National University
Of Computer and Emerging Sciences

Software Design and Architecture

Logical Models: Structural Models
Class Diagram

Sajid Anwer

Department of Software Engineering,
FAST-NUCES, CFD Campus

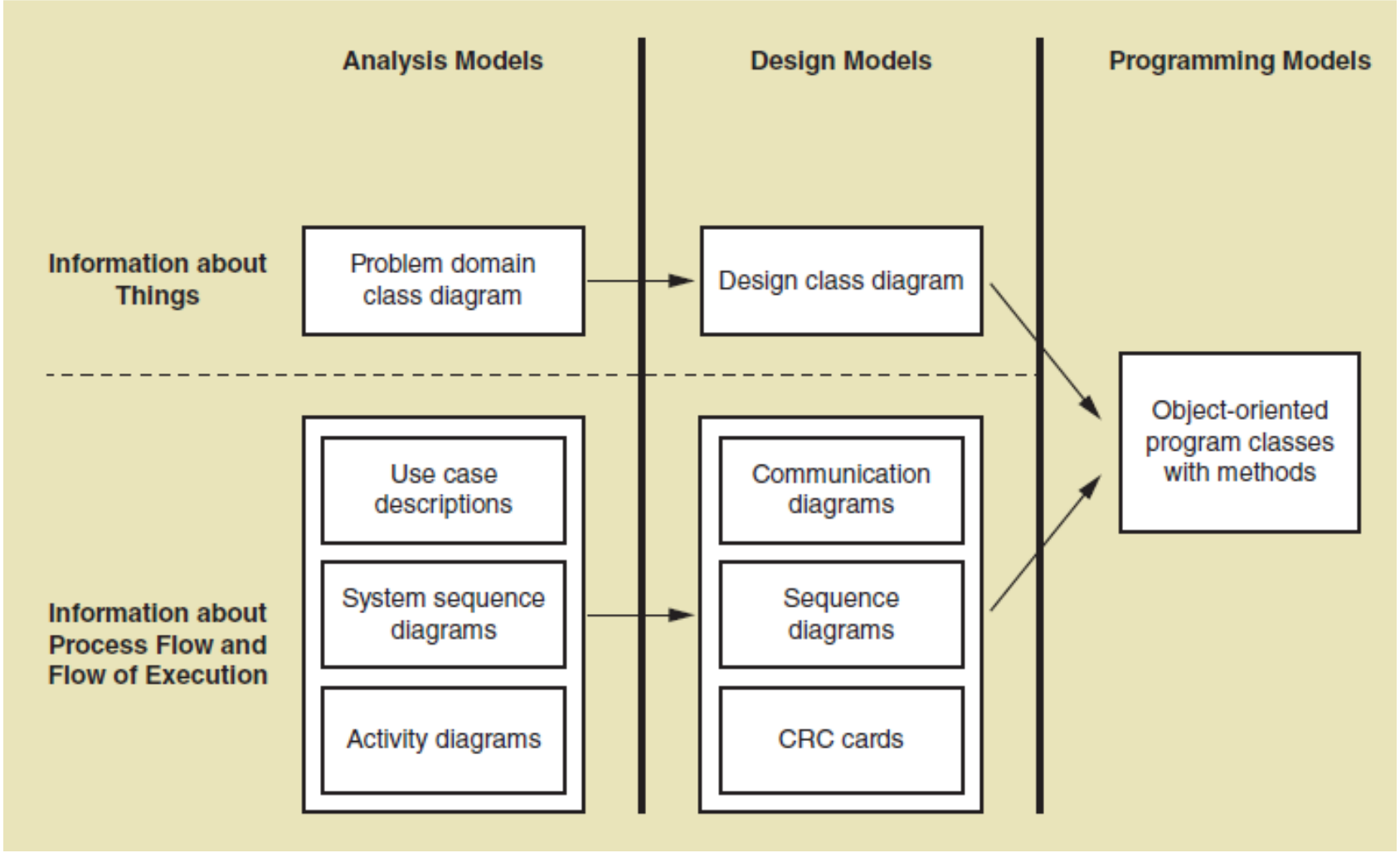
Lecture Material

- System Analysis and Design in a Changing World (Chapter 12)

Lecture Outline

- Class diagram fundamentals
- CD design guidelines
- CD constructs
- Example

Class Diagram Fundamentals



Class Diagram Fundamentals – Domain Models

- It helps to get the *rough idea* of the system structure and *potential* conceptual classes.

- illustrates meaningful *conceptual classes* in a problem domain.

- is NOT a set of diagrams describing software classes, or software objects and their responsibilities.

- It may show:
  - » concepts
  - » associations between concepts
  - » attributes of concepts

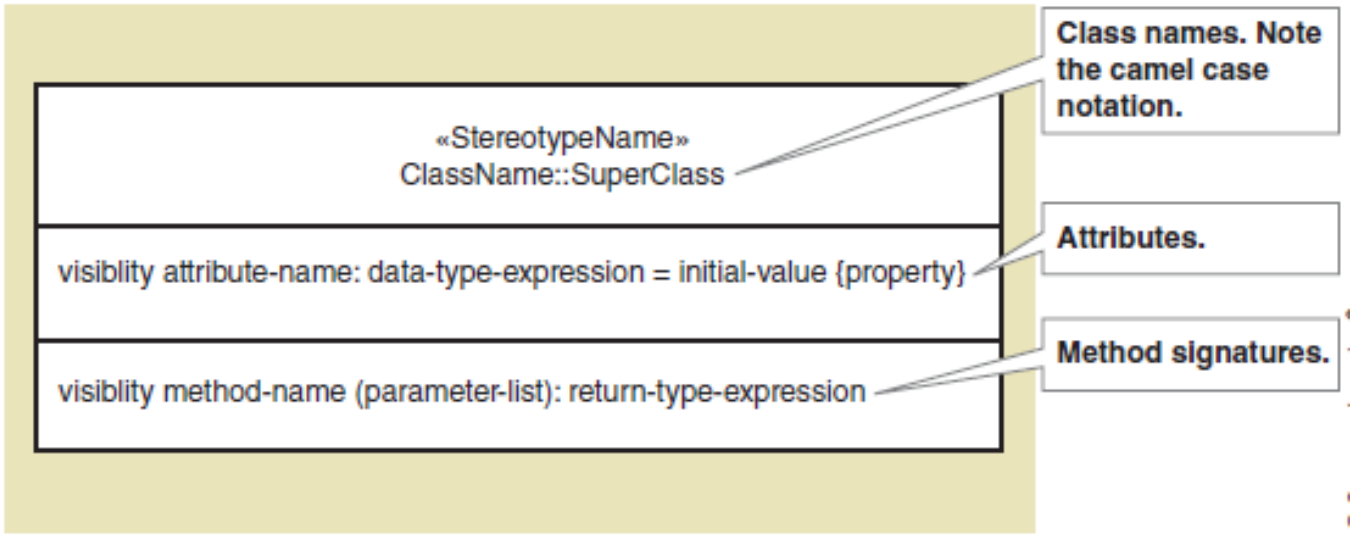
- A CD is an extended form of domain models and illustrates the specifications for software *classes* and *interfaces*

Design Class Types

- Entity classes
- Boundary classes
- Controller classes
- Data access classes

Design Class

- Typical information included:
  - Classes, associations, and attributes
  - Interfaces, with their operations and constants
  - Methods
  - Attribute type information
  - Navigability
  - Dependencies



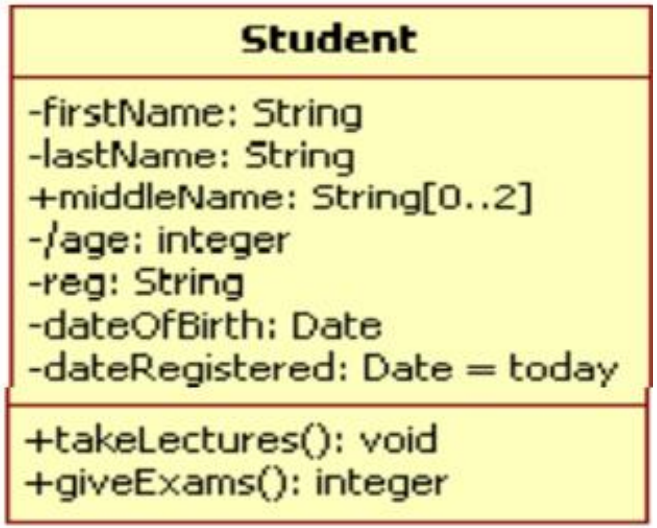
Design Class -- Attributes

- Each attribute and operation are listed *one per line* in the appropriate compartment
- Each attribute and operation name should start with a *lower case* letter
- The only exception to this is a constructor operations of a class, which will have the same name as the class itself in exactly the same uppercase and lowercase format
- Attribute and Operation names should have no spaces between multiple words in the name but should start each word with a capital letter e.g
  - » **giveQuiz instead of givequiz or**
  - » **dateRegistered instead of dateregistered**

Design Class -- Attributes

- Attributes and operations can be assigned a level of visibility on the class diagram with a visibility indicator.

- The visibility of a feature can be defined by either a keyword or a symbol
  - » There are three specific types of visibility
  - ➢ Private [represented with a symbol -]
  - ➢ Public [represented with a symbol +]
  - ➢ Protected [represented with a symbol #]
  - » There is no default value for visibility



Design Class -- Methods

- The name, parameter list and return type of an operation are collectively known as its *Signature*

- It is possible to have several operations with the *same name* and *return type* in one class provided that those operations each has a different *parameter list* to the other same name operations.

- Knowing the operation signatures provides a clear specification for the collaboration between operations.

Design Class – Methods – Design Guidelines

- Responsibilities include information that;
  - » the class maintains
  - » actions that the class carries out in support of a particular use case.
- Class responsibilities can be identified by using
  - » CRC cards
  - » Use case specifications
  - » SSD, Activity diagram, etc.

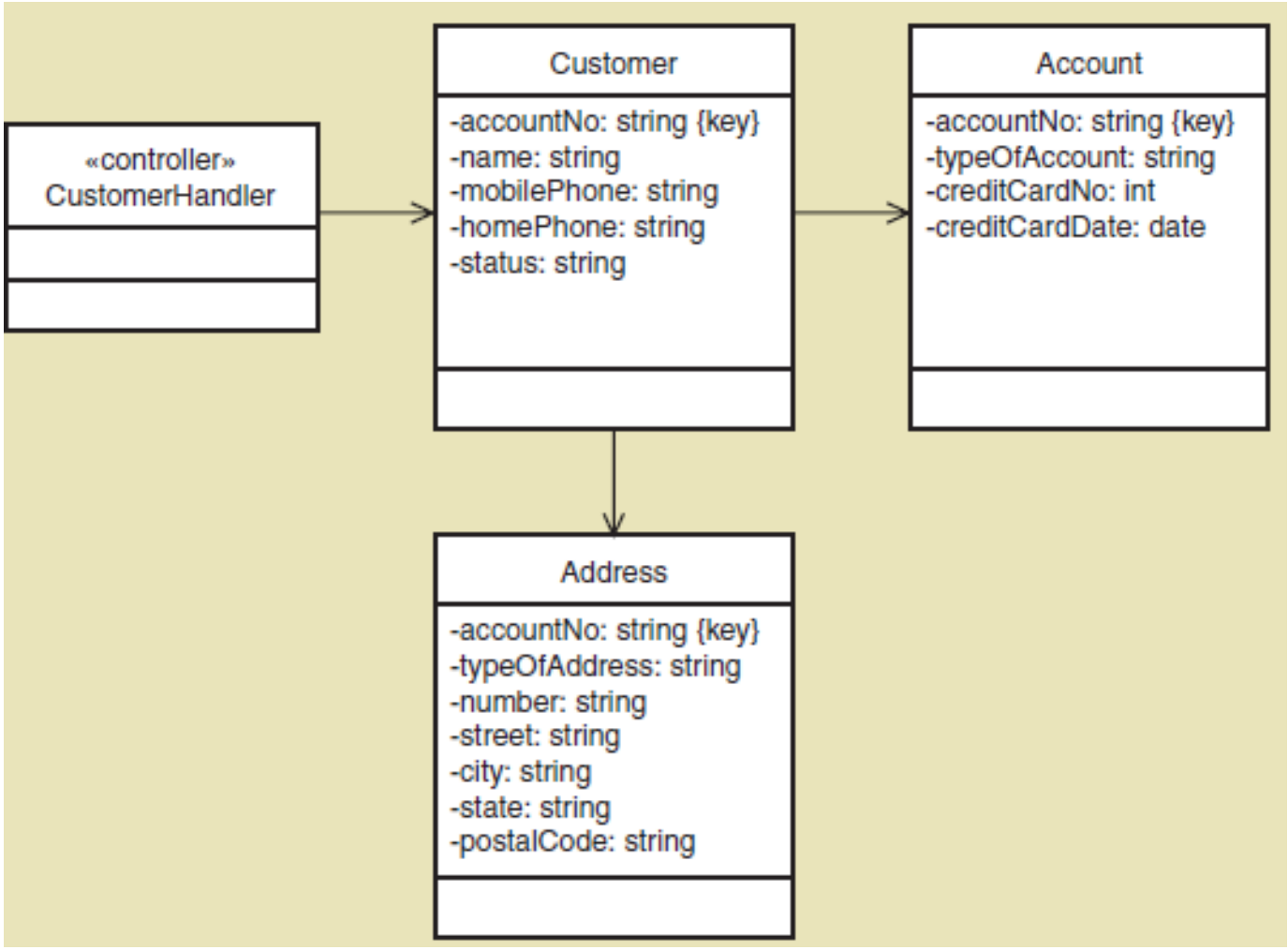
CD Fundamentals – CRC

- *CRC modeling* provides a simple means for *identifying* and organizing the classes that are relevant to system or product requirements.
- *Responsibilities* are the *attributes and operations* that are relevant for the class
- *Collaborators* are those classes that *are required* to provide a class with the information needed to *complete a responsibility*.

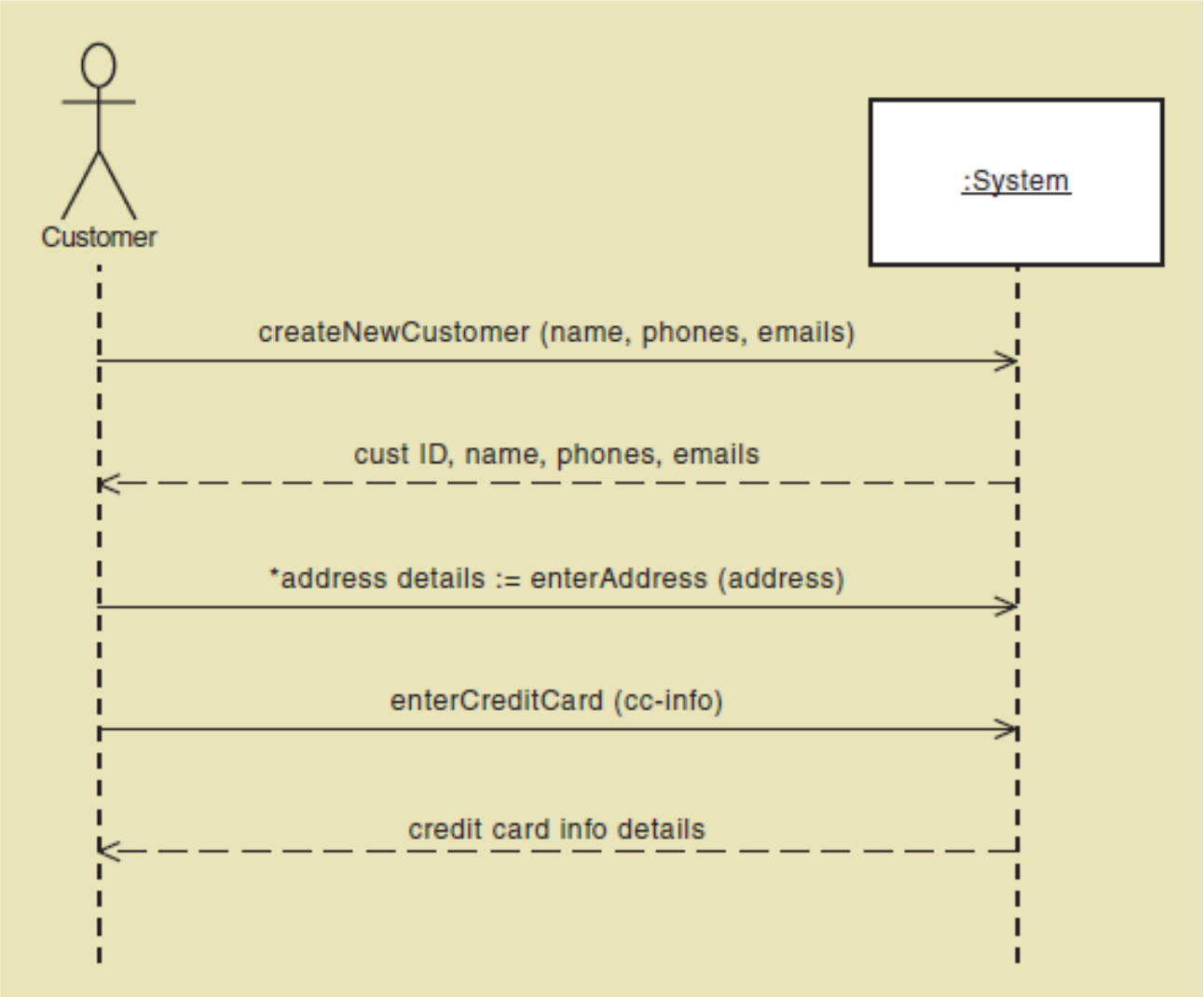
CD Fundamentals – CRC

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**

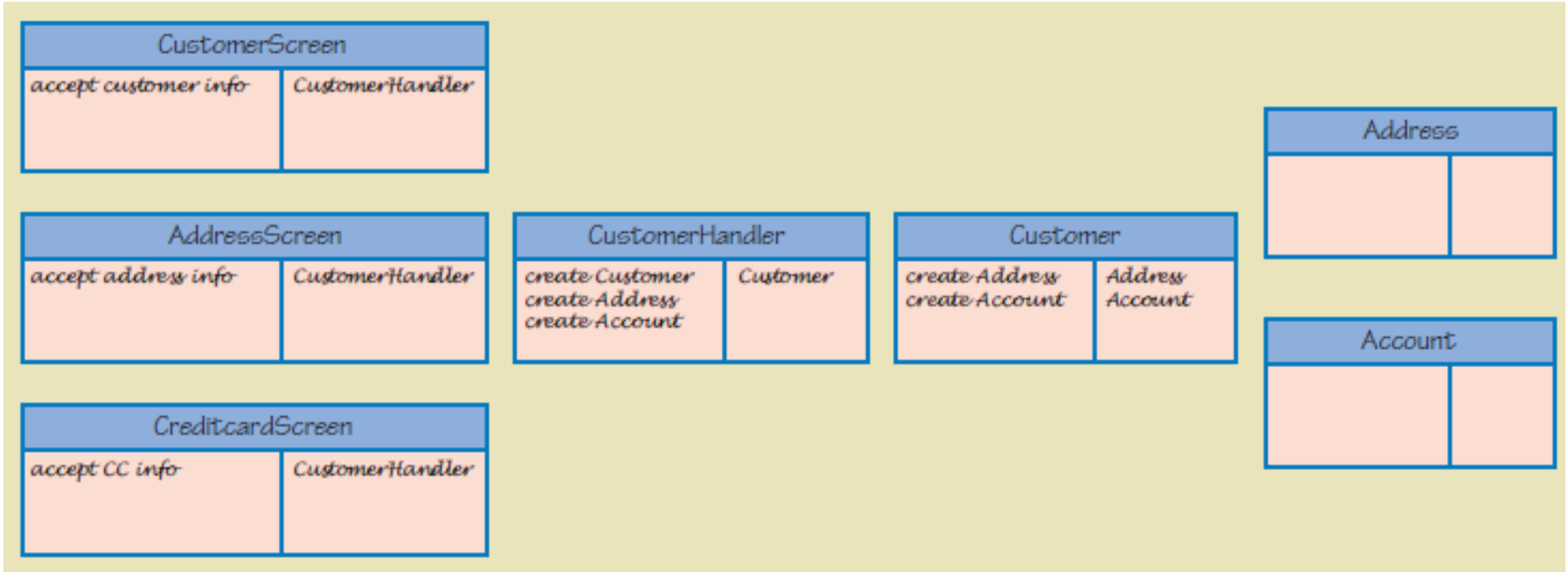
CD Fundamentals – CRC



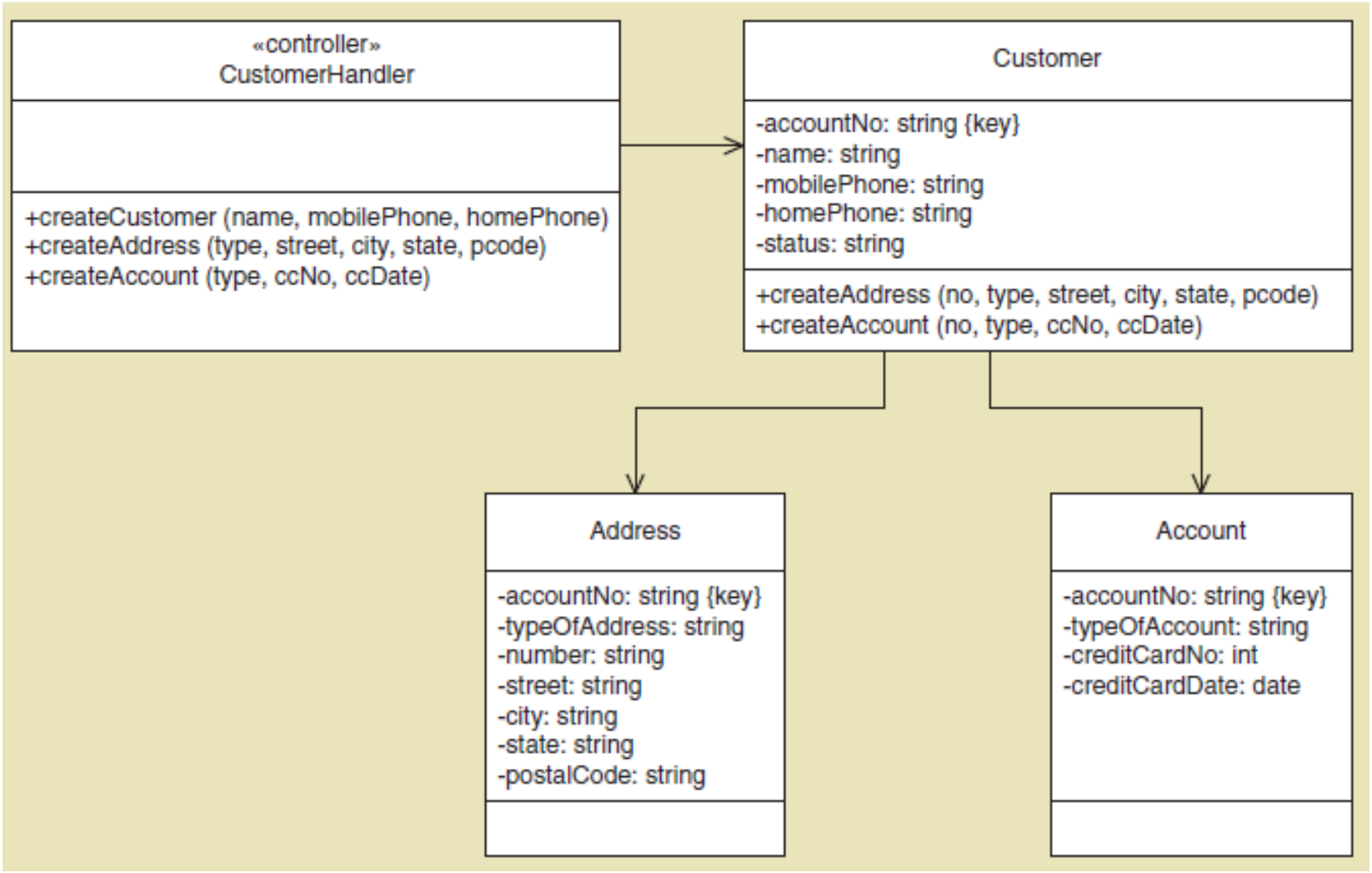
CD Fundamentals – CRC



CD Fundamentals – CRC



CD Fundamentals – CRC



Design Class – Methods – Design Issues

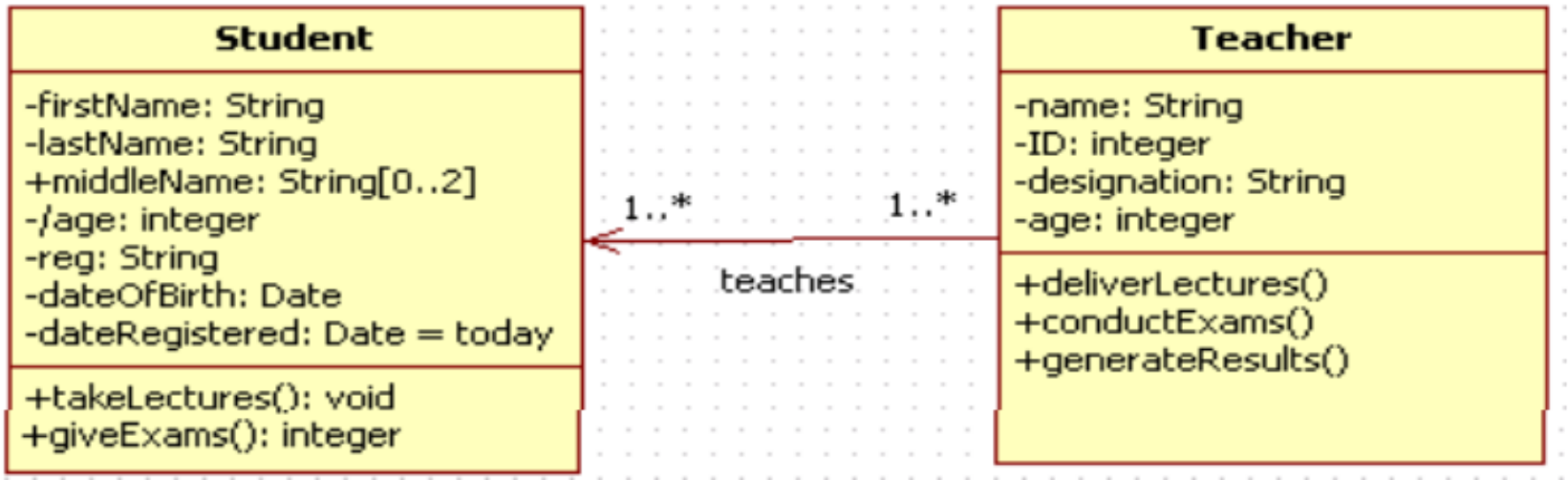
- Interpretation of the create() message.
  - » The create message is the UML language independent form to indicate *instantiation* and initialization.
    - Equivalent to calling the constructor method of a class
- Depiction of accessing methods.
  - » Accessing methods are those which *retrieve attribute values* (accessor method - get) or set attribute values (mutator method)
  - » It is common idiom to have an accessor and mutator for each attribute, and to declare all attributes private (to *enforce encapsulation*).

Adding Relations and Navigability

- When instance of one class pass messages to instances of another class, a *relation is implied* between those two classes
- The relation can be labeled with a name to indicate the nature of the relationship
- The direction or navigability is not recommended to model in class diagram, as it is purely an *implementation issues.*

Adding Relations and Navigability

- In terms of *relations multiplicity* indicates the number of object instances of the class at the far end of a relation for one instance of the class at the near end of a relationship.
  - » It is optional to add multiplicity in CD, *why?*



Relationship Types

- Dependency
- Association
  - » Simple association
  - » Is-A association (generalization)
  - » Part-Of association
    - Aggregation
    - Composition

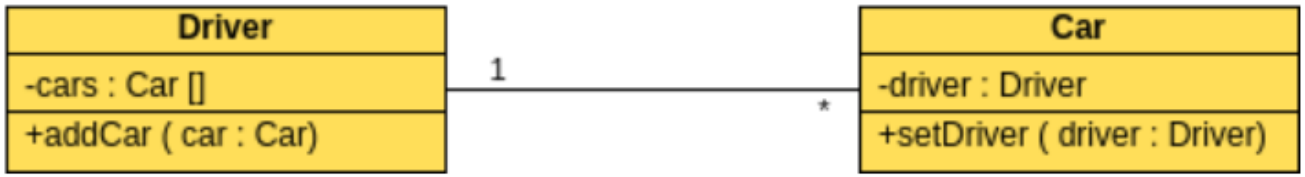
Relationship Types -- Dependency

- Dependency is the *weakest type* of relation in UML class diagrams.
- It is considered weak because the relationship between the two ends is *only temporary* or restricted to a single method or constructor.


```
class Mechanic {
  public void repair(Tool tool) {
    // tool is only used within this method
  }
}
```

Relationship Types

- Simple association
  - » A class only uses behaviors/functionalities (methods) of another class but does not change them by overriding them.
  - » A class does not inherit another class.
  - » A class does not include (own) another class as a public member.
  - » Both classes have independent lifetime where disposing of one does not automatically dispose of another.

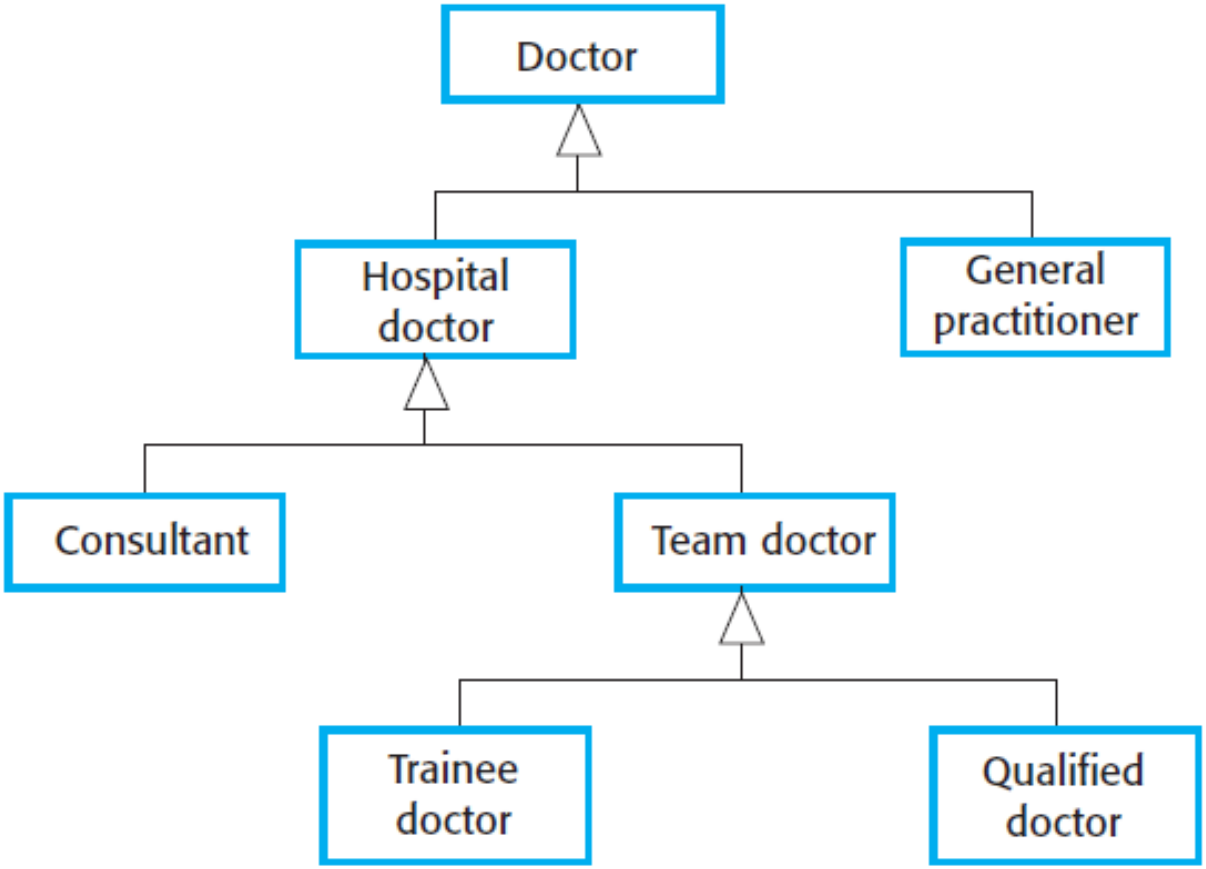


Relationship Types – Is-A Association

- Is-A Association (Generalization)
  - » Sometimes, some entities have few *commonalities* and *differences* among them, and these entities can be *generalized* based on the commonalities.
  - » In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
  - » The lower-level classes are subclasses *inherit* the attributes and operations from their superclasses. These lower-level classes then add more *specific* attributes and operations.

Relationship Types – Is-A Association

- Is-A Association (Generalization)

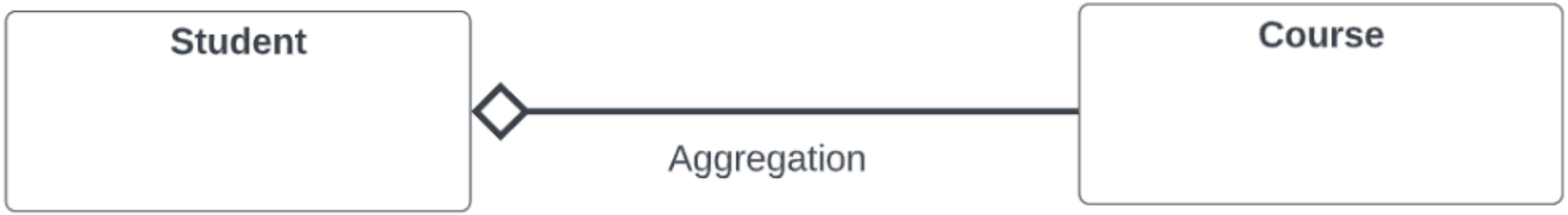


Relationship Types – Part-A Association

- Part-A Association (Aggregation)

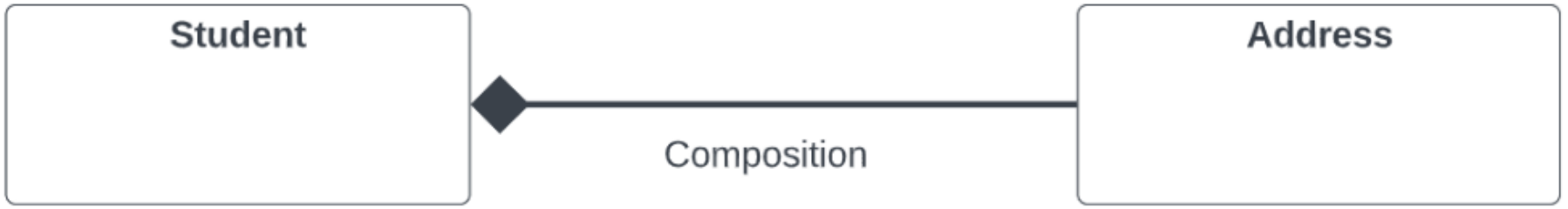
  » Aggregation is another type of composition ("has a" relation).

  » A class (parent) contains a *reference* to another class (child) where both classes can exist *independently*.

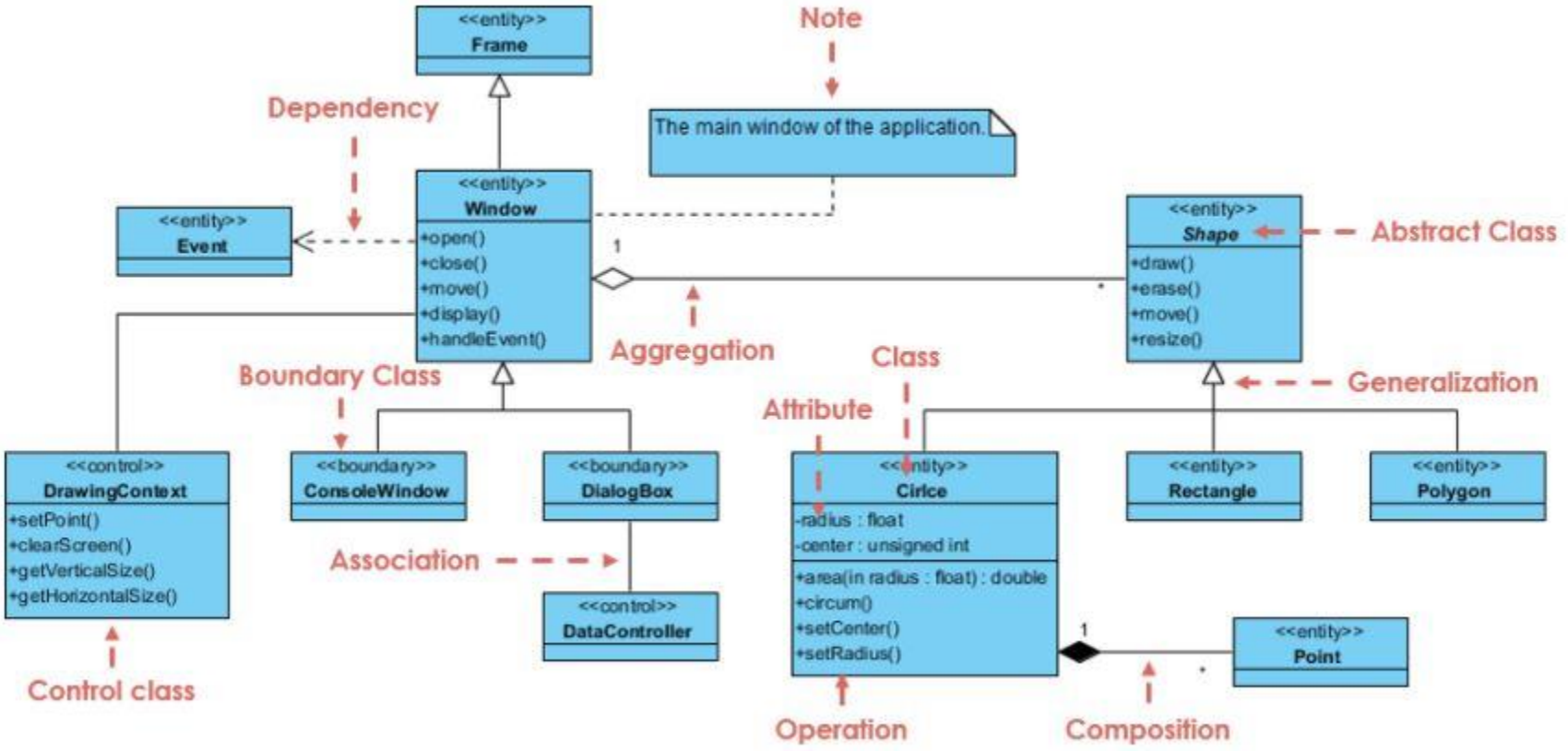


Relationship Types – Part-A Association

- Part-A Association (Composition (has-A)
  - » A class (parent) contains a reference to another class (child).
  - » The child class *doesn't exist* without the parent class.
  - » Deleting the parent class will also delete the child class.



Design Class



Interfaces

- Interfaces allow you to specify what methods a class should implement.
- Interfaces cannot have properties, while abstract classes can
- All interface methods must be public, while abstract class methods is public or protected.
- Classes can implement an interface while inheriting from another class at the same time.

Interfaces

- Improves Maintainability, information hiding
- Reduce coupling
- Implement multiple inheritance
- Most designers use a dependency arrow and the «interface» stereotype