CSE870 Software Engineering
Object-oriented Modeling:
Class Diagrams

Acknowledgements: M. Langford, M. Heimdahl

1

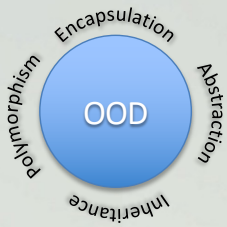
Structural Models

- ***Structural models focus on system organization***
  - What are the ***system components***, what are their ***characteristics***?
  - How are the ***relationships*** between components?
- ***UML Structural Models***
  - ***Package Diagrams*** – visualizes ***how various UML models are grouped*** together
  - ***Component Diagrams*** – visualizes ***groups of classes*** into system components
  - ***Class Diagrams*** – visualizes a system's ***object-oriented classes***
  - ***Deployment Diagrams*** – visualizes the ***physical hardware*** for a system

Class diagrams are the ***most common*** UML diagram you will encounter.

3

Object-Oriented Design



- ***Four principles of Object-Oriented Design***
  - ***Encapsulation***
    - Ability to ***group data*** into a single entity with public/protected/private access
    - Ex: Obstacle avoidance(bundle together relevant elements: detection, classification, etc.)
  - ***Abstraction***
    - Ability to ***hide data*** from external entities (public/protected/private access)
    - Ex: Obstacle detection (abstract details, such as technique used, identification, etc.)
  - ***Inheritance***
    - Ability to ***derive and add properties*** to existing entities (promotes reuse)
  - ***Polymorphism***
    - Ability to define ***alternative behavior*** to an entity based context

Not all programming languages are object-oriented.

4

Programming Paradigms

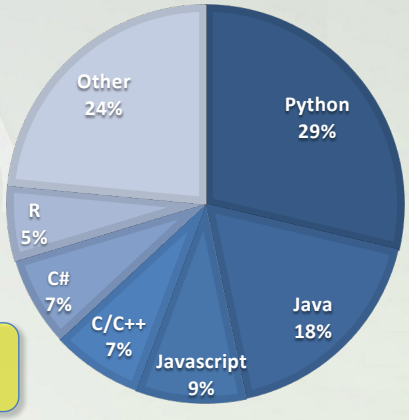
- ***Procedural Languages (Imperative)***
  - Program is a ***series of statements*** or procedure calls
  - Focus: computations/functions
  - ***Examples***: Fortran, COBOL, BASIC, C
- ***Functional Languages (Declarative)***
  - Program is a ***composition of chained functions***
  - Focus: functions; (computation, parameters, procedures)
  - ***Examples***: Lisp, Scheme, Haskell, Erlang
- ***Logic-based Languages (Logical, inference)***
  - Program stmts express facts and rules about problems
  - Focus: logic
  - ***Examples***: Prolog, Parlog, Datalog
- ***Object-Oriented Languages***
  - Program is a ***collection of interacting objects (application)***
  - Focus: data
  - ***Examples:*** Java, C#, C++, Python

6

Popularity of Programming Languages

- ***PYPL: PopularitY of Programming Languages*** [PYPL 2024]
  - Based on ***frequency of Google*** searches for tutorials
  - https://pypl.github.io/
- ***2024 Results***
  1. ***Python*** – Object-oriented
  2. ***Java*** – Object-oriented
  3. ***Javascript*** – Can be object-oriented
  4. ***C/C++*** – Can be object-oriented (C++)
  5. ***C#*** – Object-oriented
  6. ***R*** – Object-oriented properties

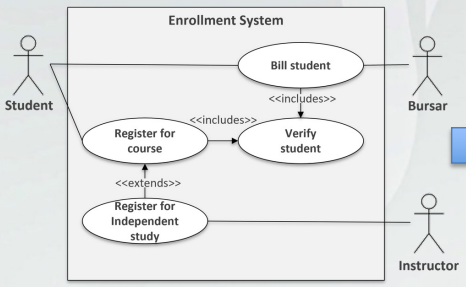
Object-oriented programming dominates.



8

Class Diagrams for Domain Models

- ***Class Diagrams can be used early in the software process***
  - ***No code has been implemented***
  - ***Domain models*** describe ***objects in the real world***
  - Parse out ***each noun from use cases***

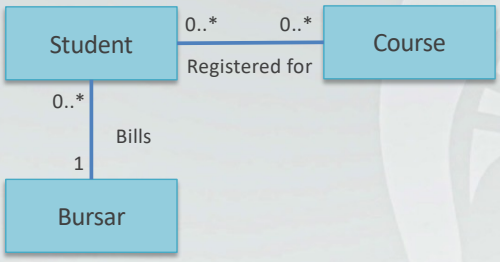


Potential Class
Student
Bursar
Instructor
Course
Independent Study

These nouns describe our problem space.

10

Elements of a Class Diagram



- ***Classes***
  - Drawn as a ***box***
  - Given a ***unique name***
- ***Associations***
  - Represents a ***relationship*** between classes
  - Drawn as a ***line***
  - ***Labeled*** to describe type
  - Annotated to indicate ***cardinality***

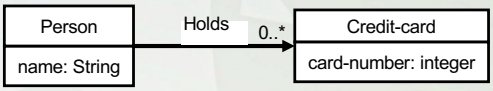
11

Multiplicity

One person holds one credit card

- One object can be related to many objects through the same association

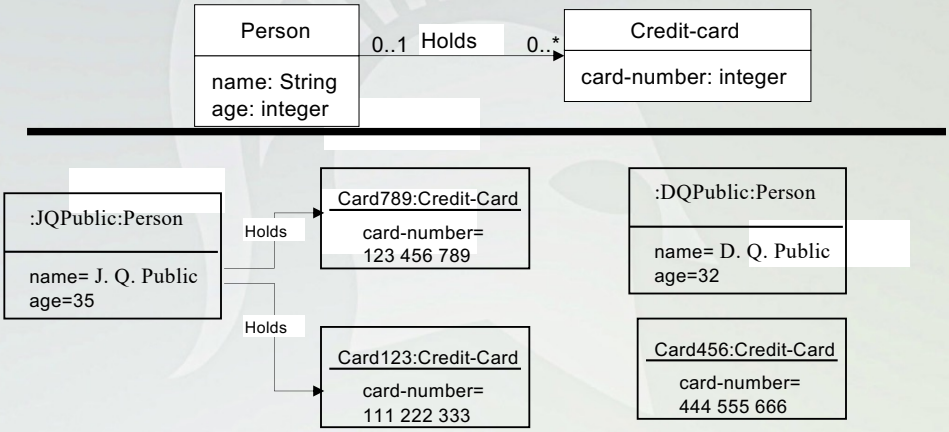
One person can hold zero or more credit cards



12

Multiplicity (Cont.)

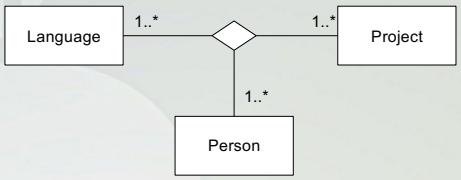
•One person can hold zero or more credit cards (0..*)
•Each card has zero or one holder (0..1)



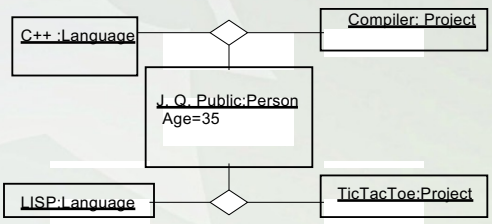
13

Higher order associations

- Ternary association
  - Project, language, person
- Seldom needed (and should be avoided)



Note: hexagons should be rectangles to represent instances

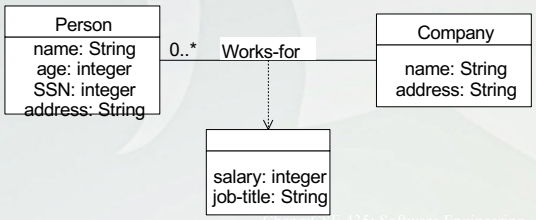


14

Link Attributes

- Associations can have properties the same way objects have properties

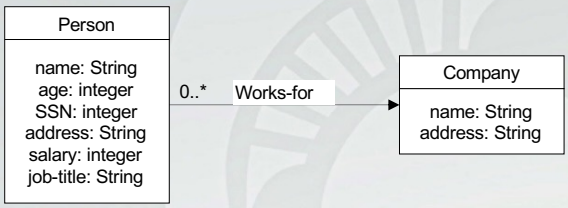
How to represent salary and job title?



Use a link attribute!

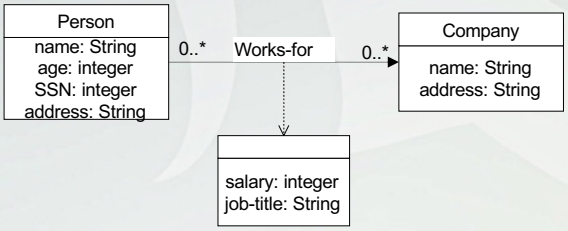
15

Folding Link Attributes



Why not this?

Salary and job title are properties of the job **not** the person

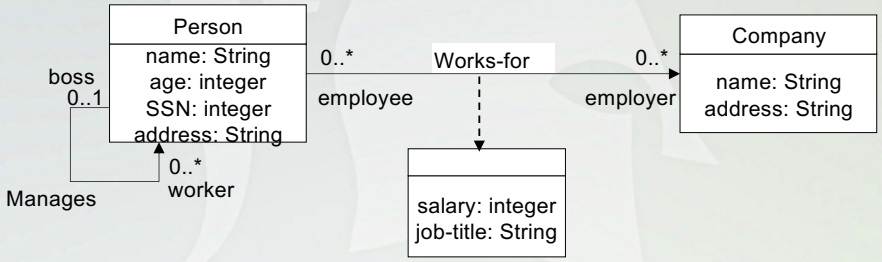


In this case, a link attribute is the only solution

16

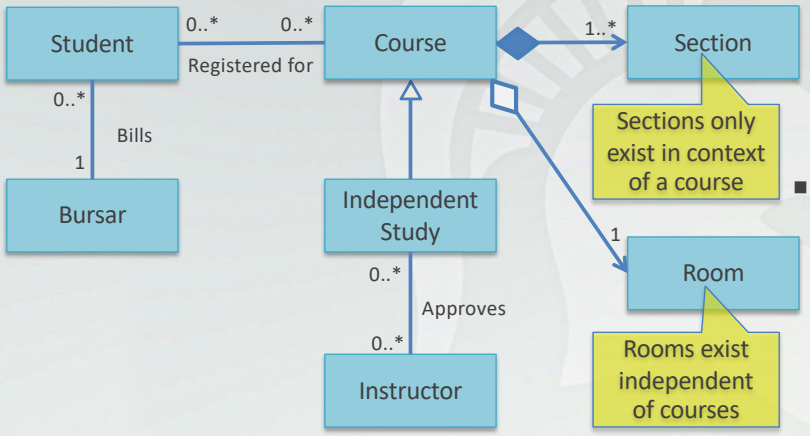
Role Names

- Attach names to the ends of an association to clarify its meaning



17

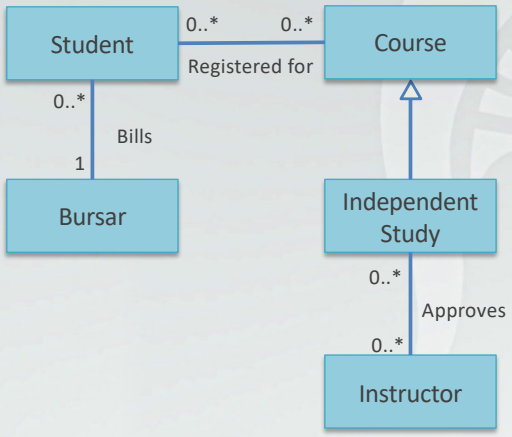
Special Types of Relationships



- ***Composition***
  - Indicates that ***one class contains another***
  - Drawn as an arrow with a ***filled diamond***
- ***Aggregation***
  - Same as composition, except ***contained class can exist independent of container***
  - Drawn as an arrow with a ***open diamond***

20

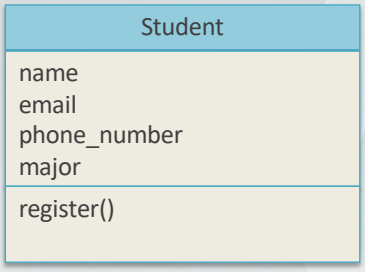
Special Types of Relationships



- ***Generalization/Inheritance***
  - Indicates that one class ***shares characteristics*** of a more general class
  - Drawn as an ***open arrow*** pointing to the more general class

21

Defining Attributes and Operations



- ***Attributes***
  - ***Characteristics*** that distinguish individual instances of a class
  - Listed ***under class name***
- ***Operations/Methods***
  - ***Behavior*** of class
  - Listed ***under attributes***

Prior to design stage, class diagrams are usually not any more detailed than this.

22

Domain vs. Design Model

- ***Domain* Models are used for *Requirements Modeling***
  - Describe the ***problem domain***
  - Objects in ***problem and solution space***
- ***Design* Models are used to model a *Software Implementation***
  - Describe ***object classes in a software system***
  - Include more ***implementation details*** (data types and hidden variable)
  - Classes may or may not correlate with classes in the domain model

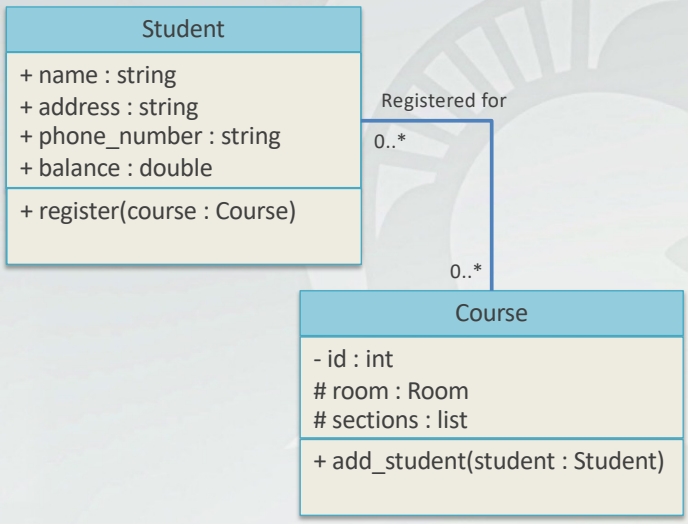
Domain Model (Requirements)

Design Model (Implementation)

Class Diagrams can be used for ***a variety of modeling purposes***.

23

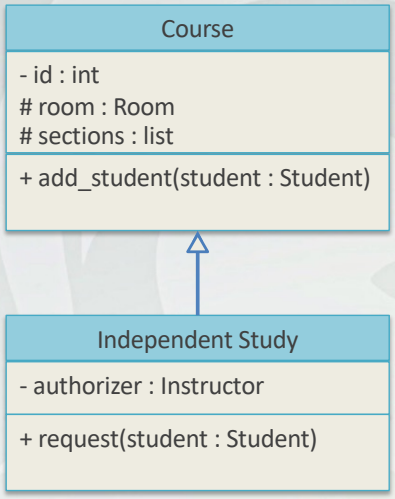
Implementation Details in Class Diagrams



- ***Data types***
  - Specifies ***type*** of each attribute
  - ***Follows attribute***, with a ***colon***
- ***Data access***
  - Indicates ***visibility*** of attribute
  - ***Precedes attribute***
  - (+) ***Public***
  - (#) ***Protected***
  - (-) ***Private***

24

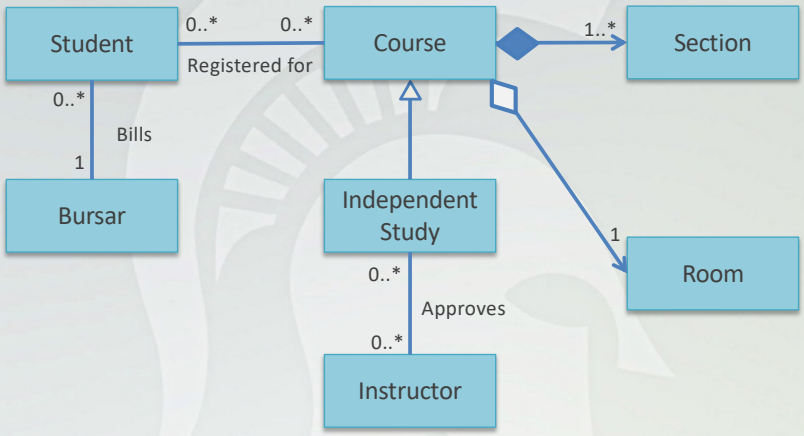
Inheritance of Attributes and Operations



- ***Parent Class:***
  - Define common attributes and operations
- ***Derived Classes***
  - Only include attributes and operations ***not in the parent*** class
  - Child class has access to all ***public and protected*** attributes

25

Domain Model Class Diagram (High-level)

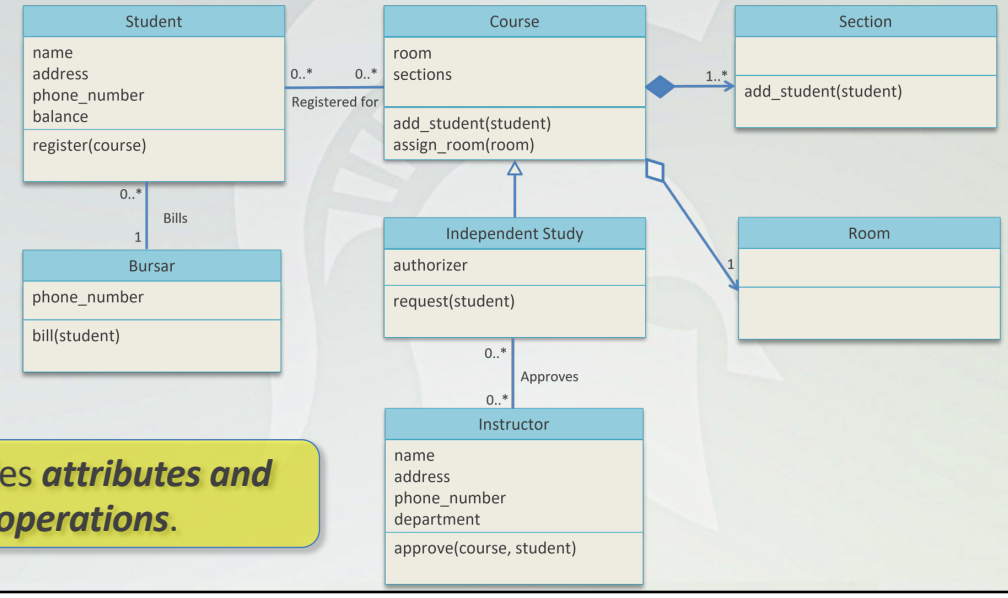


Models real world ***objects and relationships for problem domain***.

26

MICHIGAN STATE UNIVERSITY

Domain Model Class Diagram (Detailed)

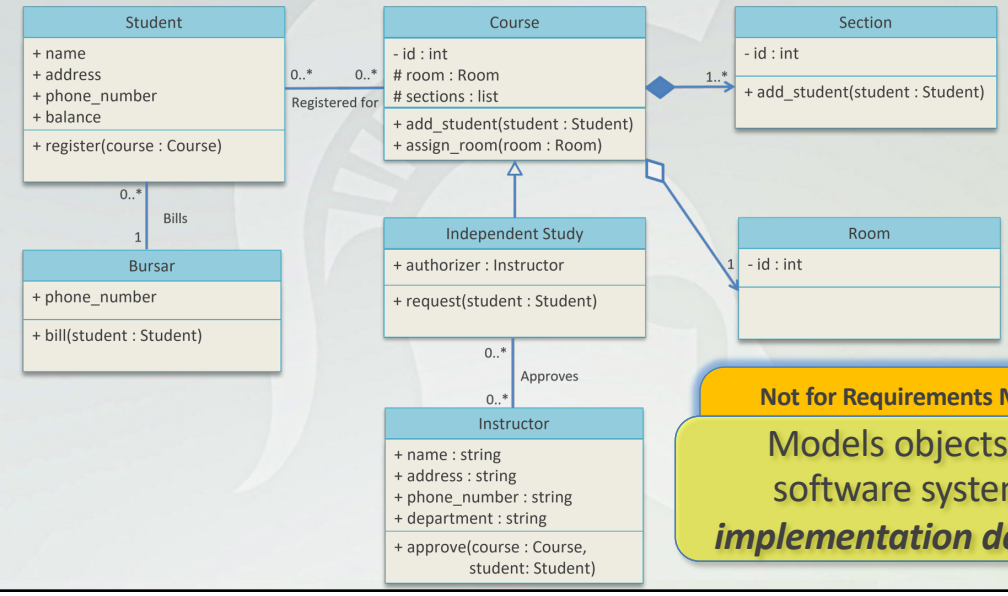


Includes ***attributes and operations***.

27

MICHIGAN STATE UNIVERSITY

Design Model Class Diagram



Not for Requirements Modeling.

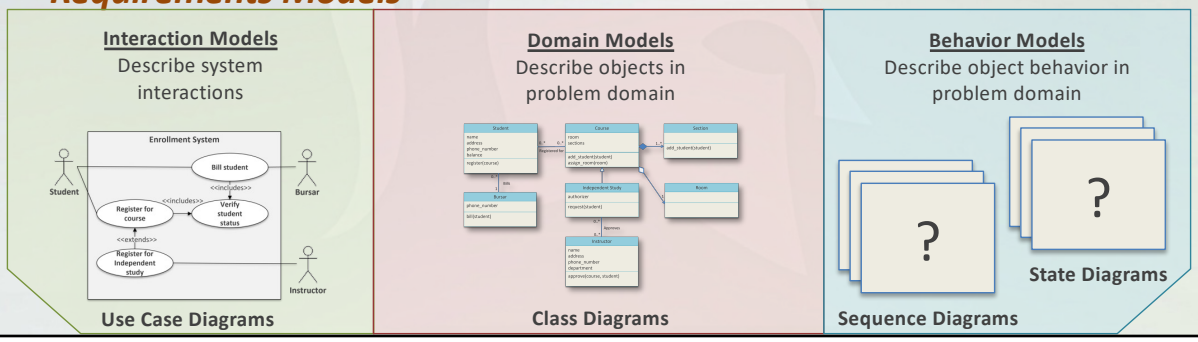
Models objects in software system, ***implementation details***.

28

Summary

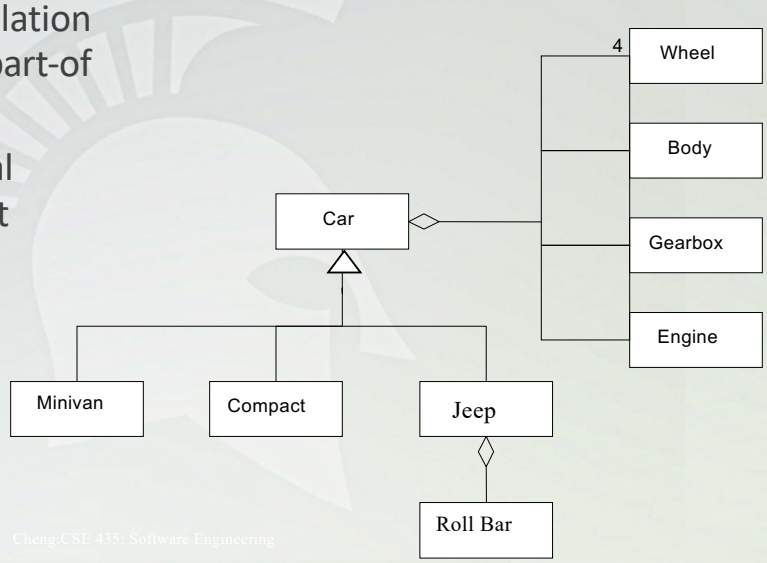
- ***Class Diagrams can be used for different types of models***
  - ***Domain Models*** – describe ***problem/solution space*** in early activities
  - ***Design Models*** – describe ***software implementation*** in later activities

- ***Requirements Models***



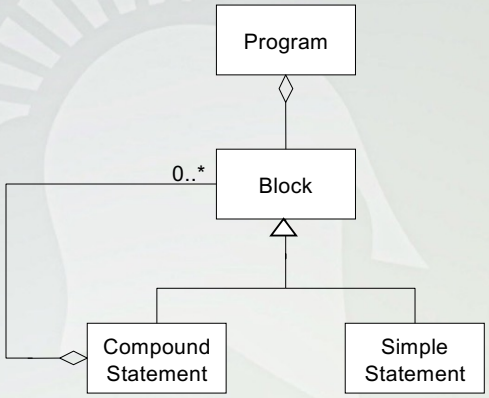
Aggregation Versus Inheritance

- Do not confuse the is-a relation (inheritance) with the is-part-of relation (aggregation)
- Use inheritance for special cases of a general concept
- Use aggregation for parts explosion



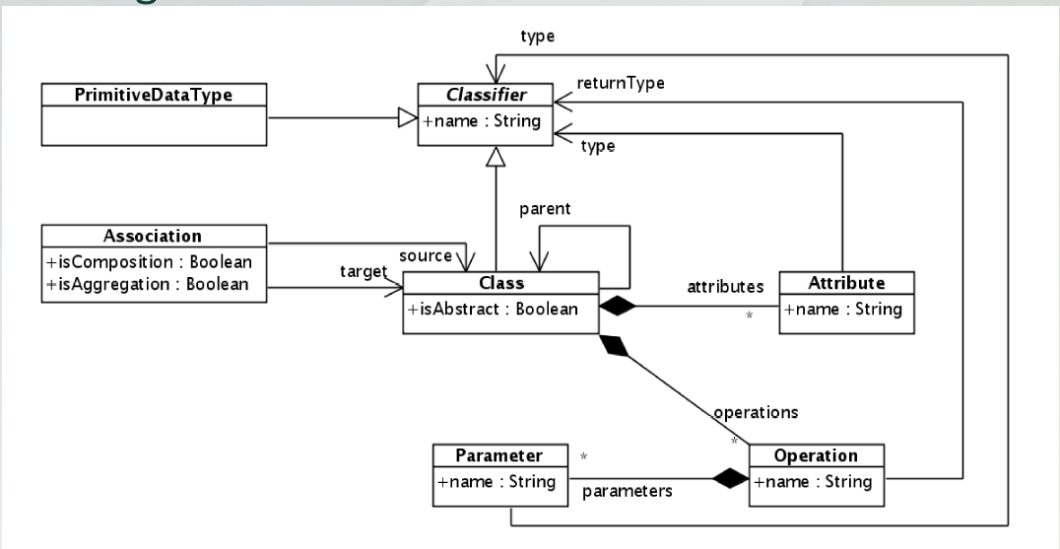
Recursive Aggregates

- A recursive aggregate contains (directly or indirectly) an instance of the same kind of aggregate



31

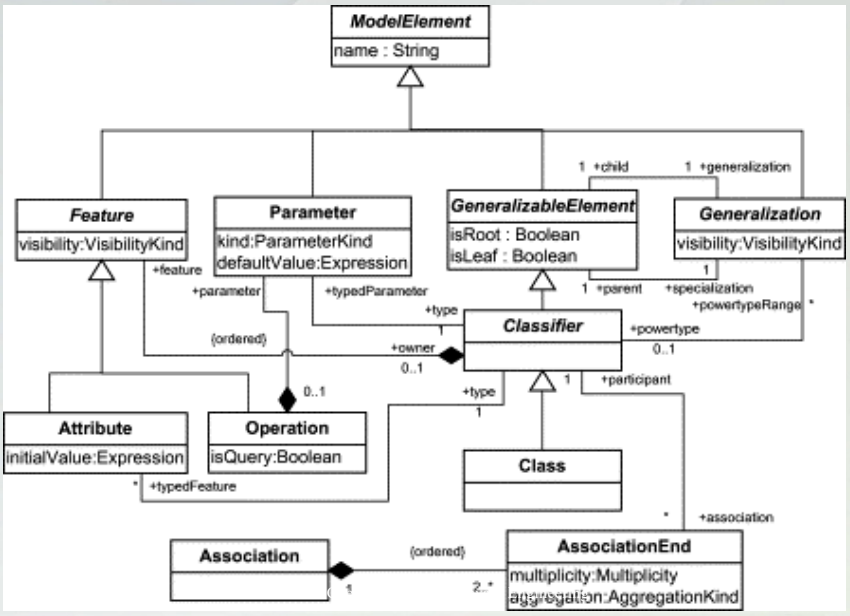
Class diagram Metamodel I



Eclipse.org

32

Class diagram Metamodel II



Science direct

33

Use Case Metamodel I

uu: use case association relationship
_i: includes
_e: extends
_g: generalization
aa: actor relationship

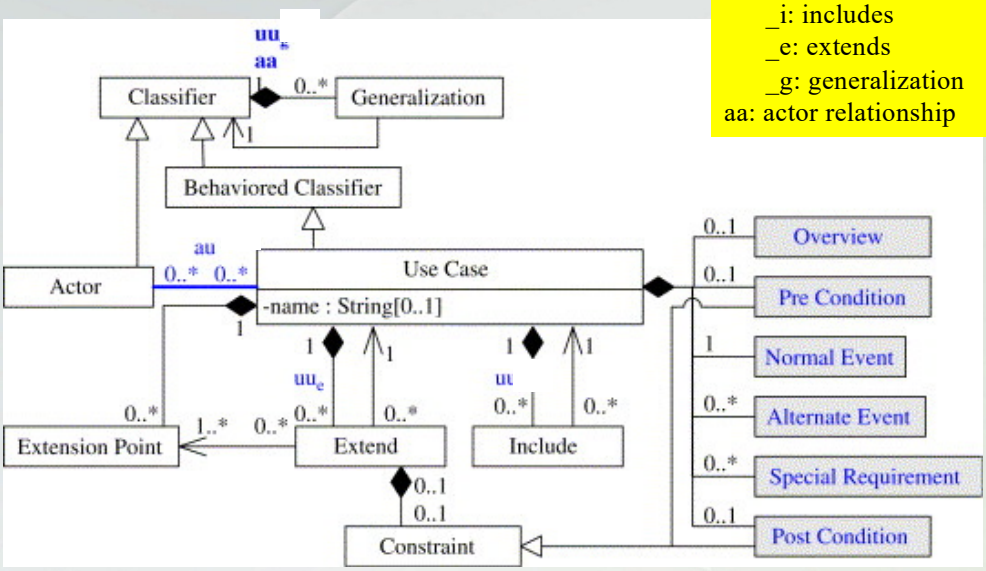


Image: Science Direct: ``Visual Modeling for Software Intensive Systems, Kooper et al, 2006.

34

Use Case Metamodel II

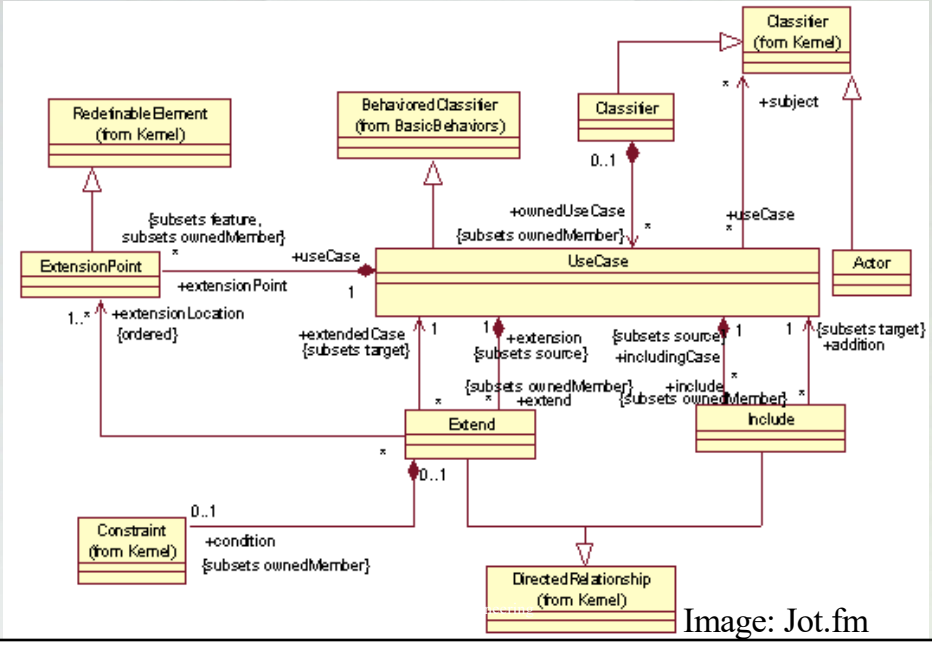
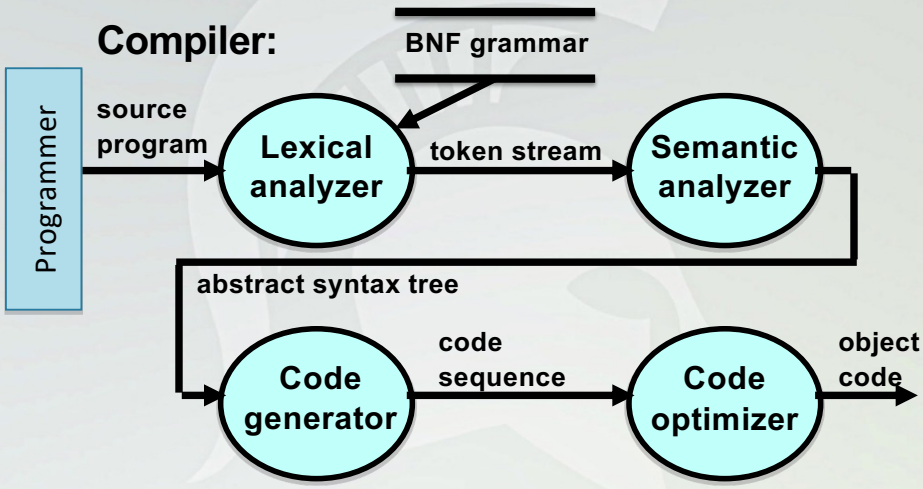


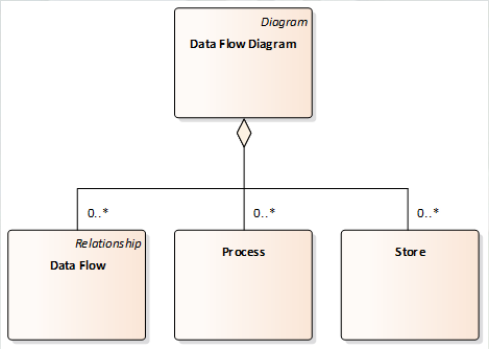
Image: Jot.fm

Data flow diagram (DFD)



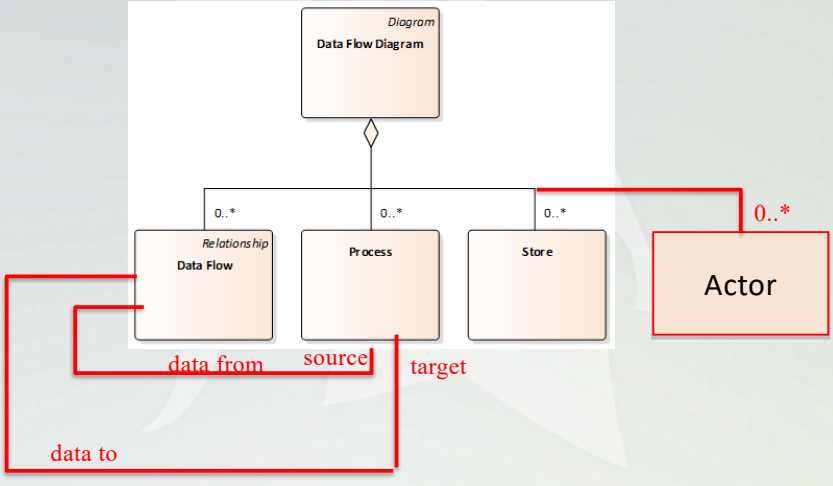
Cheng:CSE 435: Software Engineering

Possible Metamodel for DFD



37

Revised Metamodel for DFD



PC:https://community.sparxsystems.com/white-papers/tag/metamodel

38

Object Modeling Summary

- Classes
  - Name
  - Attributes
  - Operations
- Associations
  - Roles
  - Link attributes

- Aggregation/Composition
- Inheritance