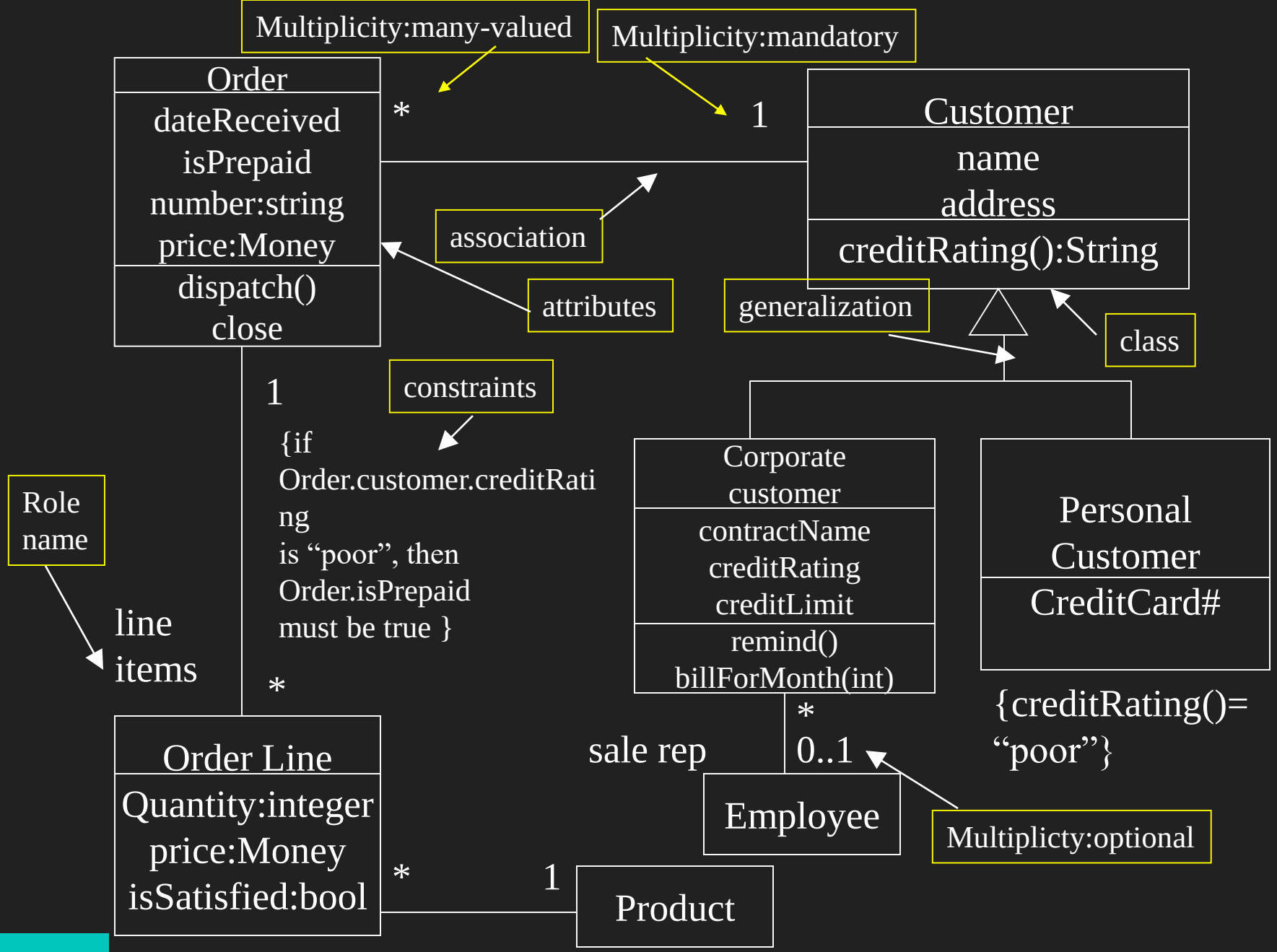
UML Class Diagram

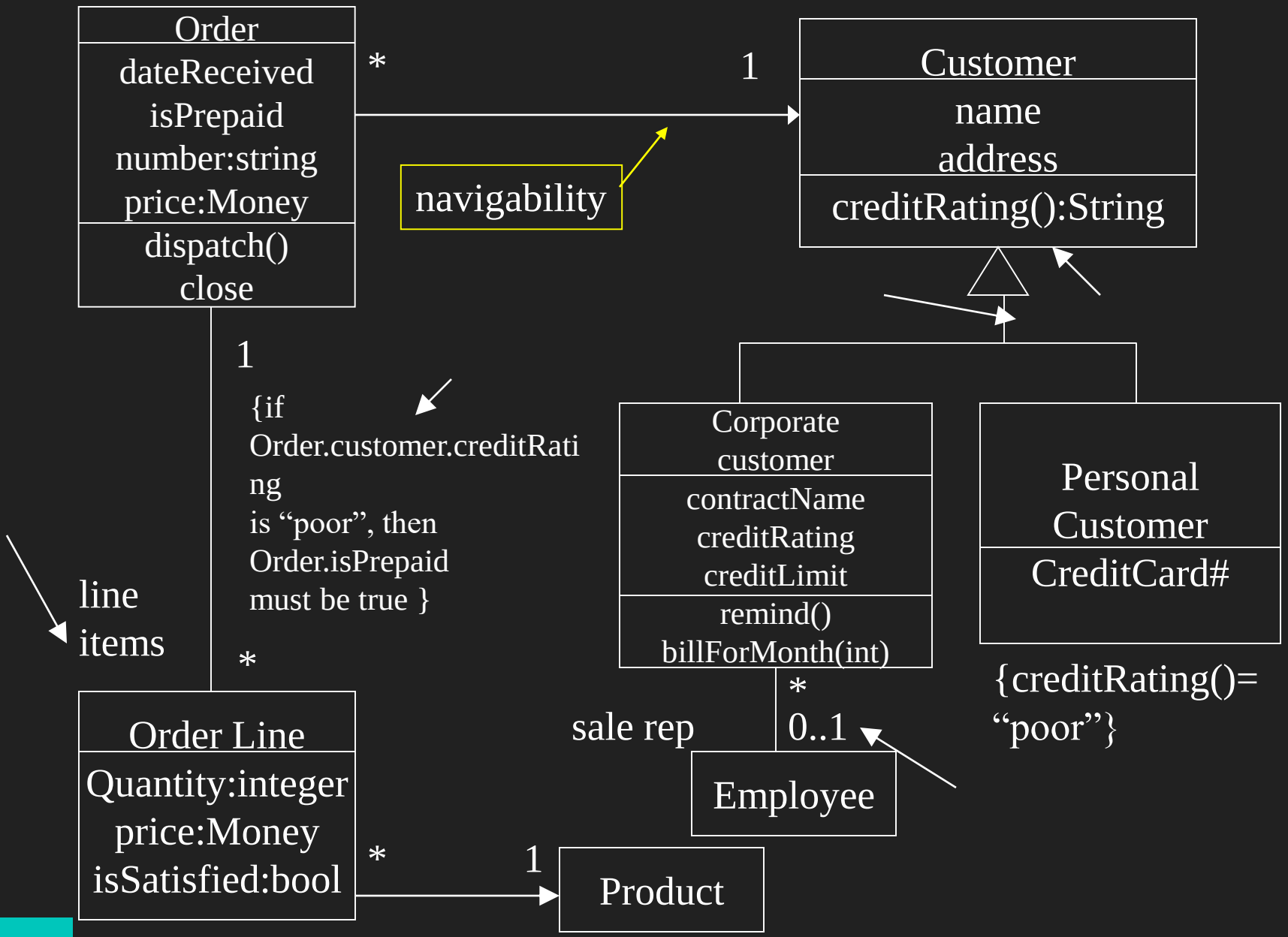
Class Diagram

- A class diagram describes the classes and the association (static relationship) between these classes (描述系統中物件的type(class)，以及這些class 的靜態關連.)
  - association (for example, a customer may rent a number of videos)
  - subtypes (a nurse is a kind of person)


Multiplicity:many-valued
Multiplicity:mandatory
Order
dateReceived
isPrepaid
number:string
price:Money
dispatch()
close
*
1
Customer
name
address
creditRating():String
association
attributes
generalization
class
1
constraints
{if
Order.customer.creditRati
ng
is “poor”, then
Order.isPrepaid
must be true }
Role
name
line
items
*
Corporate
customer
contractName
creditRating
creditLimit
remind()
billForMonth(int)
Personal
Customer
CreditCard#
{creditRating()=
“poor”}
*
0..1
sale rep
Employee
Multiplicty:optional
Order Line
Quantity:integer
price:Money
isSatisfied:bool
*
1
Product

navigability

- See next fig
- In specification model, an Order has a responsibility to tell you which Customer it is for, but not conversely
- In implementation diagram, this would indicate that Order contains a pointer to Customer
- UML 2.x remove this feature


Order
dateReceived
isPrepaid
number:string
price:Money
dispatch()
close
*
1
Customer
name
address
creditRating():String
navigability
1
{if
Order.customer.creditRati
ng
is “poor”, then
Order.isPrepaid
must be true }
Corporate
customer
contractName
creditRating
creditLimit
remind()
billForMonth(int)
Personal
Customer
CreditCard#
{creditRating()=
“poor”}
line
items
*
Order Line
Quantity:integer
price:Money
isSatisfied:bool
*
sale rep
0..1
Employee
*
1
Product

Generalization

- Generalization (In OOP term, is inheritance)
- see personal and corporate customers
- Using polymorphism, the corporate customer may respond to certain commands differently from another customer

Operations

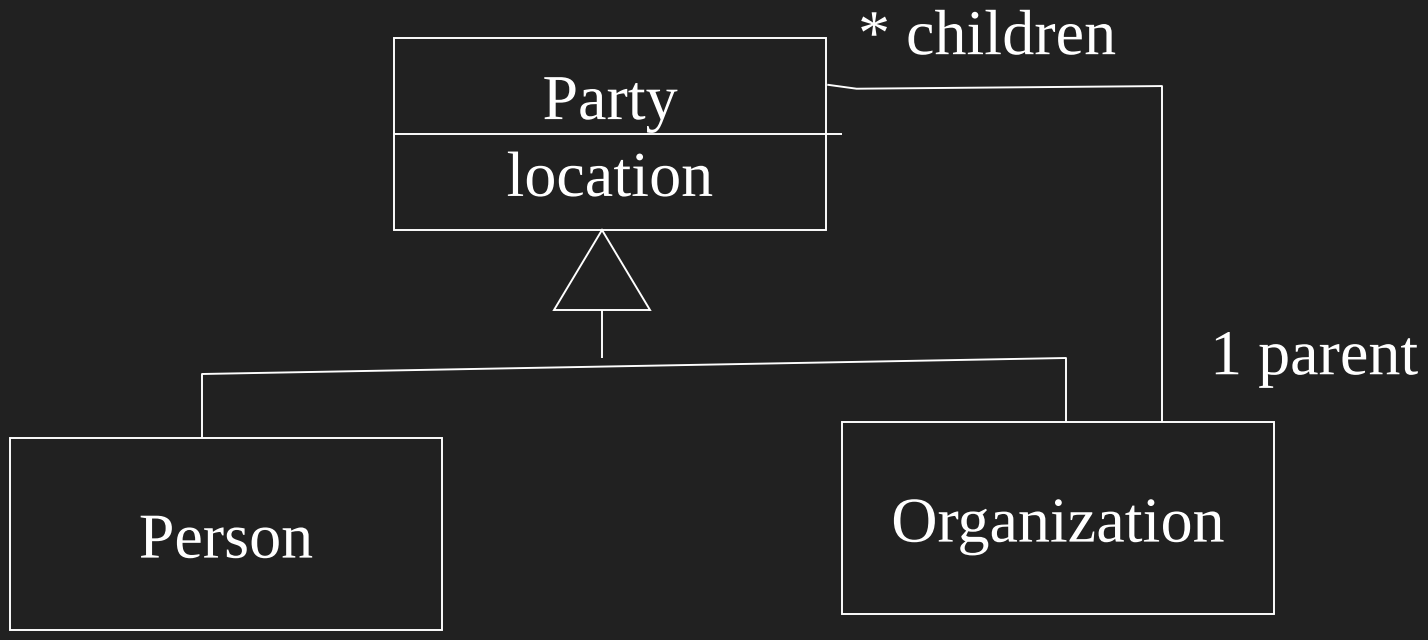
- Most obviously correspond to the methods on a class
- UML syntax is

visibility name (parameter list): return-type-expression {property string}

Class Diagrams: Advanced Concepts

Object Diagram

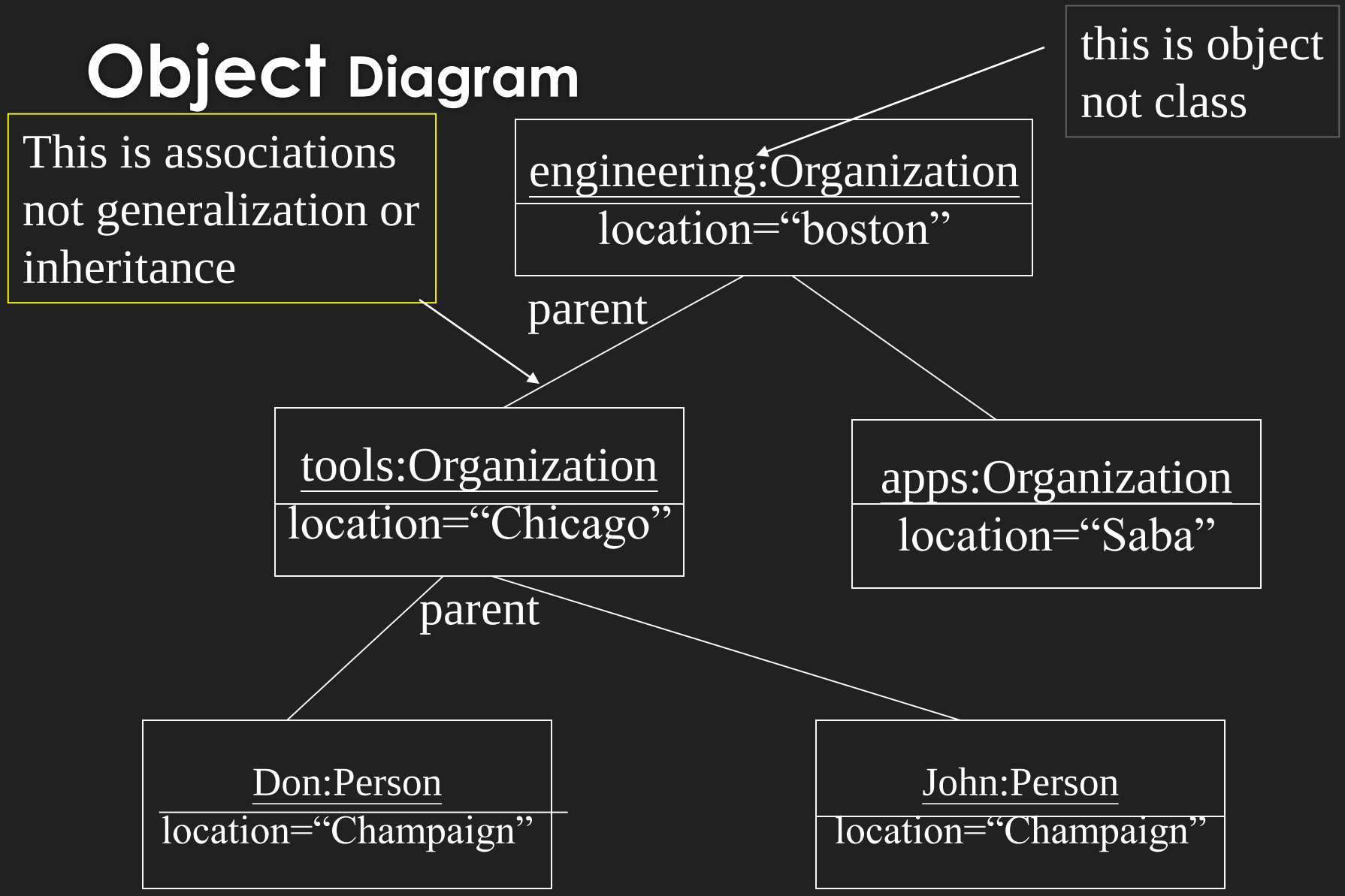
- A snapshot of the objects in a system at a point in time
- Often called instance diagram
- Let's see the class diagram first



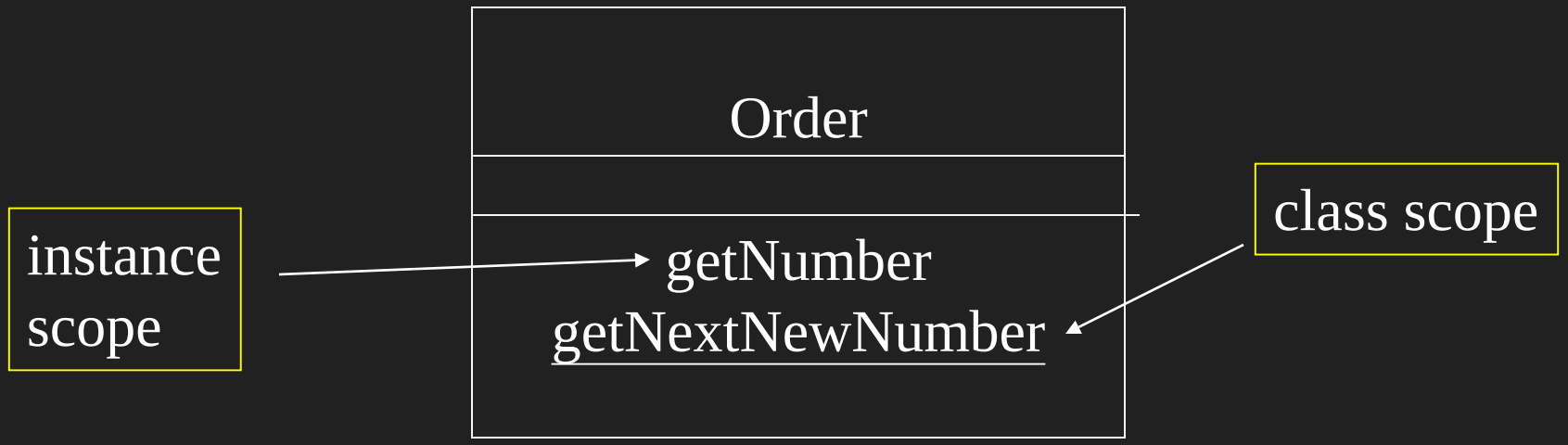
Party::dosomething()

```
Class party {
    location ;
    abstract  int computeSalary
}
Class person : party {
   computeSalary() {
      return his personal wages ;
}
Class organization : party {
    vector<party *> children ;
    computeSalary () {
        for each o in children
            total += o.computeSalary() ;
    }
}
```

Object Diagram

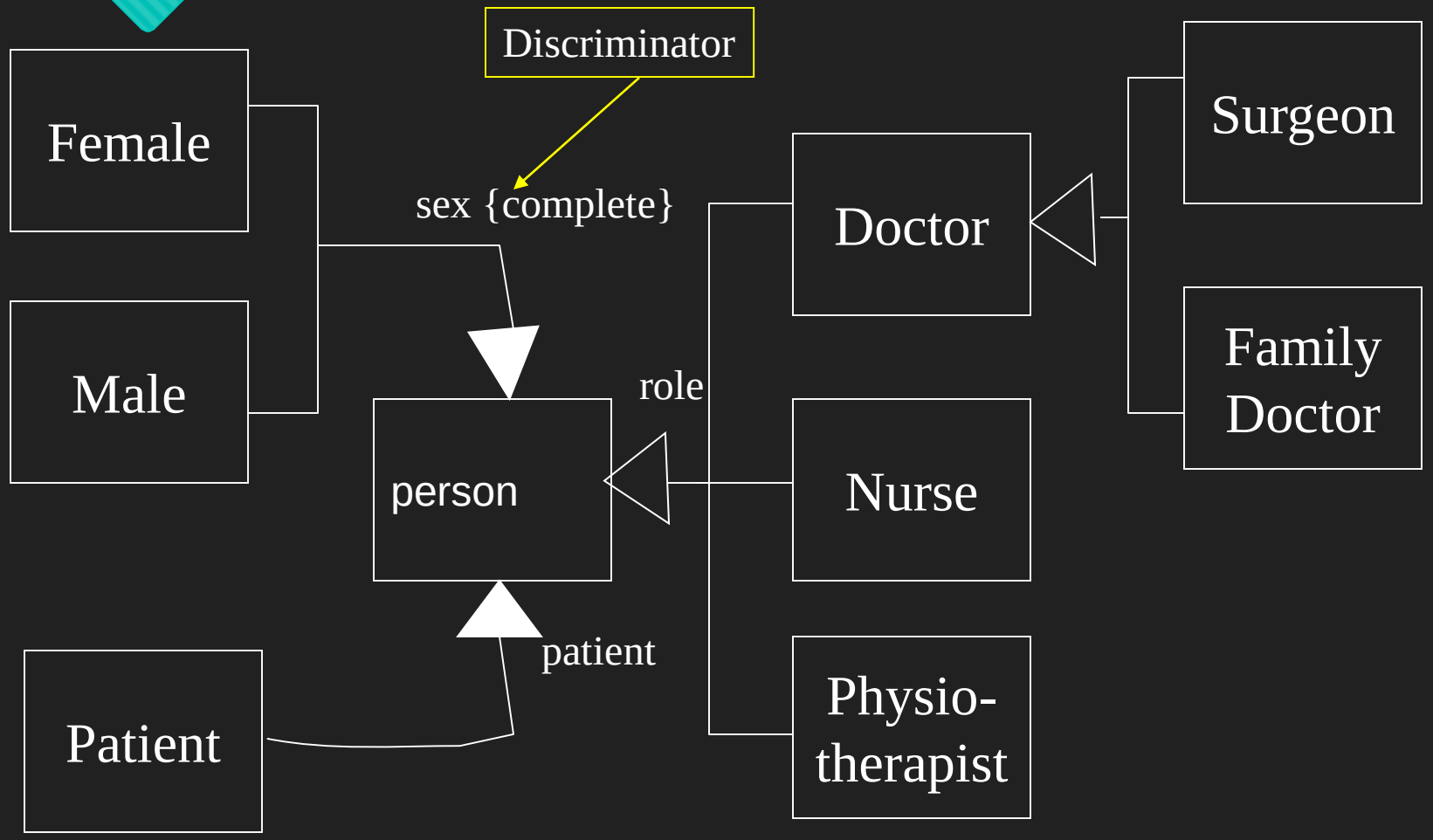


Class Scope Operations and Attributes



Class scope is equivalent to static members in C++

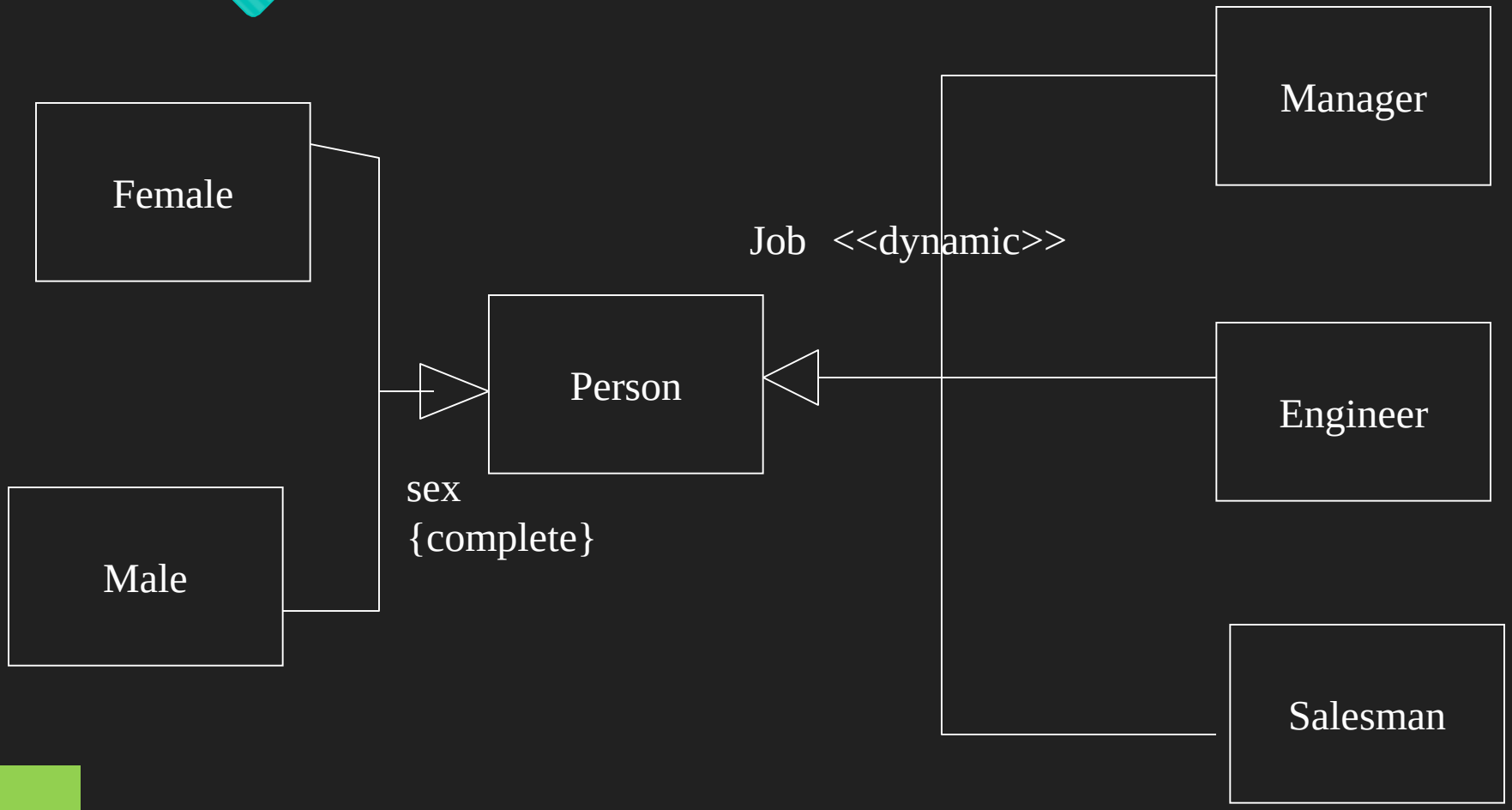
Multiple and Dynamic Classification



Multiple Classification

- Don’ t confuse with multiple inheritance.
- You should make it clear which combinations are legal by using a discriminator

Dynamic Classification

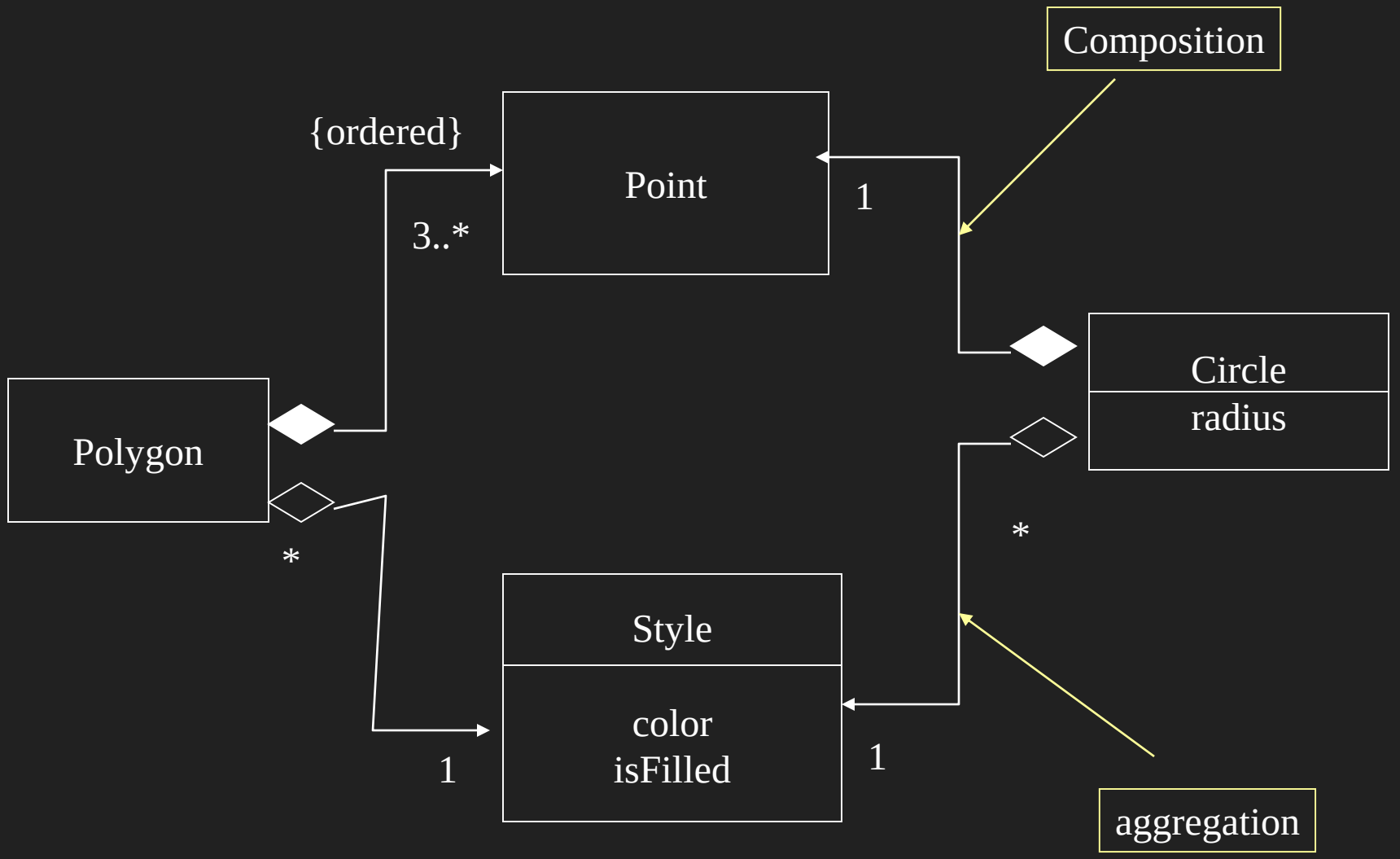


Dynamic Classification

- allow object to change type within the subtyping structure
- useful for conceptual modeling
- How to implement it? See Fowler 1997 video tape rental example (state design pattern)

Aggregation and Composition

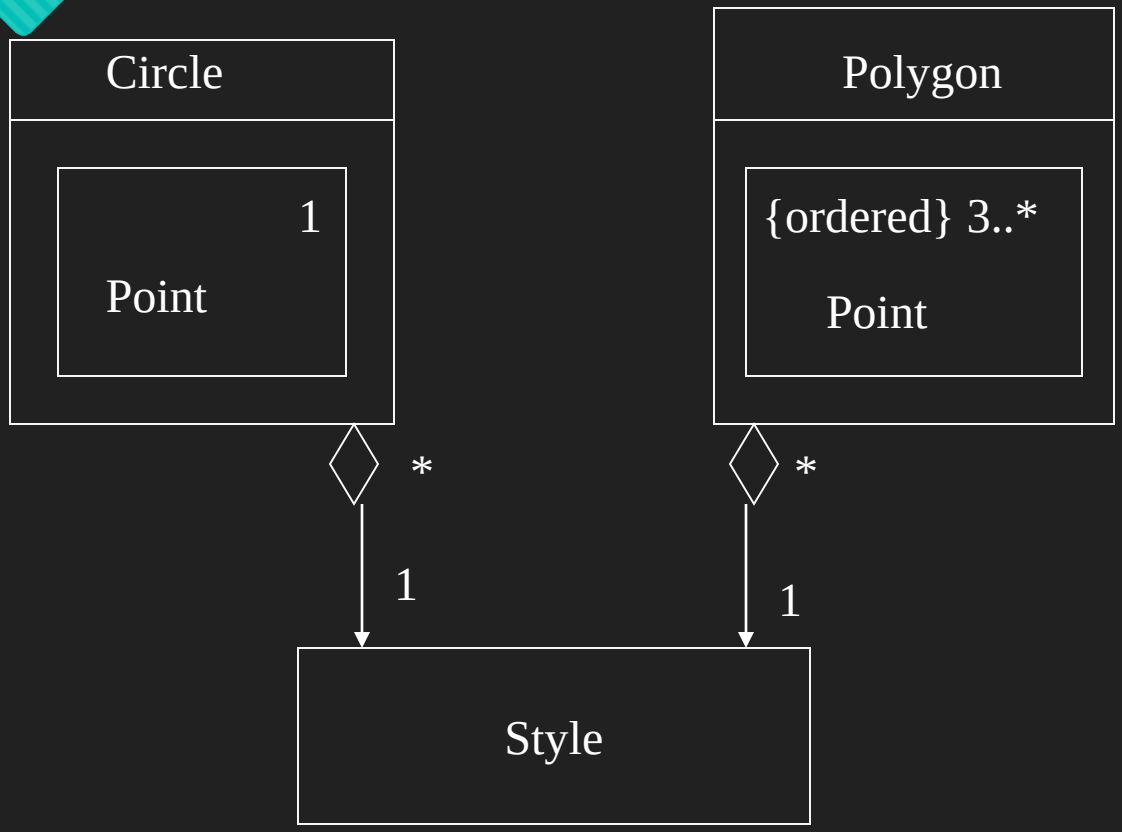
- Aggregation is the part-of relationship
- difficult things – what is the difference between aggregation and association?
  - It is vague and inconsistent
  - Anyway, UML decide to include aggregation and a stronger variety of aggregation called *composition*
- *Basically an aggregation/composition relation is still an association. They are used to emphasize the type of association*


Composition
{ordered}
Point
3..*
1
Polygon
Circle
radius
*
*
Style
color
isFilled
1
1
aggregation

Composition

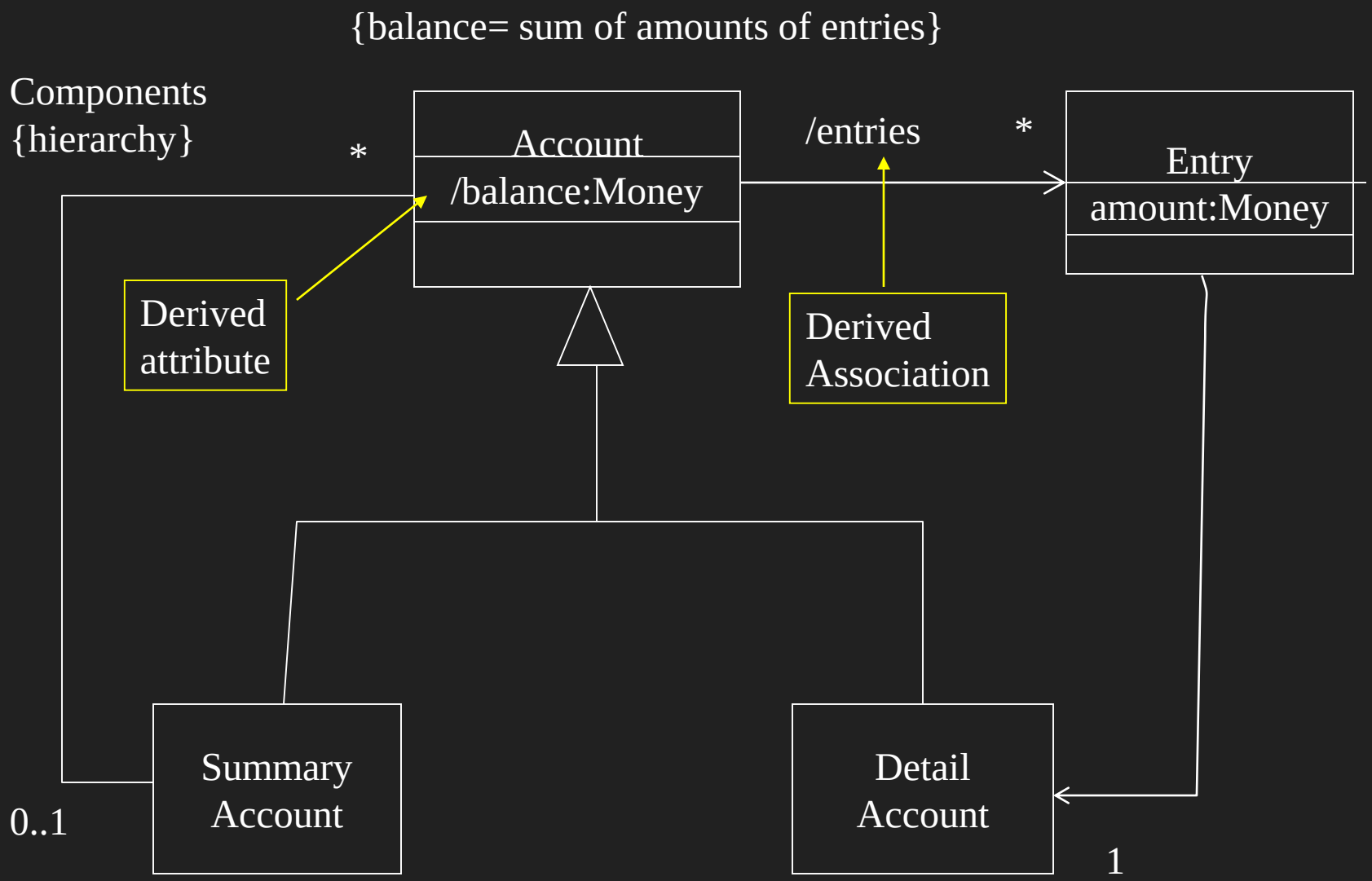
- With composition, the part object may belong to only one whole; further, the parts are usually expected to live and die with the whole
- deletion of the whole is considered to cascade to the part
- In previous graph, deleting a polygon would caused its associated *Points* to be deleted, but not the associated *Style*.

Alternative Notation for Composition



Derived Associations and Attributes

- Derived Associations and derived attributes can be calculated from other associations and attributes.
  - for example, an age attribute of a Person can be derived if you know that person's birthday.


{balance= sum of amounts of entries}
Components
{hierarchy}
*
Account
/balance:Money
/entries
*
Entry
amount:Money
Derived
attribute
Derived
Association
Summary
Account
Detail
Account
0..1
1

Note

- Entry objects are attached to detail Accounts
- The balance of an Account is calculated as the sum of Entry accounts
- A Summary Account’ s entries are the entries of its components determined recursively

Interfaces and Abstract Classes

- Programming language that use a single construct, the class, which contains both *interface* and *implementation*.
- When you subclass, you inherit both.
- A pure interface, as in Java, is a class with no implementation and, therefore, has operation declarations but no method bodies and no fields.

For example

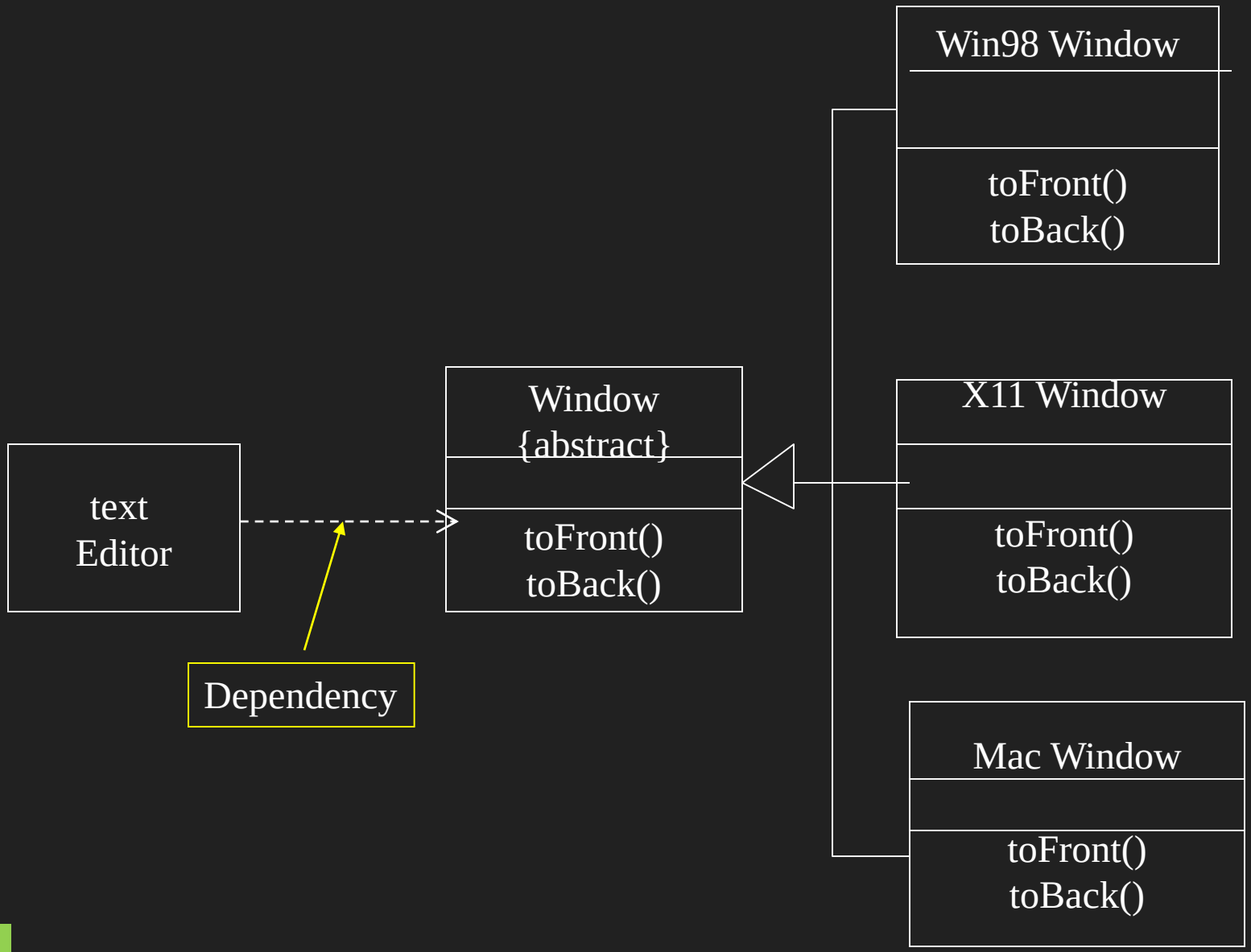
```
interface Stack {
     boolean Push(Object);
     Object   Pop();
}

Class MyStack implements Stack

    boolean Push(Objects) {
       ……..
    }

    Object Pop() {
       ……..
       return xxxx ;
    }
}
```

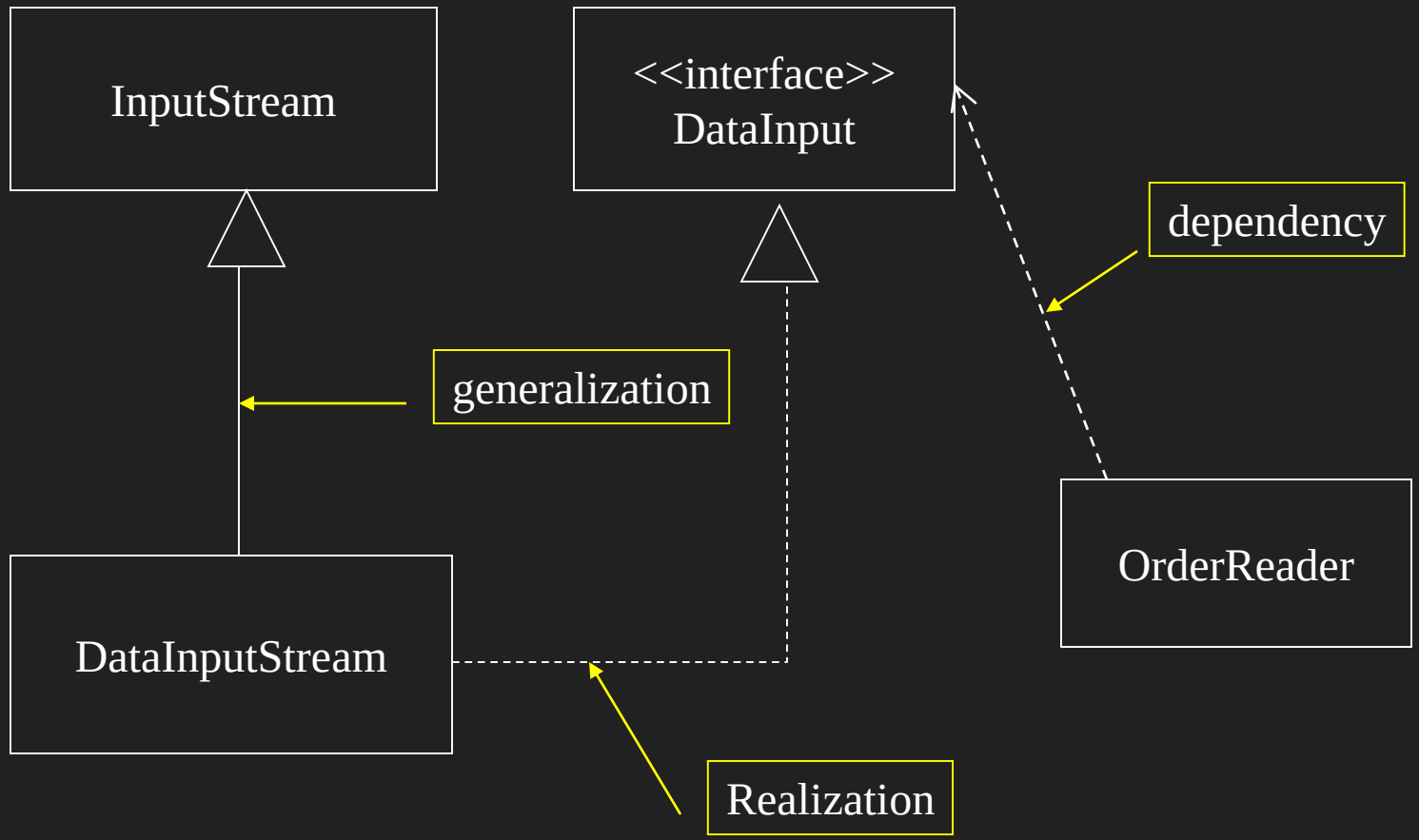
Somewhere in initialization

```
Stack  S =  new MyStack() ;
```

Somewhere in the code

```
S.push(a);
S.push(b);
S.pop(a);
S.pop(b);
```


Win98 Window
toFront()
toBack()
Window
{abstract}
toFront()
toBack()
X11 Window
toFront()
toBack()
text
Editor
Dependency
Mac Window
toFront()
toBack()


InputStream
<<interface>>
DataInput
dependency
generalization
OrderReader
DataInputStream
Realization

Abstract Class and Interface

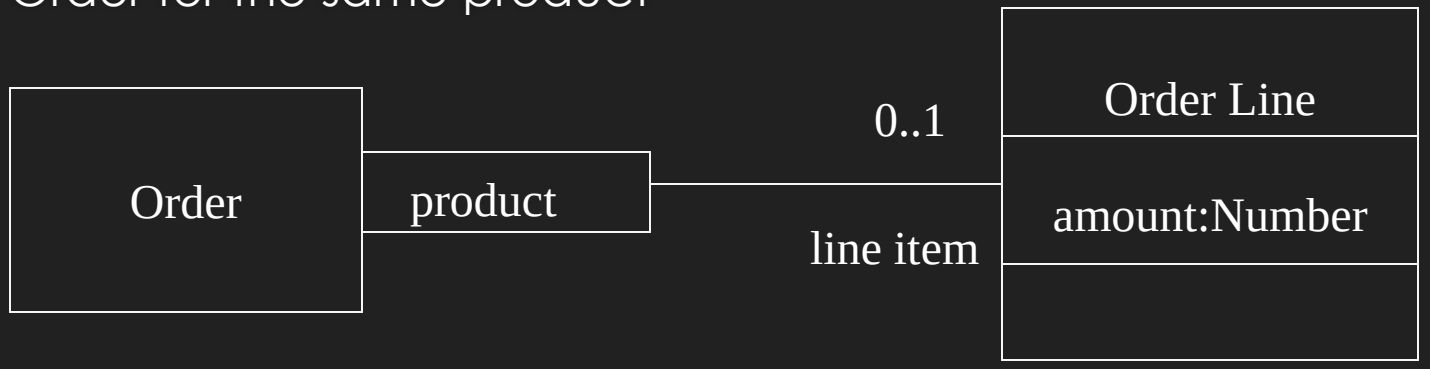
- two are similar
- abstract class allows you to add implementation of some of the methods
- an interface forces you to defer definition of all methods

Classification and Generalization

- People often talk subtyping as the “is a” relationship
- beware of that way of thinking
- for example
  1. Shep is a border Collie
  2. A border Collie is a Dog
  3. Dogs are animals
  4. A border Collie is a Breed (品種)
  5. Dog is a Species (物種）
- if you try combination 2 and 5 “A border Collie is a Species”. This is not good
- The reasons?
  - some are classification and some are generalization
- Generalization is transitive and classification is not
- “Dog are a kind of animals” is better

Qualified Associations

- equivalent to associative arrays, maps, and dictionaries
- an example, there maybe one Order Line for each instance of Product
- Conceptually, you cannot have two Order Lines within an Order for the same product

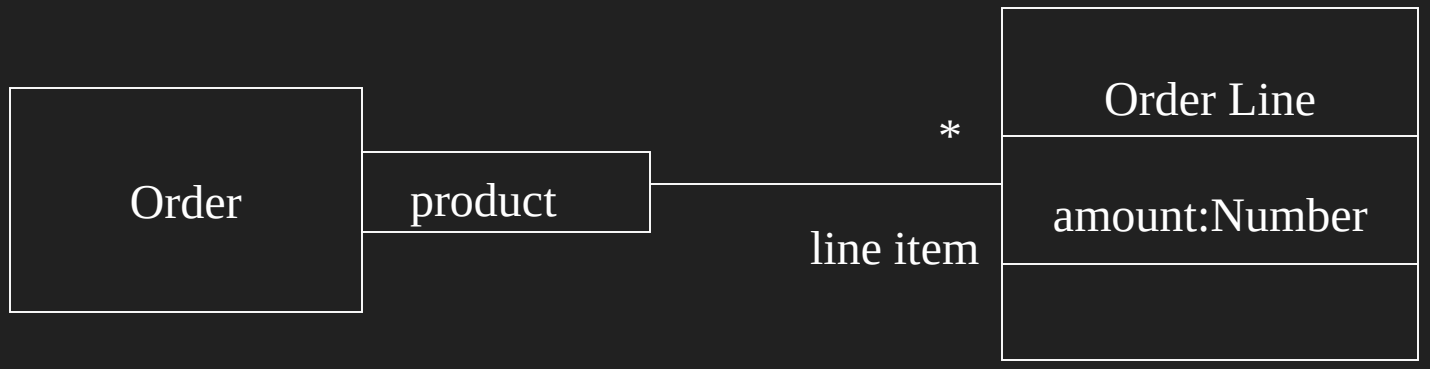


Qualified association would imply an interface like

```
class Order {

        public OrderLine getLineItem (Product aProduct);;

       public void addLineItem (Number amount, Product
forProduct);
}
```

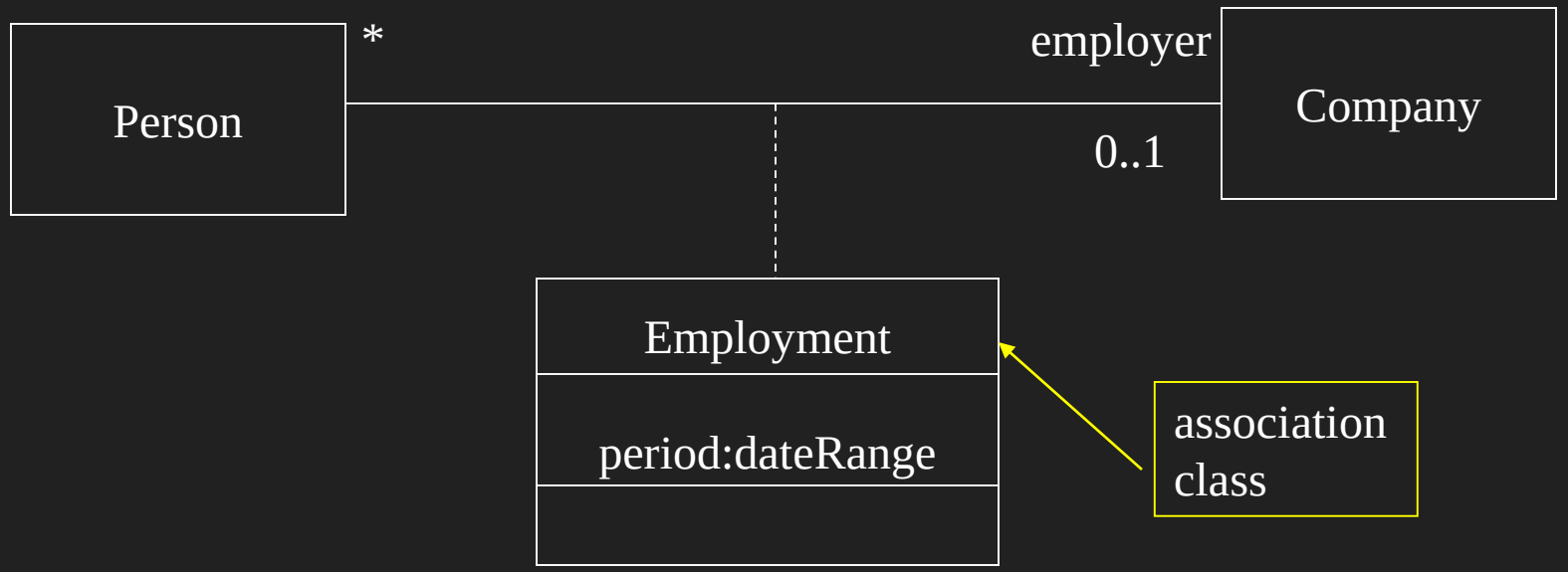
- You can have multiple OrderLines per Product but access to the Line Items is still indexed by Product
- use an associative array or similar data structure to hold the order lines

```
Class Order {
    private Map _lineItems ;
```



Association Class

- Association class allow you to add attributes, operations, and other features to association



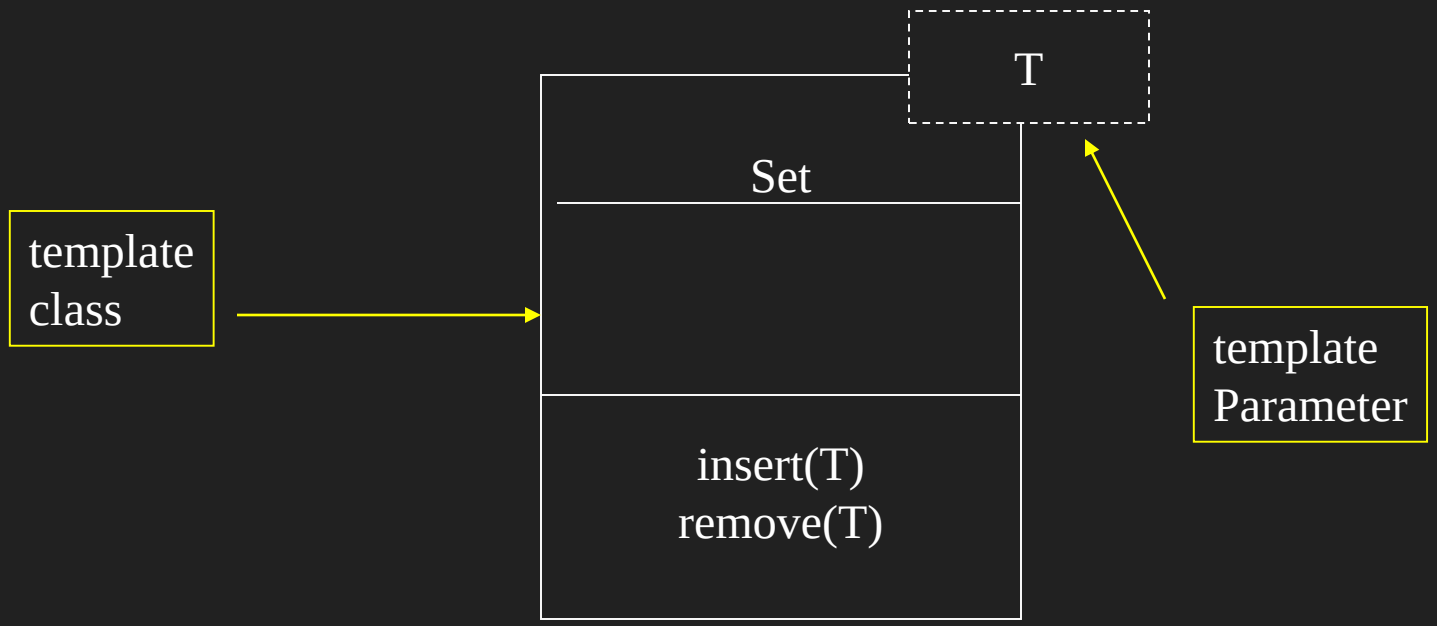
- A person may work for a single company
- We need to keep information about the period of time that each employee works for each Company
- You can redraw: make Employment a full class in its own right

Parameterized Class

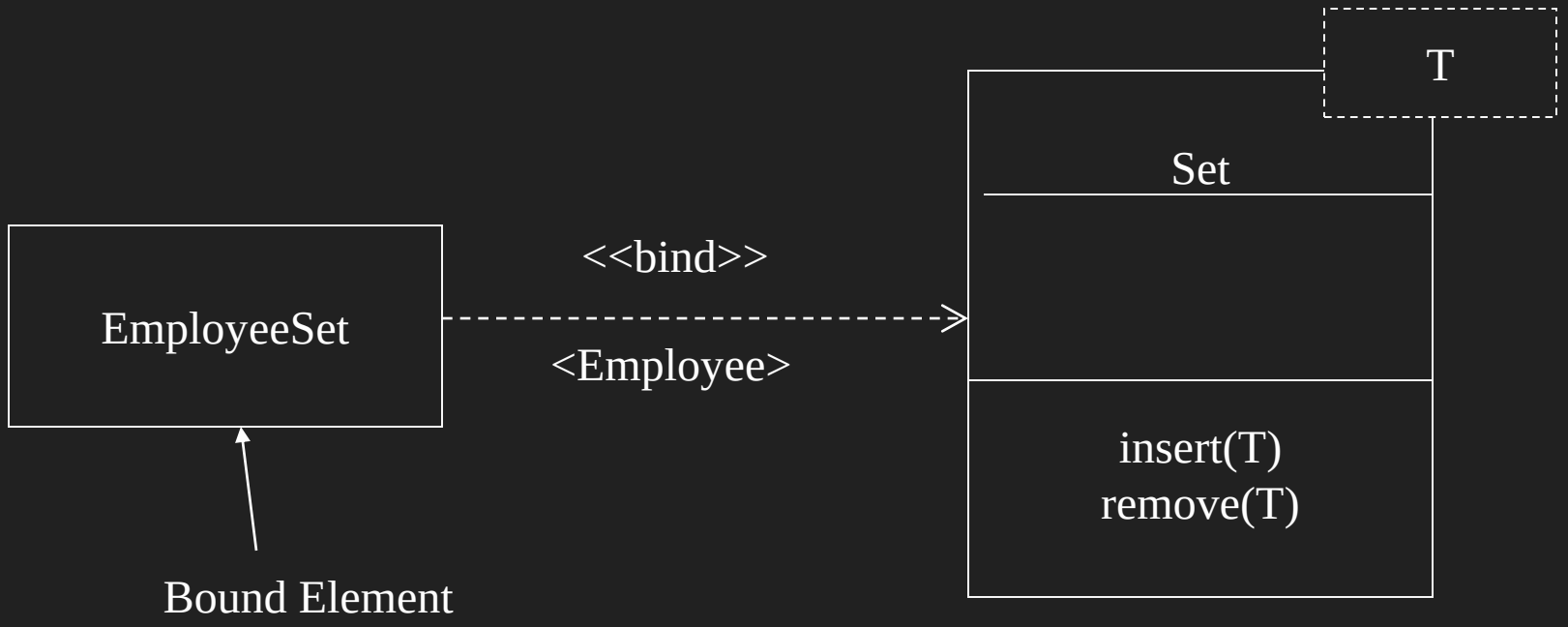
- Several language, noticeably C++, have the notion of a parameterized class or template
- ex

```
class set <T> {
    void insert (T newElement);
    void remove(T anElement);
…
…
Set <Employee> employSet ;
```

A define a parameterized class in UML



A use of a parameterized class



Visibility

- C++
  - A public member is visible anywhere in the program and may be called by any object within the system
  - A private member may be used only by the class that defines it
  - A protected member may be used only by (a) the class that defines it or (b) a subclass of that class
- In Java
  - a protected member may be accessed by subclasses but also by any other class in the same package as the owning class
- C++
  - one C++ method or class can be made a friend of a class. A friend has complete access to all members of a class