UML Class Diagram

Agenda

- What is a Class Diagram?
- Essential Elements of a UML Class Diagram
- Tips

What is a Class Diagram?

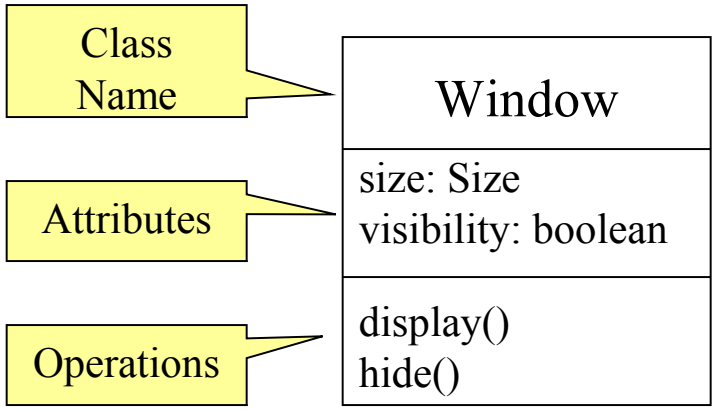
- A Class Diagram is a diagram describing the structure of a system
- shows the system's
  - classes
  - Attributes
  - operations (or methods),
  - Relationships among the classes.

Essential Elements of a UML Class Diagram

- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalization
  - Realization
  - Dependency
- Constraint Rules and Notes

Class

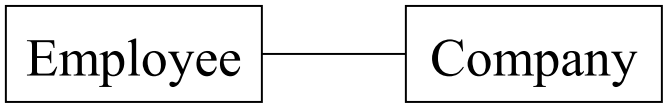
- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behavior)
  - Relationships with other classes
- Attributes and operations may
  - have their visibility marked:
  - "+" for *public*
  - "#" for *protected*
  - "–" for *private*
  - "~" for *package*



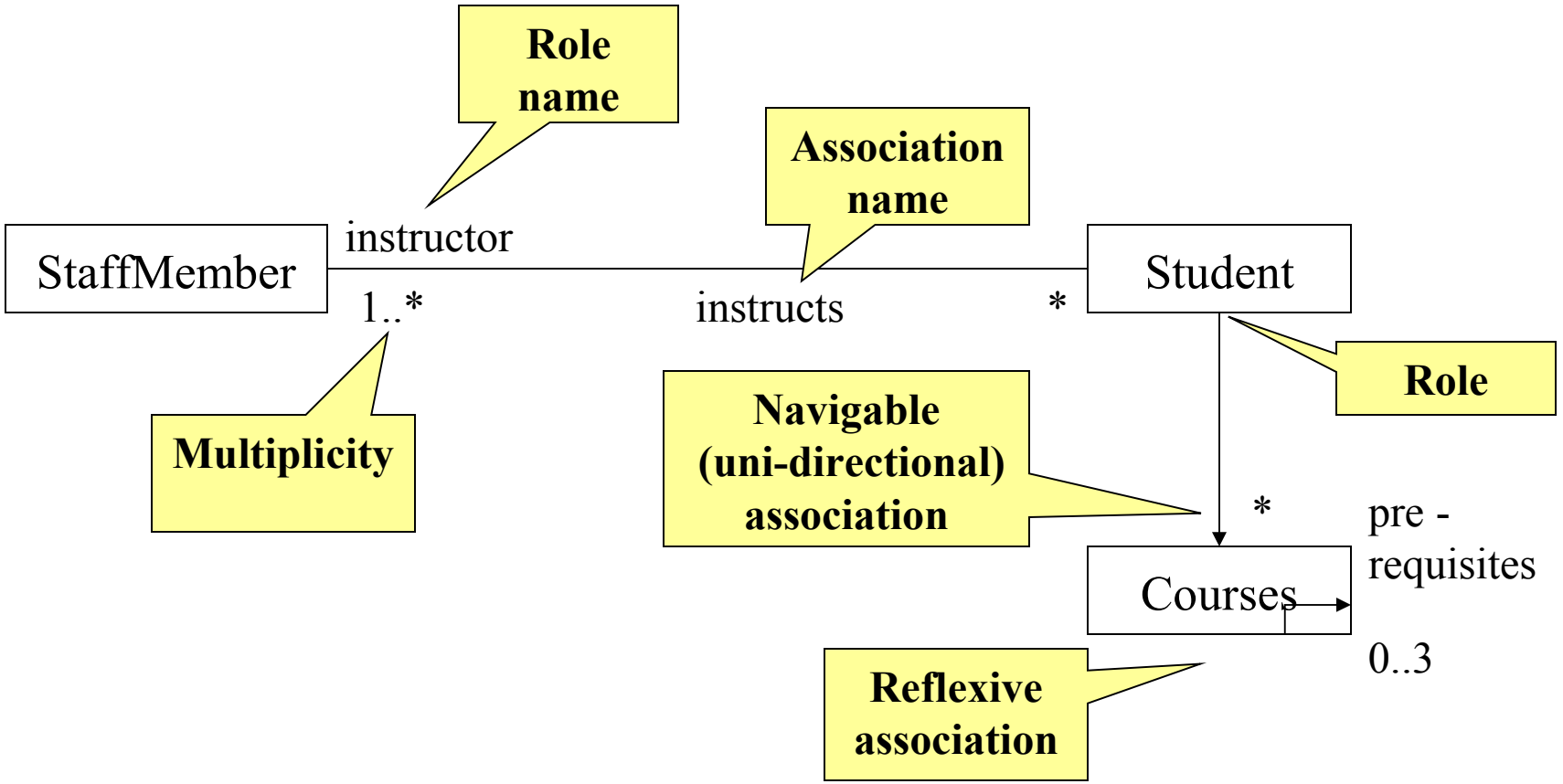
Associations

- An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.

- Example: "An Employee works for a Company"



Associations (cont.)



Associations (cont.)

- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.
- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations

Associations (cont.)

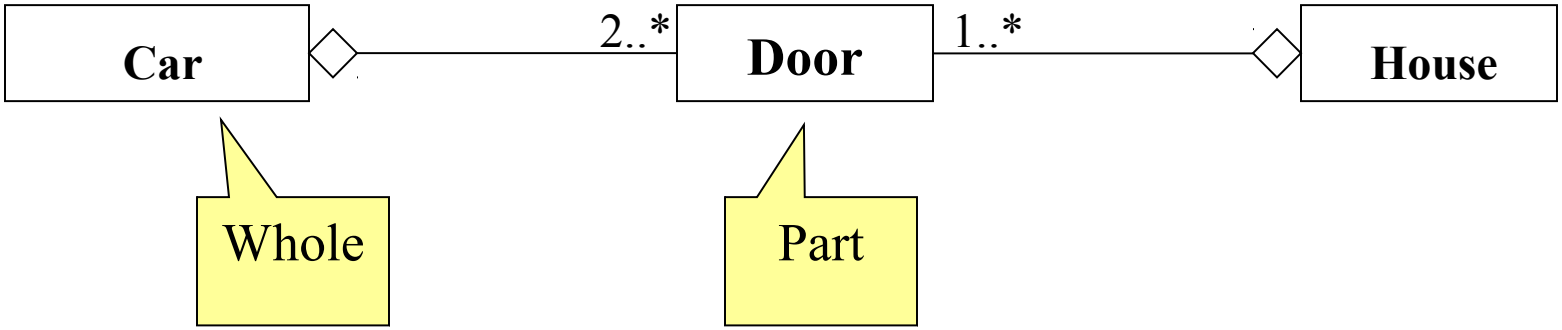
- Multiplicity
  - the number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

Multiplicity Indicators

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

Aggregation

- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a "is a part-part of" relationship.

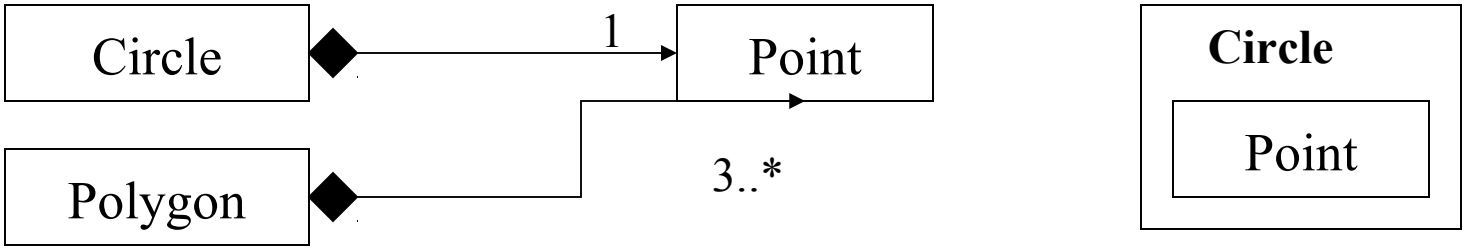


Aggregation (cont.)

- Aggregation tests:
  - Is the phrase “part of” used to describe the relationship?
    - A door is “part of” a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door **is** part of a car. A car **is not** part of a door.

Composition

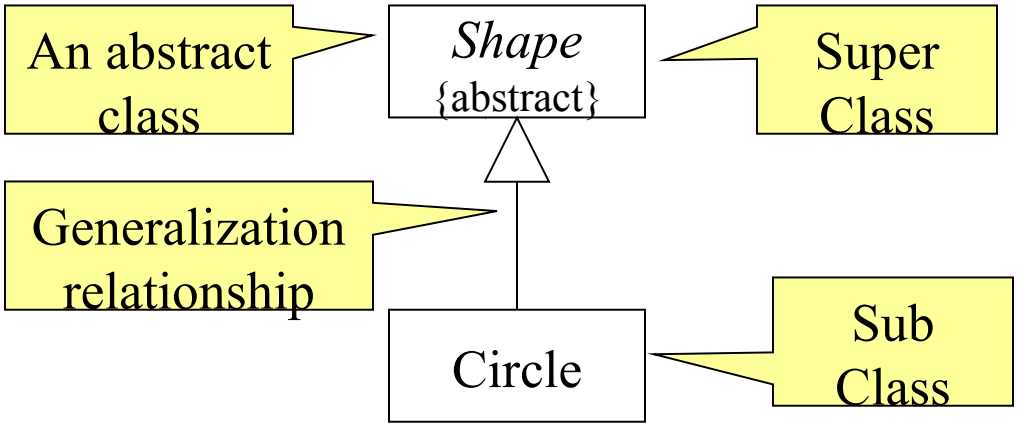
- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.



Generalization

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class).
  - "is kind of" relationship.

`{abstract}` is a tagged value that indicates that the class is abstract. The name of an abstract class should be italicized



Generalization

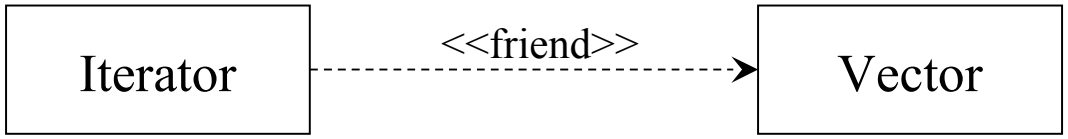
- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

Realization

- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.

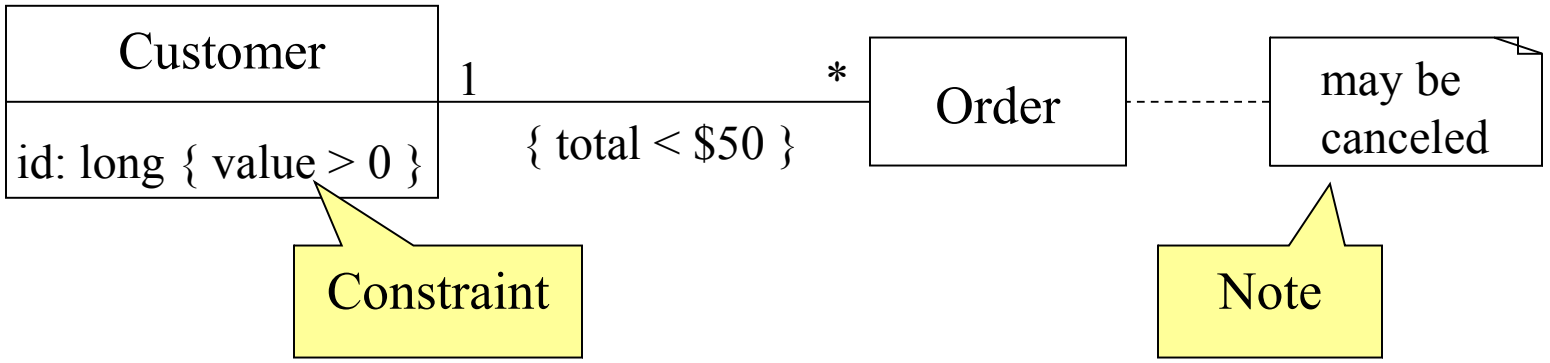
Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.
- This is different from an association, where an attribute of the dependent class is an instance of the independent class.

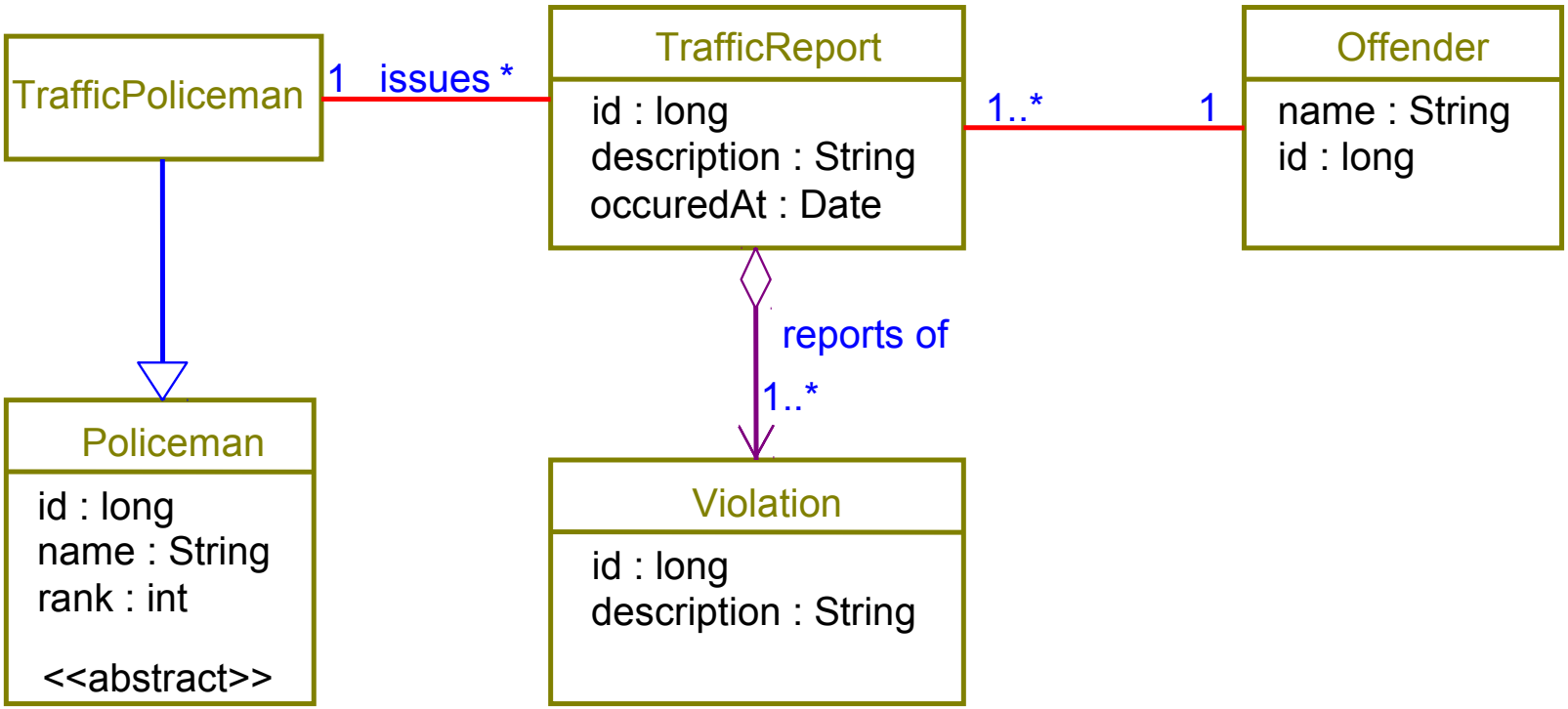


Constraint Rules and Notes

- **Constraints** and **notes** annotate among other things associations, attributes, operations and classes.
- Constraints are semantic restrictions noted as Boolean expressions.
  - UML offers many pre-defined constraints.



Traffic Violation Report System Example

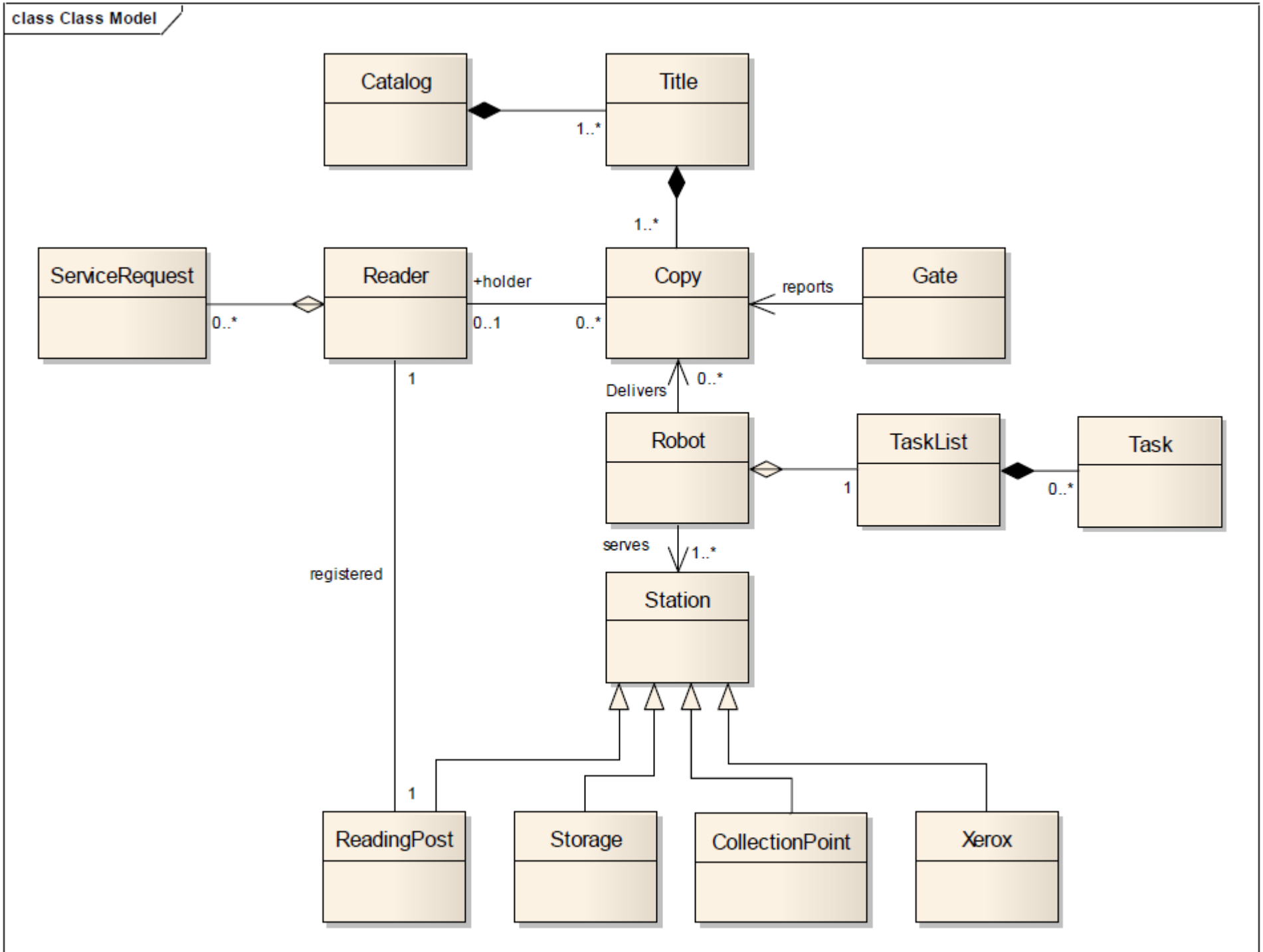


Analysis Classes Elicitation

- Consider main perspectives of the system
- ***Interface*** between the system and its actors
  - Protocols for information exchange
  - Don't concentrate on visual aspects
- ***Data*** the system uses
  - The core of the system, key concepts
- The system ***logic***
  - Controls and coordinates the behavior
  - Delegates the work to other classes
  - Decouples interface and data classes

RoboLib Example

- באולם הספרייה נמצאות עמדות קריאה
- את משימות שינוע הספרים במרחב הספרייה מבצע רובוט
- עותקי הספרים עצמם מאוחסנים על גבי מדפים במחסן סגור ומבודד
- על כל עותק מוטבע בר-קוד, המציין את כותר-הספר כפי שהוא מופיע בקטלוג, ואת מספר העותק


class Class Model
Catalog
Title
1..*
1..*
ServiceRequest
Reader
+holder
Copy
reports
Gate
0..*
0..1
0..*
1
Delivers
0..*
Robot
TaskList
Task
1
0..*
serves
1..*
registered
Station
1
ReadingPost
Storage
CollectionPoint
Xerox

Tips

- Don’t try to use all the various notations.
- Don’t draw models for everything, concentrate on the key areas.

Backup

Analysis Classes

- A technique for finding analysis classes which uses three different perspectives of the system:
  - The boundary between the system and its actors (Boundary)
  - The information the system uses (Entity)
  - The control logic of the system (Control)

Boundary Classes

- Models the interaction between the system's surroundings and its inner workings
  - User interface classes
    - Concentrate on what information is presented to the user
    - Don't concentrate on user interface details
  - System / Device interface classes
    - Concentrate on what protocols must be defined. Don't concentrate on how the protocols are implemented

Boundary Classes (cont.)

- Boundary classes are environment dependent:
  - UI dependent
  - Communication protocol dependent

Entity Classes

- Models the key concepts of the system
- Usually models information that is persistent
- Contains the logic that solves the system problem
- Is environment independent
- Can be used in multiple use cases

For example: Violation, Report, Offender.

Control Classes

- Controls and coordinates the behavior of a use case
- Delegates the work of the use case to classes
  - A control class should tell other classes to do something and should never do anything except for directing
- Control classes decouple boundary and entity classes
- Often, there is one control class per use case