University of Buea
Faculty of Engineering and Technology
Department of Computer Engineering

CEF 331

OBJECT ORIENTED MODELING AND UNIFIED MODELING LANGUAGE (UML)

By I.R. DJOUELA KAMGANG Epse NOUABO; Dr. Eng.

dorlystevia08@gmail.com
ines.djouela@supcom.tn

Course outline

Chapter 1: Modeling, lifecycle and methods

Chapter 2: Object technology

Chapter 3: Unified Modeling Language (UML) and its diagrams

Chapter 4: Use case diagram

Chapter 5: Class diagram

Chapter 6: Sequence diagram

Chapter 7: projects

Course outline

Chapter 2: Object technology

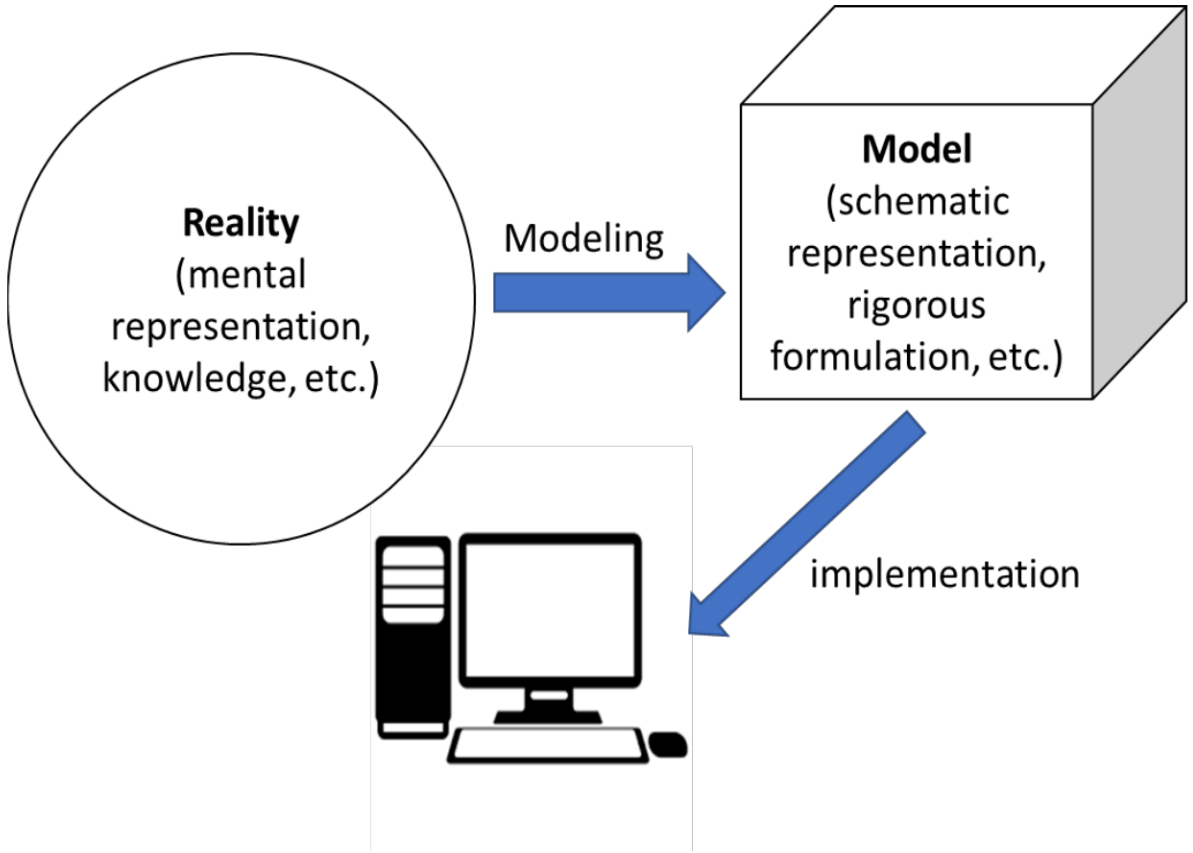
The main objective of this chapter is to understand the basic concepts and terms of object orientation and the associated UML notation.

Course outline

Chapter 2: Object technology

- Reminder: Procedural vs object-oriented methods
- What is UML (unified Modeling Language)
- Object-oriented important concepts
  - ✓ Object
  - ✓ Class
  - ✓ Association
  - ✓ Generalization
  - ✓ Aggregation and composition
  - ✓ Dependency
  - ✓ Encapsulation
  - ✓ Polymorphism
- UML diagrams

What is a Model?



- A model is a simplification of reality.
- It is used for communication among stakeholders.
- A model may provide
  - ✓blueprints of a system
  - ✓Organization of the system
  - ✓Dynamic of the system

Procedural vs object-oriented methods

PROCEDURAL OR FUNCTIONAL METHOD (MERISE)

- The system is viewed as a collection of functions.
- Moving from one phase to another is complex.
- Increases duration of project.
- Increases complexity (non iterative).
- Focuses on function and procedures for each step od the process.
- It is a top down approach

OBJECT-ORIENTED METHOD (UML)

- The system is viewed as a collection of objects.
- Moving from one phase to another is easier.
- Decreases duration of projects.
- Decreases complexity (highly iterative)
- Focuses on object, classes, modules that can easily be replaced.
- It is a bottom-up approach

What is UML?

UML is a language for

- Visualizing
- Specifying
- Constructing
- Documenting

Objected-Oriented important terminologies

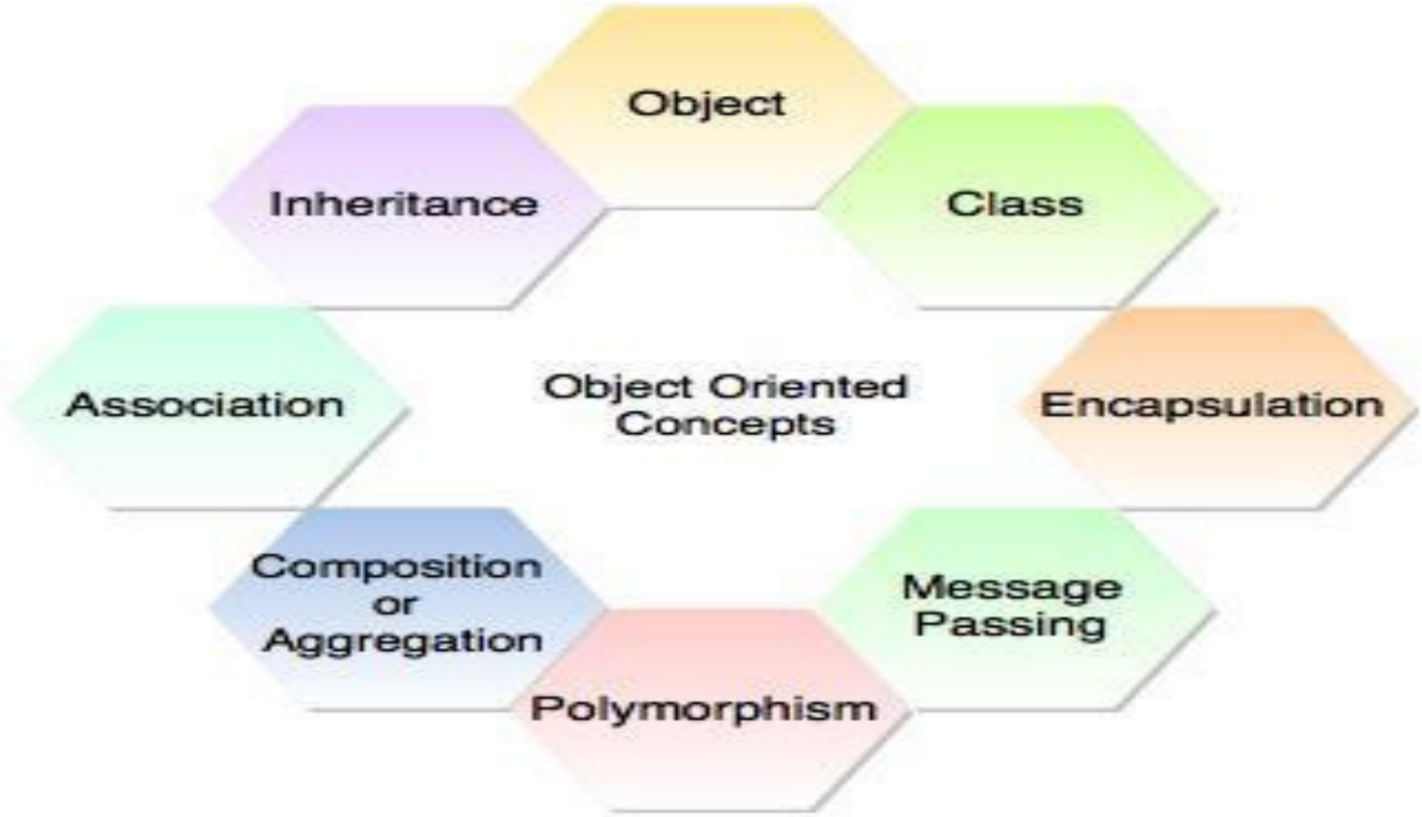


Fig. Object Oriented Concepts

Objects

Nearly anything can be an object

- External Entities: that interact with the system being modeled **E.g. people, devices, other systems**
- Things …that are part of the domain being modeled **E.g. reports, displays, signals, etc.**
- Occurrences or Events …that occur in the context of the system **E.g. transfer of resources, a control action, etc.**
- Roles played by people who interact with the system
- Organizational Units that are relevant to the application E.g. division, group, team, etc.
- Places …that establish the context of the problem being modeled E.g. manufacturing floor, loading dock, etc.
- Structures …that define a class or assembly of objects E.g. sensors, four-wheeled vehicles, computers, etc.

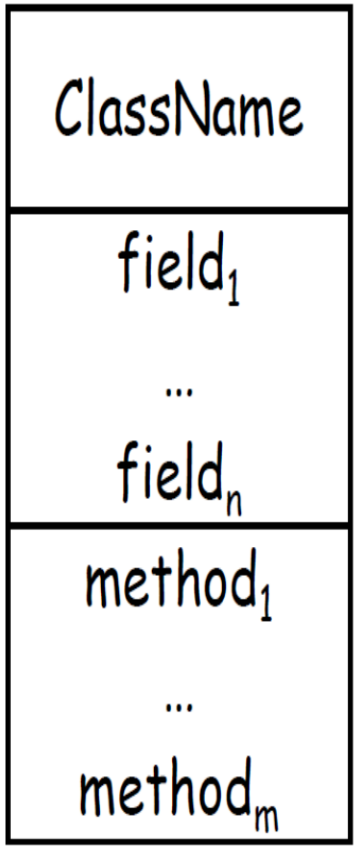
Some things cannot be objects:

- procedures (e.g. print, invert, etc).

Objects

- Each object has a unique identity.
- The state of an object is composed of a set of fields (data fields), or attributes.
- Each field has a name, a type, and a value.
- Behaviors are defined by methods.
- Each method has a name, a type, and a value.
- Each method may or may not return a value.
- Features are a combination of the state and the behavior of the object.

Classes



The top compartment shows the class name.

The middle compartment contains the declarations of the fields of the class.

The bottom compartment contains the declarations of the methods

A class describes a group of objects with

- similar properties (attributes)
- Common behavior (operations)
- Common relationship to other objects
- And common meaning ('semantics')

E.g. Tutorial: give some examples of classes.

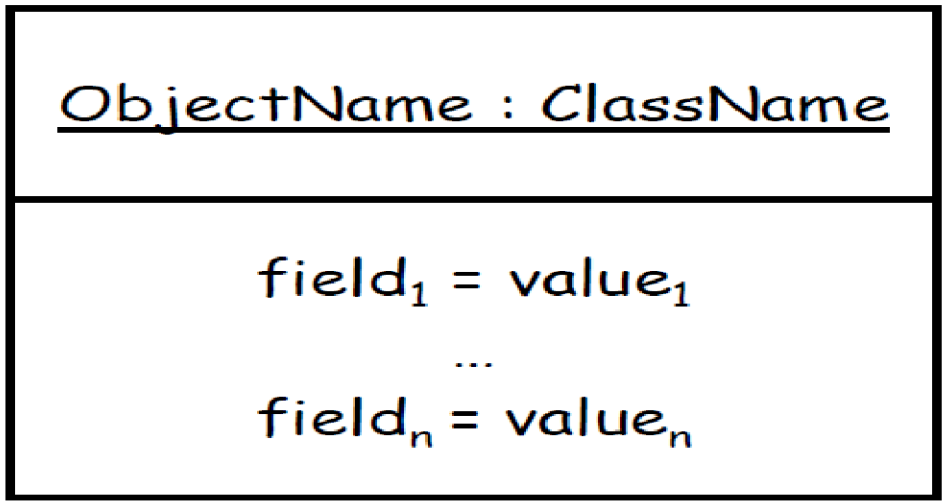
Finding Classes

- **Finding classes from source data:** Look for nouns and noun phrases in stakeholders' descriptions of the problem
  - ✓include in the model if they explain the nature or structure of information in the application.
- **Finding classes from other sources:**
  - ✓Reviewing background information;
  - ✓Users and other stakeholders;
  - ✓Analysis patterns;
- **It's better to include many candidate classes at first**
  - ✓You can always eliminate them later if they turn out not to be useful
  - ✓Explicitly deciding to discard classes is better than just not thinking about them

Visibility syntax in UML

visibility	UML Syntax
Public	+
Protected	#
Package	~
private	-

UML Notation for Object



The top compartment shows the object name and its class.

The bottom compartment contains a list of the fields and their values.

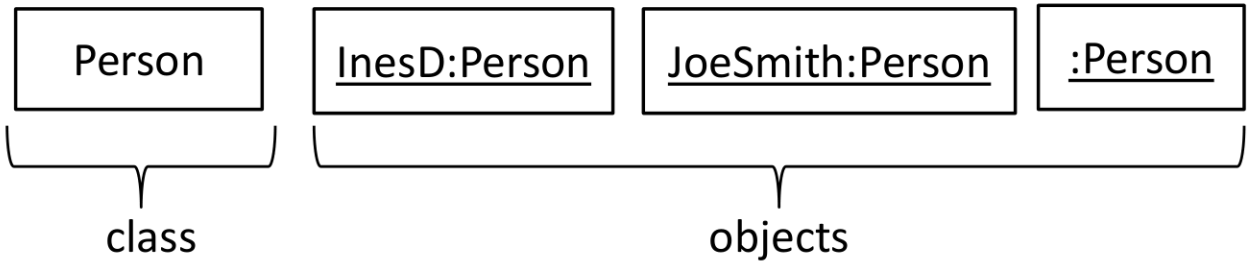
objectName -- objectName whose class is of no interest

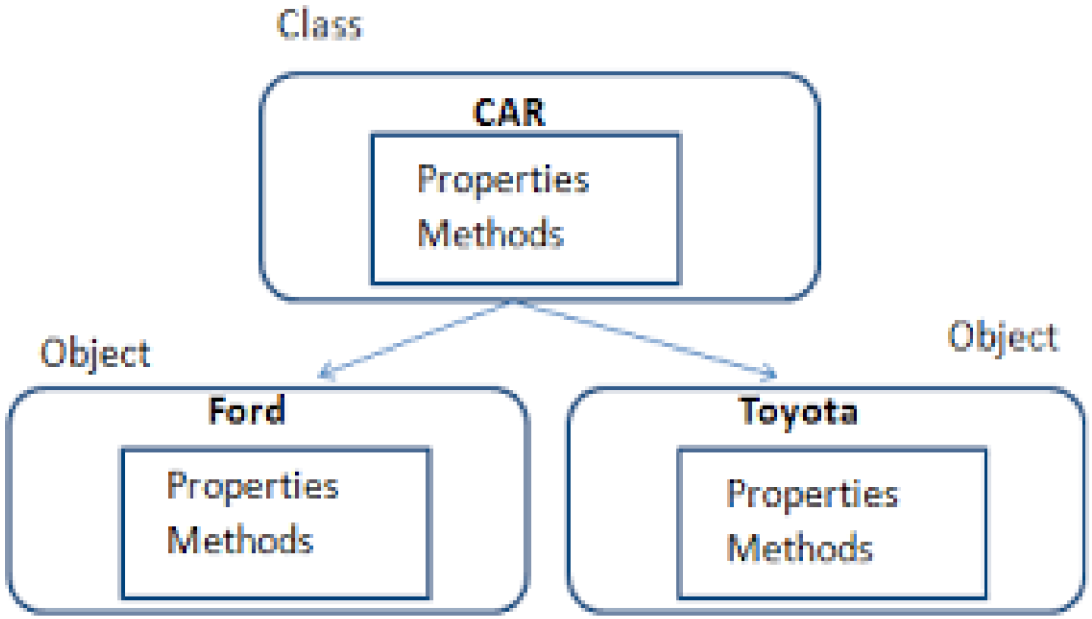
ClassName -- anonymous object of ClassName which can be identified only through its relationship with other objects.

Objects and Classes

	Interpretation in the real world	Interpretation in the model
Object	An object is a thing that can be distinctly identified	An object has an identity, a state and a behavior
class	A class represents a set of objects with similar characteristics and behavior. This objects ae called the instances of the class	A class characterizes the structure of states and behaviors that are shared by all instances

Class vs Object representation



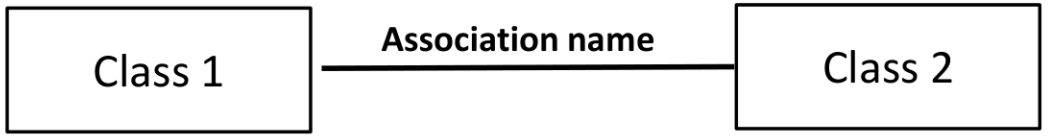


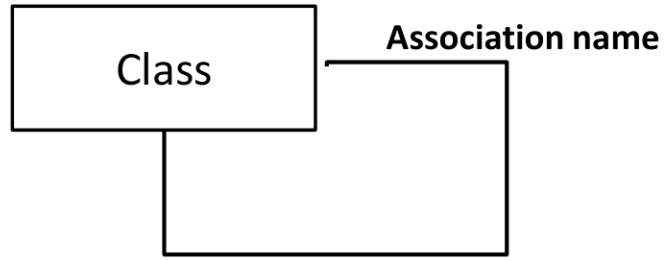
Relationships between Classes

- Objects do not exist in isolation from one another. Relationships exist among them as in real-life.
- A relationship specifies the type of link that exists between objects.
- In UML, there are different types of relationships between classes:
  - ✓ Association
  - ✓ Aggregation and Composition (“part-of”),
  - ✓ Generalization/ specialisation (“type-of”),
  - ✓ Dependency

Association

- Association represents binary relationship between classes
- An association is used to show how two classes are related to each other (communication between classes)
- The binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class.



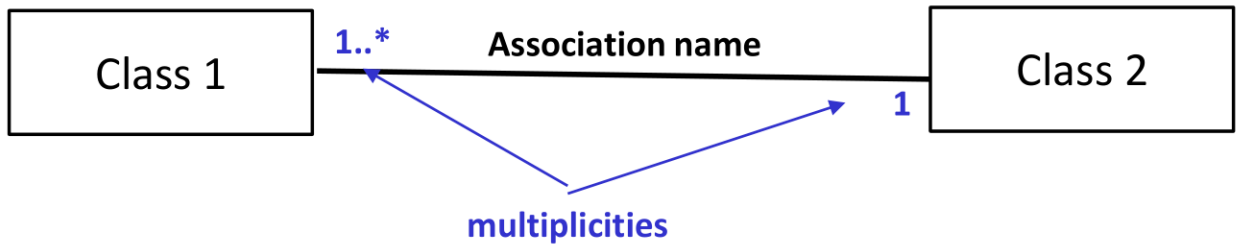


Self association

Association: multiplicity

indicator	Meaning
0…1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only n (where n>1)
0..n	Zero to n (where n>1)
1..n	One to n (where n>1)

- Multiplicity defines the number of objects associated with an instance of the association.
- UML diagrams explicitly list multiplicity at the end of association lines.
- Intervals are used to express multiplicity:



Tutorial 1: Analyzing and validating associations

Represent the following association with it corresponding multiplicities

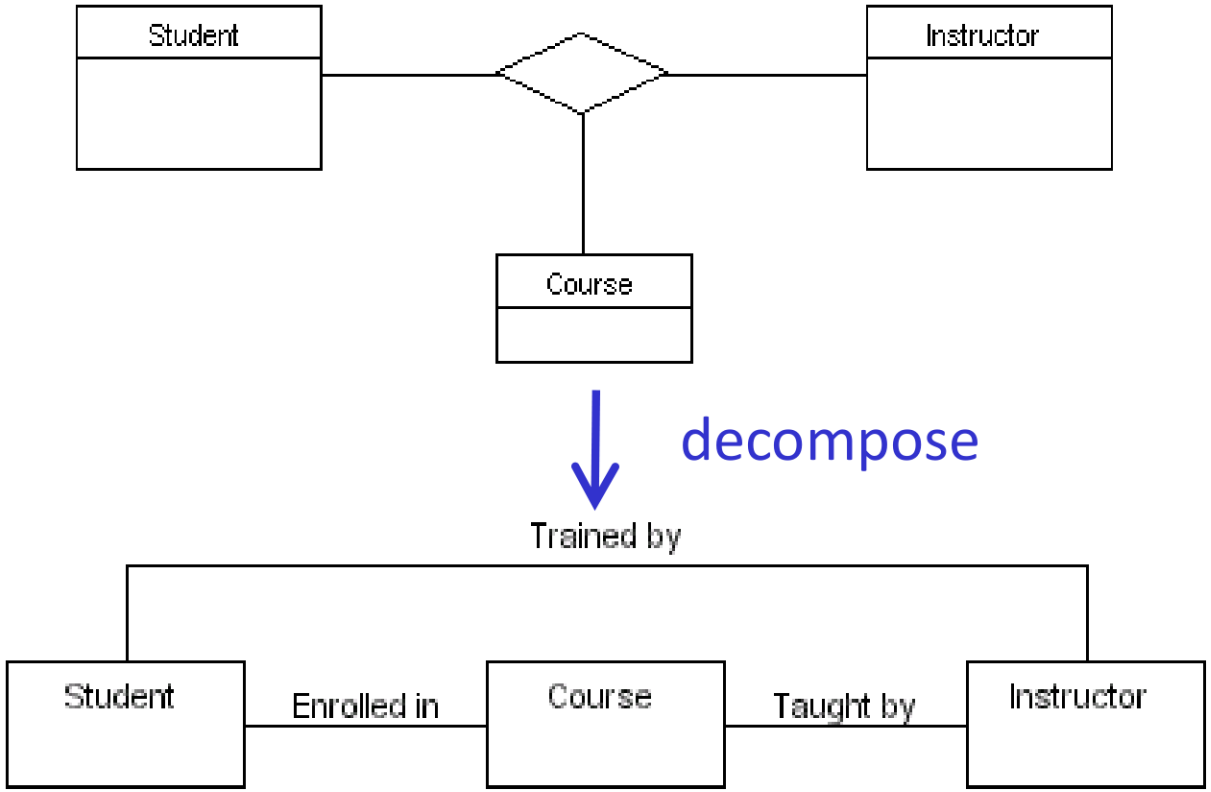
- A student can take up to six courses
- Student has to be enrolled into at least one course
- Up to 300 students can enroll in a course
- A class should have at least 10 students.

Tutorial 2: Analyzing and validating associations

Represent the following association with it corresponding multiplicities

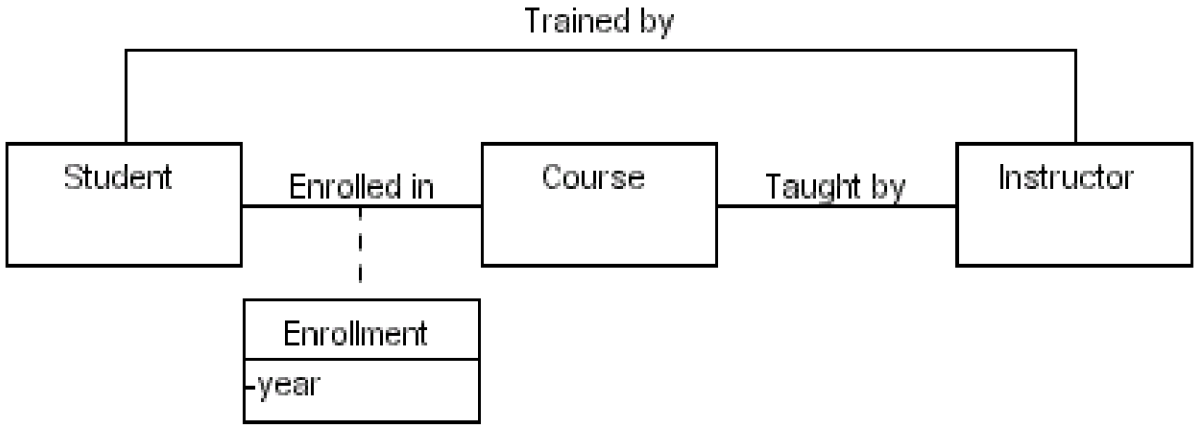
- A company has many employees,
- An employee can only work for one company.
- A company can have zero employees
- It is not possible to be an employee unless you work for a company

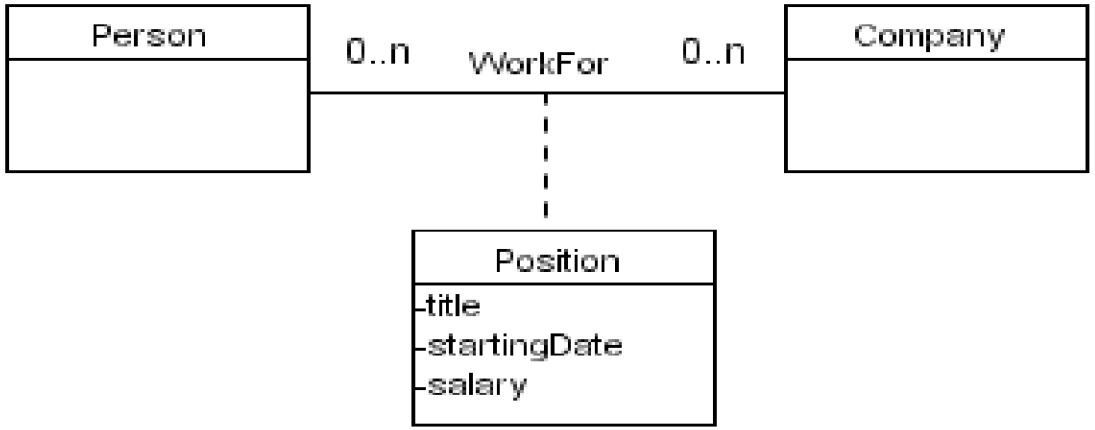
N-Ary Associations



Association classes

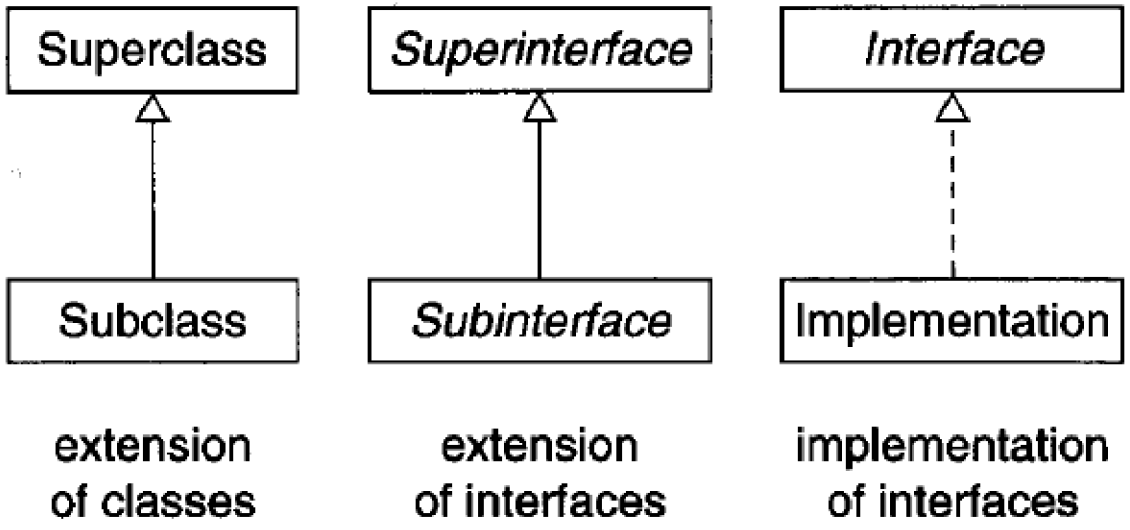
- Sometimes, it is necessary to describe an association by including some attributes which do not naturally belong to the objects involved in the association.

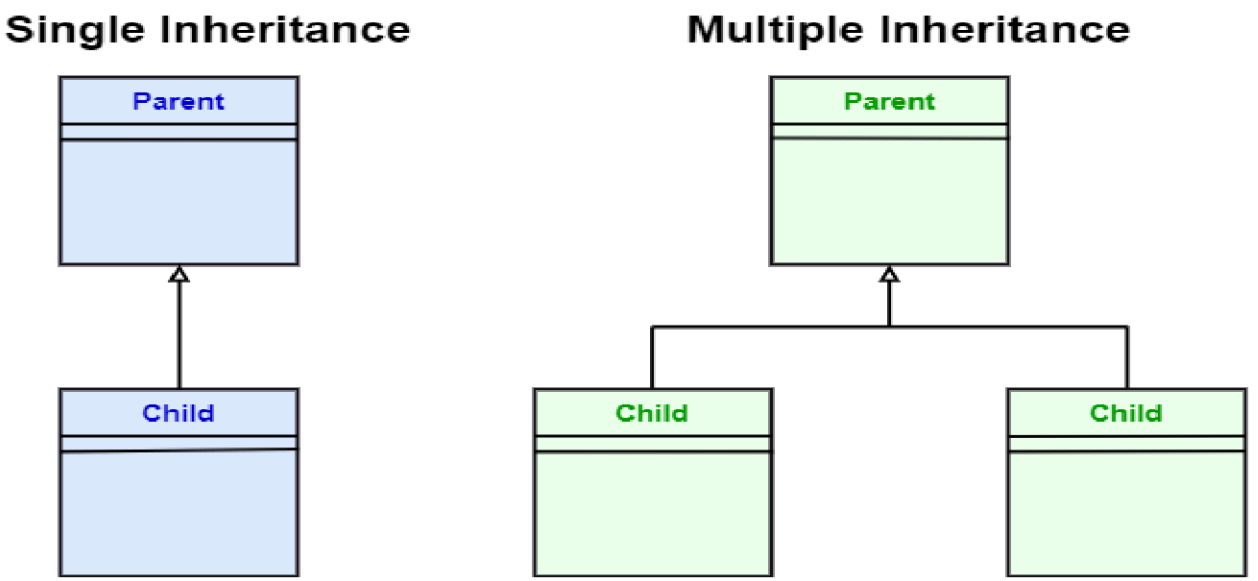




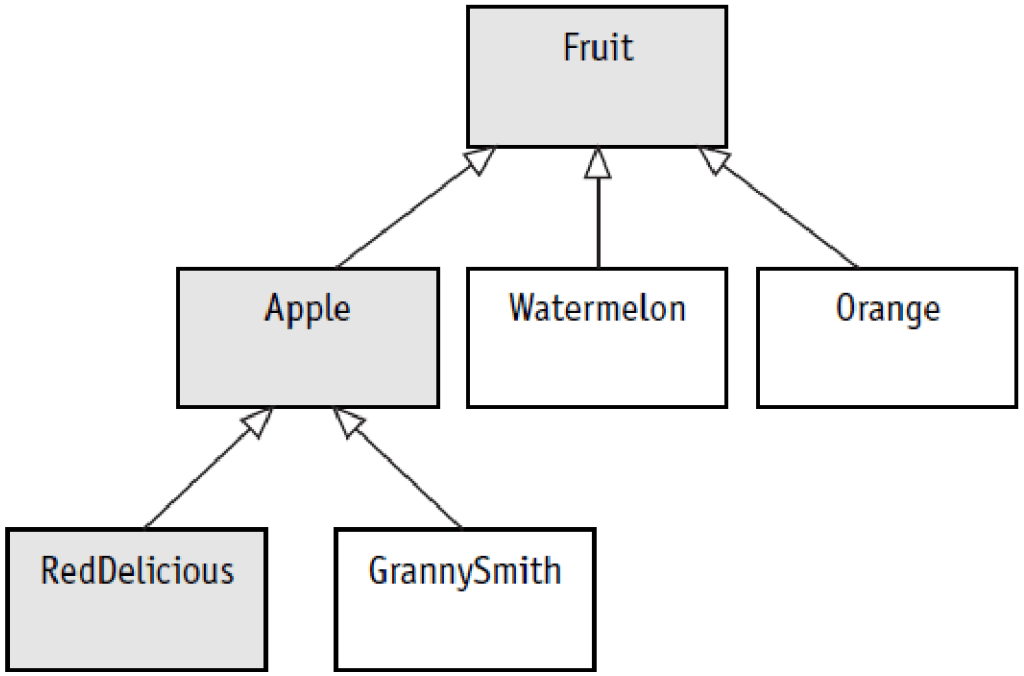
Generalization a.k.a Inheritance

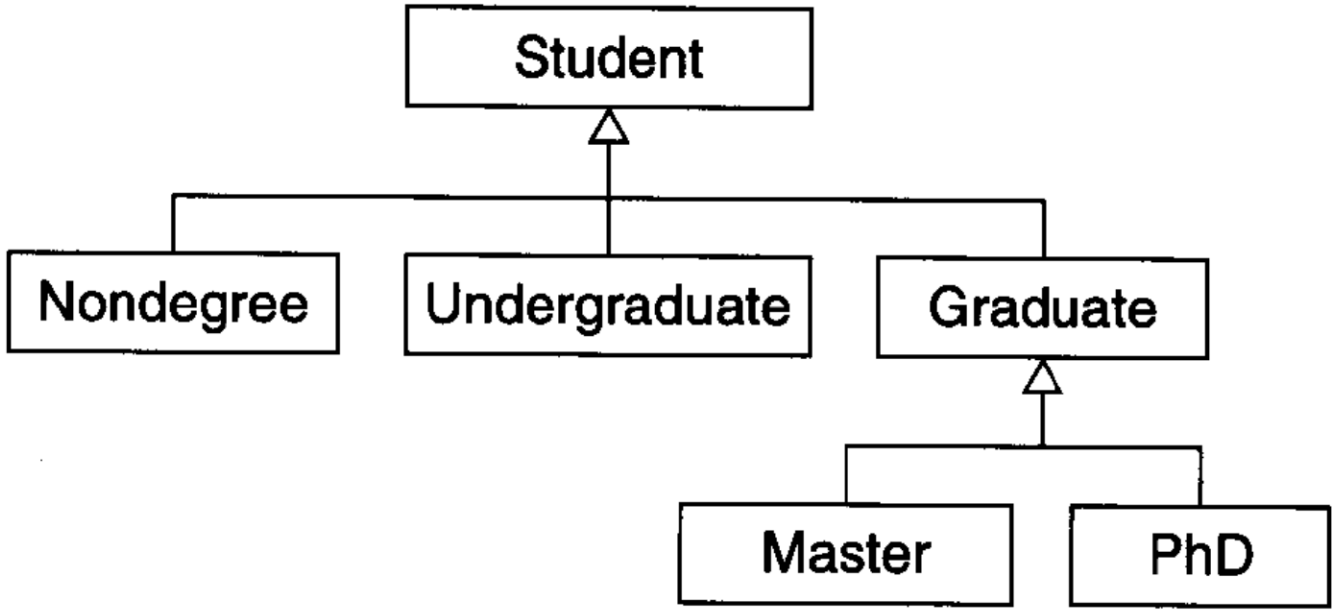
- Define a relationship among classes and interfaces -- the "is-a(n)" or the "type-of" or the "is a kind of" Relationship.
- Subclasses inherit attributes, associations, & operations from the superclass.





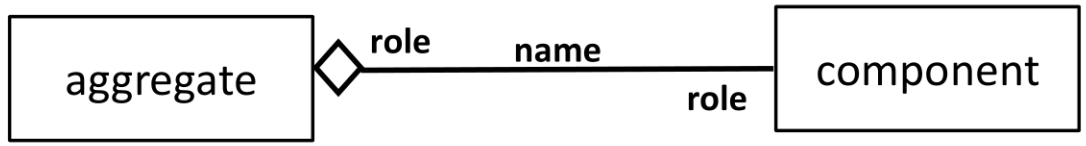
Generalization Example



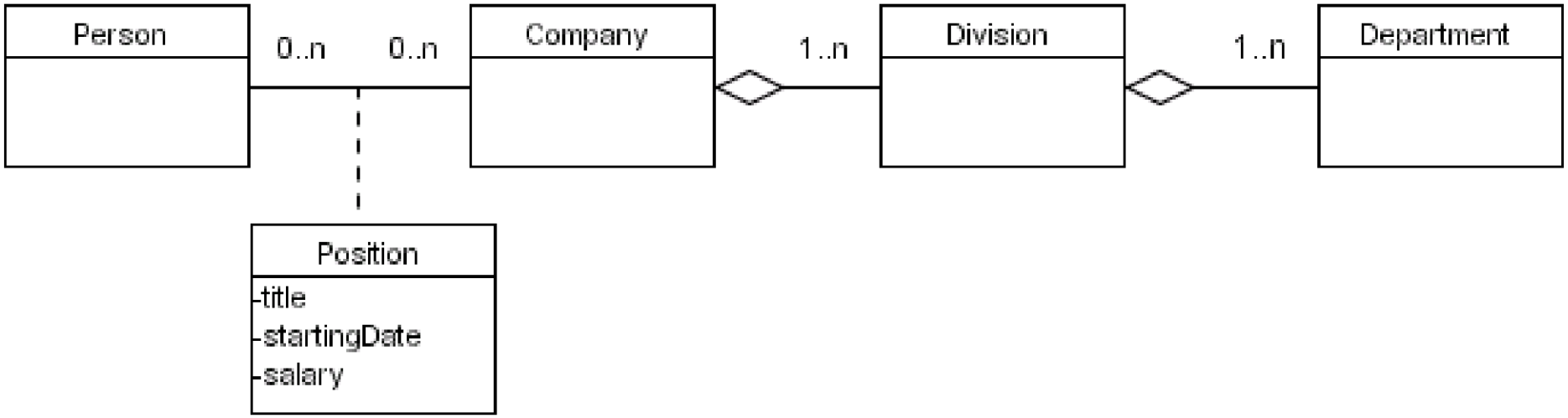


Aggregation

- Aggregation is a special form of association– it represents the "Has-a" or "part-of" relationship.
- In UML, a link is placed between the "whole" and the "parts" classes with a diamond head attached to the "whole" class to indicate that this association is an aggregation.
- Multiplicity can be specified at the end of the association for each of the "part-of" classes to indicate the quantity of the constituent parts.
- Typically, aggregations are not named, and the keywords used to identify aggregations are "consists of", "contains" or "is part of".

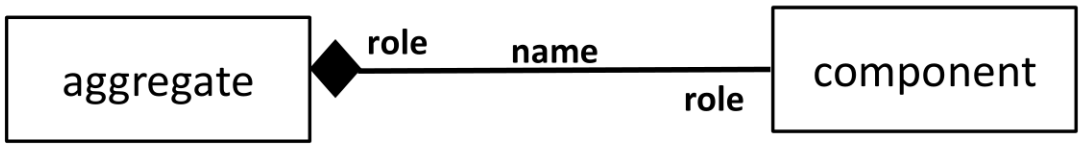


Aggregation example

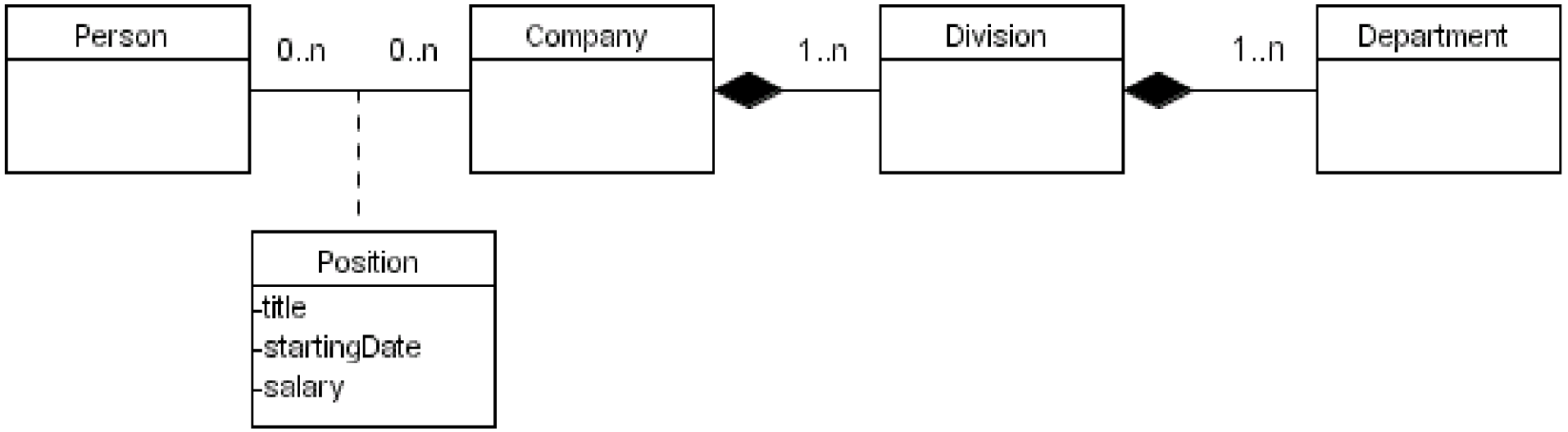


Composition

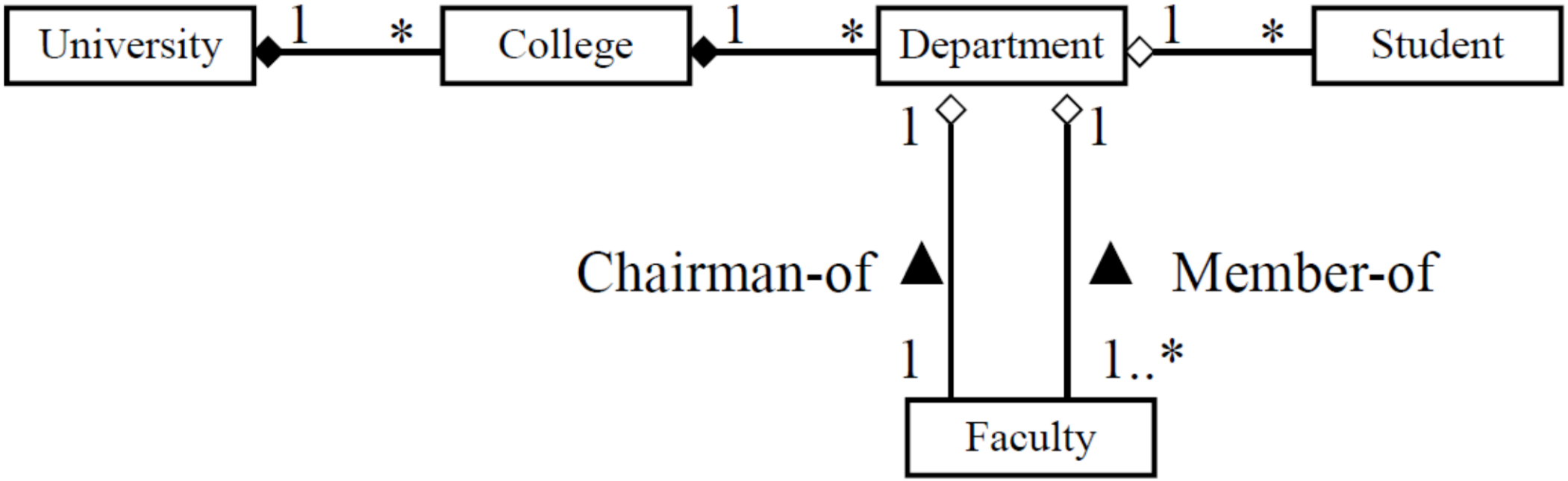
- A stronger form of aggregation is called composition, which implies exclusive ownership of the "part-of" classes by the "whole" class, i.e. a composite object has exclusive ownership of the parts objects.
- This means that parts may be created after a composite is created, but such parts will be explicitly removed before the destruction of the composite.



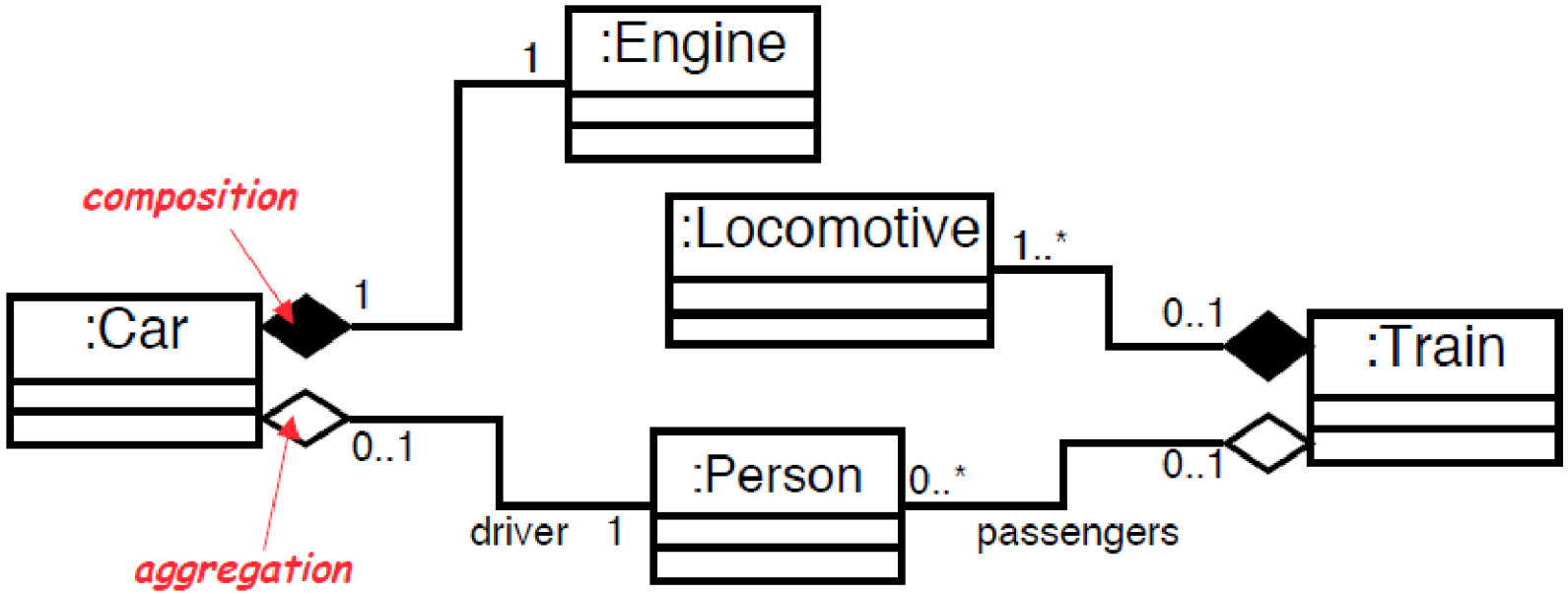
Composition example



Aggregation and Composition example 1

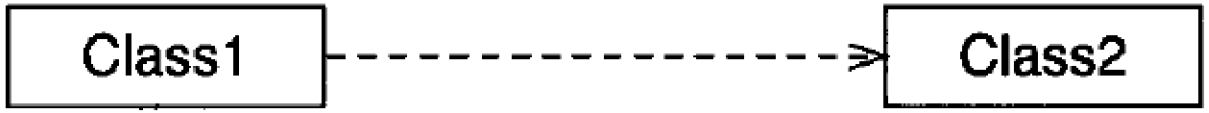


Aggregation and Composition example 2



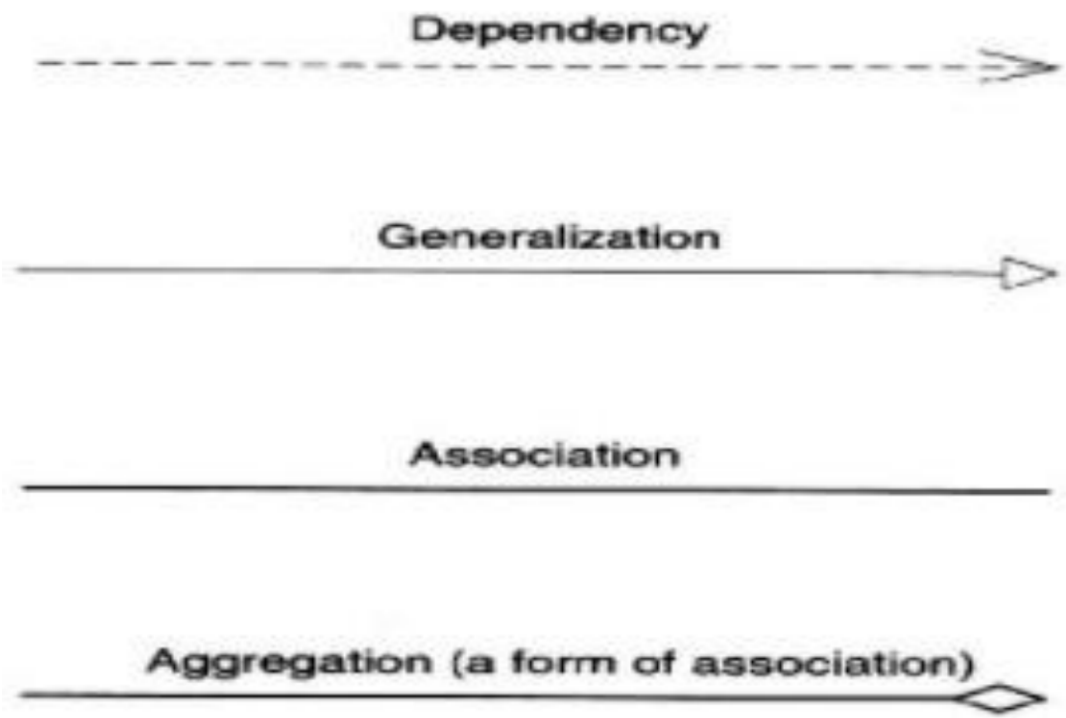
Dependency

- Dependency is a relationship between entities such that the proper operation of one entity depends on the presence of the other entity, and changes in one entity would affect the other entity.



Class1 depends on Class2.

summary


Dependency
Generalization
Association
Aggregation (a form of association)

Tutorial

- We are asked to build a system for keeping track of the time workers of an enterprise spend working on customer projects.
- We divide projects into activities, and the activities into tasks. A task is assigned to a worker, who could be a salaried worker or an hourly worker.
- Each task requires a certain skill, and resources have various skills at various level of expertise.

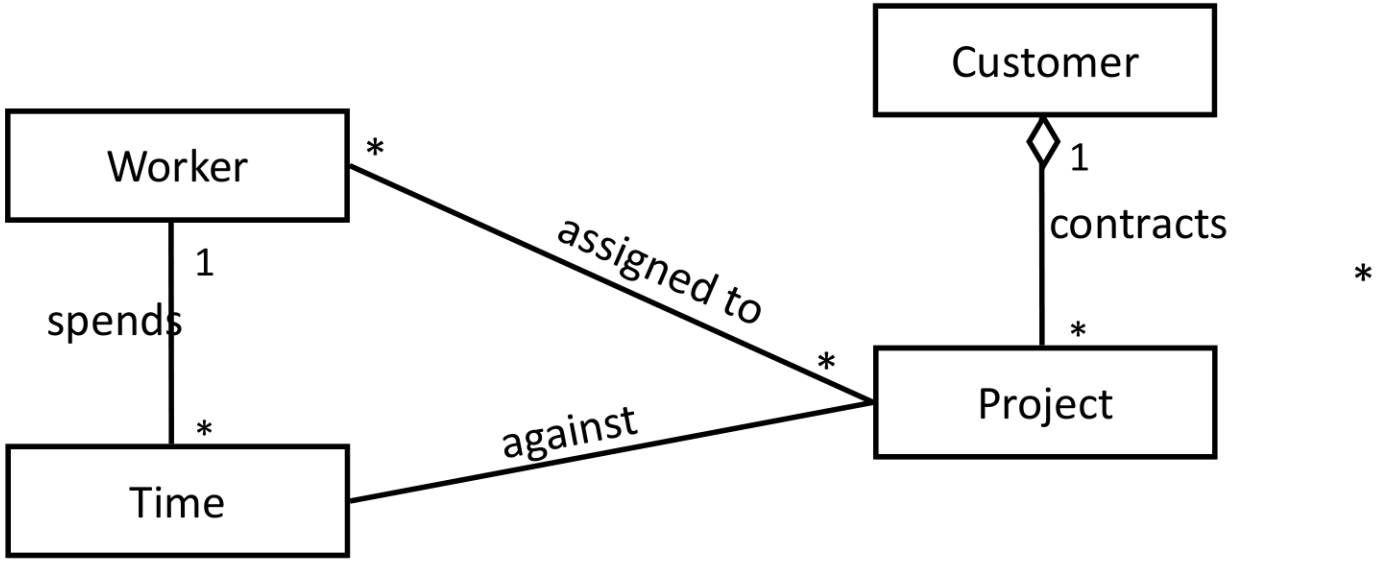
Task: Analyze the written requirements

Identifying classes and associations

- To identify objects and classes, perform a textual analysis to extract all nouns and noun phrases from the problem statement.
- To identify associations, extract the verbs of the problem statement

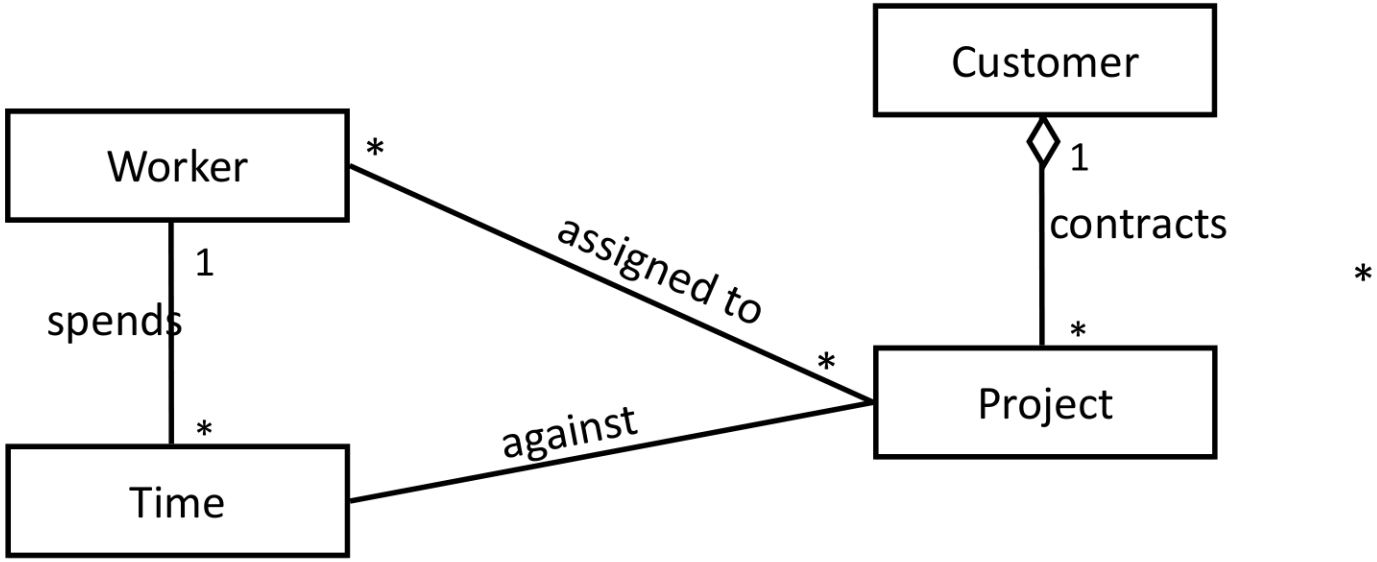
Example

- We are asked to build a system for keeping track of the time our workers spend working on customer projects.



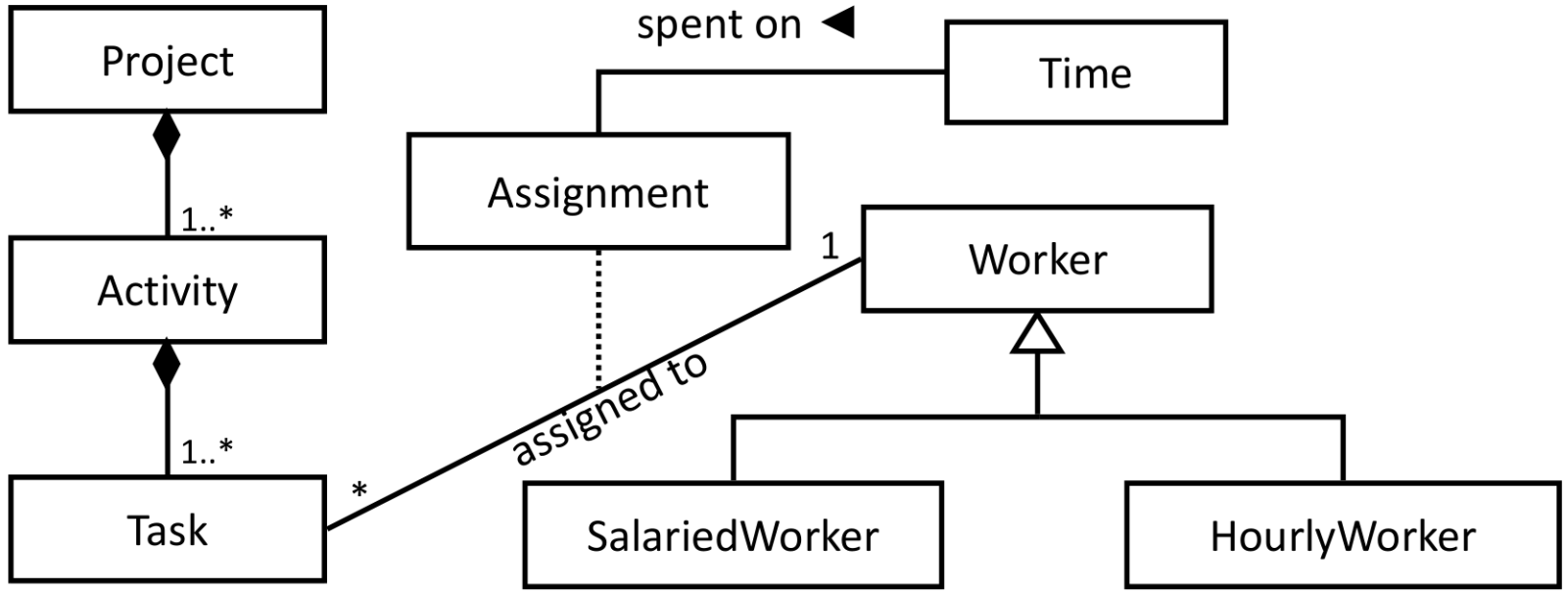
Example

- We are asked to build a system for keeping track of the time our workers spend working on customer projects.



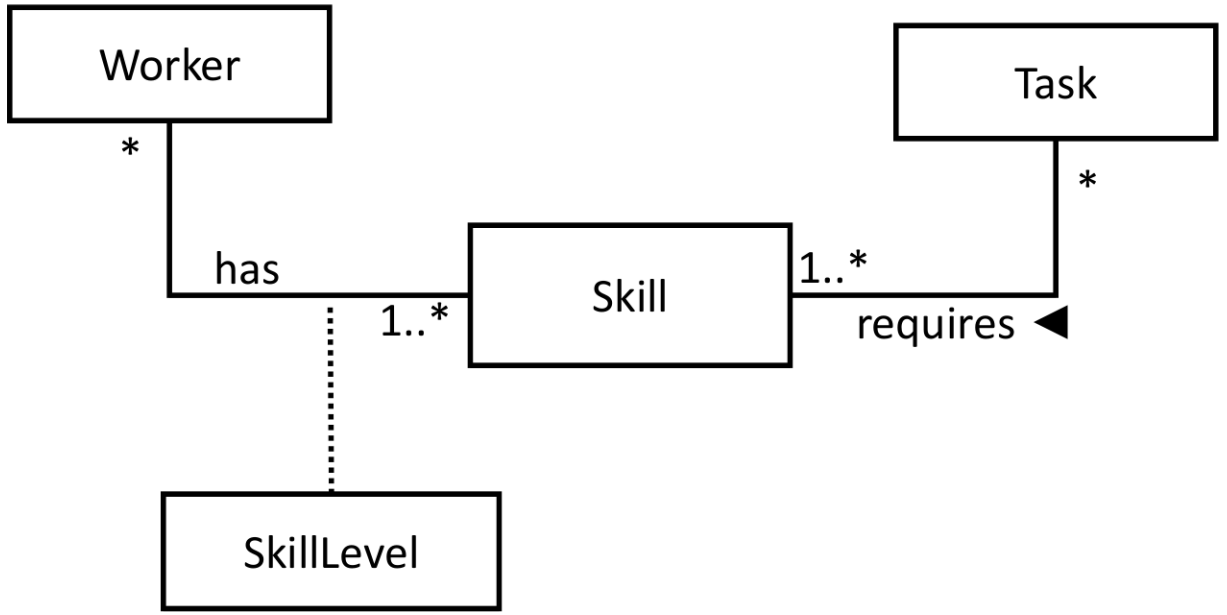
Example

- We divide projects ~~into~~ activities, and the activities ~~into~~ tasks. A ~~task is assigned~~ to a worker, who ~~could be a~~ salaried worker or an hourly worker.

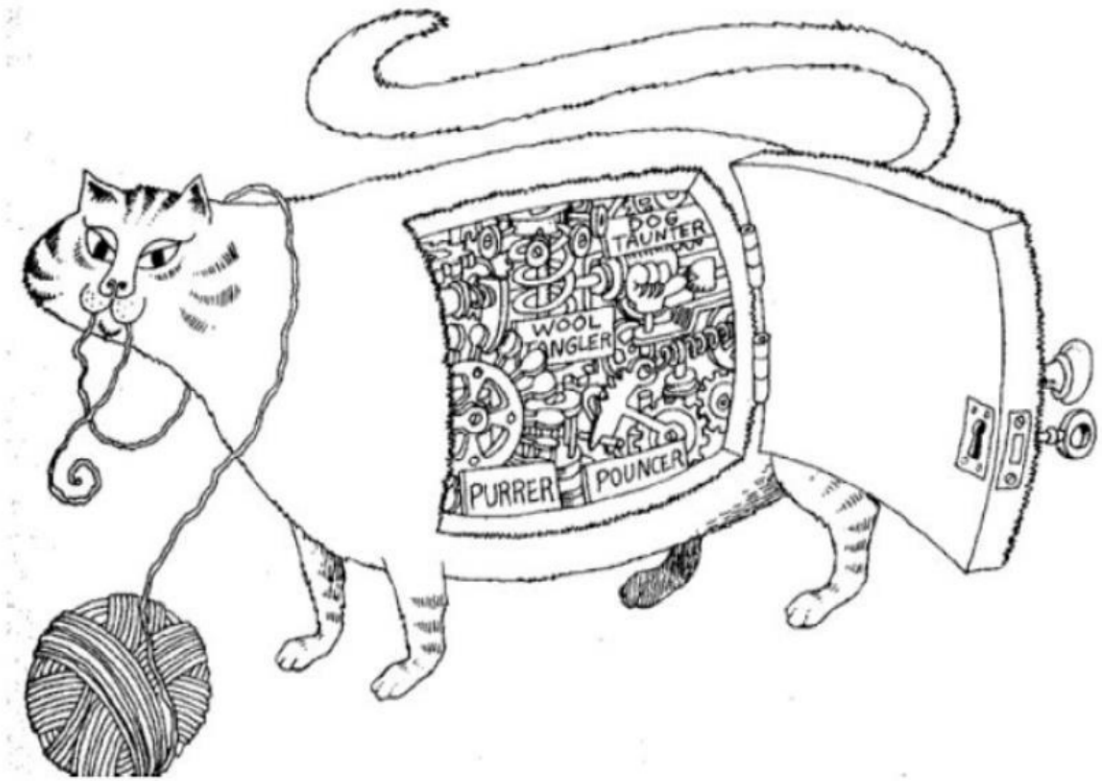


Example

Each task requires a certain skill, and workers have various skills at various level of expertise.



Principle of Encapsulation



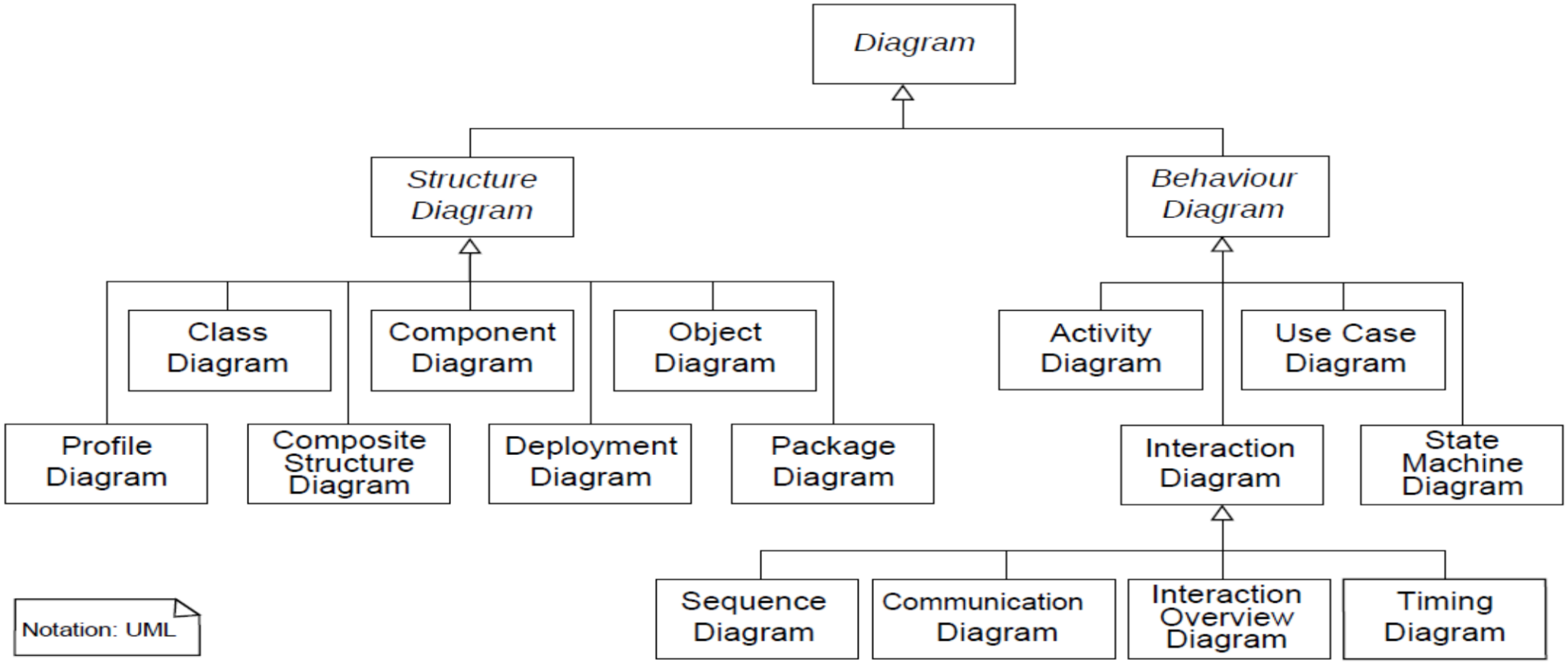
Encapsulation hides the details of the implementation of an object.

- Information Hiding: separates the external aspects of an object , that are accessible to other objects from internal implementation details.
- Just as a car hides its parts under the hood, an object encapsulates (i.e., hides) its instance variables and the statements of its methods.

Principle of Polymorphism

- Ability to interchange modules dynamically without affecting the system
- Refers to a contractual interface with multiple interchangeable implementation

UML different diagrams



UML most useful diagrams

- Use Case Diagrams: model relation of actors to system functions. Document who can do what in a system
- Interaction diagrams
  - ✓Sequence Diagrams: shows interactions between objects. Used to implement a use case
  - ✓Collaboration Diagrams: same as sequence but also shows context - i. e. objects and their relationships
- Activity diagram: models the sequential flow of activities i. e. action states
- Class diagram

Thank you for your attention