SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Class Diagram

Dr. Ayesha Hakim

Introduction to UML

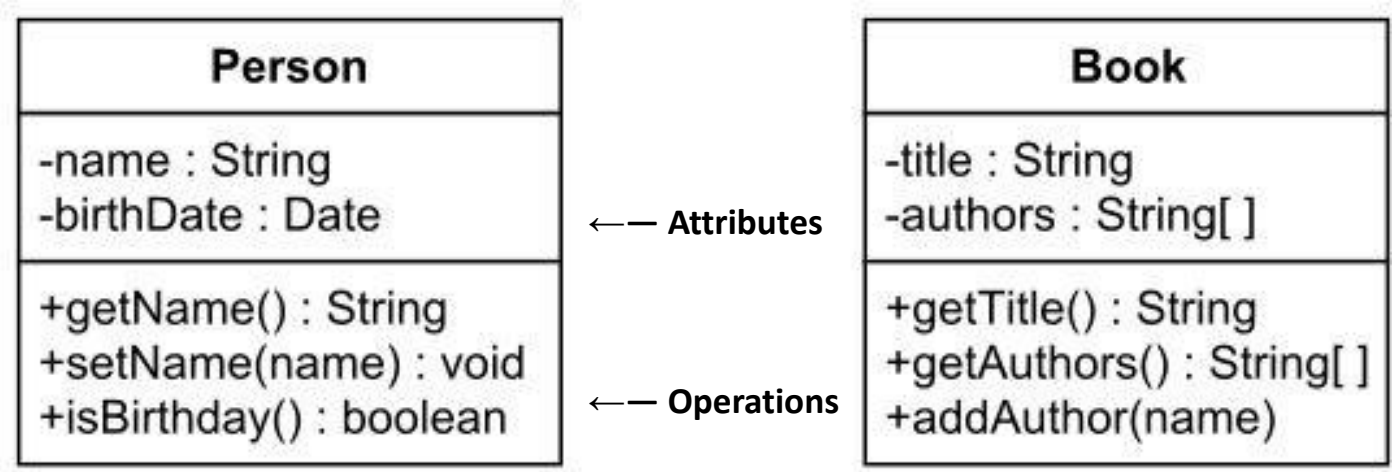
- UML (Unified Modeling Language) is a graphical language for **modeling the structure and behavior** of object-oriented systems.
- UML is widely used in industry to design, develop and document complex software.
- We will focus on creating **UML class diagrams**, which describe the internal structure of classes and relationships between classes.

Class Diagrams

- Class diagrams are the main building block in object-oriented modeling.
- A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations.
- A class diagram contains a rectangle for each class. It is divided into three parts.
  - The name of the class.
  - The names and types of the fields.
  - The names, return types, and parameters of the methods.

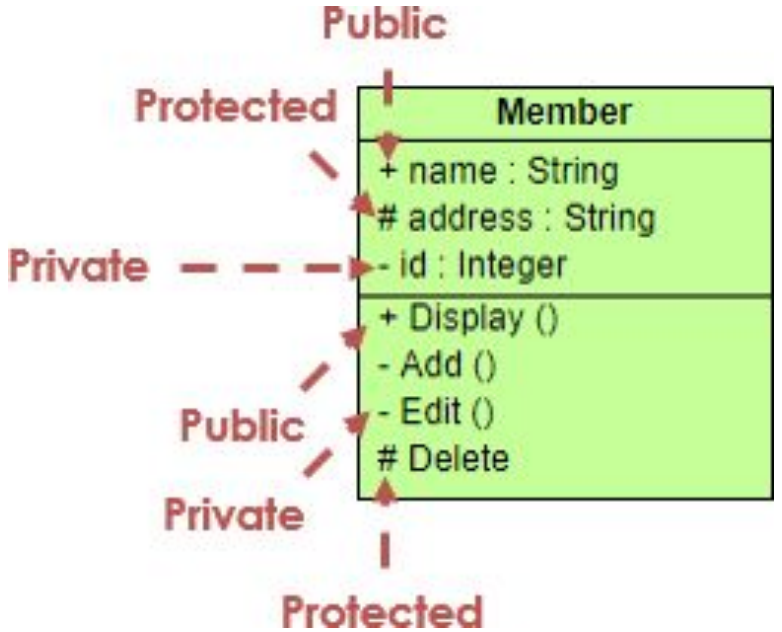
Example

- For example, a Person class and a Book class might be modeled like this:



Marks for UML-supported visibility types

Mark	Visibility type
+	Public
#	Protected
-	Private
~	Package



Visibility

Public visibility, denoted with a + sign, allows all other classes to view the marked information.
Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.

Example Explained

- The class diagram indicates that a Person object has private fields named name and birthDate, and that it has public methods named getName, setName and isBirthday.
- A Book object has private fields named title and authors.
- A Book object also has public methods named getTitle, getAuthors and addAuthor.

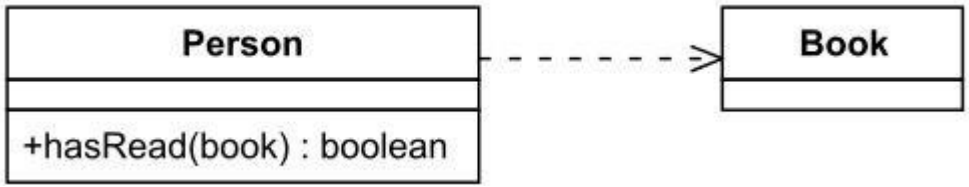
Use Relationships

- Often, objects and/or methods of one class use objects/methods from another class.
- For example, a person might read and/or own a book, and these relationships might be modeled in the UML diagram, so that they will be implemented in the corresponding program.
- UML class diagrams include the following **types of use-relationships**, in order from weakest to strongest.
  - Dependency
  - Unidirectional Association
  - Bidirectional Association

Dependency

- An object of one class might use an object of another class in the code of a method.
- If the object is not stored in any field, then this is modeled as a dependency relationship.
- For example, the Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database).

Dependency Example

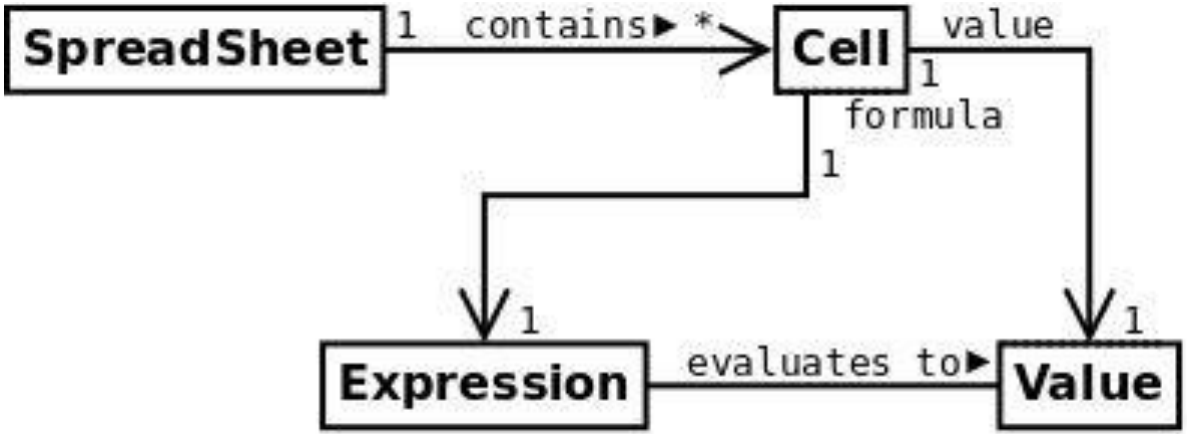


Unidirectional Association

- Associations represent logical connection or relationships between classes.
- Two classes are related, but only one class knows that the relationship exists.
- A unidirectional association is drawn as a solid line with an open arrowhead pointing to the known class.

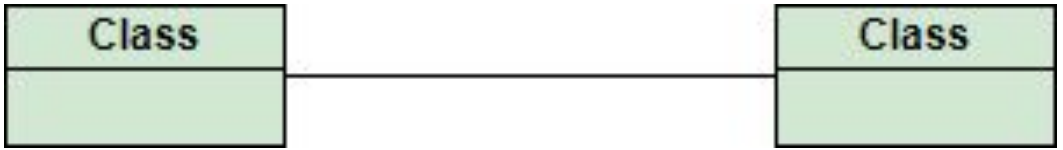
Unidirectional Association

- An object might store another object in a field.
- For example, people own books, which might be modeled by an owns field in Person objects.
- However, a book might be owned by a very large number of people, so the reverse direction might not be modeled.
- The *'s in the figure indicate that a book might be owned any number of people, and that a person can own any number of books.


SpreadSheet
1
contains
*
Cell
value
1
formula
1
1
1
Expression
evaluates to
Value

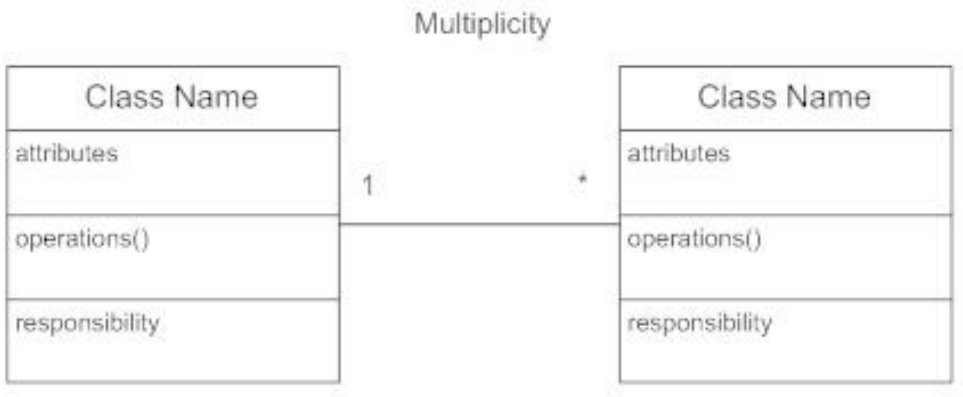
Bidirectional Association

- An association is a linkage between two classes. Associations are always assumed to be bi-directional; this means that both classes are aware of each other and their relationship, unless you qualify the association as some other type.
- A bi-directional association is indicated by a **solid line** between the two classes.



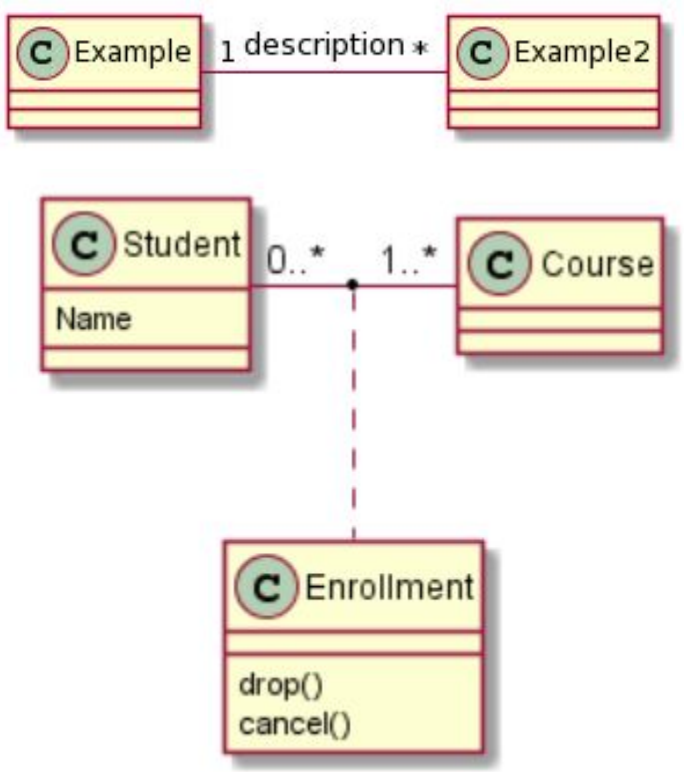
Multiplicity

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class through a given association. This relation is often expressed as a string showing the lower and upper bounds at the endpoints of a connection.



For **example**, one company will have one or more employees, but each employee works for just one company.

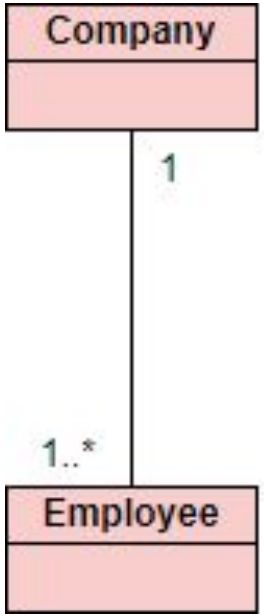
Multiplicity



The numbers are now many can be in the relationship. The first says that there can only be 1 Example class related to any number “*” of Example2 classes.

In the second example, there can be 0 or more students enrolled in 1 or more courses. The “*” in the “1..*” next to courses, could be replaced with say a “10” and that would mean that a student could not enroll in more than 10 courses.

Multiplicity



Multiplicities examples:

1	Exactly one, no more and no less
0..1	Zero or one
*	Many
0..*	Zero or many
1..*	One or many

Multiplicity

Let's look at an example from a hypothetical banking system. In our system, each customer must have at least one account at our bank and should not have more than five accounts in total. We could model this with the following UML association:

Multiplicity in a UML diagram

The text `1 .. 5` placed at the connection's end defines the range of possible accounts. To interpret this diagram, we would start reading from left to right :

> *One customer must have between one and five bank accounts*

Multiplicity

But that's not all: We can also describe the relation between bank accounts and customers. Let's say that every account must belong to at least one customer, but for joint accounts, it could also have two owners:

Multiplicity in the reverse direction

Here we had to place the lower and upper bound on the other end, and consequently, we must now read from right to left:

> *One* bank account must belong to one up to two customers*.

Multiplicity

Finally, we can combine both multiplicities into the same diagram to express both directions:

Both multiplicities combined in a single diagram

It's essential to keep in mind that this diagram combines the two expressions mentioned above into a single diagram, so the correct way to read it would be:

✓ One customer has one to five bank accounts **and** one bank account belongs to one or two customers.

Navigability

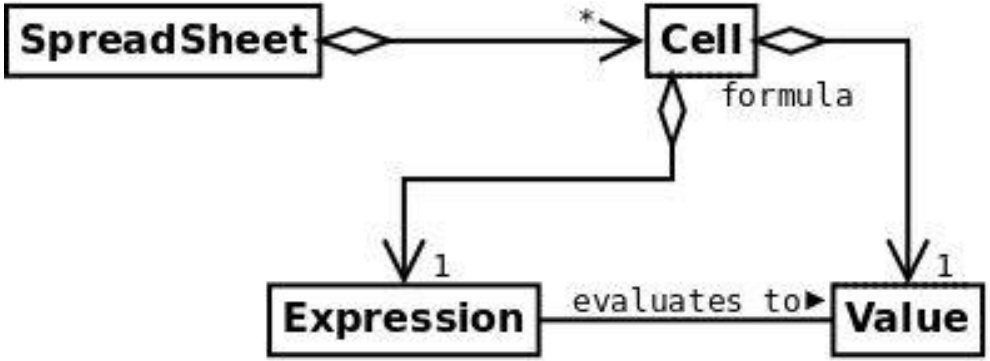
- Navigability arrows indicate whether, given one instance participating in a relationship, it is possible to determine the instances of the other class that are related to it.
- The diagram in the previous slide suggests that, given a spreadsheet, we can locate all of the cells that it contains, but that we cannot determine from a cell in what spreadsheet it is contained.
- Given a cell, we can obtain the related expression and value, but given a value (or expression) we cannot find the cell of which those are attributes.

Bidirectional Association

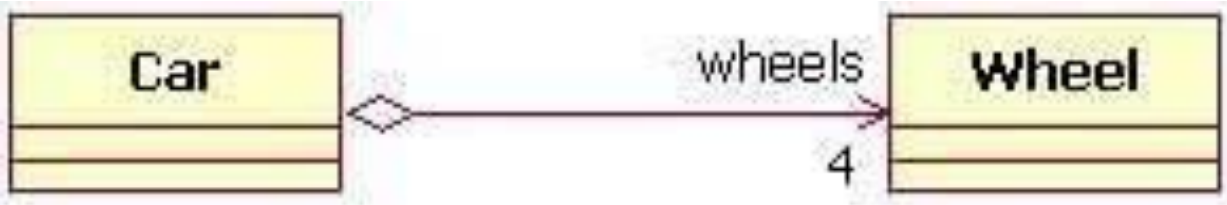
- Two objects might store each other in fields. For example, in addition to a Person object listing all the books that the person owns, a Book object might list all the people that own it.

Aggregation

- Denoted by an arrowhead drawn as an unfilled diamond, aggregation can be read as “is part of” or, in the opposite direction as “has a”.
- This diagram suggests that cells are part of a spreadsheet and that an expression and a value are each part of a cell.



Another example of aggregation

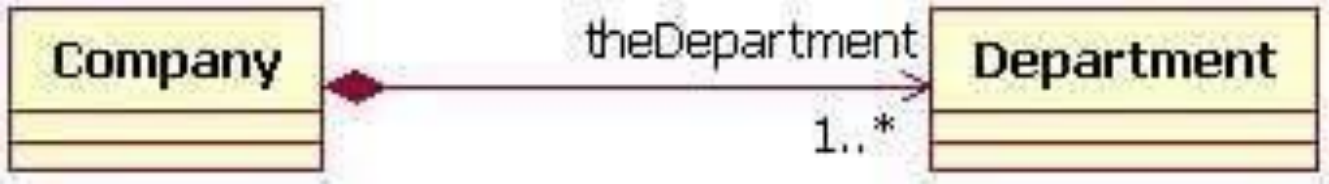


To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

Composition

- The composition relationship is a **special form of the aggregation** relationship, but the child (part) class's instance lifecycle is dependent on the parent (whole) class's instance lifecycle.
- It denotes a strong ownership between Class A, the whole, and Class B, its part.
- In the following, which shows a composition relationship between a Company class and a Department class, notice that the composition relationship is drawn like the aggregation relationship, but this time **the diamond shape is filled.**
- In the relationship modeled in the figure, a Company class instance will always have at least one Department class instance.
- Because the relationship is a composition relationship, when the Company instance is removed/destroyed, the Department instance is automatically removed/destroyed as well.
- Another important feature of composition is that the part class can only be related to one instance of the parent class (e.g. the Company class in our example).

Example of composition



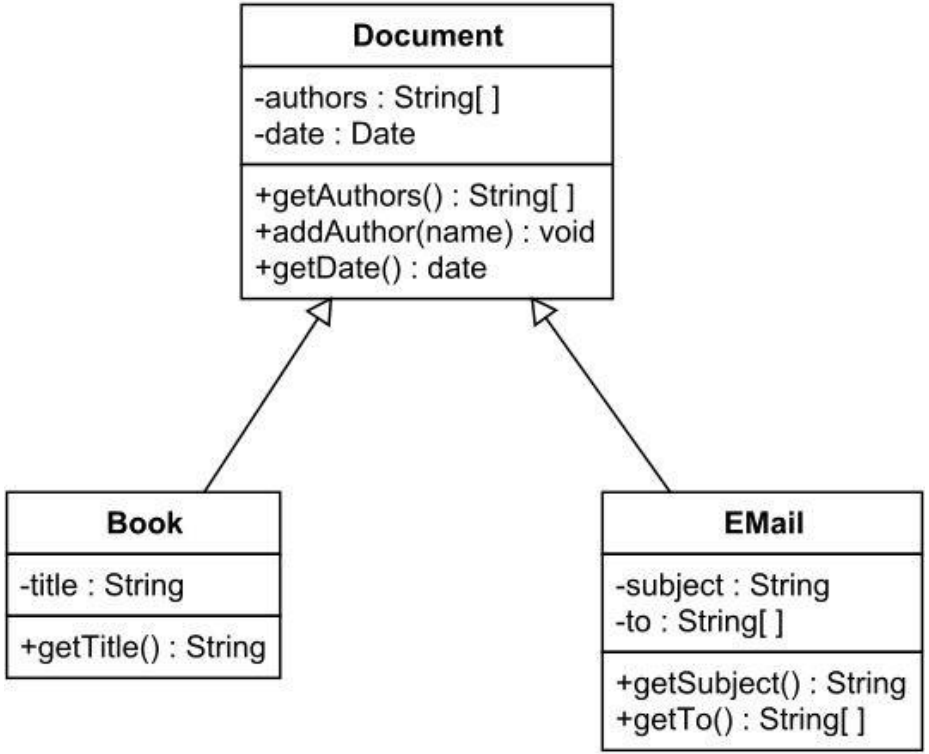
Illustrate composition with a **filled diamond**. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond ends in both composition and aggregation relationships point toward the "whole" class (i.e., the aggregation).

Inheritance Relationships: Generalization

- Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another.
- A class extends another class.
- For example, the Book class might extend the Document class, which also might include the Email class.
- The Book and Email classes inherit the fields and methods of the Document class (possibly modifying the methods), but might add additional fields and methods.

Generalization example

The child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.

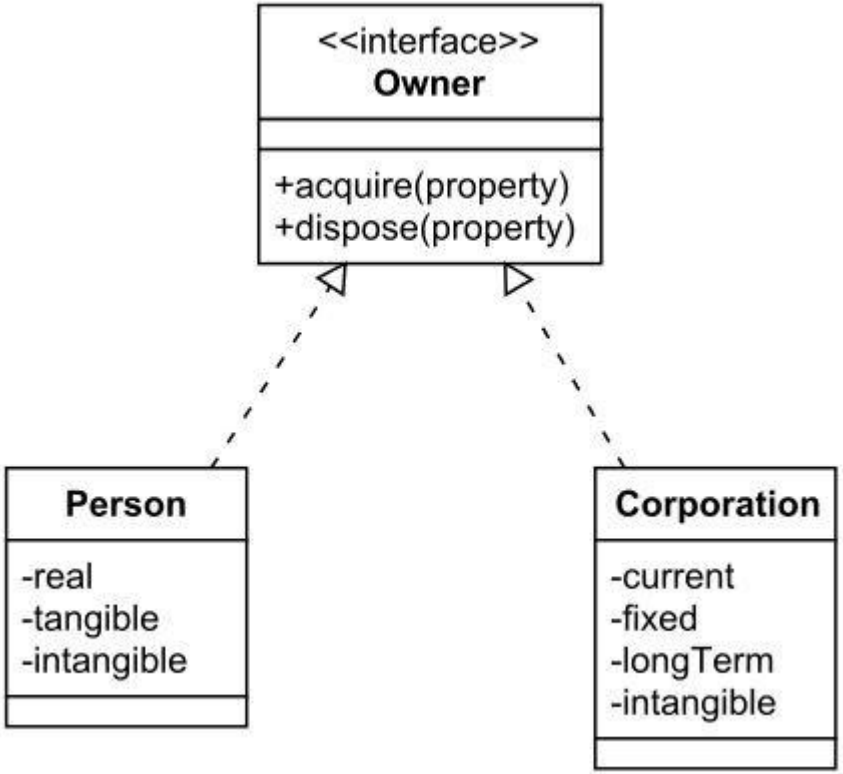


Inheritance Relationships: Realization

- A class implements an interface.
- For example, the Owner interface might specify methods for acquiring property and disposing of property.
- The Person and Corporation classes need to implement these methods, possibly in very different ways.

Realization Example

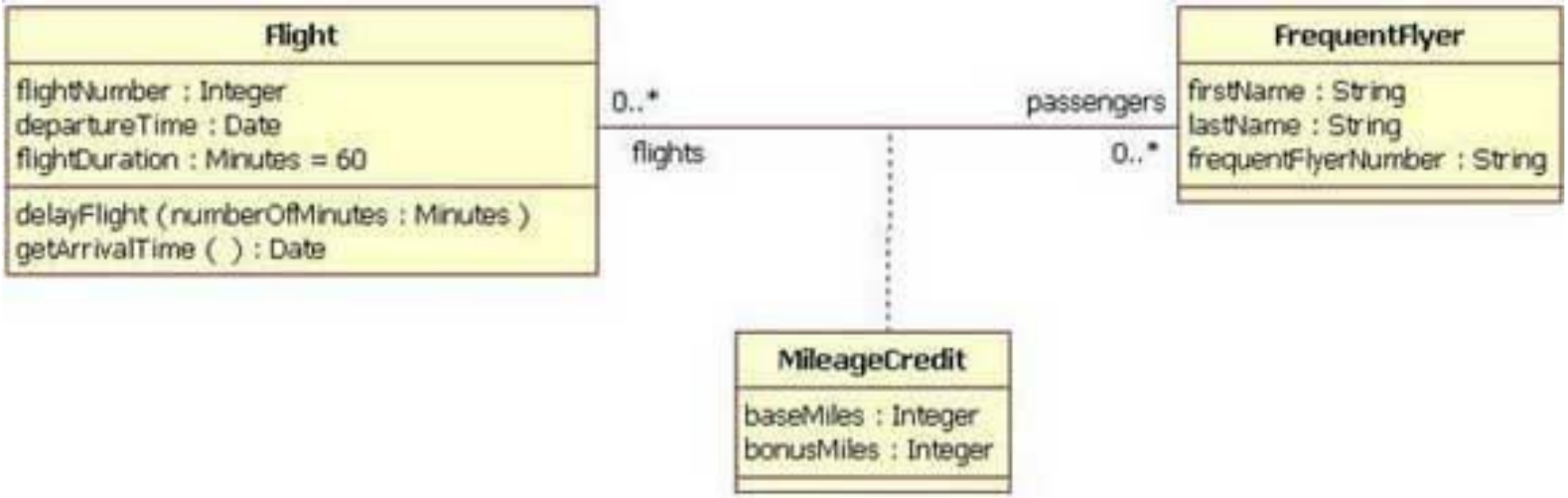
Realization denotes the implementation of the **functionality defined in one class by another class.** To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function.



Association Class

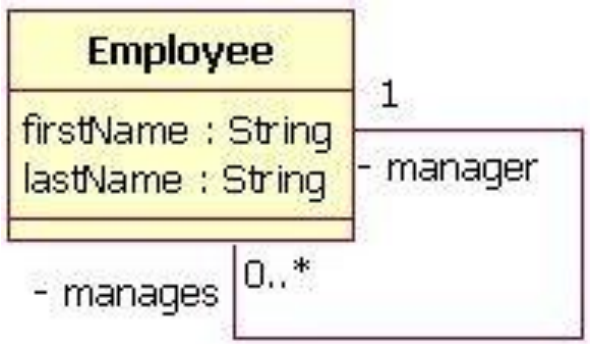
- In modeling an association, there are times when you need to include another class because it includes valuable information about the relationship.
- For this, you would use an association class that you tie to the primary association. An association class is represented like a normal class.
- The difference is that the association line between the primary classes intersects a dotted line connected to the association class.

Association class example



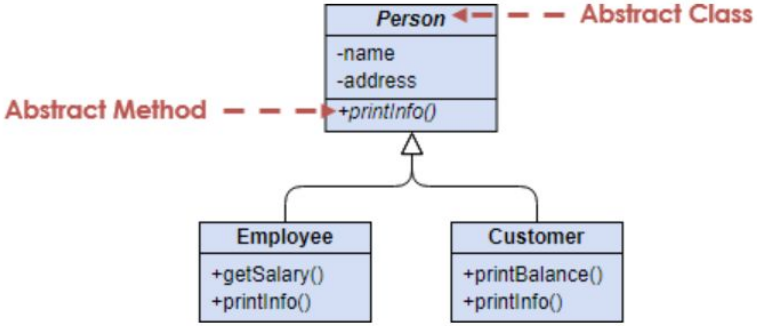
Reflexive associations

- A class can also be associated with itself, using a reflexive association.
- An Employee class could be related to itself through the manager/manages role.
- When a class is associated to itself, this does not mean that a class's instance is related to itself, but that an instance of the class is related to another instance of the class.



Abstract Classes and methods

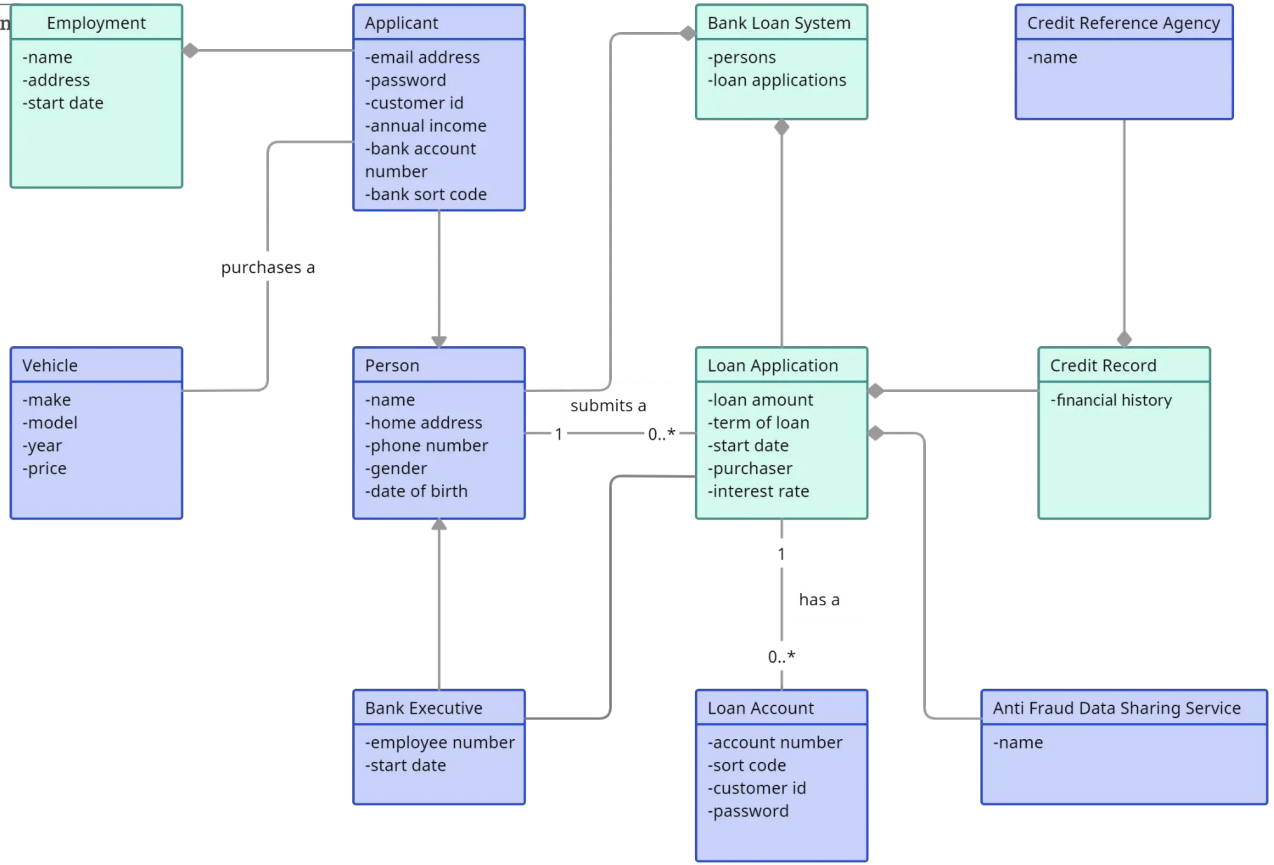
In an inheritance hierarchy, subclasses implement specific details, whereas the parent class defines the framework its subclasses. The parent class also serves a template for common methods that will be implemented by its subclasses.



The name of an **abstract Class** is typically shown in italics; alternatively, an abstract Class may be shown using the textual annotation, also called stereotype {abstract} after or below its name.

An **abstract method** is a method that do not have implementation. In order to create an abstract method, create a operation and make it italic.

Class Diagram : Example



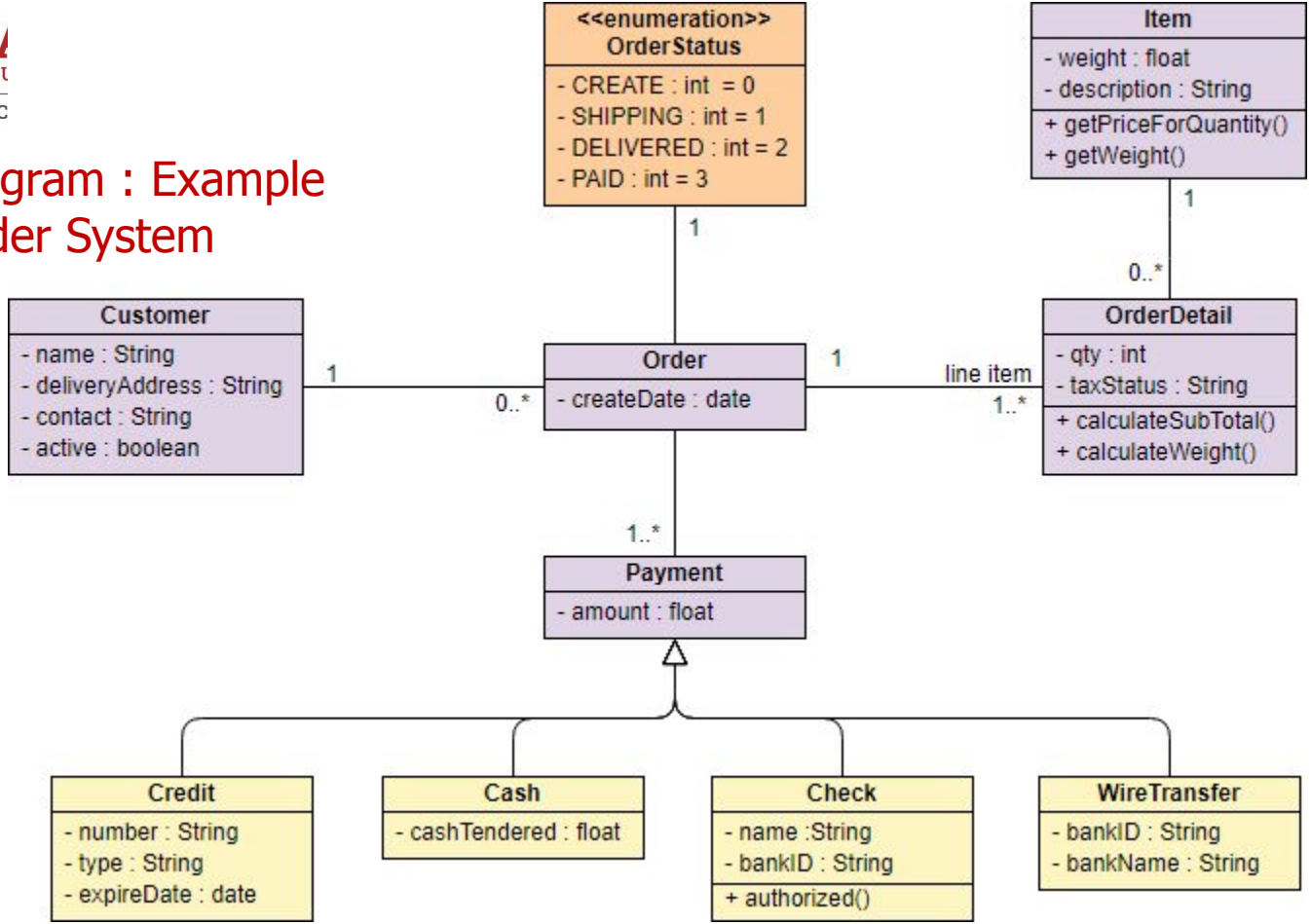
Example (contd.)

In the example, a class called “loan account” is depicted. Classes in class diagrams are represented by boxes that are partitioned into three:

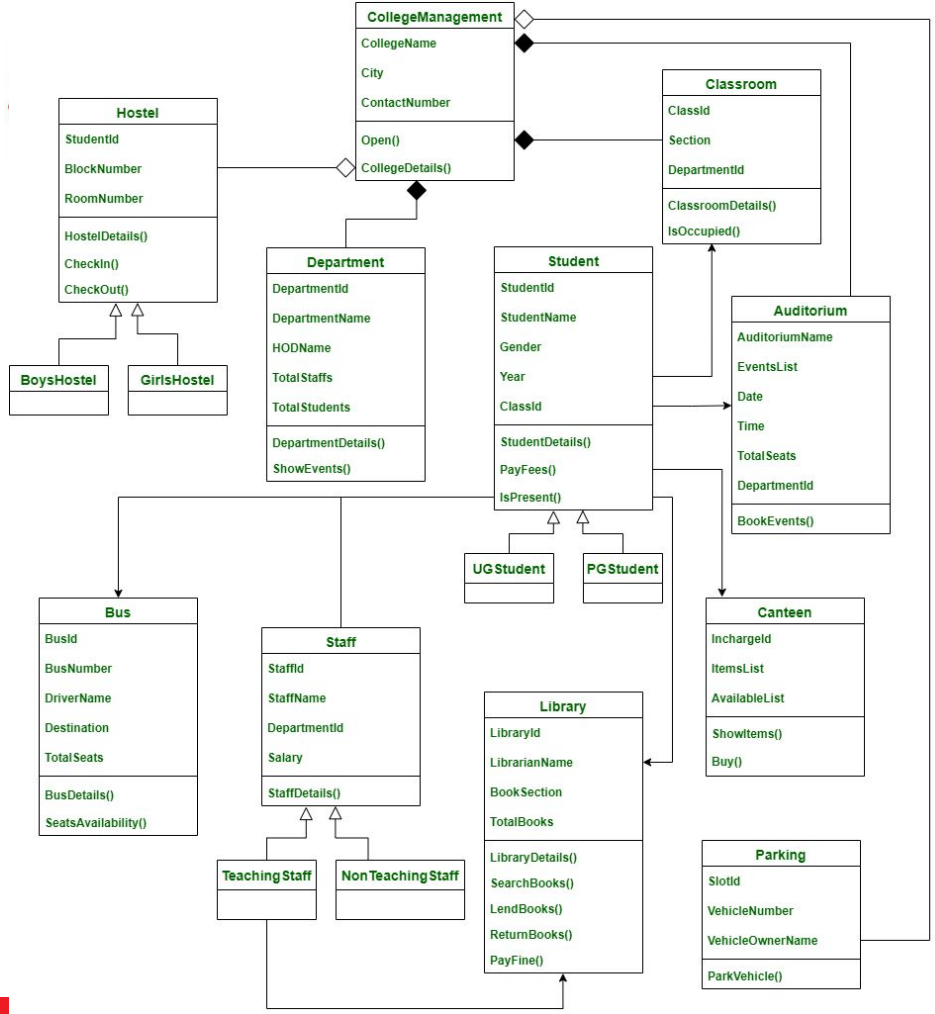
- The top partition contains the name of the class.
- The middle part contains the class’s attributes.
- The bottom partition shows the possible operations that are associated with the class.

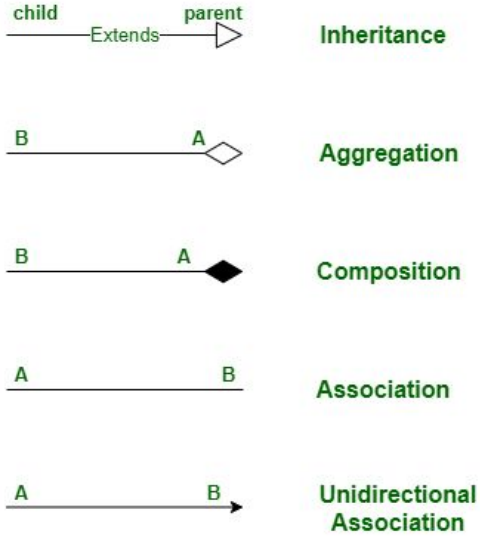
The example shows how a class can encapsulate all the relevant data of a particular object in a very systematic and clear way.

Class Diagram : Example
Sales Order System



College Management System





Converting UML Class Diagram to Java Code

References:

1. https://youtu.be/rtqQeYKHHNg?si=jQ38V6pmju7Wy3Qk
2. https://youtu.be/qOKK5yAnEc0?si=9dMtFaP3oeeJF1Qz

Practice Questions:
https://docs.google.com/document/d/1DzyhK4q3aCOCj83P4zSyvk7mwvu7Un9gchTu5XcThsA/edit?usp=sharing

Solutions: Shared as attachment

Questions?