator (CRC) modeling* [Wir90]: a simple means for identifying and organizing the classes relevant to system or product requirements.



Class: FloorPlan

Description:

Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Class Types

- *Entity classes*, also called *model* or *business classes*, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).
- *Boundary classes*: interface (e.g., interactive screen or printed reports) that the user sees and interacts with.
- *Controller classes* manage a "unit of work" [UML03] such as:
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.

Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem
- Each responsibility should be stated as generally as possible
- Information and the behavior related to it should reside within the same class
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.

Collaborations

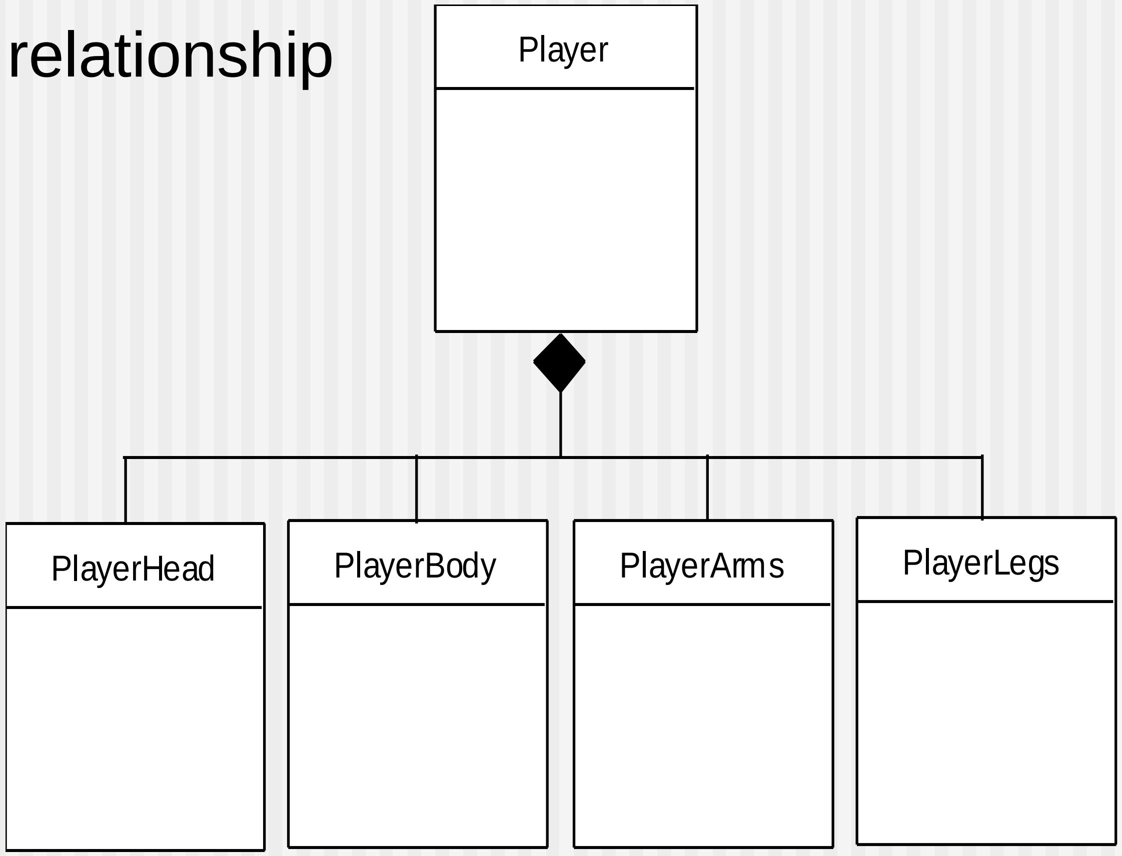
- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.

- Collaborations identify relationships between classes

- Three different generic relationships between classes [WIR90]:
  - the *is-part-of* relationship
  - the *has-knowledge-of* relationship
  - the *depends-upon* relationship

Composite Aggregate Class

is-part-of relationship



Reviewing the CRC Model

- All participants in the review are given a subset of the CRC model index cards.
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate).
- All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.
- The review leader reads the use-case deliberately.
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.
- When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card.
  - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.

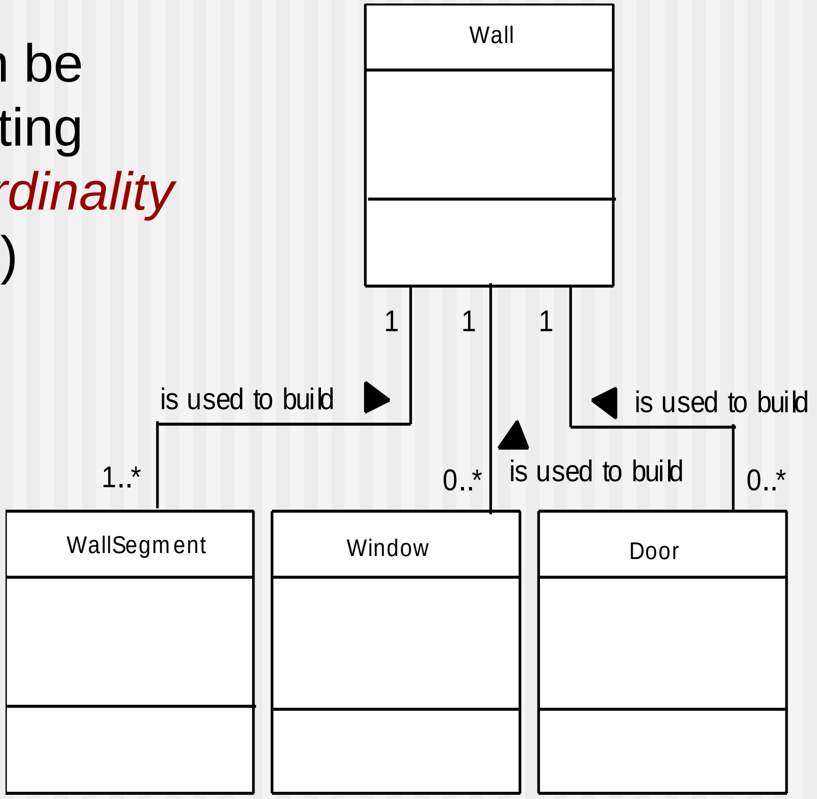
10.5 Associations and Dependencies

- Two analysis classes are often related to one another
  - In UML these relationships are called *associations*

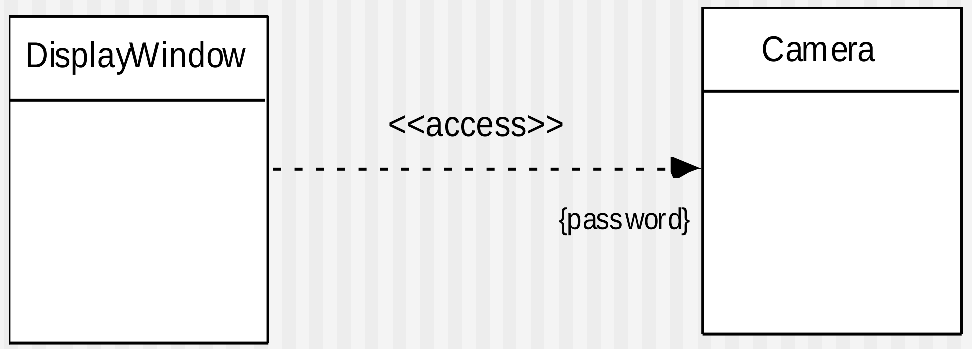
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class

Multiplicity

Associations can be refined by indicating *multiplicity* (= *cardinality* in data modeling)



Dependencies



Which one depends on which?

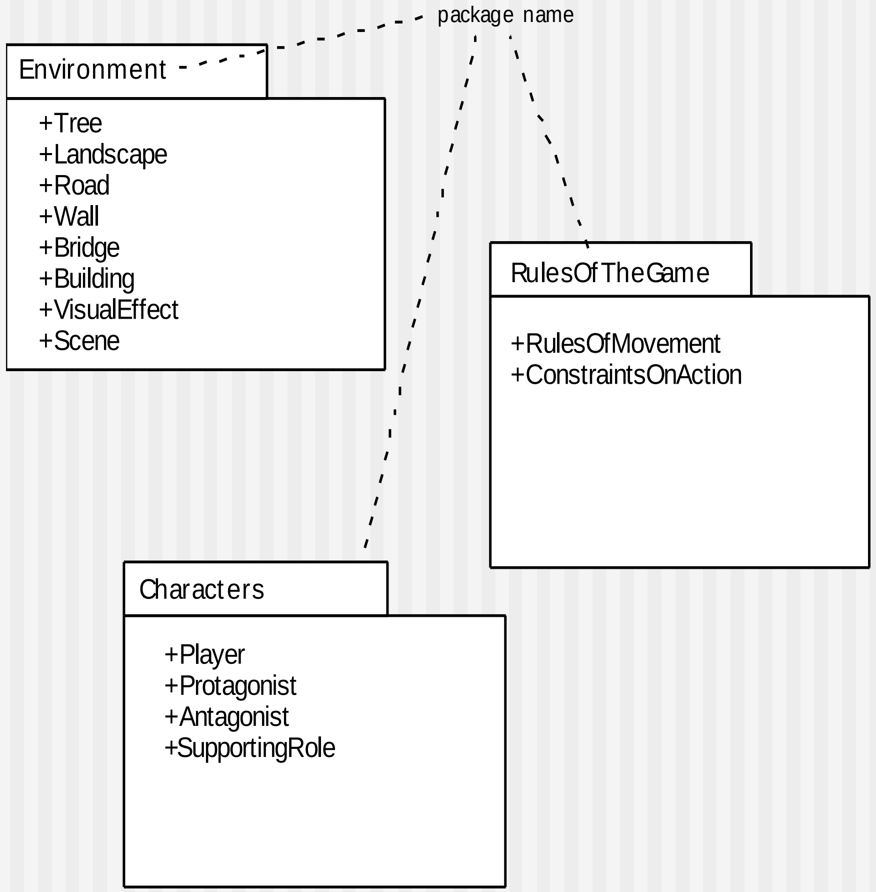
10.6 Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are grouped as packages

accessibility or visibility

- '+' preceding the analysis class name in each package: the classes have public visibility and are therefore accessible from other packages.
- '-': an element is hidden from all other packages
- '#' : an element is accessible only to packages contained within a given package.

Analysis Packages


package name
Environment
+Tree
+Landscape
+Road
+Wall
+Bridge
+Building
+VisualEffect
+Scene
RulesOfTheGame
+RulesOfMovement
+ConstraintsOnAction
Characters
+Player
+Protagonist
+Antagonist
+SupportingRole