Lesson-7:Class Diagram (Object Modeling)

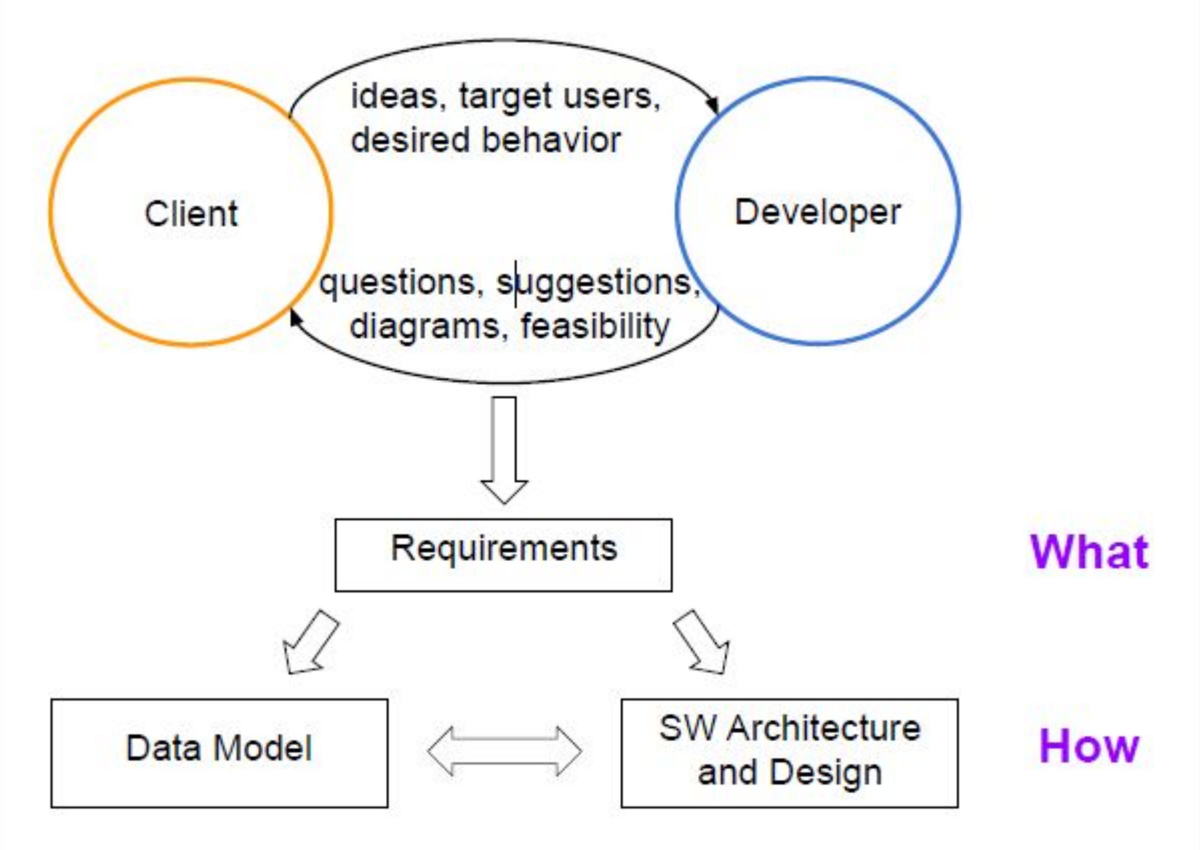
CS 438

Ali Aburas PhD

Today’s Goals

- UML
- Class Diagram

From Requirements to System Design


ideas, target users, desired behavior
Client
Developer
questions, suggestions, diagrams, feasibility
Requirements
What
Data Model
SW Architecture and Design
How

What is UML?

- Unified Modeling Language.
- Developed in the mid 90’s, improved since.
- Standardized notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
  - Use case diagrams
  - Component diagrams
  - Class and Object diagrams
  - Sequence diagrams
  - Statechart diagrams
  - ……

Are UML diagrams useful?

- **Communication**
  - Forward design (before coding)
    - Brainstorm ideas (on whiteboard or paper).
    - Draft and iterate over software design.
- **Documentation**
  - Backward design (after coding)
    - Obtain diagram from source code.

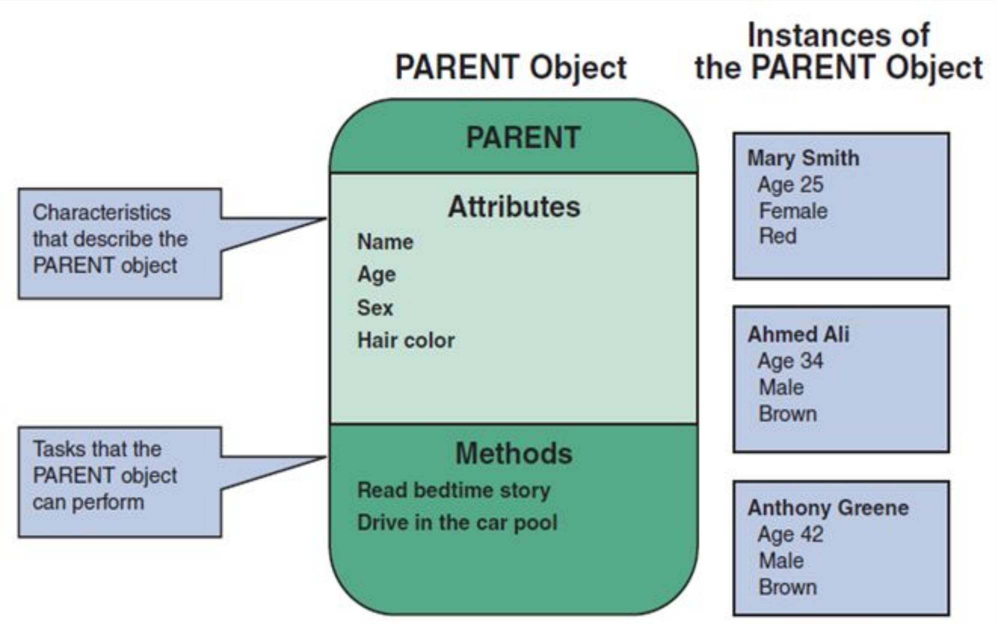
Design models

- When you use the UML to develop a design, you should develop two kinds of design model:

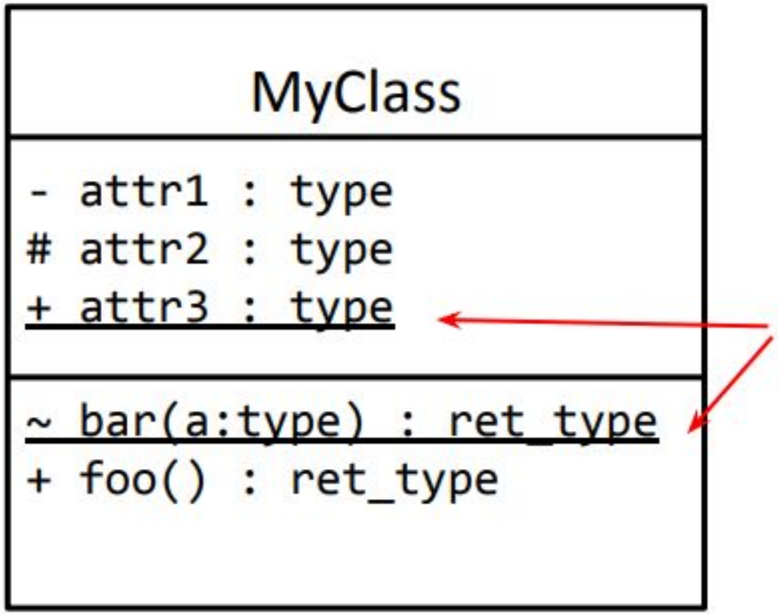
1. ***Structural models***, which describe the static structure of the system using object classes and their relationships. Important relationships that may be documented at this stage are generalization (inheritance) relationships, uses/used-by relationships, and composition relationships.
2. ***Dynamic models***, which describe the dynamic structure of the system and show the expected runtime interactions between the system objects. Interactions that may be documented include the sequence of service requests made by objects and the state changes triggered by these object interactions.

UML Class Diagram

- **UML Class Diagram** represents a static structural view of the system and it describes the classes and their structure, and their relationships among classes in the system
- **Class** is a description of a set of objects that share the same attributes, methods/operations, and relationships.
- **Object**
  - Entity from the real world.
  - Instance of a class



UML class diagram: basic notation



Name

Attributes

<visibility> <name> : <type>

Static attributes or methods are underlined

Methods

<visibility> <name>(<param>) : <return type>*

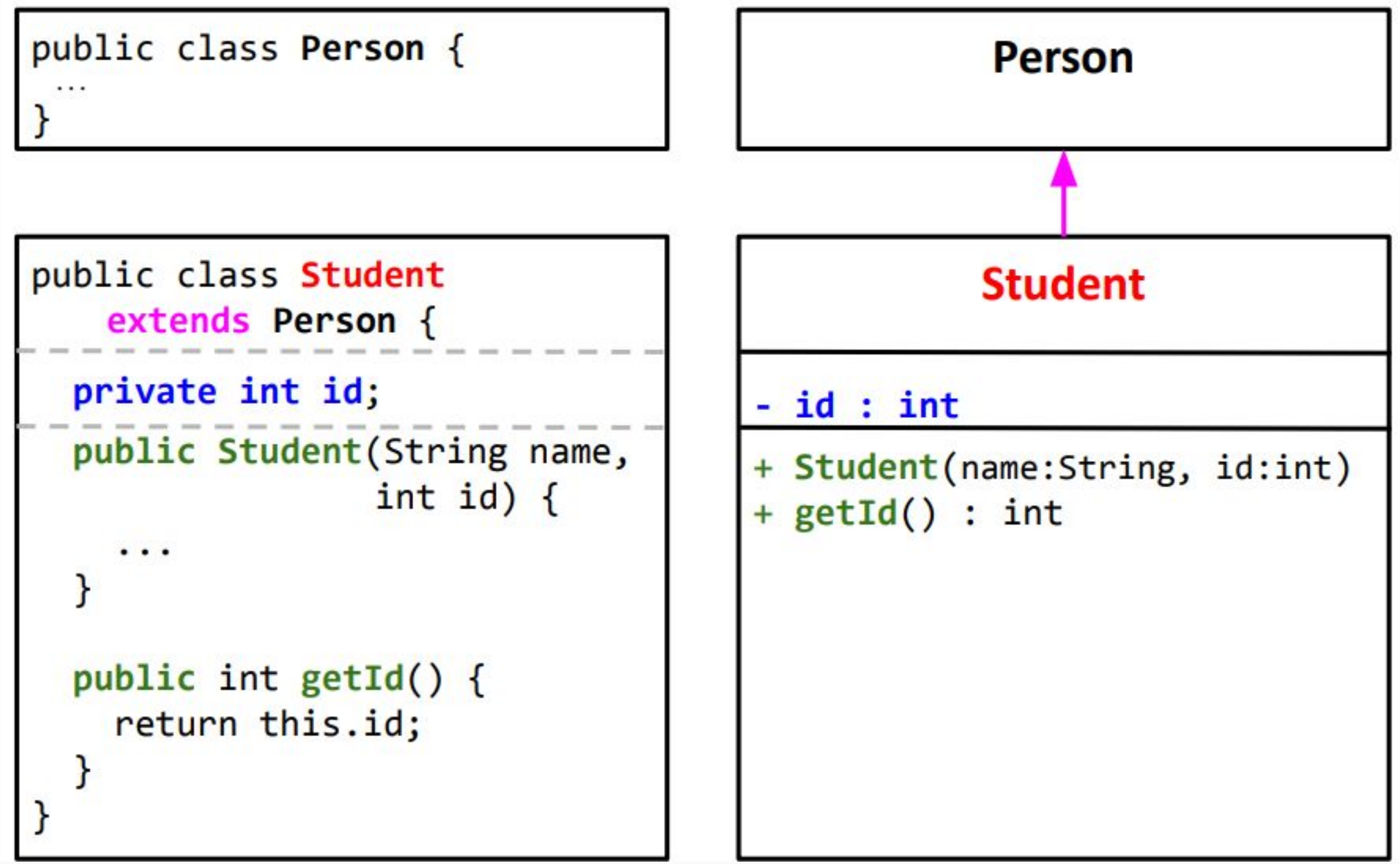
<param> := <name> : <type>

Visibility

- private
~ package-private
# protected
+ public

1. **Attributes**:describe the characteristics of an object. **Attributes** of an object are defined during the system development process
2. **Methods**: tasks or functions that the object performs when it receives a message, or command

UML class diagram: concrete example

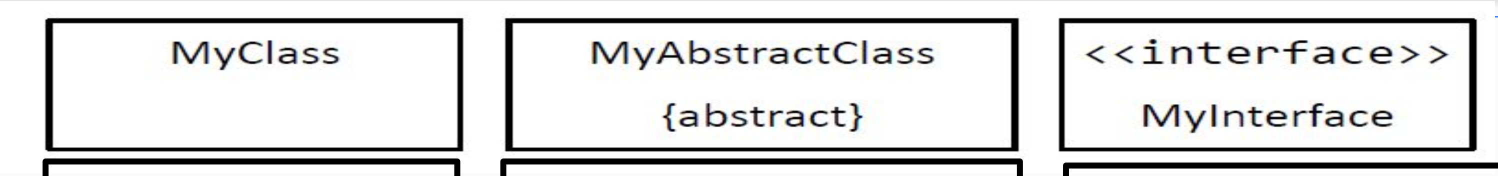


```java
public class Person {
  ...
}
```

```java
public class Student
    extends Person {
  private int id;
  public Student(String name,
                 int id) {
    ...
  }

  public int getId() {
    return this.id;
  }
}
```

Classes, abstract classes, and interfaces



```
public class MyClass
{
public void op() {
...
  }
public int op2() {
...
  }
}
```

```
public abstract class
MyAbstractClass {
public abstract void op();
public int op2() {
...
}
}
```

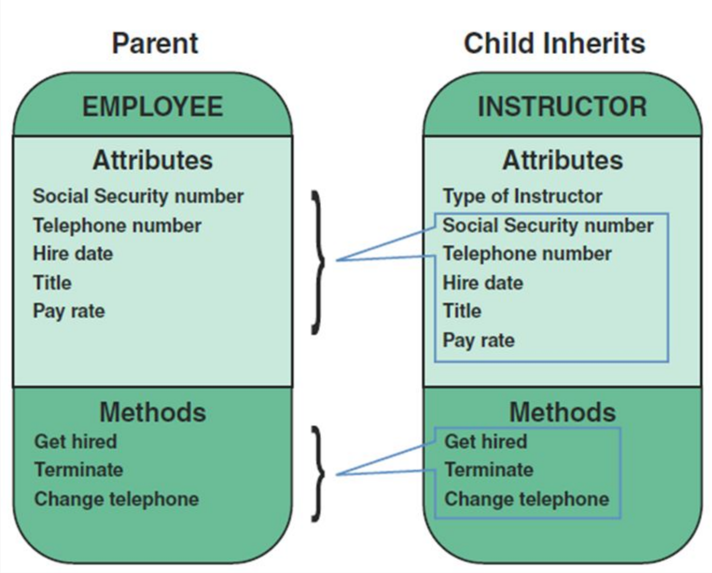
```
public interface MyInterface
{
public void
op();
public int
op2();
}
```

An **abstract class** is a class that cannot be instantiated on its own. Instead, it serves as a blueprint for other classes that extend it. Abstract classes can contain both **abstract methods** (methods without implementation) and **concrete methods** (methods with implementation). It is typically used when you want to define common functionality for a group of subclasses, but also leave some methods for those subclasses to implement

An **interface** class is like a contract for classes. It is a completely abstract class, meaning all of its methods are **abstract**. Interfaces are used to specify behaviors that a class must implement, but they don’t provide any concrete behavior themselves.

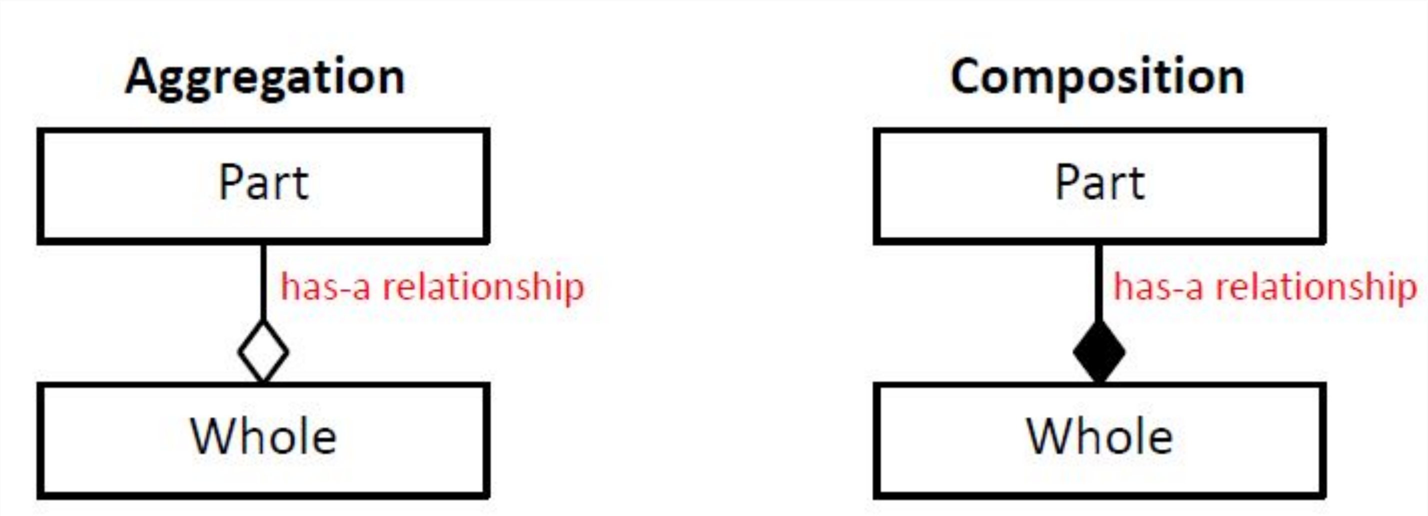
Relationships Among Objects and Classes

- Relationships
  - Enable objects to communicate and interact as they perform business functions and transactions
  - Describe what objects need to know about each other
- Inheritance
  - Enables an object to derive one or more of its attributes from another object



- An inheritance relationship exists between the **INSTRUCTOR** and **EMPLOYEE** objects.
- The **INSTRUCTOR** (child) object inherits characteristics from the **EMPLOYEE** (parent) class and can have additional attributes of its own.

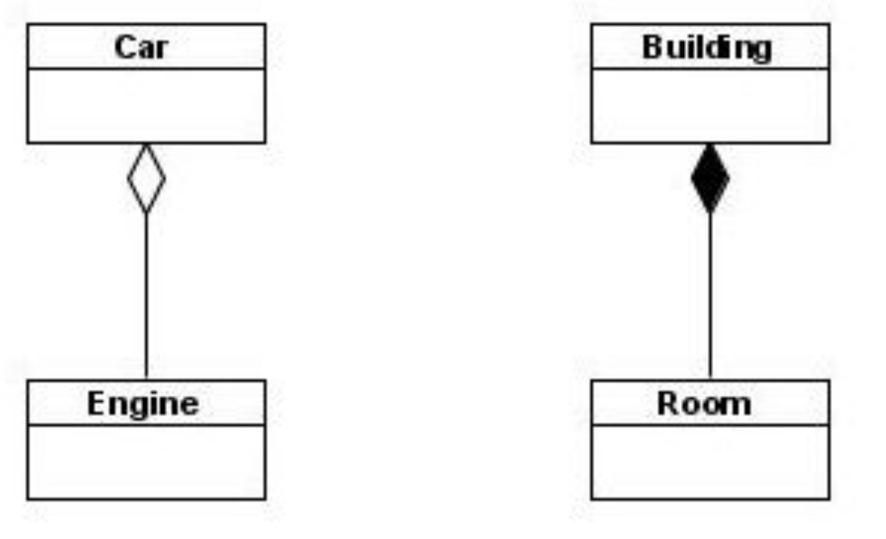
UML class diagram: Aggregation & Composition



- Existence of Part does not depend on the existence of Whole.
- Lifetime of Part does not depend on Whole.
- No single instance of whole is the unique owner of Part (might be shared with other instances of Whole).

- Part cannot exist without Whole.
- Lifetime of Part depends on Whole.
- One instance of Whole is the single owner of Part.

Quiz: Aggregation or Composition? Why?



In **aggregation**, the part may have an independent lifecycle, it can exist independently. When the whole is destroyed the part may continue to exist.

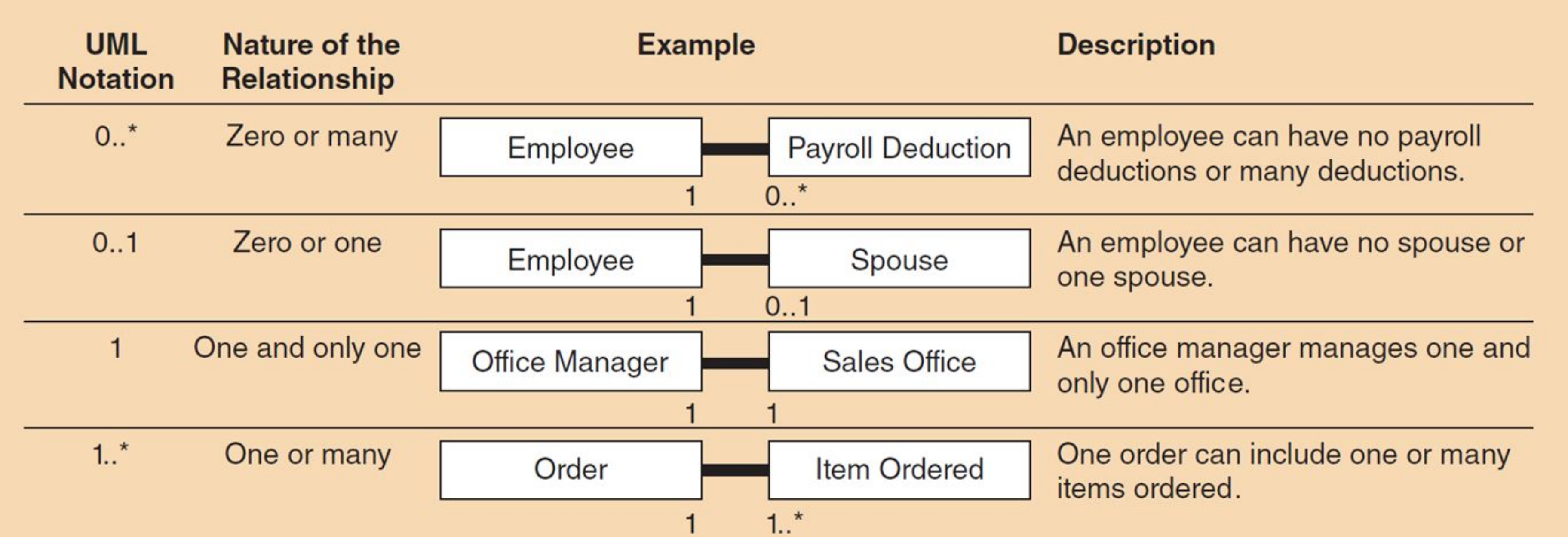
For example, a car has many parts. A part can be removed from one car and installed into a different car. If we consider a salvage business, before a car is destroyed, they remove all saleable parts. Those parts will continue to exist after the car is destroyed.

Composition is a stronger form of aggregation. The lifecycle of the part is strongly dependent on the lifecycle of the whole. When the whole is destroyed, the part is destroyed too.

For example, a building has rooms. A room can exist only as part of a building. The room cannot be removed from one building and attached to a different one. When the building ceases to exist so do all rooms that are part of it.

UML notations

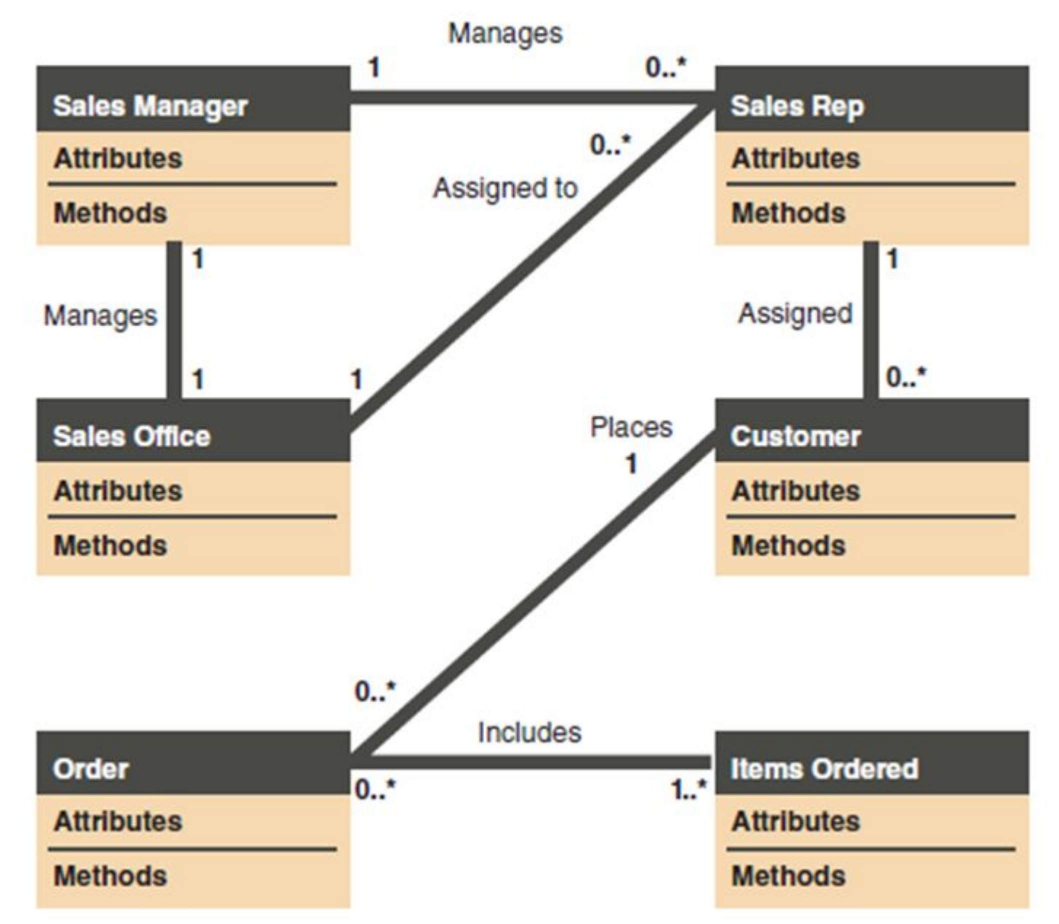
- Examples of UML notations that indicate the nature of the relationship between instances of one class and instances of another class



UML Notation	Nature of the Relationship	Example	Description
0..*	Zero or many	Employee 1 — 0..* Payroll Deduction	An employee can have no payroll deductions or many deductions.
0..1	Zero or one	Employee 1 — 0..1 Spouse	An employee can have no spouse or one spouse.
1	One and only one	Office Manager 1 — 1 Sales Office	An office manager manages one and only one office.
1..*	One or many	Order 1 — 1..* Item Ordered	One order can include one or many items ordered.

UML notations

Class diagram for a **sales** order use case (attributes and methods omitted for clarity)

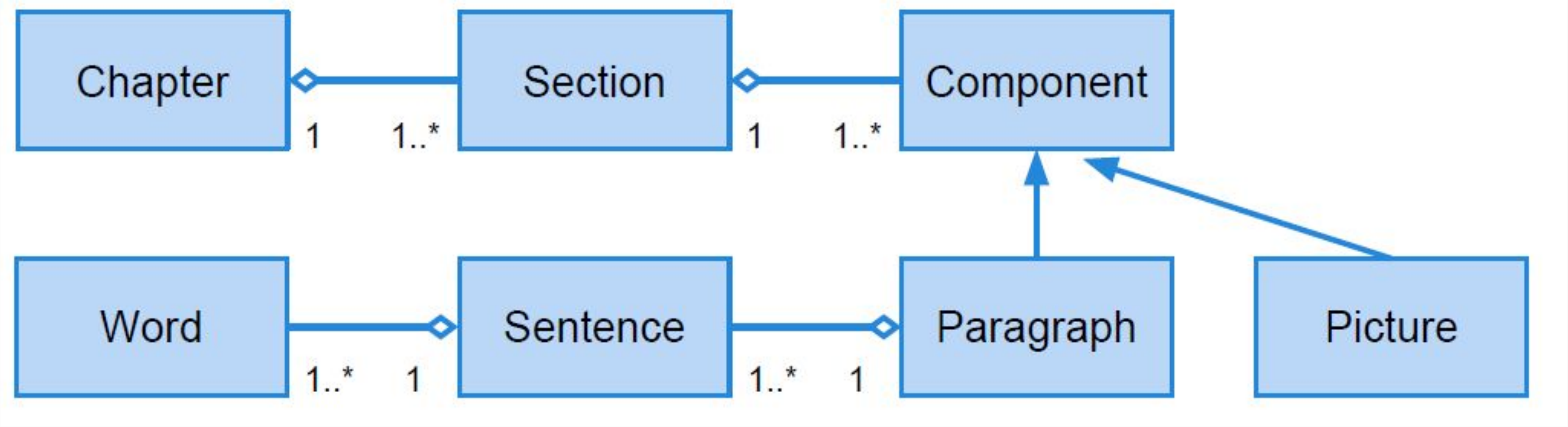


Class Diagram Activity

1. Draw a class diagram for a book chapter. A chapter comprises several sections, each of which comprises several paragraphs and/or figures. A paragraph comprises several sentences, each of which contains several words.

Class Diagram Activity

1. Draw a class diagram for a book chapter. A **chapter** comprises several **sections**, each of which comprises several **paragraphs** and/or **figures**. A paragraph comprises several **sentences**, each of which contains several **words**.

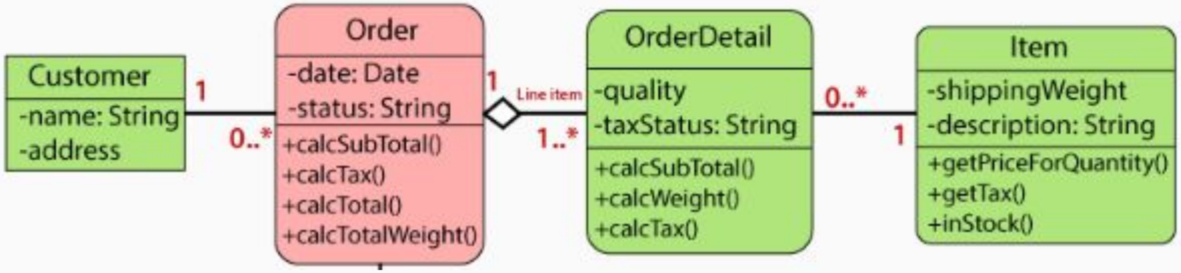


Class Diagram Activity

2. A customer can place an order. A customer can buy several items within a single order. The system calculates the tax, the total price, and the total shipping weight for an order. An order containsinformation (such as quantity, tax status) about the item. An item contains info like its shipping weight and description of the item.

Class Diagram Activity

2. A **customer** can place an **order**. A customer can buy several items within a single order. The system calculates the tax, the total price, and the total shipping weight for an order. An **order contains** information (such as quantity, tax status) about the **item**. An item contains info like its shipping weight and description of the item.



Properties of a good software design

- A good design **allows changes** to be made.
    - While also **protecting what works** from any side effects of those changes.
- **Design Attributes**
    - Simplicity
    - Modularity
        - Low Coupling
        - High Cohesion
        - Information Hiding
        - Data Encapsulation