

Interfaces, UML, Dependencies, Coupling

AND THE ANSWER TO LIFE AS A DEVELOPER

COMP3522 OBJECT ORIENTED PROGRAMMING 2: WEEK 3



ABSTRACT BASE CLASSES IN PYTHON

Remember this?

We say that Python uses **DUCK TYPING**:

"If a bird walks, swims, and quacks like a duck, then call it a duck"

For example, if an object can be concatenated, indexed, and converted to ASCII, that is, if it does everything a string can do, then let's treat the object like a string.

Duck typing is when an object's suitability for typing takes place at runtime



Informal Interface Recap

Consider a function `add(x, y)`
that adds the two
parameters

- `def add(a, b):`
- `return a + b`

We can invoke this function
using ints or using strings!

- `print(add(1, 1))`
- `print(add('nanoo','nanoo'))`

Informal Interface Recap

From our example last class

```
class Enemy():  
    def take_damage(damage):  
        print("Enemy took {}  
damage".format(damage))  
  
class Player():  
    def take_damage(damage):  
        print("Player dodged {}  
damage!".format(damage))  
  
def attack(character):  
    character.take_damage(10)
```

What would happen if `attack(my_player)` and `attack(my_enemy)` were called?

Abstract Base Classes (ABC)

While duck typing is a great way to implement polymorphism, it can be difficult to predict and ensure if the given object has the common interface (public methods and attributes) that your code expects.

Enter: Abstract Base Classes

Abstract base classes are a class that defines a set of attributes, methods and properties, but does not implement it. Any class that inherits from an ABC has to implement and override these attributes / methods (or, the child class has to implement the interface defined by the ABC).

Formal interface: Abstract Base Class (ABC)

In Java, Interfaces and Abstract classes are different. Interfaces only define method signatures, while abstract classes can define method signatures and attributes.

Abstract Base Classes are Python's answer to Interfaces and Abstract Classes!

We can create our own ABCs with the abc module

<https://docs.python.org/3/library/abc.html#module-abc>

Let's see how Abstract
Base Classes Work In
Python

Example

```
import abc

class Bird(abc.ABC):

    @abc.abstractmethod
    def fly(self):
        pass

    @abc.abstractmethod
    def lay_egg(self):
        pass
```

Abstract methods tagged and mandatory

We decorate abstract methods with `@abc.abstractmethod`

If an abstract method is not implemented by a subclass, Python will generate a `TypeError: Can't instantiate abstract class X with abstract methods Y`

An implementing class is an instance of the ABC

```
class Parrot(Bird):  
    def fly(self):  
        print("Flying")  
  
p = Parrot()  
>>> isinstance(p, Bird)  
True
```

[abc_example.py, abstractbaseclass.py](#)

Virtual subclasses

We can also register a class as a virtual subclass

The class will be treated as a subclass of the ABC

```
@Bird.register
class Robin:
    pass
r = Robin()
>>> issubclass(Robin, Bird)
True
>>> isinstance(r, Bird)
True
```

This is not commonly used

You can read the rationale for its including in Python here: <https://www.python.org/dev/peps/pep-3119/#overloading-isinstance-and-issubclass>

UML REVIEW

We will use UML this term

Modelling language consisting of set of diagrams

Used to help developers to specify, visualize, construct, and document software system

1. Class diagrams
2. Collaboration diagrams
3. Sequence diagrams

Let's review them



Class diagram

Purpose is to describe the static structure of the system

Shows the classes

- Attributes
- Operations

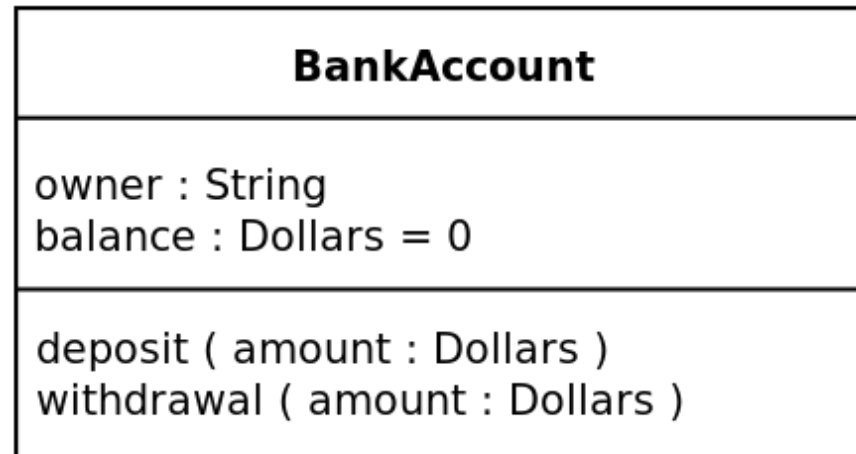
Relationships among objects

Language agnostic

Class diagram

Rectangle indicate a class

- Top section is the name of the class
- Middle section are the attributes and types
- Bottom section are the methods



Class diagram - Relationships

Arrows indicate how two classes are related

Association – Reference based relationship between two classes. A contains instance variables B



Generalization/Inheritance – Indicates that one of the two related classes is a specialized form of the other. A is a subclass of B



Class diagram - Relationships

Arrows indicate how two classes are related

Realization/Implementation – A relationship between two classes where one class implements the behaviors that the other class specifies. A is implementing interface B

A - - - -> B

Dependency – Communication between two classes, where one class does not necessarily contain an instance of another class. A contains methods that use parameter B

A - - - -> B

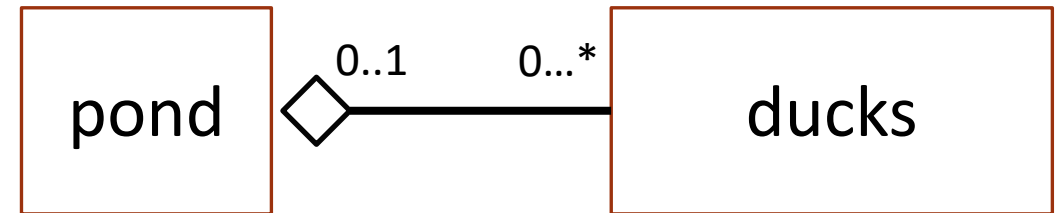
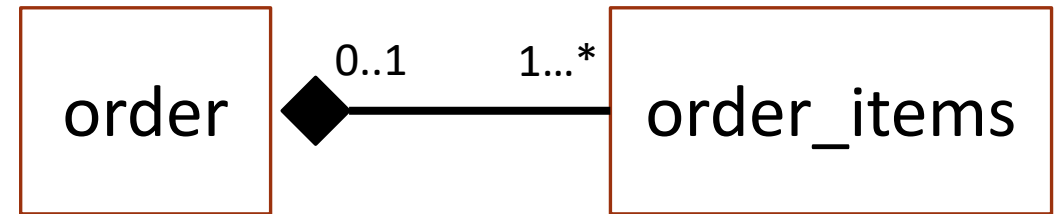
Class diagram

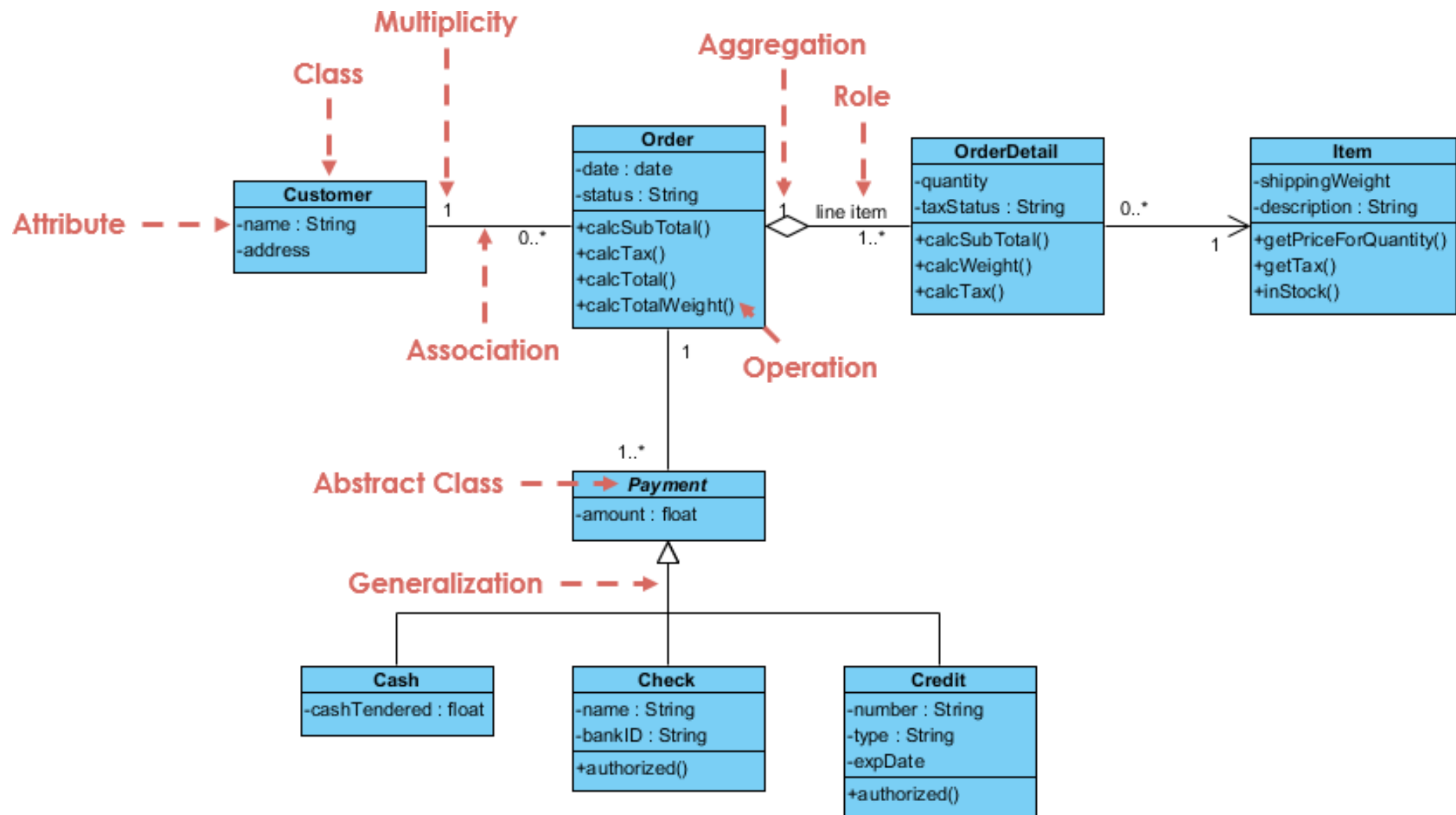
Composition: When a class is made up of and has references to objects of another class. In a composition relationship, the parts cannot exist without the whole.

Eg: If an *order* is destroyed, then so are the *order_items* that are composed by *order*

Aggregation: When a class references an object of another class. In an aggregation relationship, the referenced class can live on even if the class referencing it is destroyed.

Eg: If a *Pond* is destroyed, the ducks can live on.



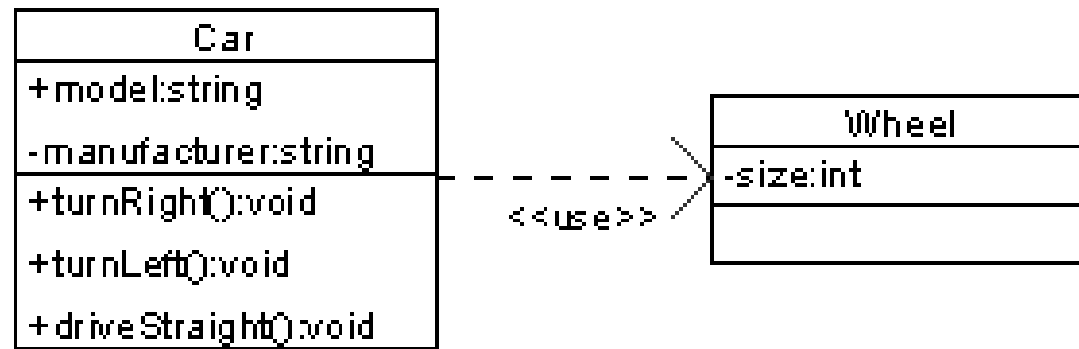


Class diagram



Car is composed of carburetors. Car has exactly 1 carburetor. Carburetor can belong to 0 to 1 cars

Class diagram

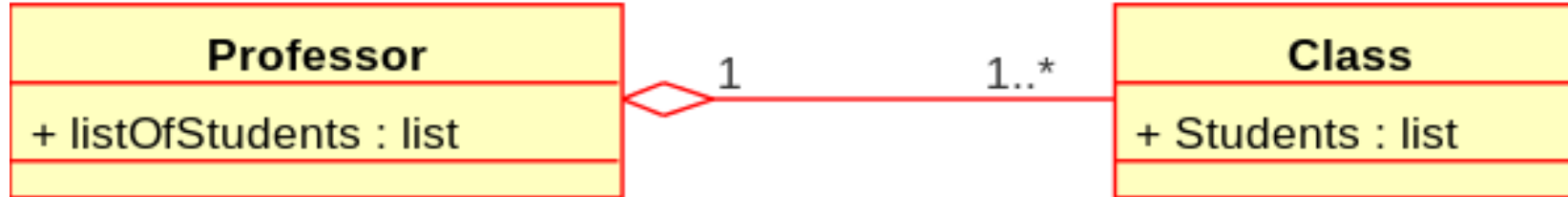


Car

- 2 strings named model, manufacturer
- Methods to turnRight, turnLeft, driveStraight

Car has a relationship with wheel. Some method uses wheel as a local variable to parameter

Class diagram



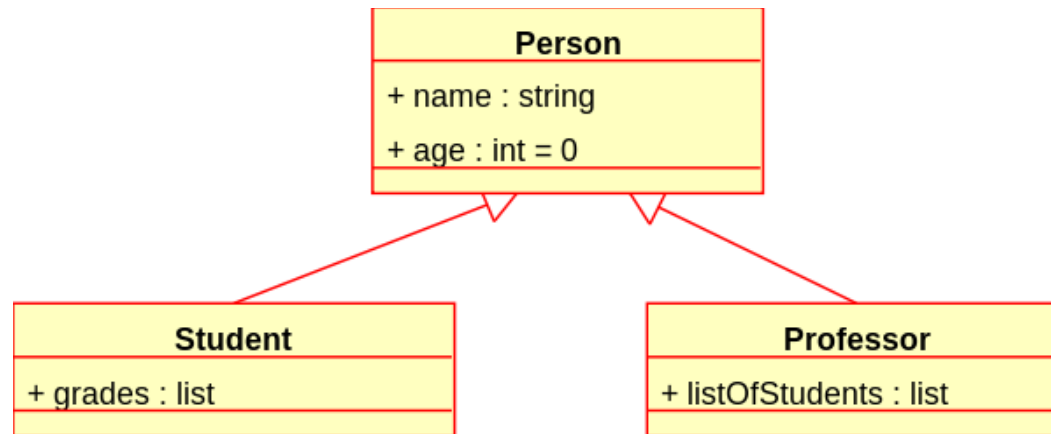
Professor is an aggregate of classes. Professor has 1 to many classes

- Has a list named `listOfStudents`

Each class has exactly 1 Professor

- Has a list named `Students`

Class diagram



Person class has a string name and int age as attributes

Student is a subclass of Person

- Has a list named grades

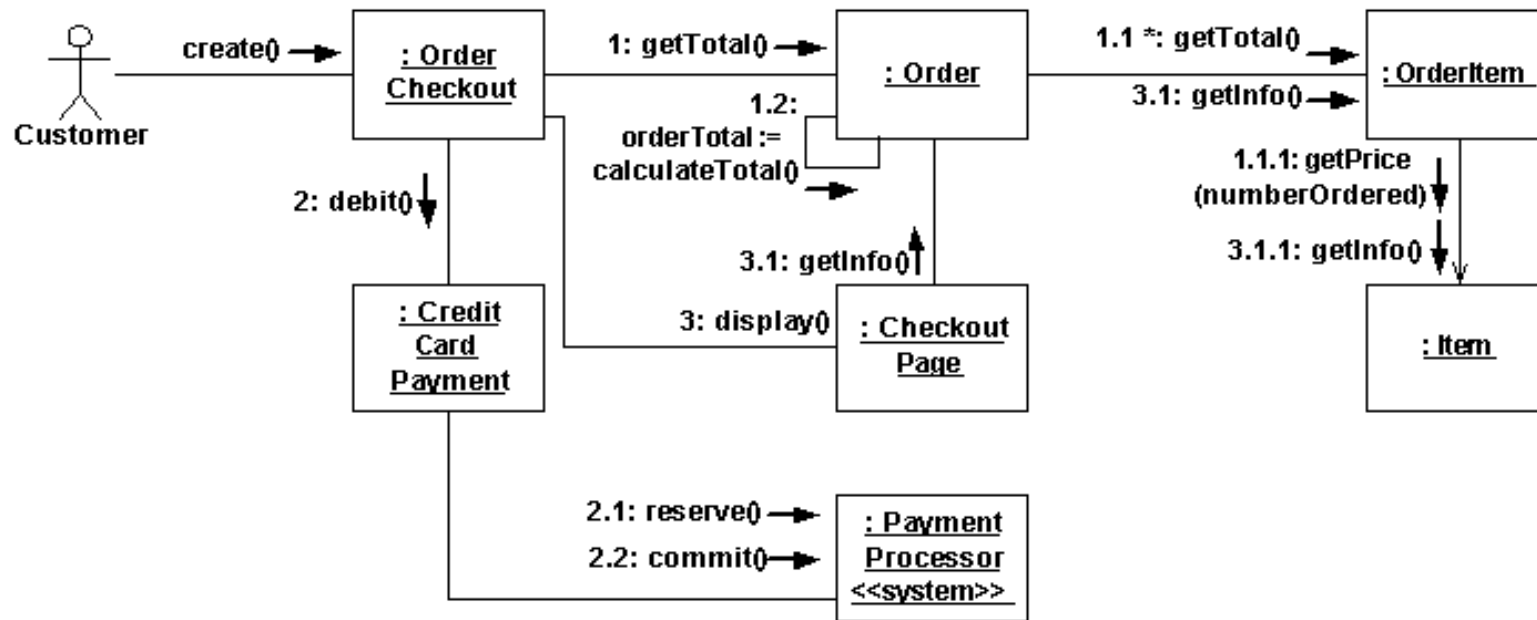
Professor is a subclass of Person

- Has a list named listOfStudents

Collaboration (communication) diagram

Presents interactions between objects as a sequence of messages between objects from a **structural perspective**

Shows how objects interact in a particular use case or subset of a use case



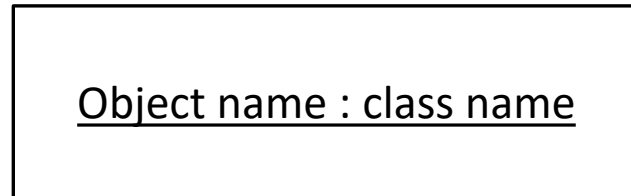
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml-collaboration-diagram/>

Collaboration (communication) diagram

Notations (Object, Actor, Links, Messages)

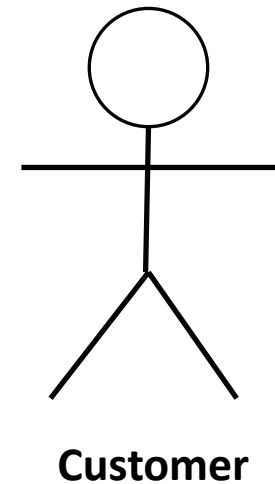
Object

- Object represented by a square containing the name of the object and its class underlined, separated by a colon



Actor

- Actor is the invoker of the interaction
- Actor is named and has a role

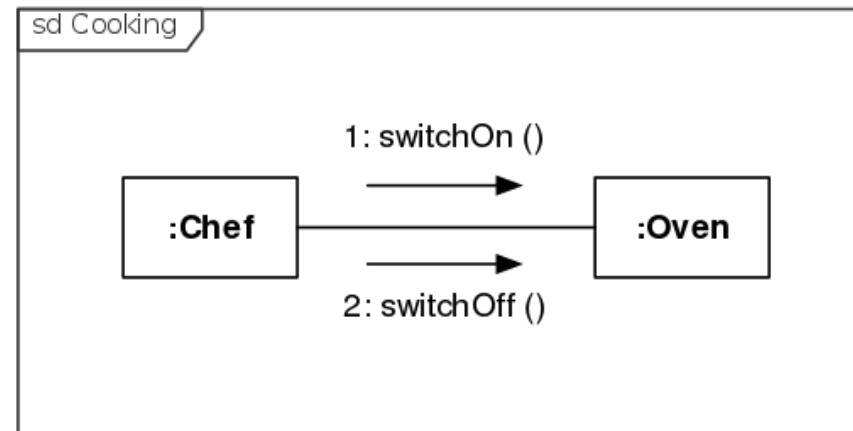


Collaboration (communication) diagram

Notations (Object, Actor, Links, Messages)

Links

- Connects objects and actors. They are instances of associations in the class diagram
- Messages are sent across these links
- Links are represented as a **solid line** between two objects

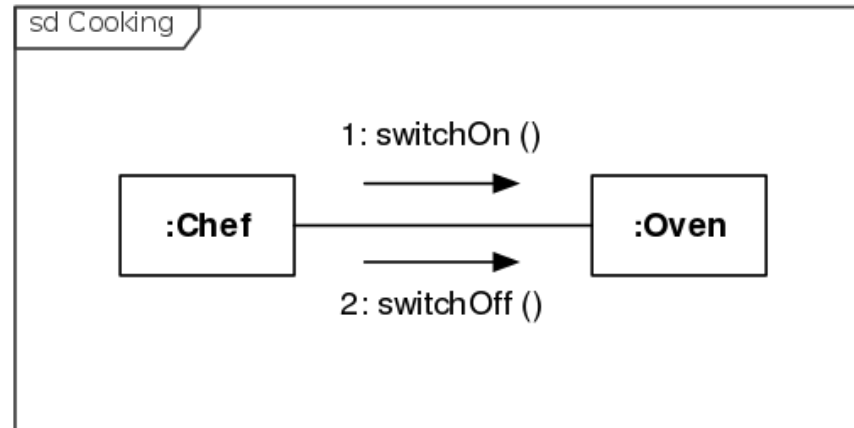


Collaboration (communication) diagram

Notations (Object, Actor, Links, Messages)

Messages

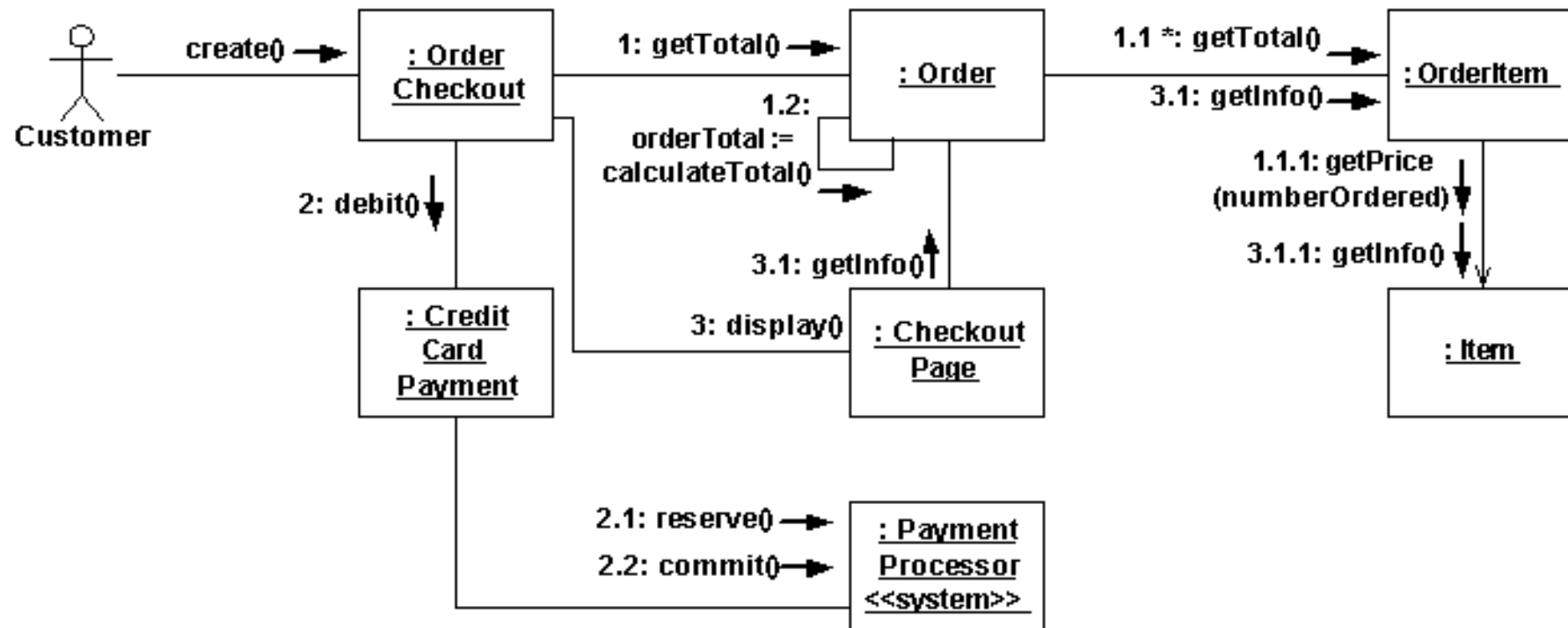
- Communication between objects that convey information with the expectation that activity occurs
- It is shown as a **labeled arrow** near a link
- Directed from sender to receiver
- Messages are numbered to indicate a sequence. Messages in the same call have additional decimals: 1.1, 1.2 etc



Collaboration (communication) diagram

Let's see this diagram again

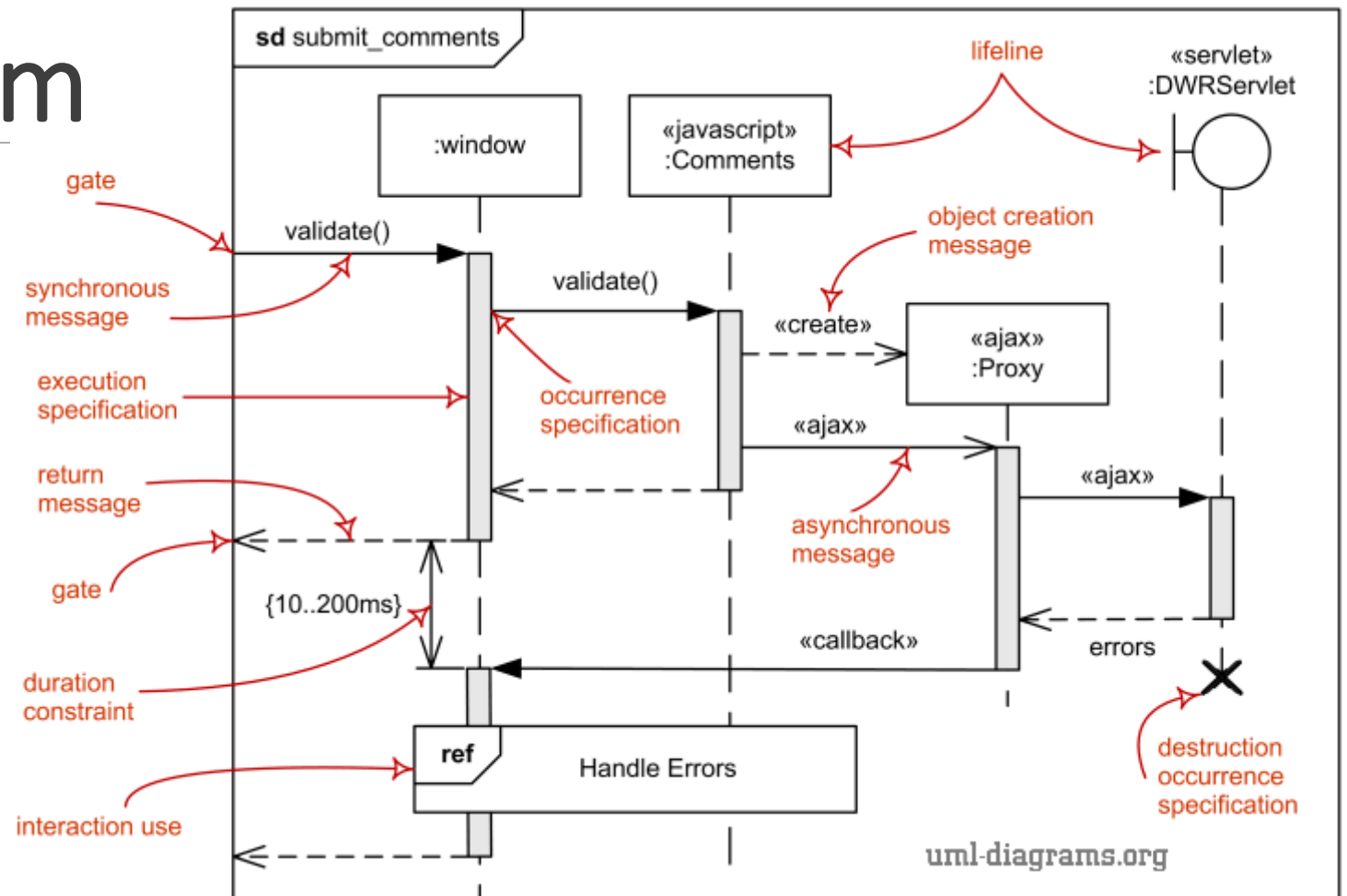
What use case is this collaboration diagram illustrating?



Sequence diagram

Presents **interactions between objects** as a sequence of messages between objects

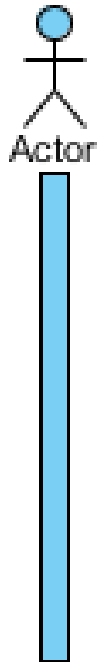
While a collaboration diagram is a structural approach, A sequence diagram is a **time-based perspective** includes the **lifetime** of the interactions between communicating objects



Sequence diagram

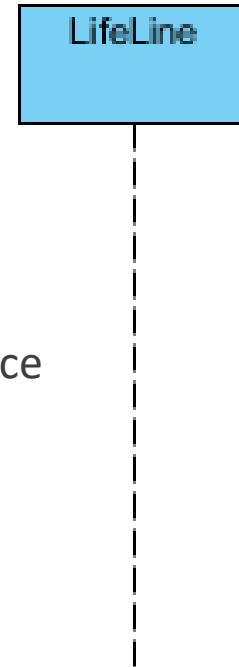
Actor

- Interacts with the subject
- External to the subject
- Played by human users, external hardware, other subjects



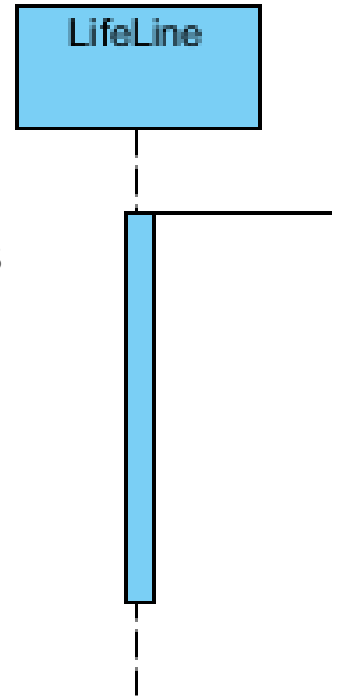
Lifeline

- Represents a single individual participant in the interaction
- Denoted by name and class type in the top box
- Dotted line indicates place in sequence



Activations

- Thin rectangle on the lifeline
- Represents the duration an element is performing an operation



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

Sequence diagram

Messages: Indicates communication between lifelines of an interaction

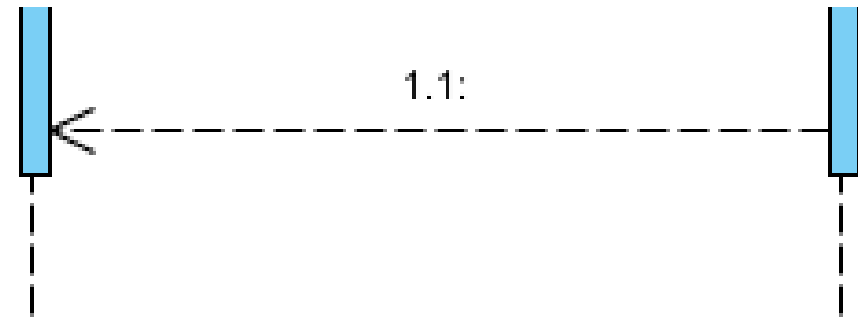
Call message

- Represents invoking operation on the target lifeline
- I.e: Call a method on target lifeline



Return message

- Represents passing information back to the caller
- I.e: Return statement in method



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>

Sequence diagram

Messages: Indicates communication between lifelines of an interaction

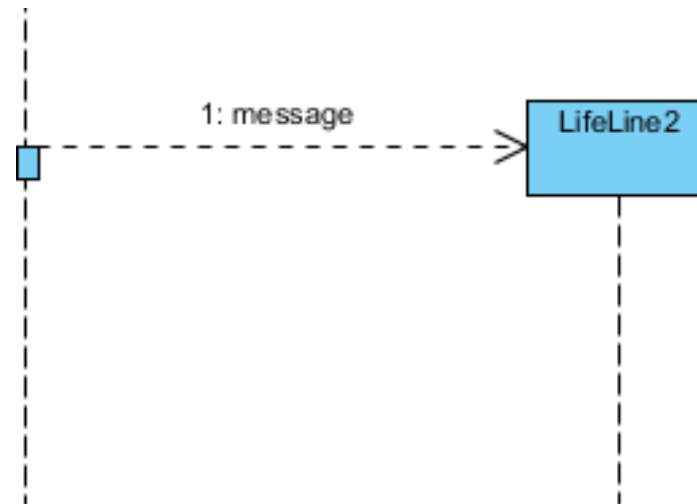
Self message

- Represents invoking message on the same lifeline

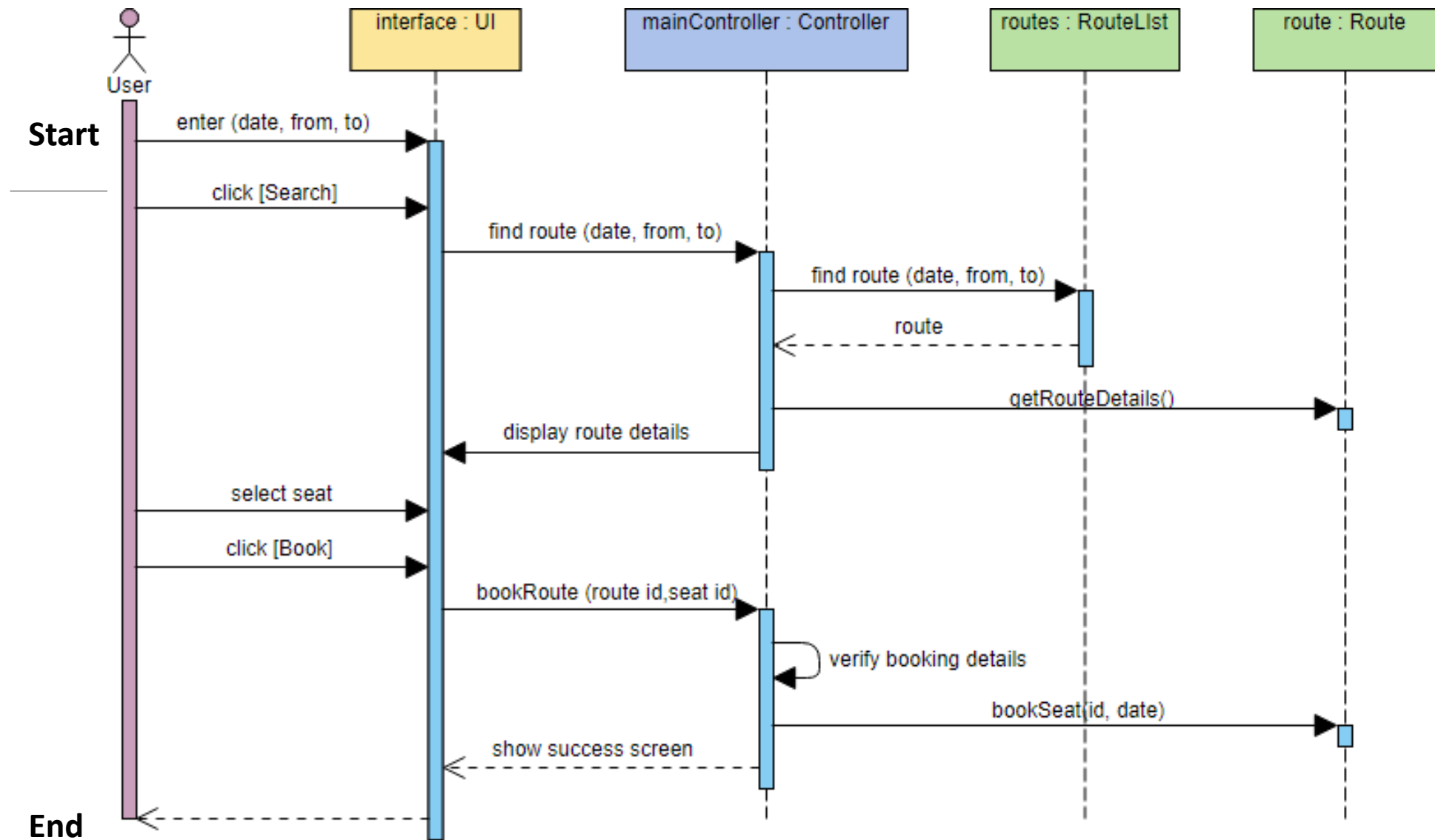


Create message

- Represents instantiation of the target lifeline



<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>



Dependencies : When your classes get clingy



Dependencies

When one entity depends on another entity.

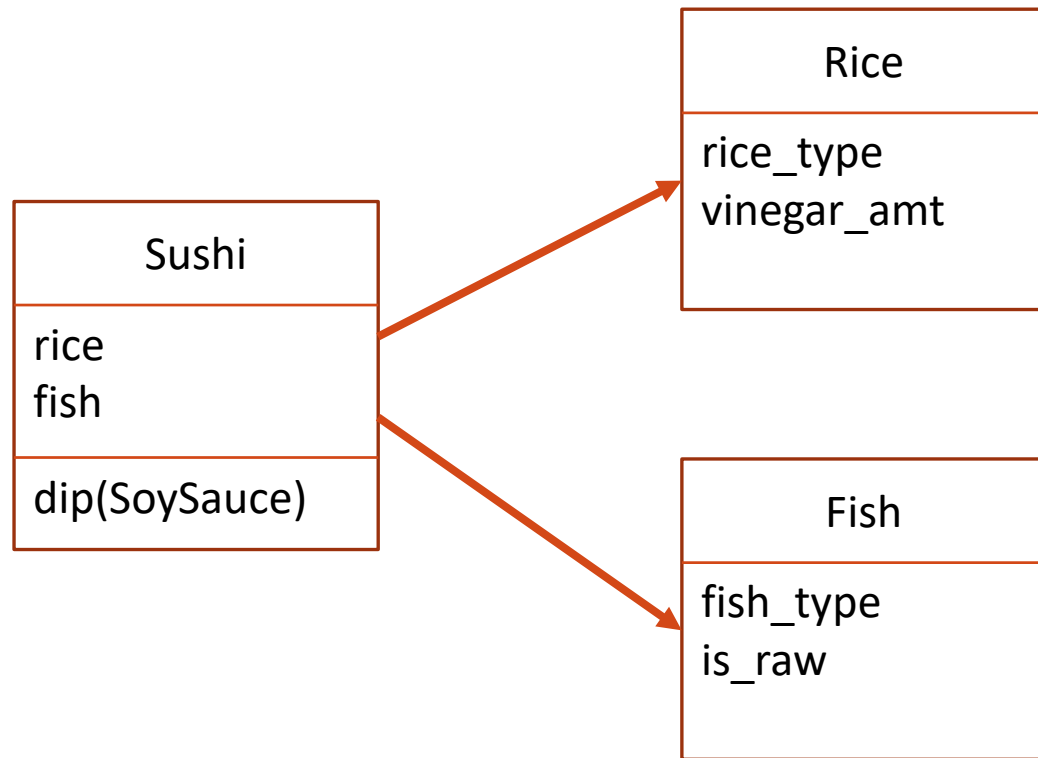
If entity A uses entity B, then A is said to be dependent on B.

In this context an entity could be a Module, Class, Package or Sub-system of related code



For example, say we were simulating a sushi restaurant. **Class Sushi would be dependent on Class Rice and Class Fish**

Why are dependencies bad?

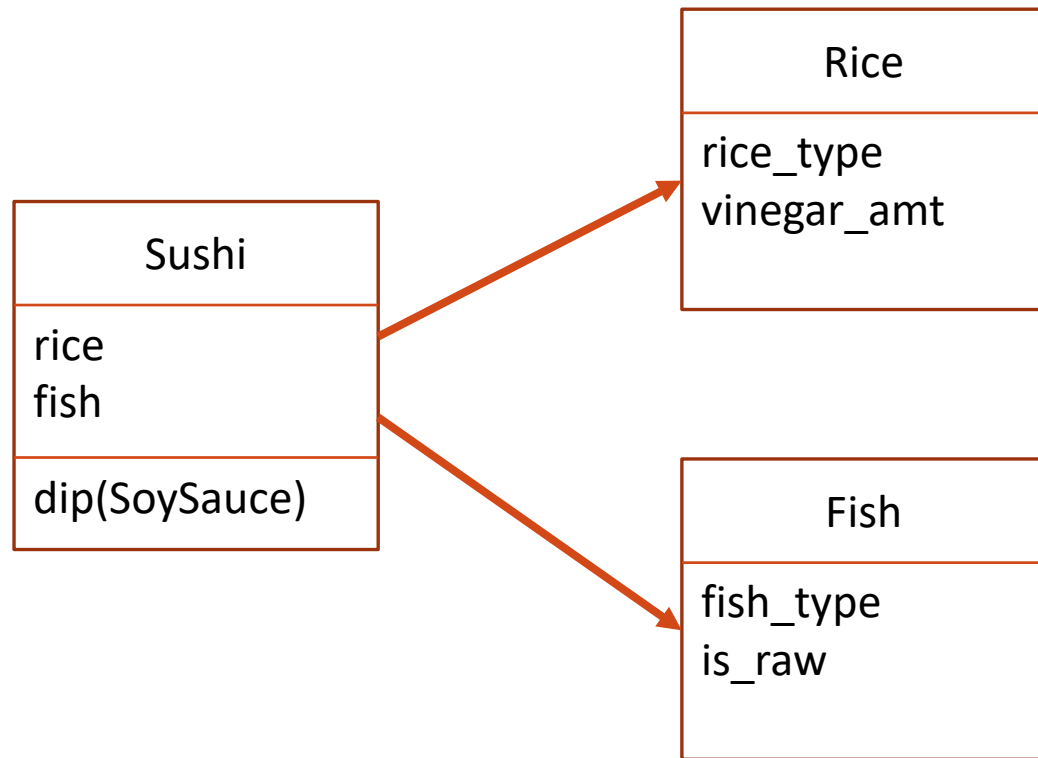


If class A is dependent on class B then any change in class B would possibly require us to change class A as well.

This may not sound like much.

Imagine if A and B were sub-systems or packages in a larger application. Now this is an issue.

Why are dependencies bad?

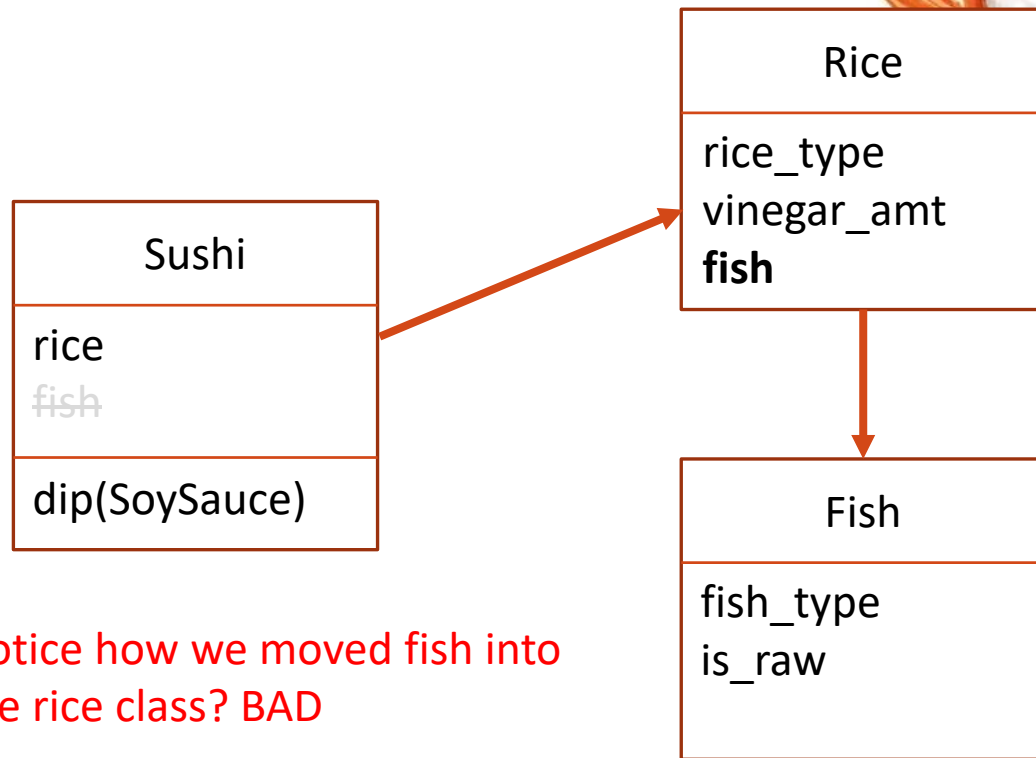


If Entity A is coupled with Entity B then Entity A will not be able to function without Entity B.

Coupling is a form of a **strong** dependency.

We try to avoid coupling where possible

Coupling

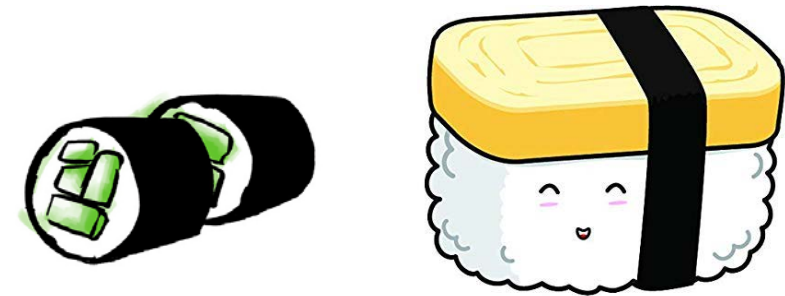


Notice how we moved fish into the rice class? BAD

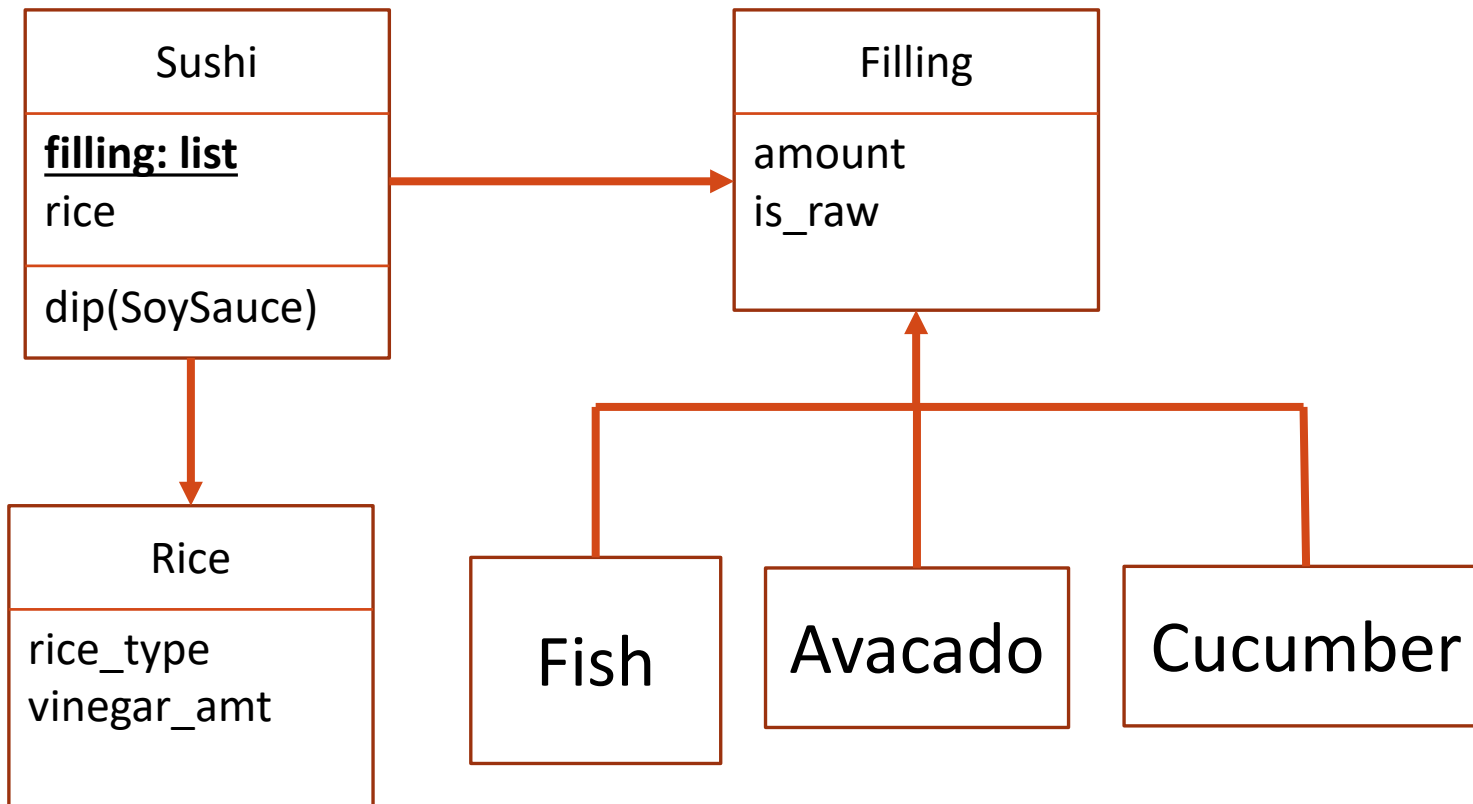
Say fish and rice are coupled. This is bad because rice will always come with fish. We wouldn't be able to have other kinds of sushi without fish.

(What about the Tamago!! :O)

We don't want strong dependencies like these. We want to decouple our code if possible so it can be flexible and polymorphic.



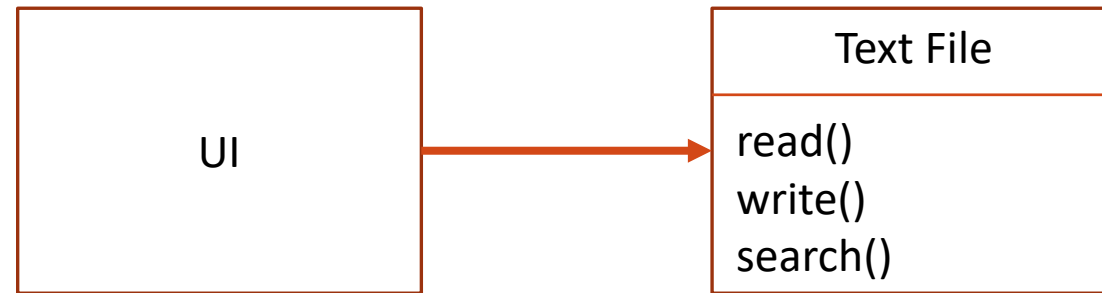
Decoupling sushi – Inheritance!



Inheritance and Interfaces are the most common and effective way of decoupling code.

NOTE: We still have a dependency! Entities will always be dependent on each other, that's how OOP works but it is better to have weak dependencies like these

A more realistic example



Say we were developing an app with a UI that was populated with some data from a text file.

After a few releases and years of development, for some reason we decide to switch out to a more secure source of data , perhaps an encrypted JSON file or even perhaps a SQLite Database. We would have to edit all the modules/classes that dealt with our app's UI!

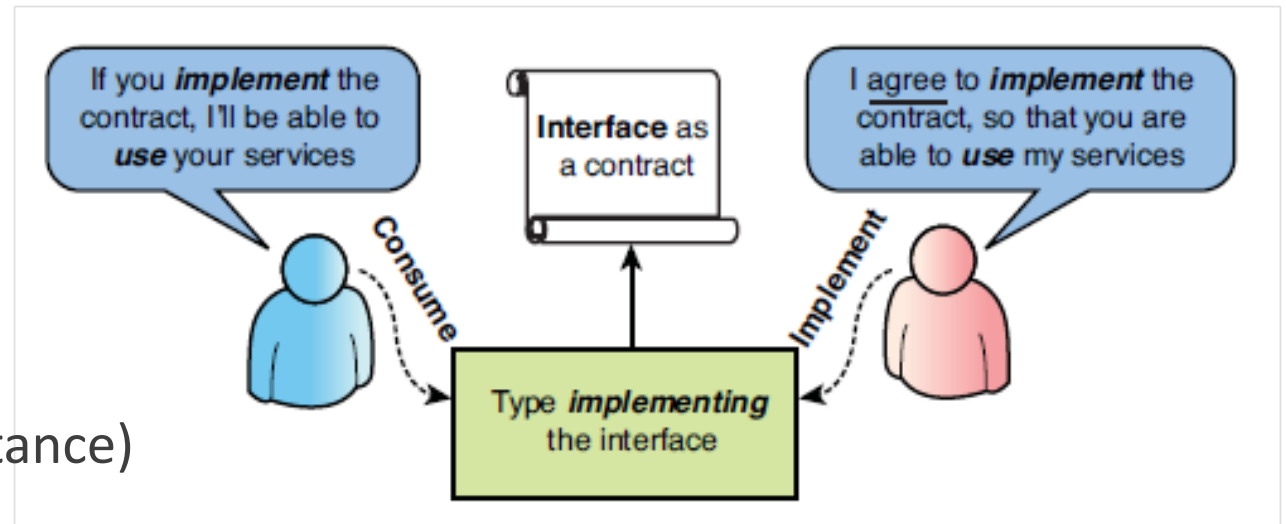
Remember Interfaces?

An Interface is a contract demanding that the class implement certain methods and behaviors.

In more general OOP terms this refers to the **public methods and attributes that are provided by a class**. These public attributes and methods are the only way to communicate (or interface) with the class.

In Python there are two kinds:

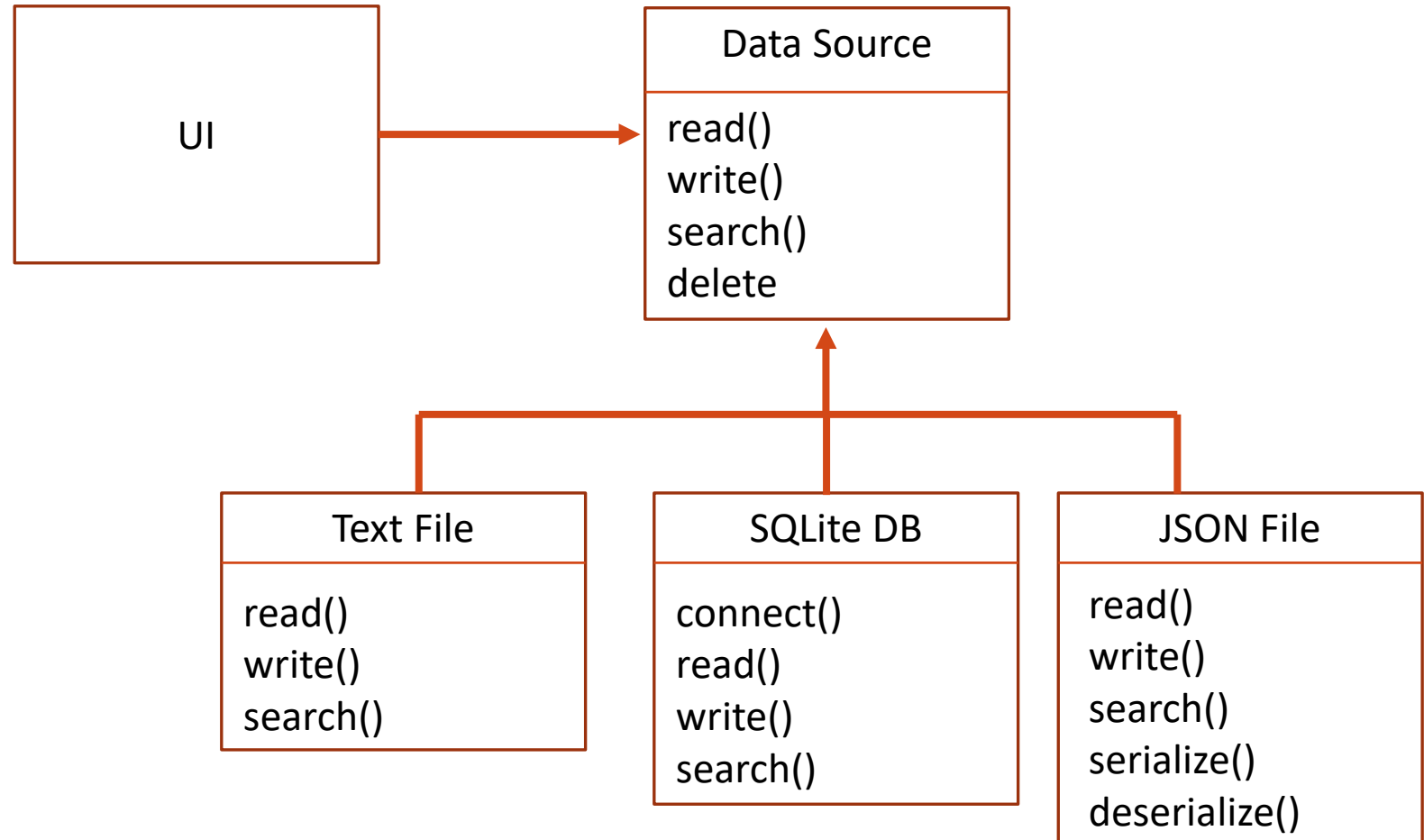
1. Informal interfaces
 - a) Duck typing
 - b) Protocols
2. Formal interfaces
 - a) Abstract base classes (using inheritance)



A more realistic example

We can decouple our system to instead depend on a data source which is an **abstraction** that **hides** different data sources.

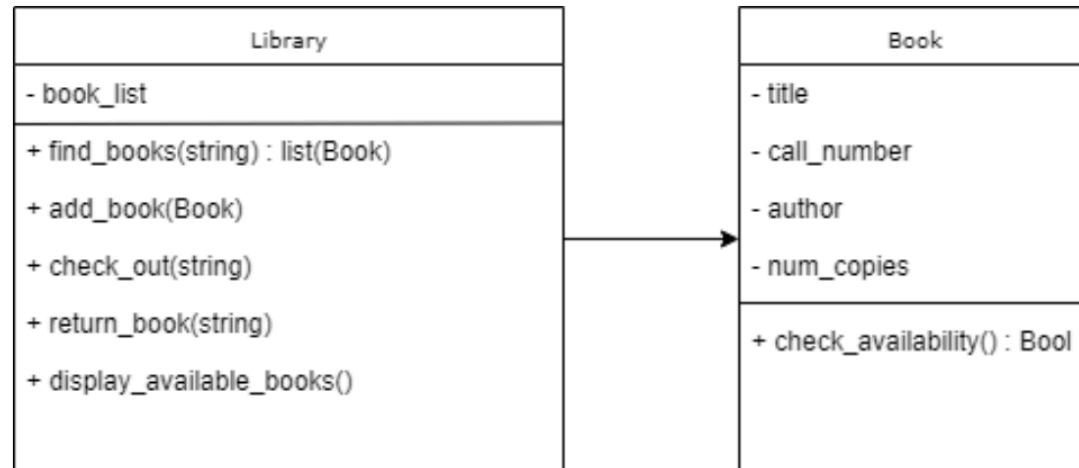
Our UI would just have to be **dependent on a common interface** provided by the Data Source and not be concerned with how that data source is implemented



Can you identify the dependencies and coupling in this next scenario? How would you solve it?



Library catalogue refactoring (Lab 2)



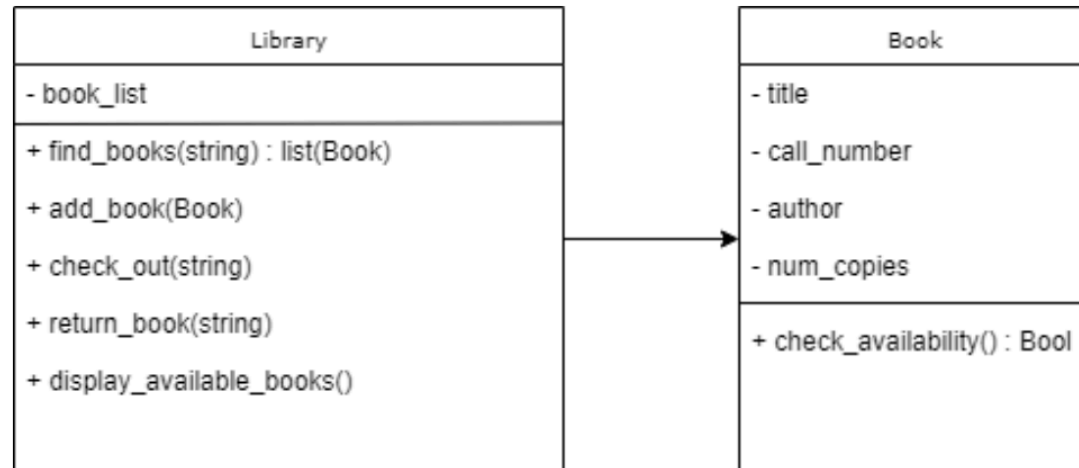
Library is a class that contains a list of books

Problem is that Library is tightly coupled with Book

If Book changes, we'll need to make changes to Library as well

We need a way to de-couple Library from Book

Library catalogue refactoring (Lab 2)



Can we split up Library into two separate classes?

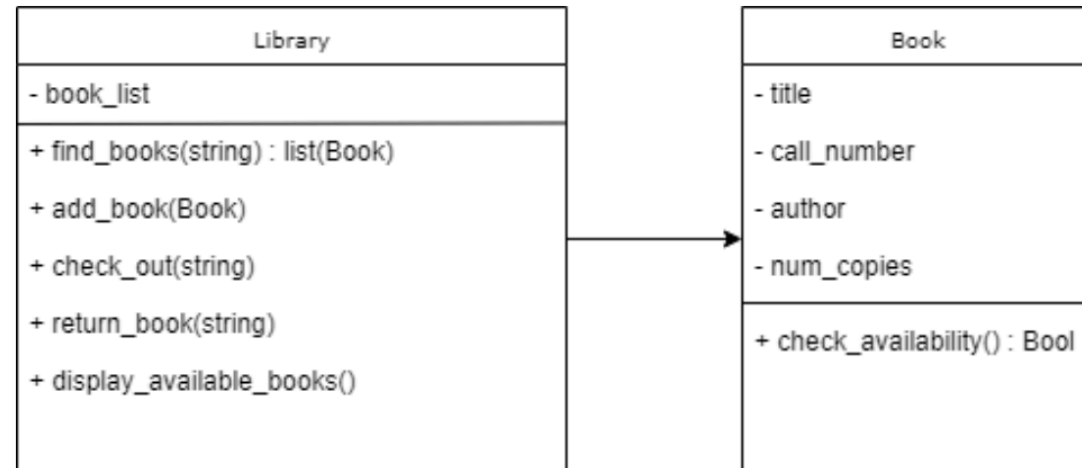
One class handles the “front end” of the library that the public interacts with

- Check in/out and displaying of books

The other class handles the “back end” of the library that employees interact with

- Adding new books to the library, logic to find books

Library catalogue refactoring (Lab 2)



What if this Library just expanded their catalogue of items and it now includes books, DVDs and scientific journals

How would you break this down? What would the class diagram look like?

Break into groups of 2/3 and draw the diagrams. I'll come around to answer questions/check out the diagrams

Functions parameters

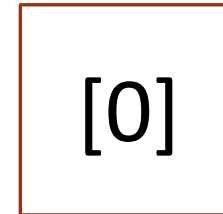
PASS BY VALUE? PASS BY REFERENCE? OR SOMETHING ELSE???

Pass by value

Check out this code, and let's assume it's pass by value

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



ham

A new object is created in memory, and the variable `ham` is assigned to it

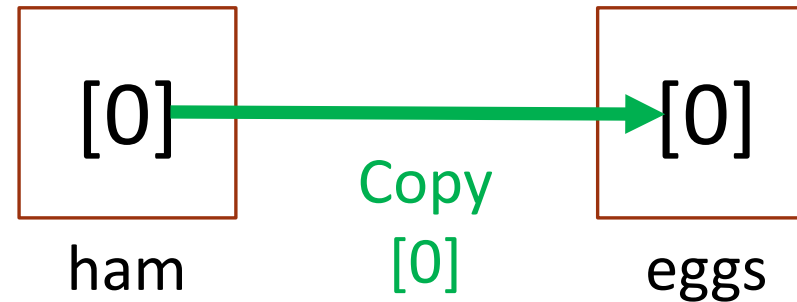
<http://stupidpythonideas.blogspot.com/2013/11/does-python-pass-by-value-or-by.html>

Pass by value

Check out this code, and let's assume it's pass by value

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



eggs is a new parameter variable. A new piece of memory is created and a copy of [0] is placed into the memory of eggs

Pass by value

Check out this code, and let's assume it's pass by value

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```

[0]

ham

[0, 1]

eggs

eggs has the value 1 appended to it. This does not change the original ham

Pass by value

Check out this code, and let's assume it's pass by value

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```

[0]

ham

[2, 3]

eggs

eggs has now changed its value to [2,3]

Pass by value

Check out this code, and let's assume it's pass by value

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```

[0]

ham

[2, 3]

eggs

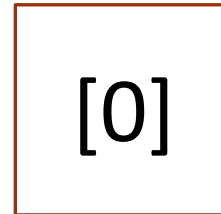
When we print ham, its value still remains as [0]

Pass by reference

Check out this code, and let's assume it's pass by reference

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



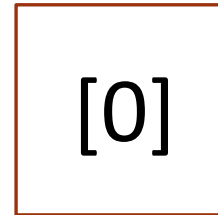
A new object is created in memory, and the variable ham is assigned to it

Pass by reference

Check out this code, and let's assume it's pass by reference

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



ham
eggs

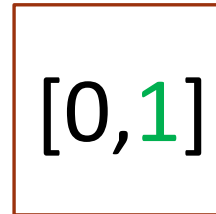
eggs is a reference to ham. Eggs and ham are both referencing the same object in memory. There is NO COPYING

Pass by reference

Check out this code, and let's assume it's pass by reference

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



[0, 1]

ham
eggs

eggs has the value 1 appended to it. This changes the original value ham is assigned to

Pass by reference

Check out this code, and let's assume it's pass by reference

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



ham
eggs

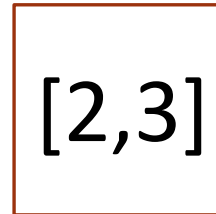
eggs has now changed its value to [2,3]. Again this changes the original value ham is assigned to

Pass by reference

Check out this code, and let's assume it's pass by reference

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



[2,3]

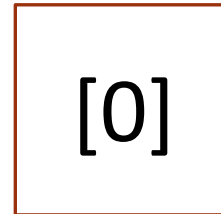
ham
eggs

When we print ham, its value was changed within the function to [2,3]. These changes are reflected even outside the function

Pass by ???

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



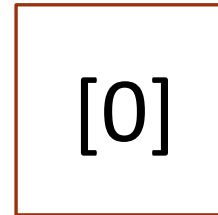
ham

A new object is created in memory, and the variable `ham` is assigned to it

Pass by ???

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



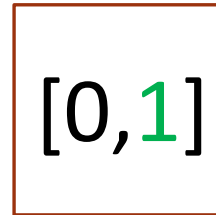
ham
eggs

eggs is a new variable that points to the same value ham is pointing at. Eggs and ham are both pointing at the same object in memory. There is NO COPYING

Pass by ???

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```



[0, 1]

ham
eggs

eggs has the value 1 appended to it. This changes the original value ham is assigned to

Pass by ???

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```

[0,1]

ham

[2, 3]

eggs

The object [2,3] is created in memory and eggs has now changed to point to it

Pass by ???

```
def spam(eggs):  
    eggs.append(1)  
    eggs = [2, 3]
```

```
ham = [0]  
spam(ham)  
print(ham)
```

[0,1]

ham

[2, 3]

eggs

When we print ham, its value is [0,1]

Pass by ???

Soooo, what is it called?

You might see some of the following ways to call it:

Call-by-Object

Call by Object Reference

Call by Sharing

Pass-by-object-reference

Some people don't bother giving it a name because it's an entirely different concept from pass by value or pass by reference

That's about it for today!

Next lecture:

SOLID Design Principles

Law of Demeter

