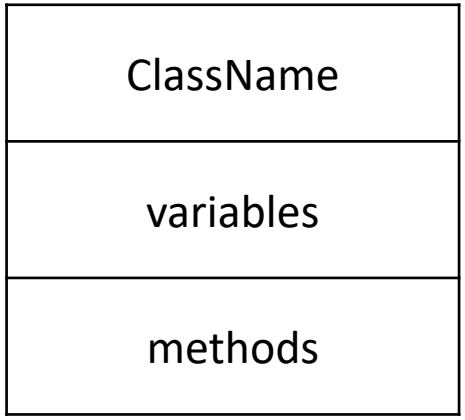
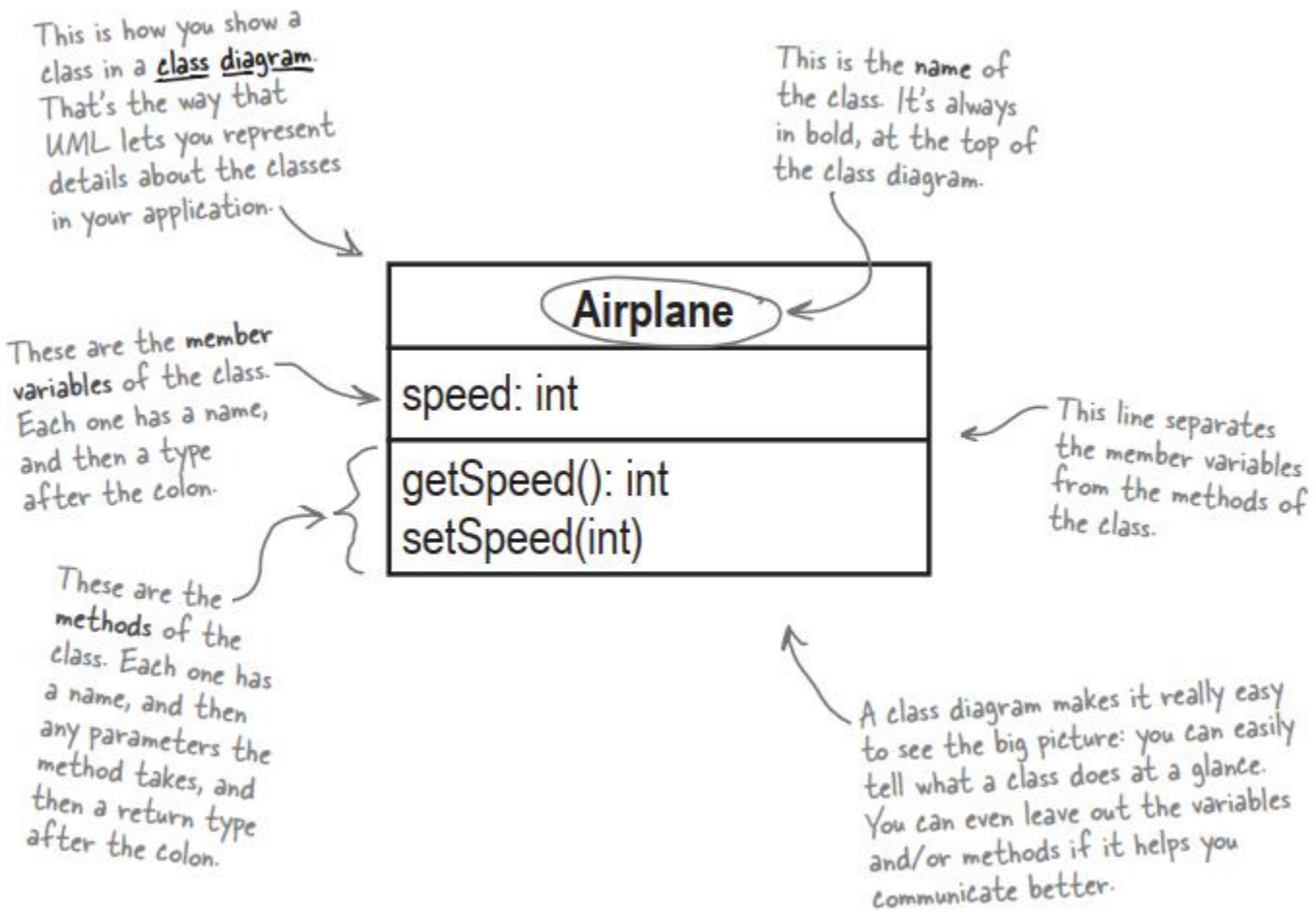
UML

the Unified Modelling Language

- It's pretty hard to look at 200 lines of code and focus on the big picture.
- UML, the Unified Modeling Language, is a language used to communicate just the details about your code and application's structure that other developers and customers need, without getting details that aren't necessary.
- It is graphical in nature, so that it is easy to visualize, understand and discuss the information presented in the diagram.

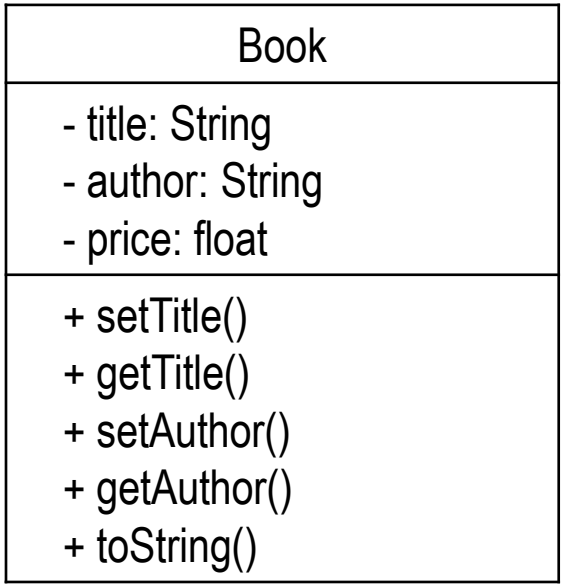
- There are four of the most common diagrams: class diagrams, object diagrams, sequence diagrams and package diagrams
- Classes are the basic components of any OO software system and UML class diagrams provide an easy way to represent these.
- A class consists of :
  - A unique class name
  - A list of member variables / attributes
  - A list of methods





- Quiz: can you write the basic skeleton (rough code) for this Airplane class?

- For attributes and methods visibility modifiers are shown (+ for public access, – for private access).
- The data type of instance variables are written behind the variables after a colon (:)
- Instance variables normally being kept private and methods normally made public



- UML allows us to suppress any information we do not wish to highlight in our diagrams – this allows us to suppress irrelevant detail and bring to the readers attention just the information we wish to focus on. Therefore the following are all valid class diagrams
- For example, a class diagram without access modifiers/data type/instance variables/methods is available.
- In many cases, class diagrams omit the public/private notations because they're not needed for clear communication.

- Class diagrams are just a way to communicate the basic details of a class's variables and methods. It also makes it easy to talk about code without forcing you to understand the language for implementation
- UML is a precise diagramming notation that will allow program designs to be represented and discussed
- By using a standard like UML, we can all speak the same language and be sure we're talking about the same thing in our diagrams.

- UML syntax
- As UML diagrams convey precise information, there is a precise syntax that should be followed.
- Instance variables should be shown as: *visibility name* : *type multiplicity*
- Visibility is one of:
  - '+' public
  - '-' private
  - '#' protected
  - '~' package
- Multiplicity is one of:
  - 'n' exactly n
  - '*' zero or more
  - 'm…n' between m and n

- `- customerId : int [1]`
  - a private variable customerId is a single int value.
  - this would often be shown as - customerId : int
- `#itemCodes : String [1..*]`
- A protected variable itemCodes is one or more String values
- `validCard : boolean`
- A variable validCard, of unspecified visibility, has unspecified multiplicity

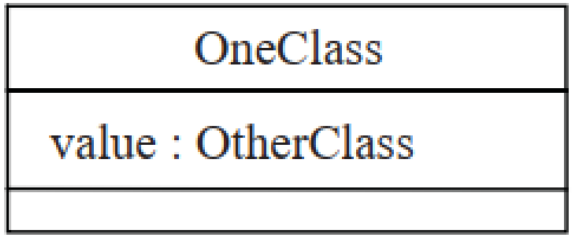
- Operations also have a precise syntax and should be shown as:

$$visibility\ name\ (arg1 : type1, arg2 : type2) : returntype$$

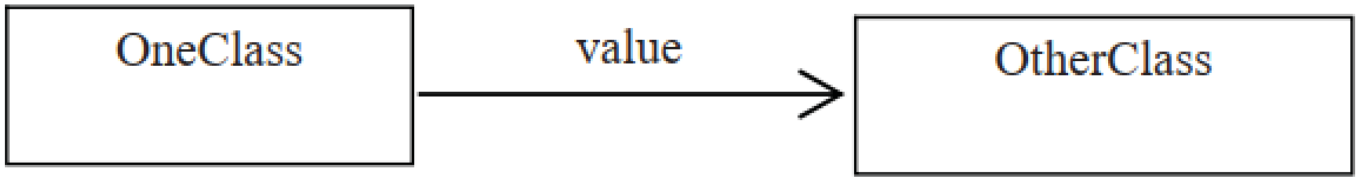
- **+ addName (newName : String) : boolean**
- This denotes a public method ‘addName’ which takes one parameter ‘newName’ of type String and returns a boolean value

- Exercise
- Draw a diagram to represent a class called ‘BankAccount’ with a private variable balance (this being a single integer) and a public method depositMoney() which takes an integer parameter, ‘deposit’ and returns a boolean value. Fully specify all of this information on a UML class diagram.

- All Java programs will be made up of many classes and classes will relate to each other – some classes will make use of other classes.
- Class diagrams can denote relationships between classes (called ***association***). For example



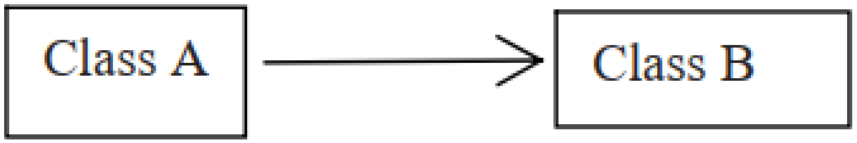
- We could denote exactly the same information by the diagram below



- Types of Association
- There are various different types of association denoted by different arrows:
  - Dependency
  - Simple association
  - Bidirectional association
  - Aggregation and Composition

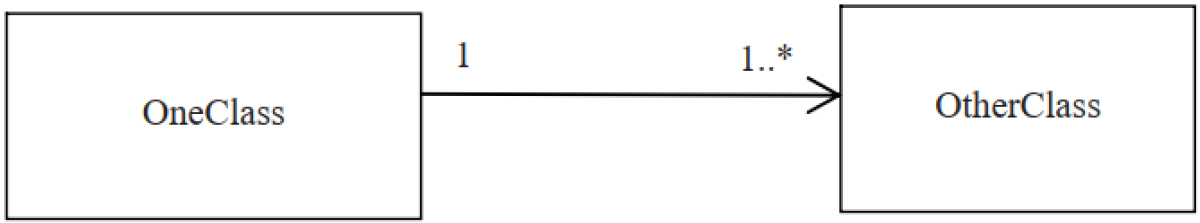
- Dependency
- The most unspecific relationship between classes
- Class A in some way uses facilities defined by Class B
- Changes to Class B may affect Class A
- E.g. Class A has a method which is passed a parameter object of Class B, or uses a local variable of that class

- Simple association
- Class A 'uses' objects of Class B
- Typically Class A has an attribute (instance variable) of Class B
- A Class A object can access the Class B object(s) with which it is associated. The reverse is not true – the Class B object doesn't 'know about' the Class A object
- A simple association typically corresponds to an instance variable in Class A of the target class B type.



- Strictly we could use an association when a class has a String instance variables – but we would not do this because the String class is part of the Java API.

- Additionally we can show multiplicity at both ends of an association

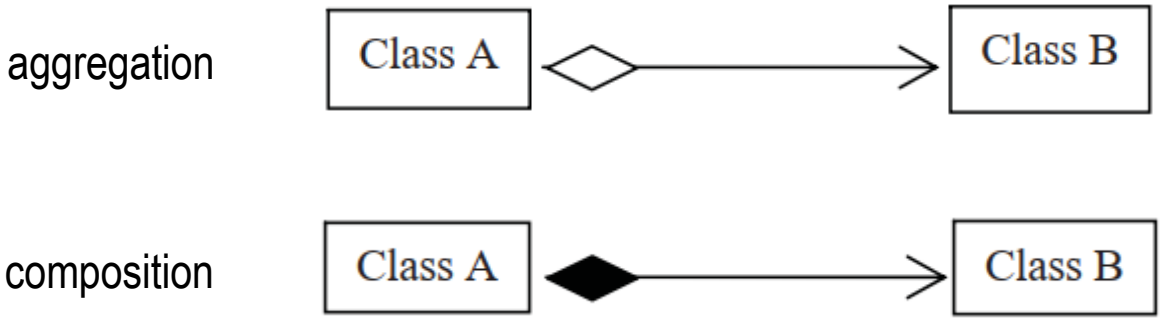


- This implies that 'OneClass' maintains a ***collection*** (e.g. array) of objects of type 'OtherClass'

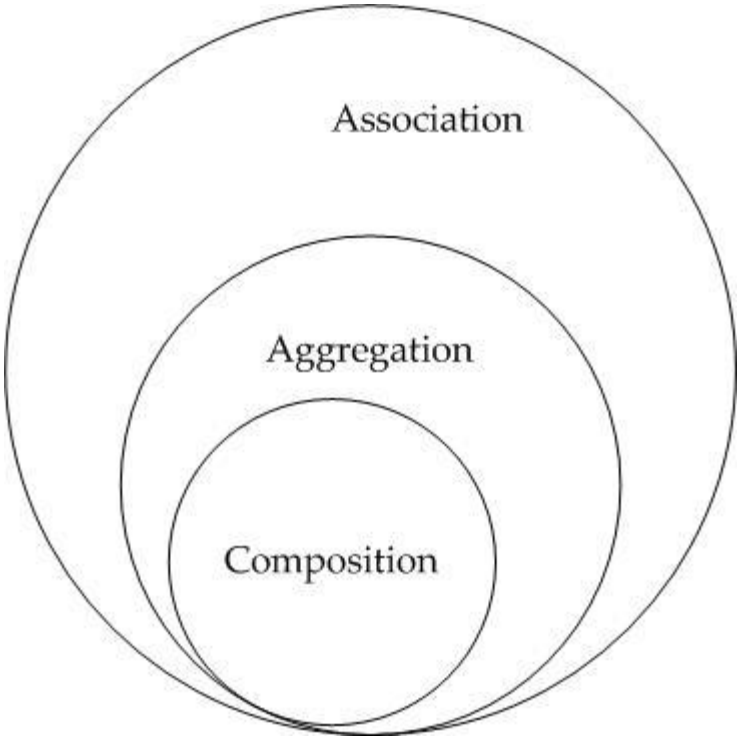
- Bidirectional association
- Each refers to the other class
- A Class A object can access the Class B object(s) with which it is associated
- Object(s) of Class B ‘belong to’ Class A
- Implies reference from A to B
- A Class B object can access the Class A object(s) with which it is associated

- An example of a bidirectional association may between a ‘Degree’ and ‘Student’. That is, given a Degree we may wish to know which Students are studying on that Degree. Alternatively starting with a student we may wish to know the Degree they are studying.
- As many students study the same Degree at the same time, but students usually only take one Degree

- Aggregation and composition
- Aggregation/Composition denotes a situation where Object(s) of Class A 'HAS-A' Class B
- Composition implies a stronger belonging relationship (i.e. as Class A vanishes, Class B vanishes as well)
- For example, a car has tires, but at the garage tires may be removed and placed on a rack to be repaired. -> aggregation
- A house has bedrooms, a living room, and a kitchen. If the house is demolished, they are also deleted -> composition
- So in representing composition, sometimes we say A 'OWNS' B

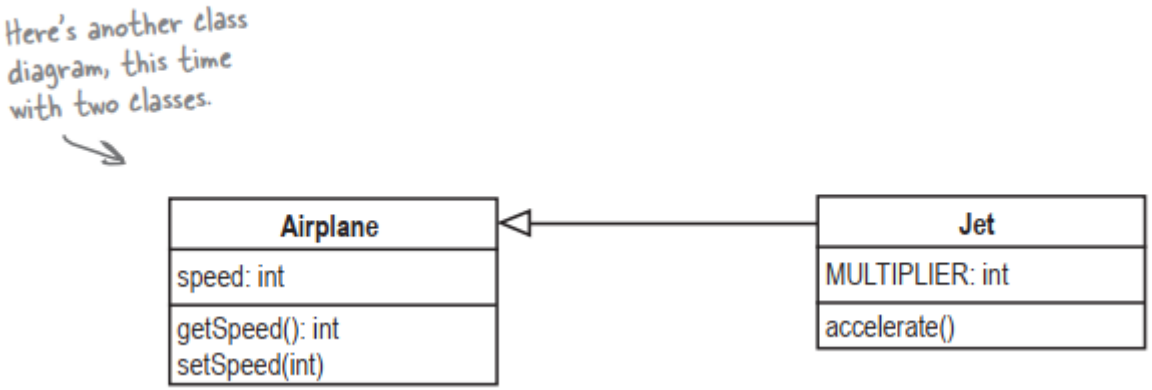


- Association is generalized concept of relations. It includes both Composition and Aggregation
- Don’t forget that using local variables of a specific class is also a kind of association.

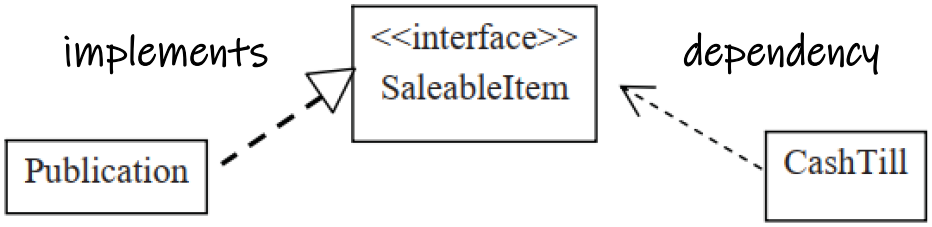


- Class diagrams can also denote following relations:
- Inheritance
- Interface
- Keywords
- Notes

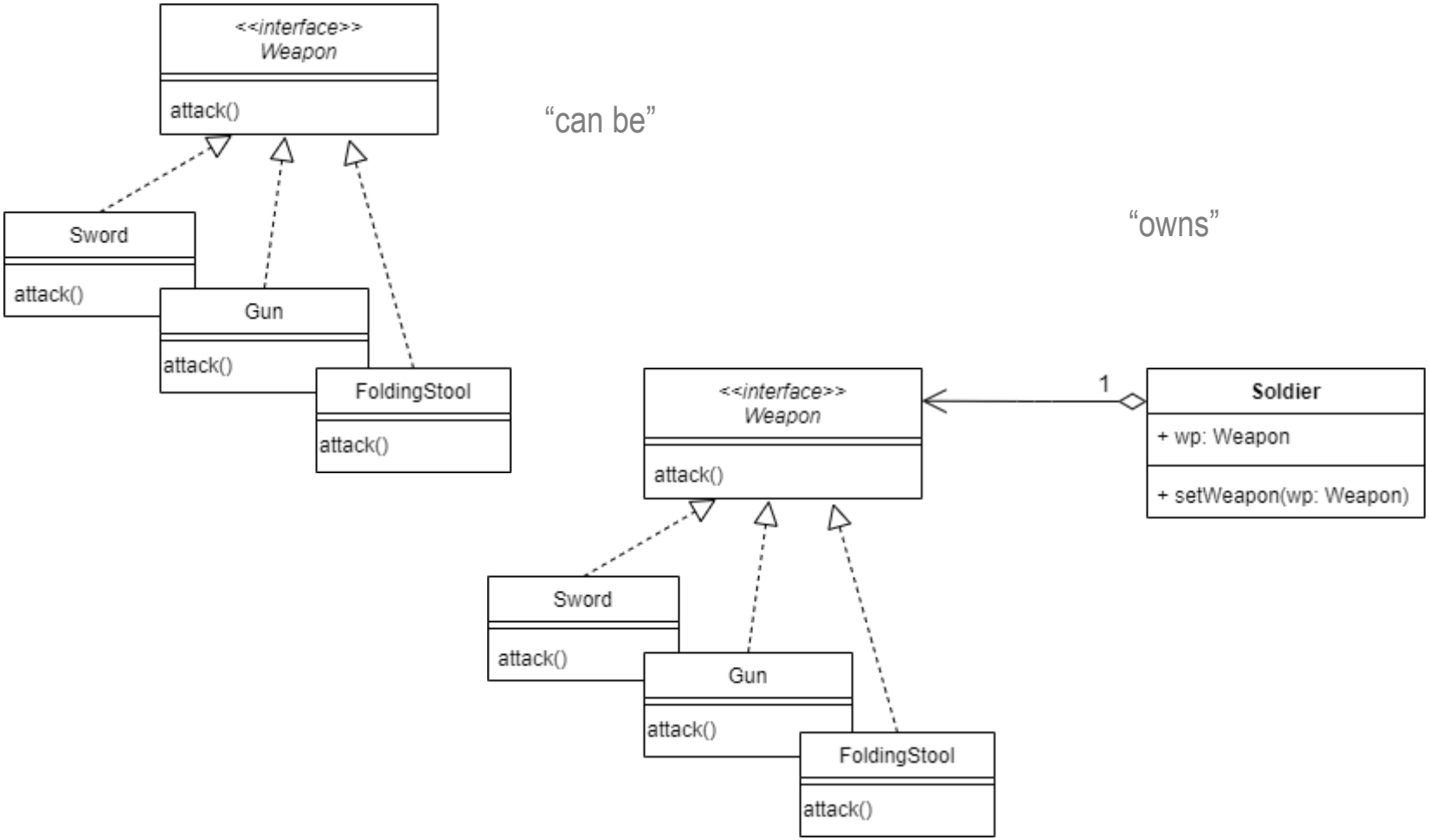
- Inheritance:
- Class B ‘inherits’ Class A means Class B ‘IS-A’ Class A



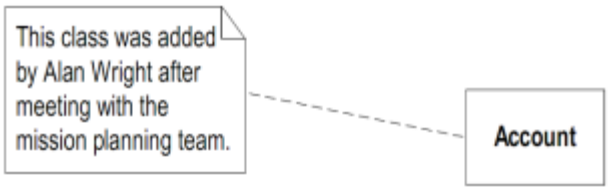
- Interface
- Interfaces are similar to inheritance however with interfaces only the interface is inherited.
- The methods defined by the interface must be implemented in every class that implements the interface.
- Interfaces can be represented using the <<interface>> keyword:



- Inheritance vs. composition/aggregation



- Keywords
- Use ‘<<‘ and ‘>>’ to surround keywords, such as interface/abstract
- Notes - To comment on a diagram element



- Exercise
- Try to illustrate the use of the following UML diagram. Note: please describe the *scenario* of the diagram, not the '*meaning*' of it (e.g. do not say "class A has 3 objects of class B").