UML

Object Orientation and UML class diagrams

QA

Learn. To Change.

Objectives

- At the end of this session you'll:
  - Learn what is UML and its value in describing your design
  - Understand Interfaces, Inheritance and Polymorphism
    - Including multiple Inheritance
  - Understand Aggregation
  - Learn about the Association Qualifiers

Why design?

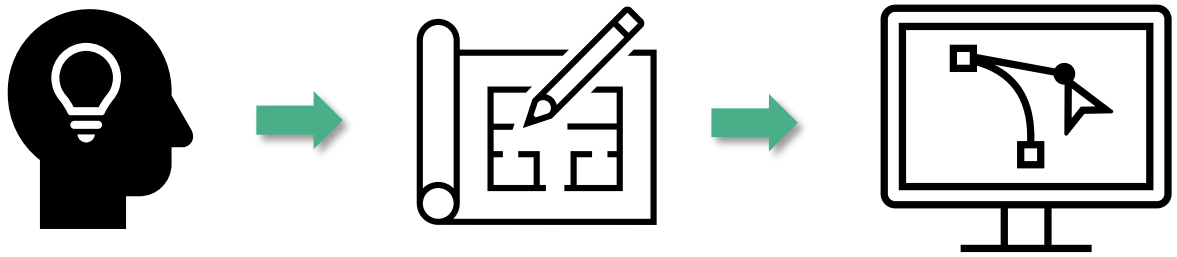
Planning matters:
Good OO design starts *before* coding. UML communicate the structure and behaviour of your app.

Thinking visually:
It helps make abstract OO ideas
Defines relationships, responsibilities, and dependencies

Catching problems early:
Poor OO design leads to rigid, hard-to-maintain code.
UML spots issues like poor cohesion or tight coupling

Why Build Models?

- Cheaper than building the real thing
- To provide a clear abstraction of concepts
- To gain understanding, before building
- To provide a common vision for users and developers
- To manage, scope and document complexity



UML diagrams

UML standards has 13 different types of diagrams:

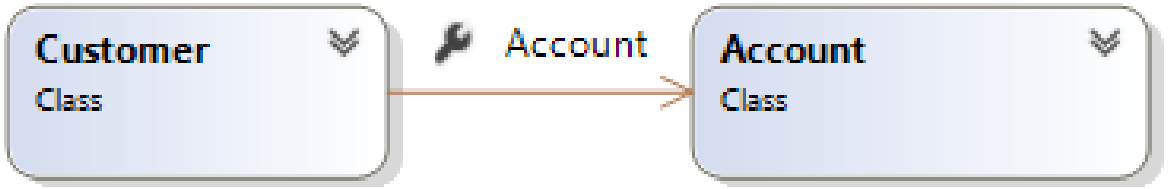
Class	Activity	Object	Use case
Sequence	Package,	State	Component
Communication	Composite structure	Interaction overview	Timing
Deployment			

In this section we will explore the Class diagram

What is UML?

- **A graphical and textual notation for concepts**
  - Rich and expressive, but can be used simply
  - Text allowed/encouraged when graphics prove difficult
- **A meta-model of the notation**
  - Defines the syntax and semantics of the notation
  - Useful for tool builders and code generators
- **UML is independent of process and lifecycle**
  - The unified process was designed for UML
  - It specifies a lifecycle and workflows

UML Association

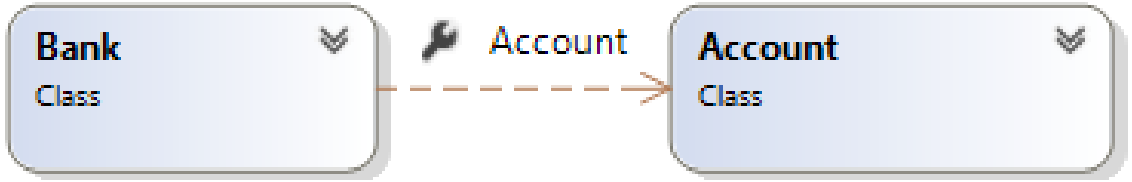


Association example:
Customer has a reference to Account

```
class Customer
{
    Account account;

    public Customer()
    {
        account = new Account();
    }
}
```

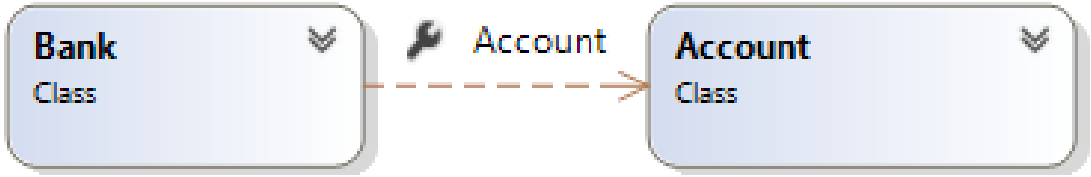
Dependency - looser association




```
class Bank
{
    public void Register(Account acc)
    {
        acc.Open();
    }
}
```

Passed as a parameter and used only by this method

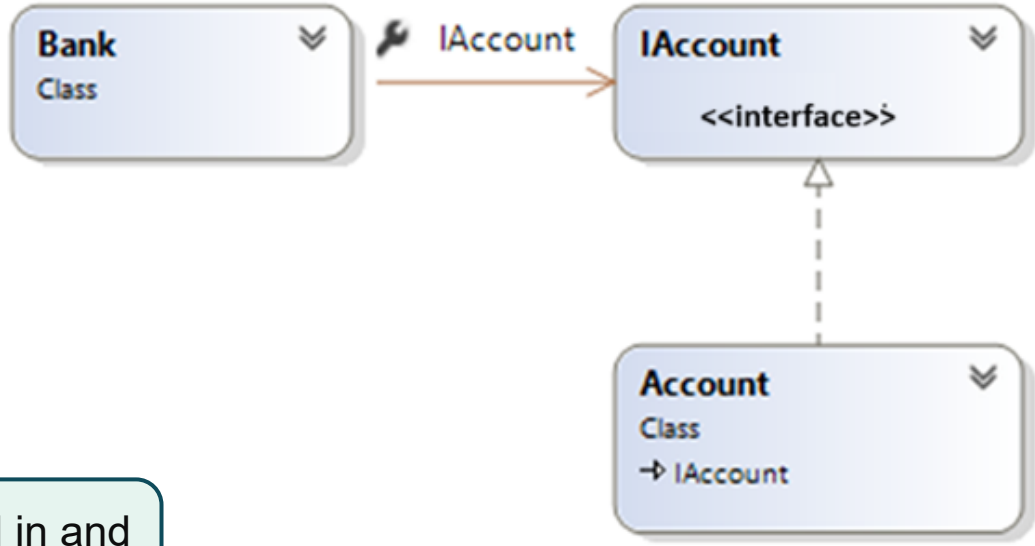
UML Dependency - another example




```
class Bank
{
    public void CreateAccount(int id)
    {
        Account account = new Account(id);
        accounts.Add(account);
    }
}
```

Created and used within the method
The object only persists within the method

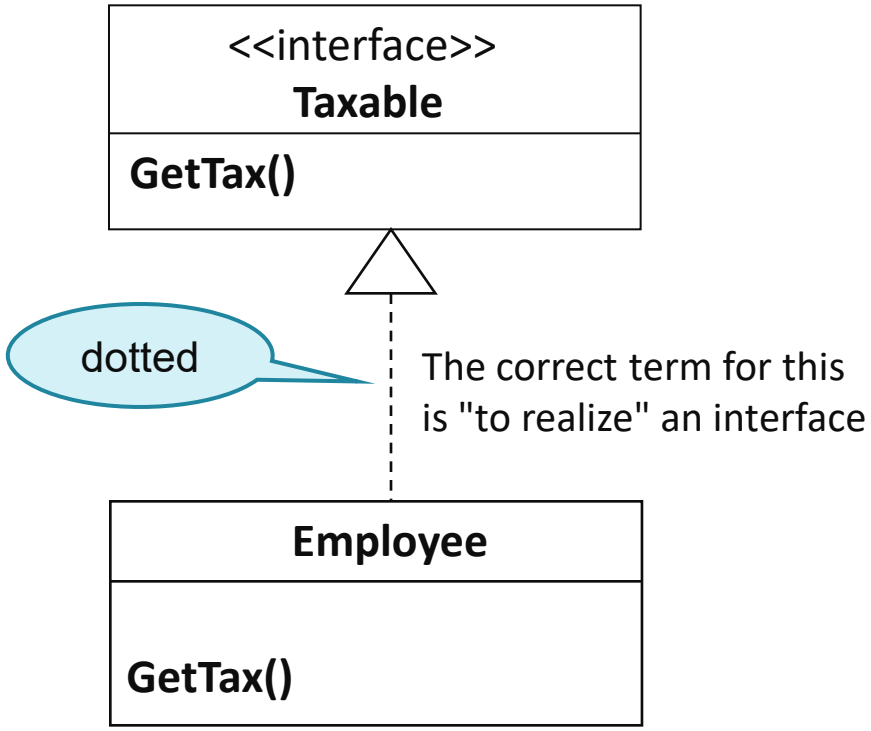
Dependency – reducing hard coupling

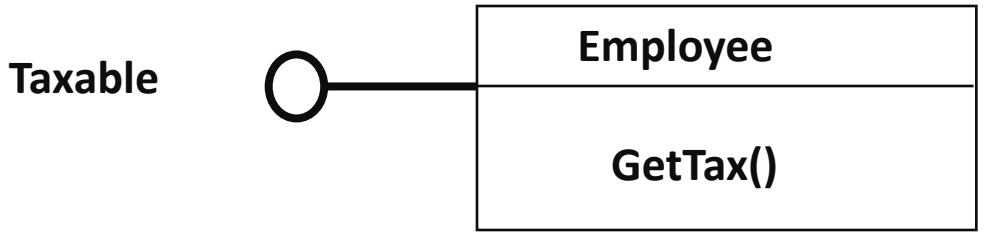



```
class Bank
{
    public void Register(IAccount acc)
    {
        acc.Open();
    }
}
```

A kind of account is passed in and used only within the method

Interface Notation in UML



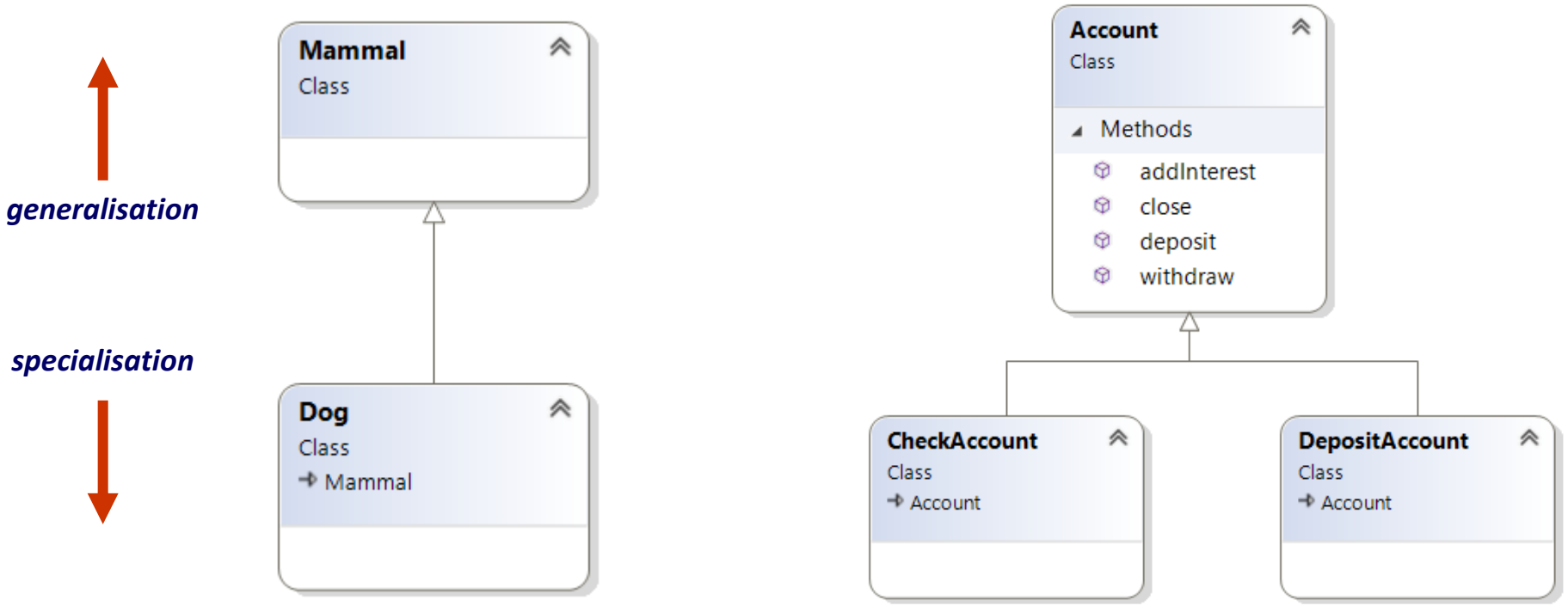


Would you say "an Employee is a kind of taxable (item) ?

Or does it sound better to say "an Employee supports all Taxable Operations?

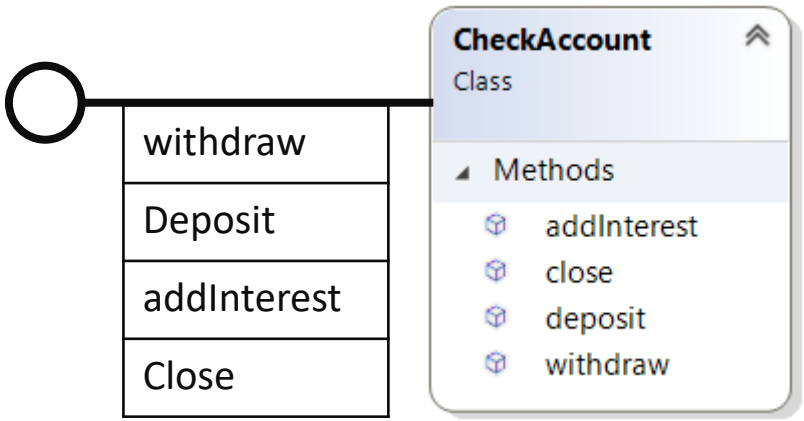
Abstract Base Class

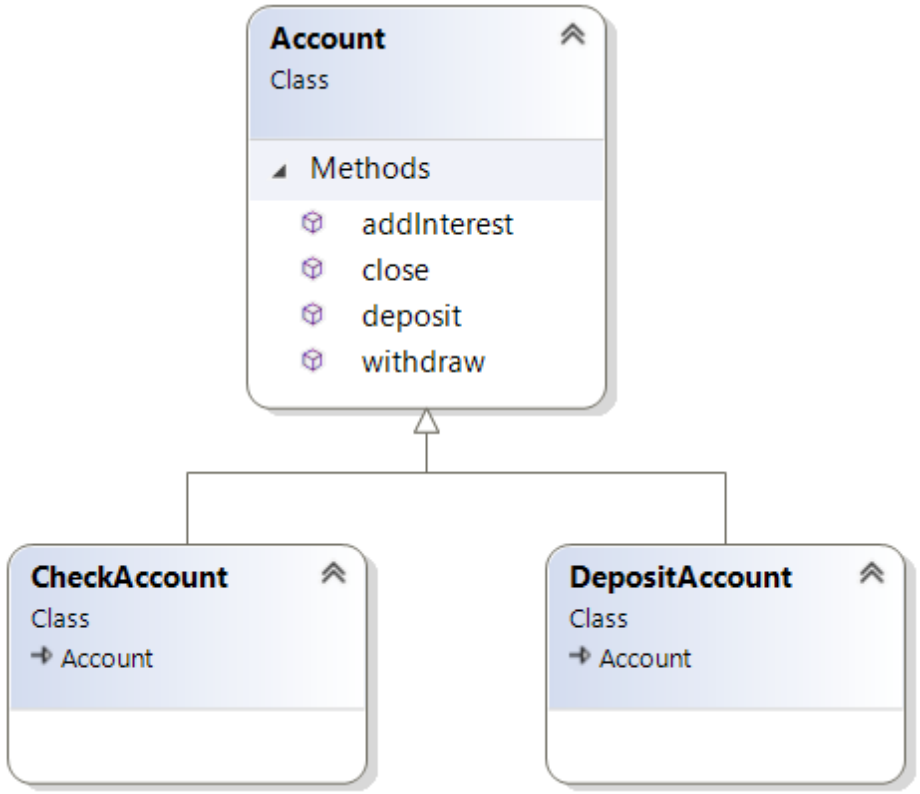
- You could view an interface as a degenerate version of an Abstract Base Class



You can't create a mammal, or an Account. They are abstract classes

What is the Difference ?

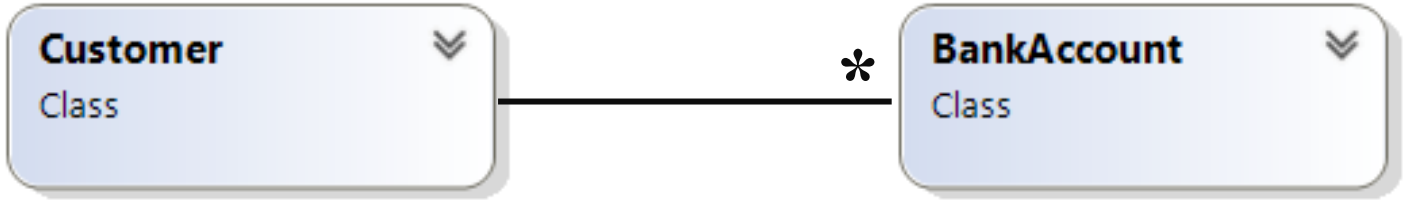




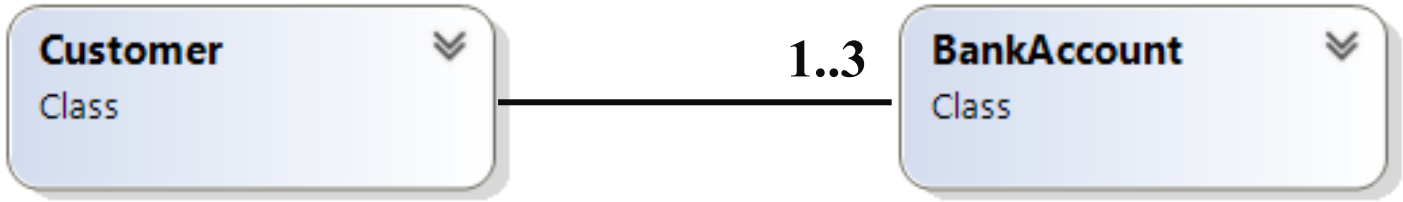
Inheritance Of Interface And
Inheritance Of Implementation

Aggregation

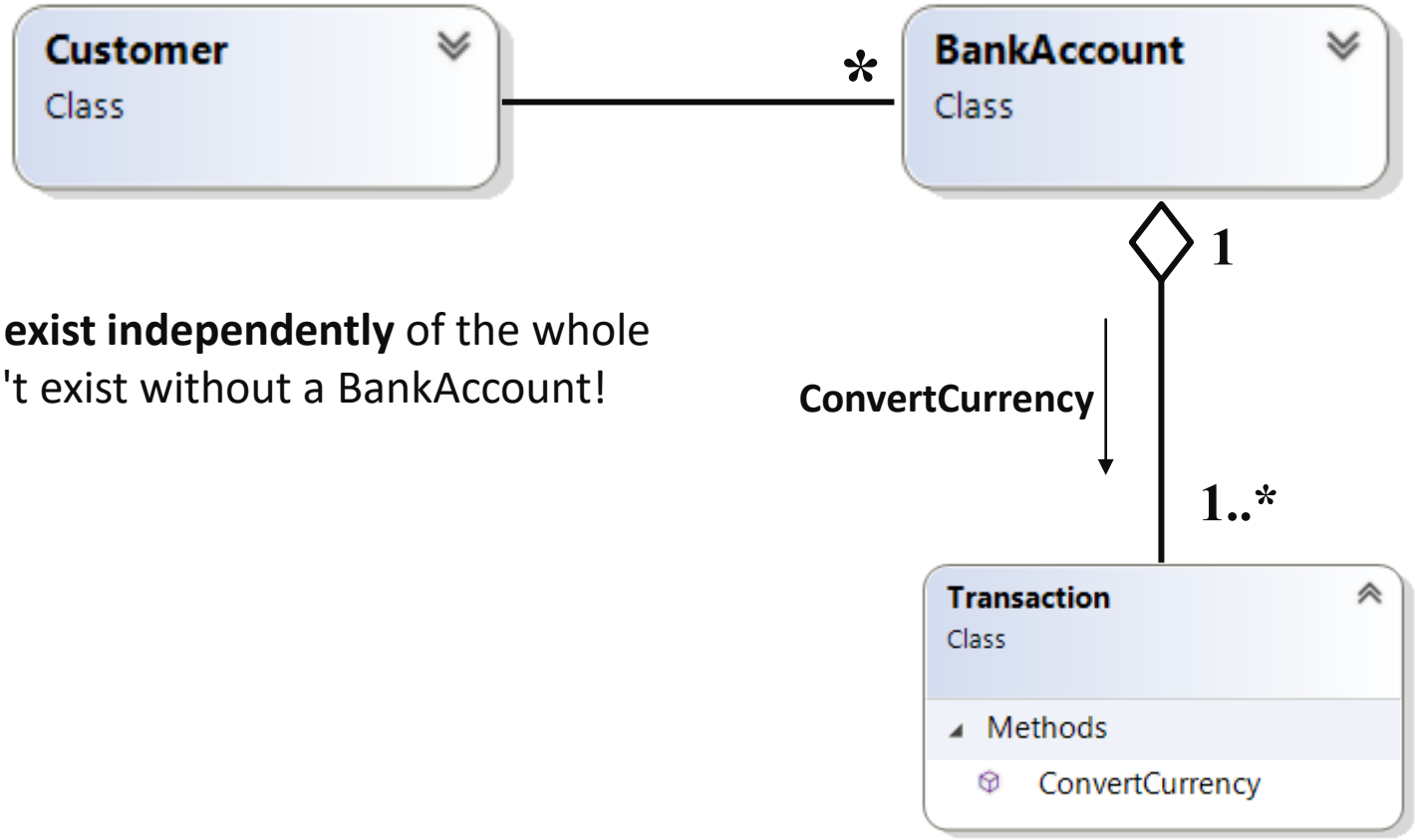
- Is a weak form of whole-part relationship. The part can exist independently of the whole Here the BankAccount object can exist and operate even when Customer referring to it.



- In the above example, a customer can have zero to many accounts
- You can use numbers to restrict creating of BankAccounts

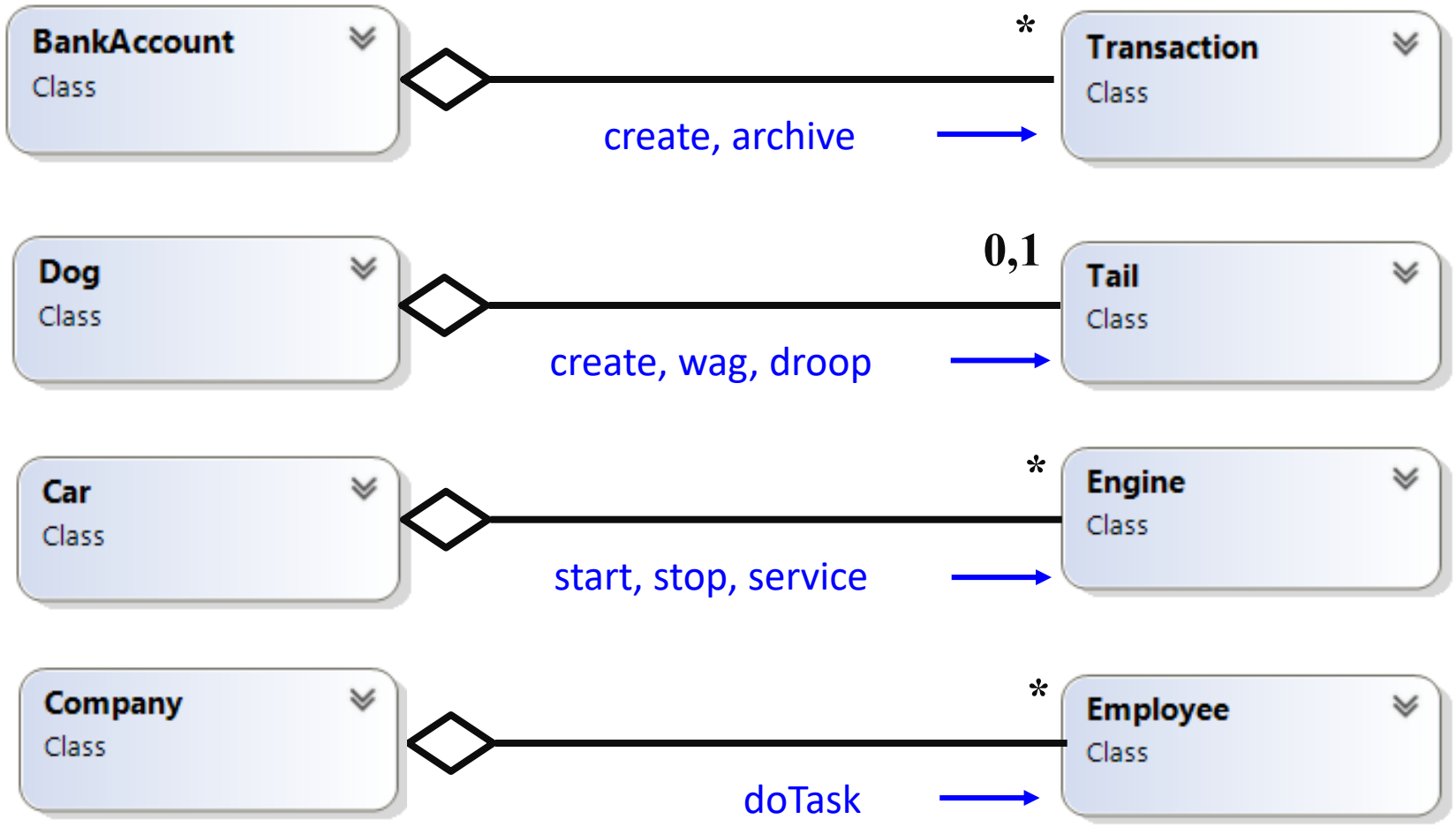


Composition – Delegation of work to other objects



The part **cannot exist independently** of the whole
Transactions can't exist without a BankAccount!

Aggregation

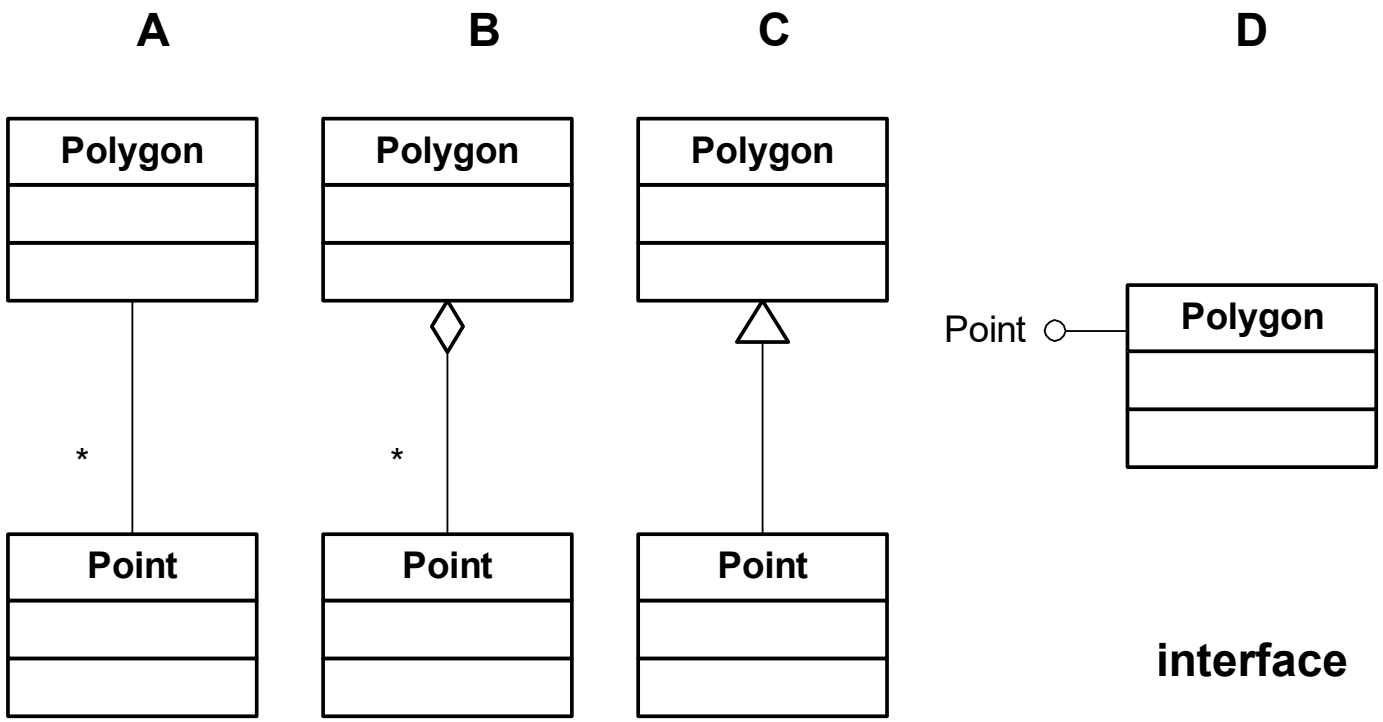


Quiz

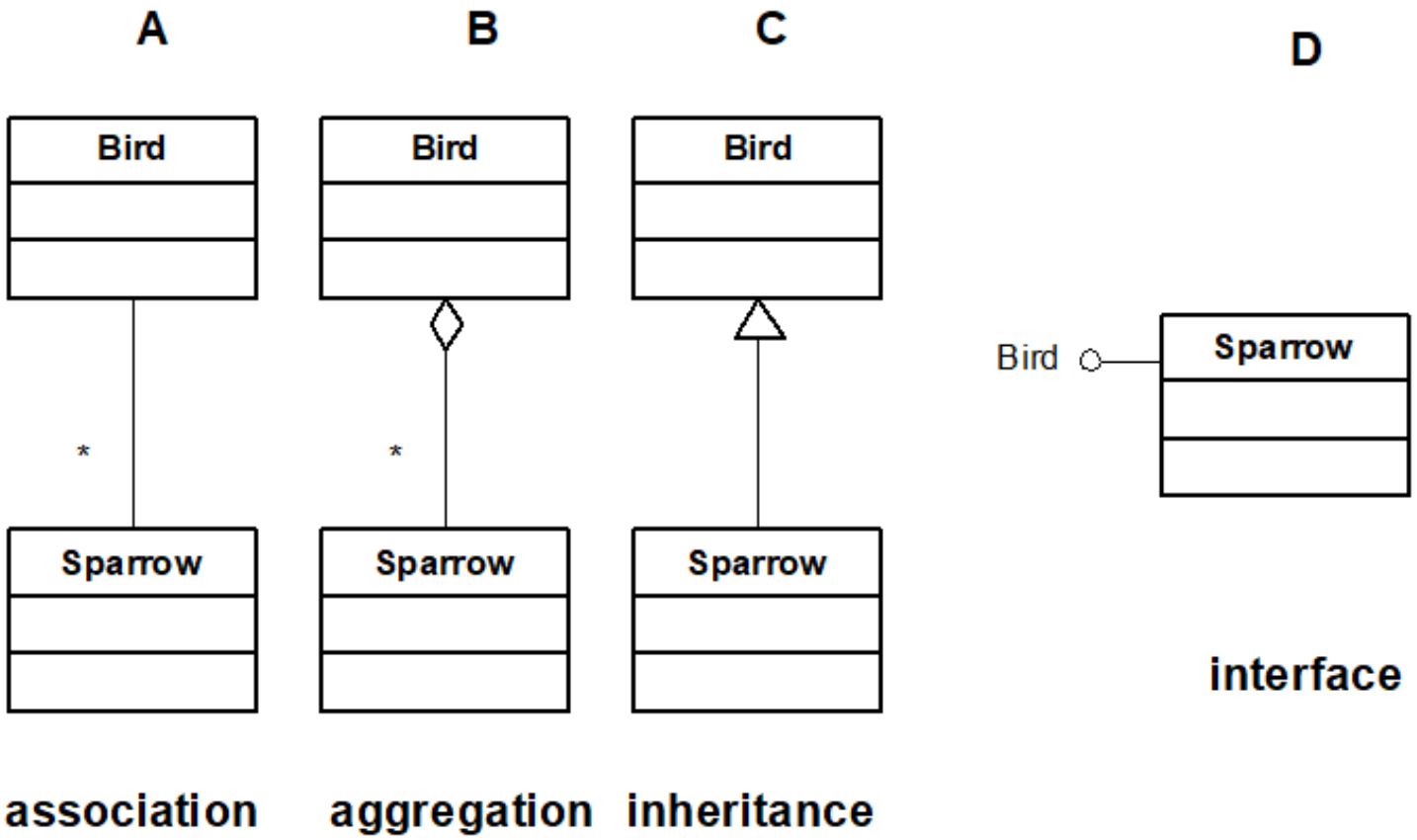
- As a group , decide for each of the following whether each **aggregation** or **association** would be most appropriate

1. **A track on a CD**
2. **A mobile phone has a battery**
3. **A portfolio contains assets**
4. **A car parked in a car park**
5. **An organisation has many departments**
6. **A person driving a car**
7. **A guitar has strings**

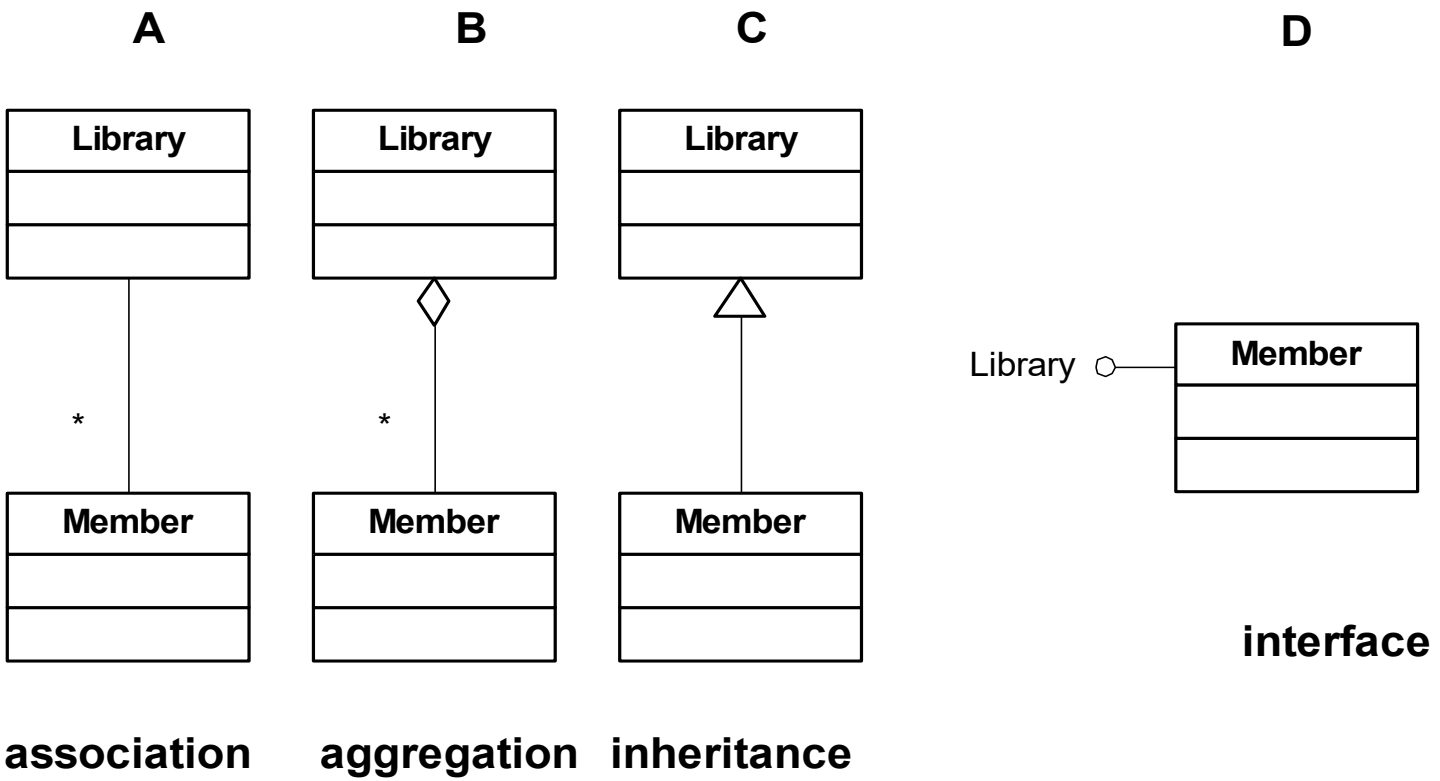
Polygon / Point



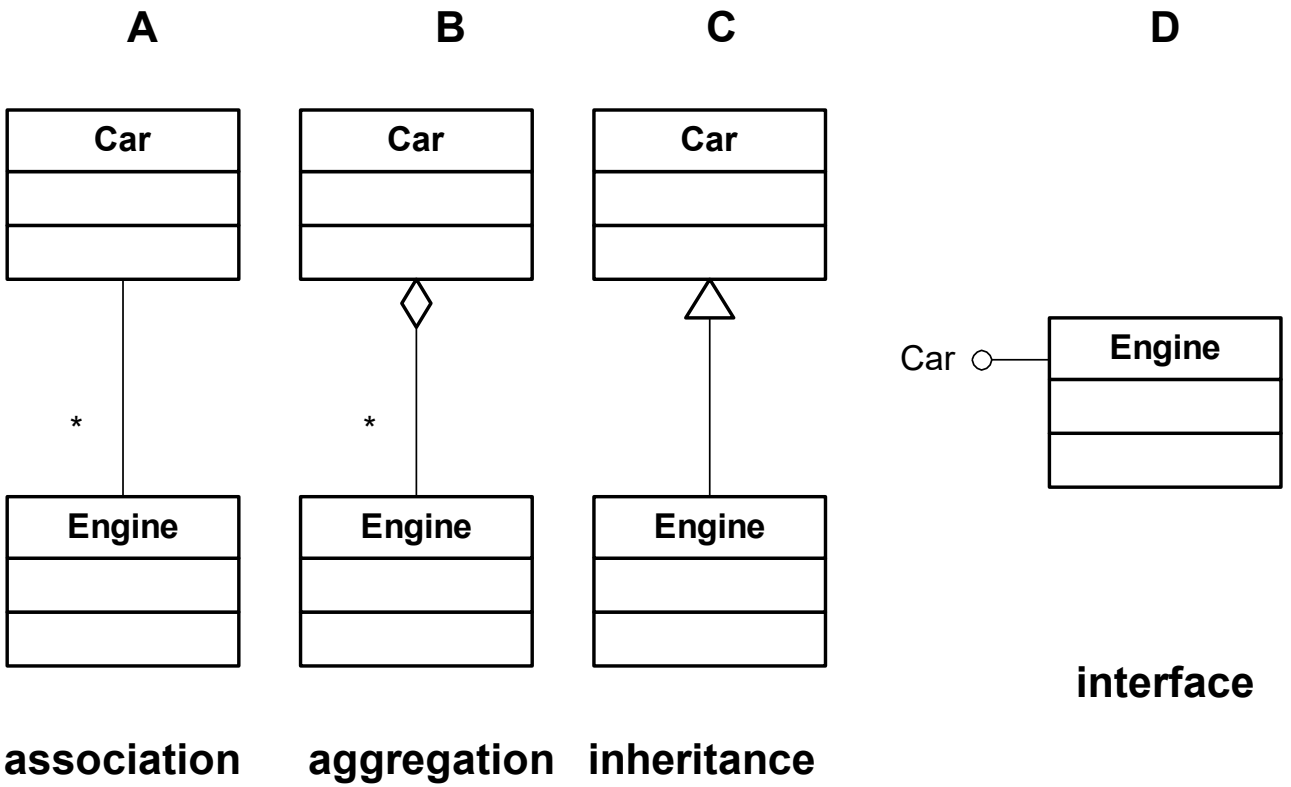
Bird or Sparrow



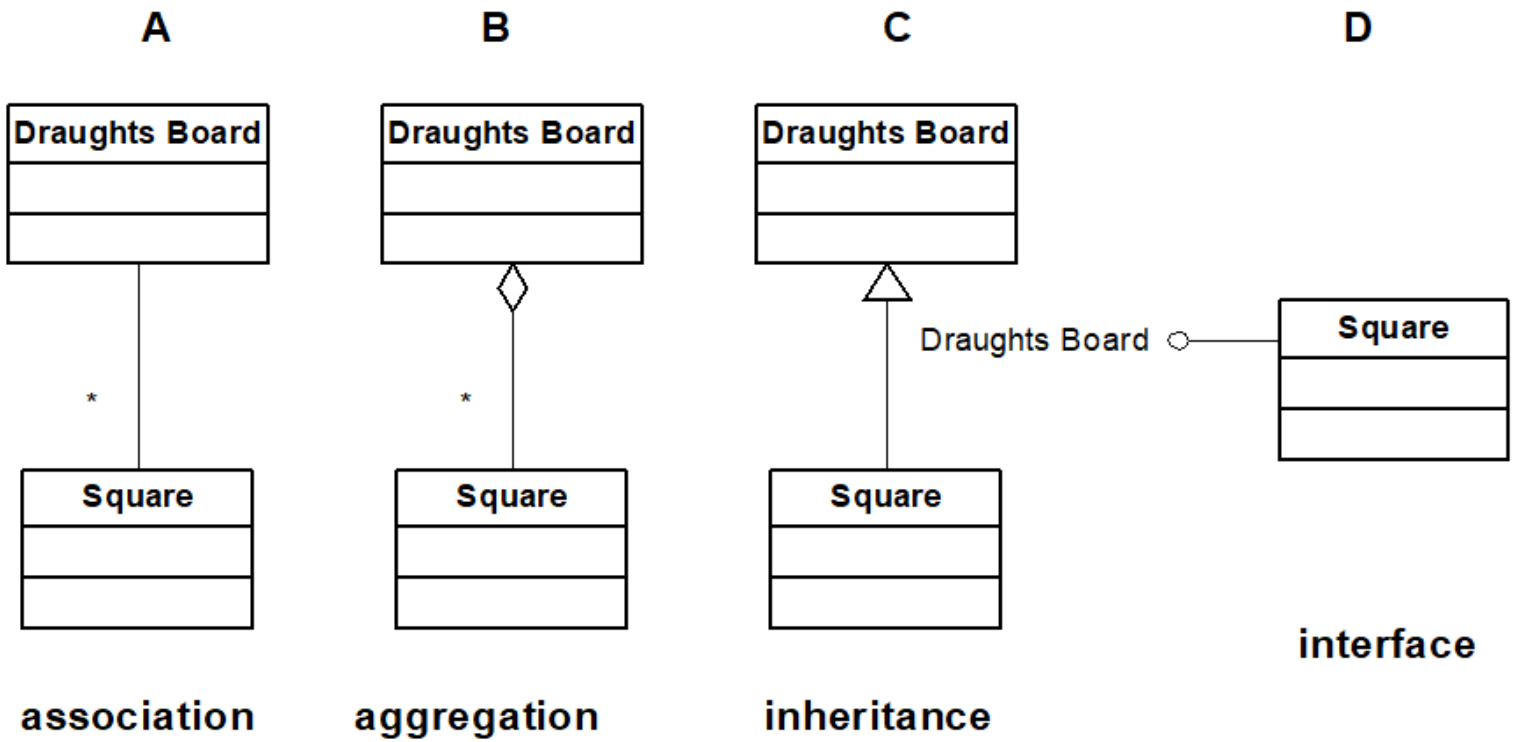
Library / Member in a Library Checkout System



Car / Engine


A
B
C
D
Car
Car
Car
*
*
Engine
Engine
Engine
Car
Engine
interface
association
aggregation
inheritance

Draughts Board



SUMMARY

In this part of the course, you have…

- Learned what UML is and its value in describing your design
- Explored and understand the reason for Interfaces, Inheritance and Polymorphism and how to use them. Including multiple Inheritance
- Have understood the use of Aggregation
- Learned about the Association Qualifiers and multiplicity
- Learned to described Encapsulation

Lab

You may read **0-UML tutorial.docx**
after the course to revise the lectures

Please do one or more of the following labs:

- **UML Class diagram Trains**
- **Extra lab - Game challenge**
- **Extra lab - Library Class diagram lab**

QA

Learn. To Change.