



# Sunbeam Infotech

Exploring new ideas, Reaching new heights!



# Hierarchy

- Multiple classes can be related with each other in a hierarchial fashion.

- There are four types of hierarchies

- Is-a (generalization) → Inheritance

- Has-a (association)

- Use-a (dependency)

- Creates-a (instantiation)

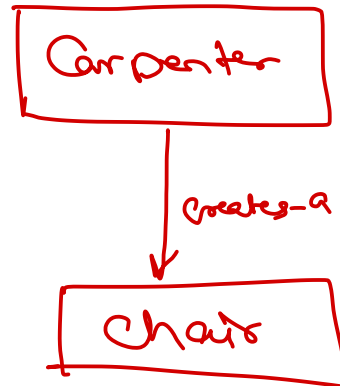
(generalization)

Fruit  
↑ is-a  
Apple

↑ is-a  
Kashmiri  
Apple

(specialization)

Processor  
↑ has-a  
mother board  
↑ has-a  
Computer



- Is-a hierarchy represents inheritance. All members of a class are inherited to inherited class.

child

parent

- Has-a hierarchy represents association. An object is attached with or part of another object.

- Use-a hierarchy represents an object is dependent on another object for one/more functionalities.

- Creates-a hierarchy represents an object is created by another object.



# Typing

- Typing may be weak or strong typing.
- Few programming languages are type-less. In few types are inferred at compile time or run time.  
*← Python*
- Few programming languages have weak type check, while few have strong type check.

*shell script*

*Scala*

*C*

*C++*

*Java*

- Typing may be static or dynamic. It is also referred as polymorphism.
- Poly-morphism means "taking many forms".
- Static or Compile-time polymorphism
  - Methods with same name and different arguments.
  - Method to be called is decided by the compiler, depending on arguments.
- Dynamic or Run-time polymorphism
  - Method is redefined in inherited class with same prototype. *→ same name, same args*
  - Method to be called is decided at runtime, depending on type of object.



# Method overloading

- Methods with same name and different arguments.
- Based on arguments each method will be given different name internally by the compiler, called as "Name mangling". *terminal > nm exe path*
- Method to be called is decided by the compiler, depending on arguments.
- Return type of method is not considered (because collecting return value is not compulsory).
- Constructors can be overloaded, but destructors cannot be overloaded.

functions must be in same scope.  
(same class or global)

```
int sum(int a, int b) {  
    return a + b;  
}
```

```
int sum(int a, int b, int c) {  
    return a + b + c;  
}
```

*math() {  
 r1 = sum(1, 2);  
}*

*r2 = sum(1, 2, 3);*

*false polymorphism*

*different args*

*count  
type  
order*


```
sqn(int) {  
    sqn(float);  
}
```

```
- run(int, float) {  
    }  
- run(float, int) {  
    }
```

```
int div(int a, int b) {  
    }  
double div(int a, int b) {  
    }
```

*int r1 = div(12, 3);  
double r2 = div(12, 5);*



{ date d1;  Param less  
date d2(1,1,2000);

} param. 

{ date \*p1 = new date;  
date \*p2 = new date(1,1,2000);

delete p1; → No args. passed  
delete p2;

}

fun1.c

```
fun(10);
```

fun2.cpp

```
void fun(int a){  
    ||  
}
```

fun1.c

error: fun is undefined.

```
fun(10);
```

```
extern "C"  
void fun(int a){  
    ||  
}
```

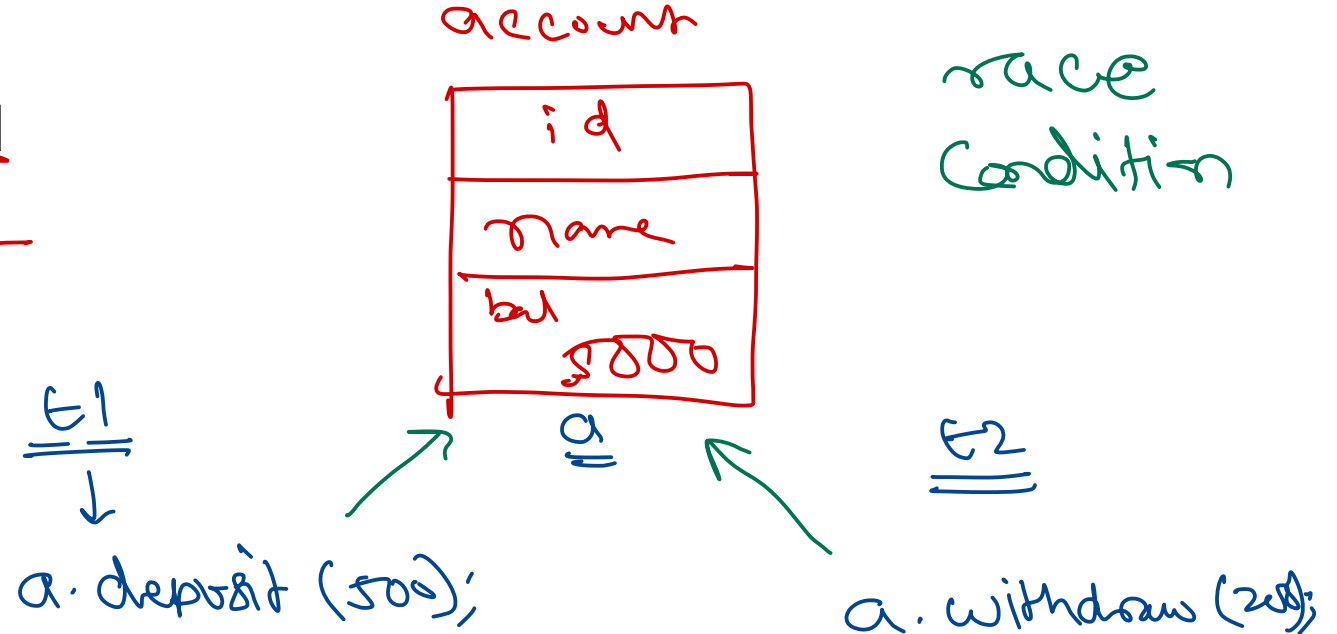
Suppress  
name  
mangling  
↓  
fun();

# Concurrency

- Concurrency deals with how object behaves when it is accessed/modified by multiple threads at the same time.
- In other words, it decides how race conditions are handled or synchronization is done.
- Few programming languages offer built-in keywords and APIs for multi-threading & synchronization.

↳ e.g. java → "synchronized".

In C++, it is done using OS system calls



# Persistence

- Storing state of object so that it can be recreated outside its scope.
- State of object can be stored in file, database or other medium.
- Few programming languages provide APIs for serialization & de-serialization.

↳ ex. Java, C#, ..

C → FILE \*

In C++  
data - can be saved into  
file in customized way.

↓  
(no standard  
method).

ifstream → to read from  
file.

ofstream → to write into  
file.

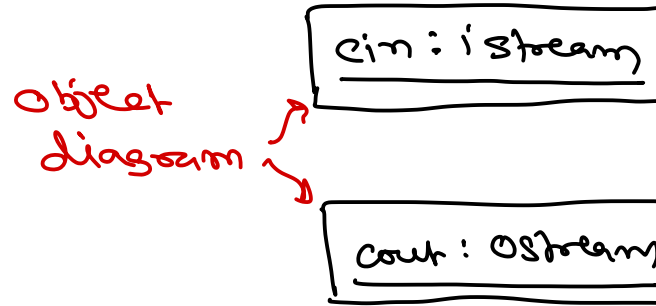
fstream → to read/write  
from file.





# UML diagrams

- Class diagram
- Association
- Aggregation
- Composition
- Inheritance
- Dependency

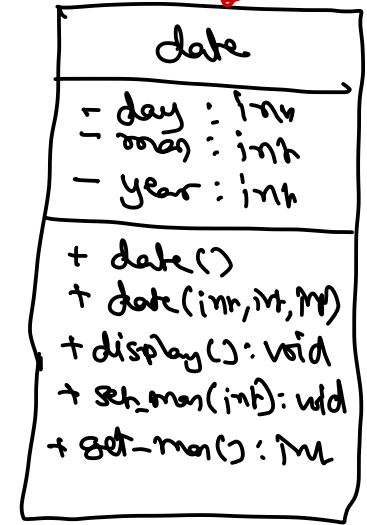


- private  
+ public  
# protected

short class,  
diag



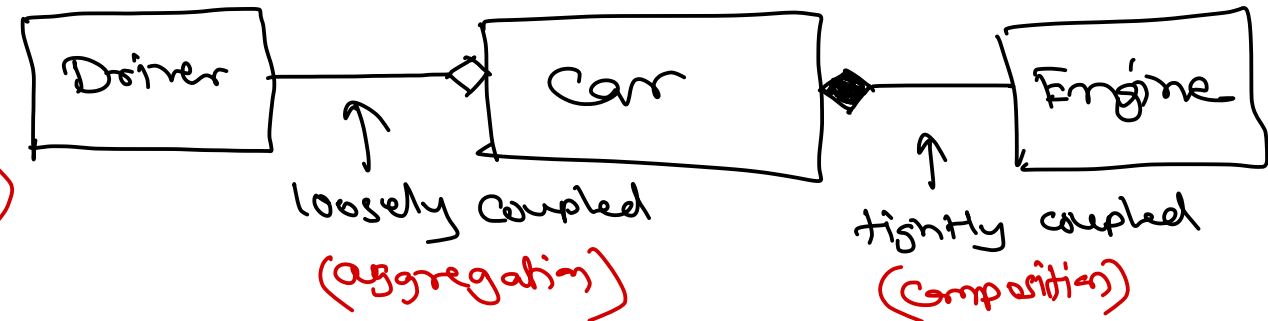
detailed class diag



① requirement analysis } OOA

② design } OOD → UML

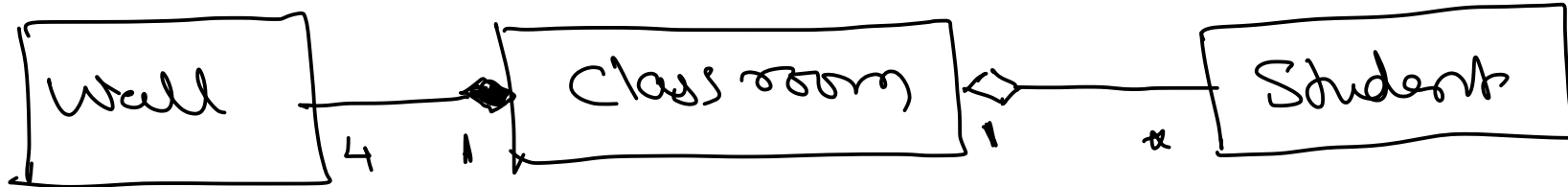
③ implementation } OOP



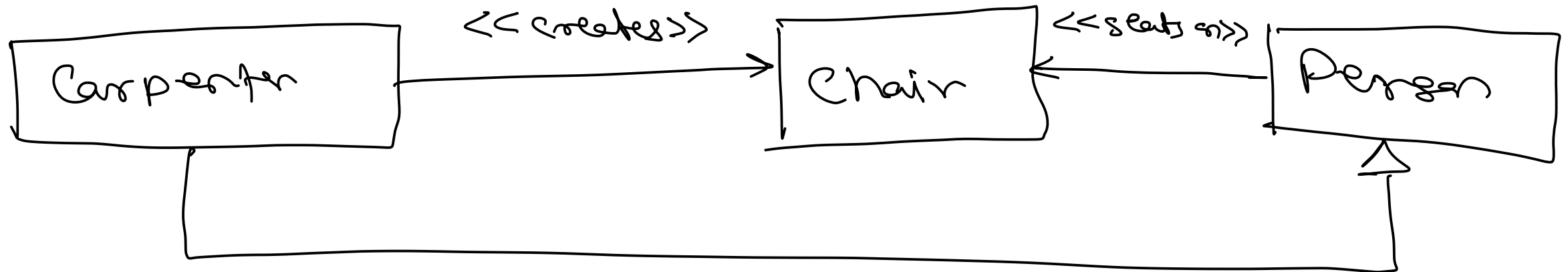
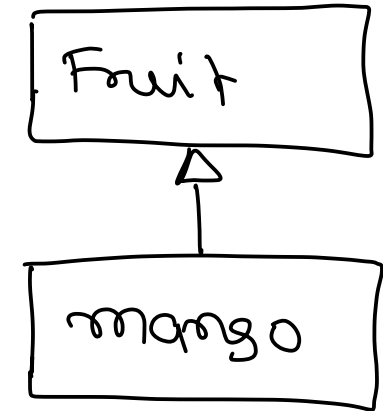
# UML diagrams

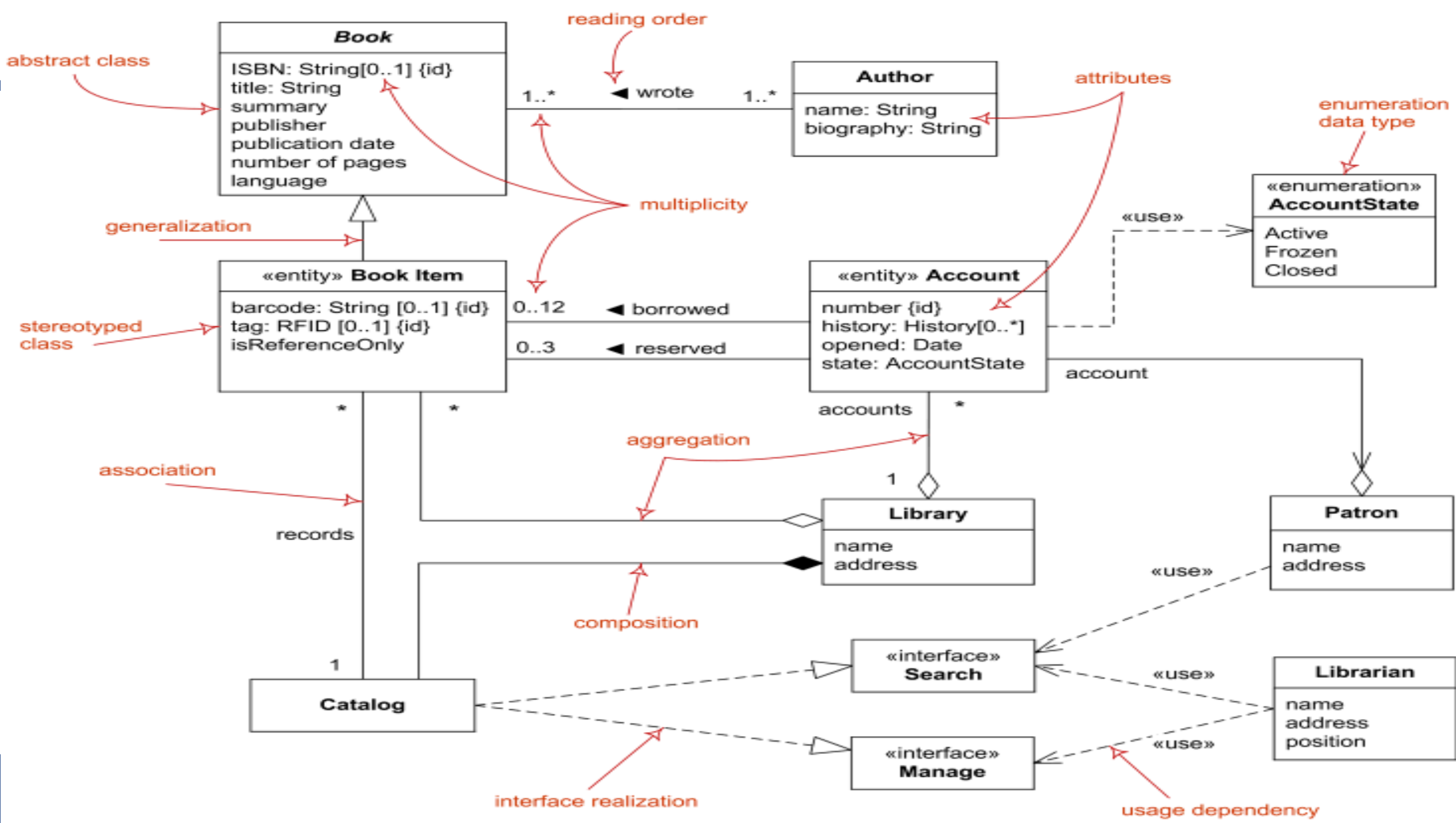
Composition

Aggregation



Cardinality of association relation.





# Association

has-a

- Association represents association (generic relation) between two classes.
- Specifically it can be
  - Aggregation: Loose coupling
    - Represent has-a relation.
  - Composition: Tight coupling
    - Represent part-of relation.
- Association can also have cardinality.

• 1-1

• 1-\*

• \*-1

• \*\_\*

```
class engine {  
    =  
};
```

```
class car {  
    engine e;  
    driver d;  
};
```

```
class driver {  
    =  
};
```



# Composition

- Inner object is part-of outer object.
- e.g. Room has a wall.
- e.g. Person has a birthdate.\*
- Size of outer object includes size of inner object.
- Inner object can be initialized using member initializer list.
- While creating outer object, inner object constructor is executed before constructor of outer object.
- Destructor execution sequence is reverse.

```
class date {  
    int day;  
    int month;  
    int year;  
    =
```

```
};
```

```
date d1;
```

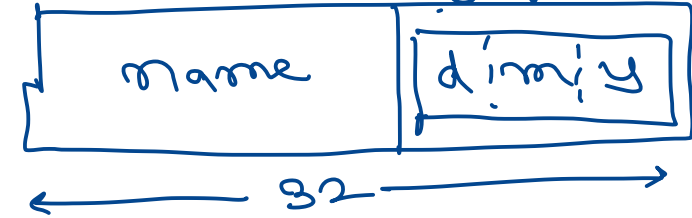


→ by default paramless ctor of inner object is executed

```
class person {  
    char name[20];  
    date birth;  
    =
```

```
};
```

```
person p1;  
    birth
```



```
class date {
```

```
=
```

```
};
```

```
class address {
```

```
=
```

```
};
```

```
class person {
```

```
char name[20];
```

```
date birth; ①
```

```
address addr; ②
```

```
public:
```

```
person(char *n, ---, ---)
```

```
: addr(---), birth(---)
```

```
{
```

```
=
```

```
}
```

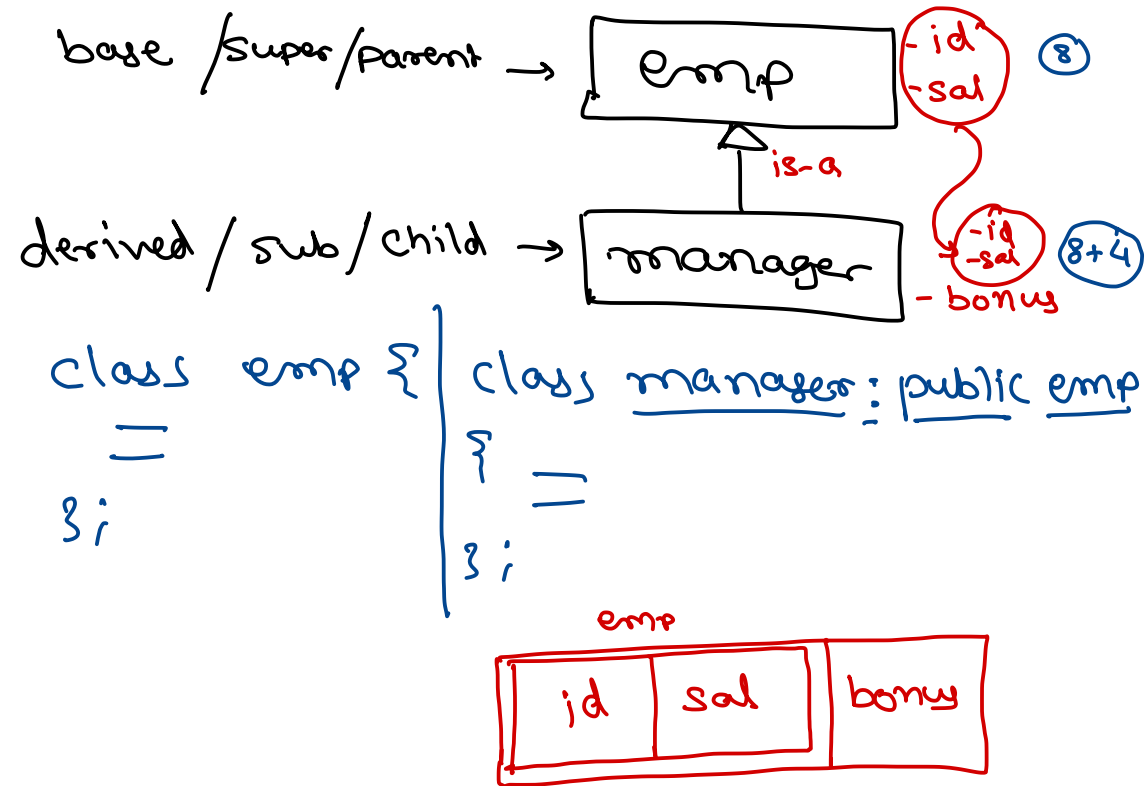
```
};
```

order of constructor execution of inner objects is same as their declaration in outer object/class.

# Inheritance

- Inheritance is implementation of generalization concept.
- e.g. Mango is a fruit.
- e.g. Employee is a person.
- All members of parent class are inherited into child class.
- Parent class: base or super-class.
- Child class: derived or sub-class.
- In C++, size of derived class includes size of base class.
- While creation base class constructor is executed before derived class.

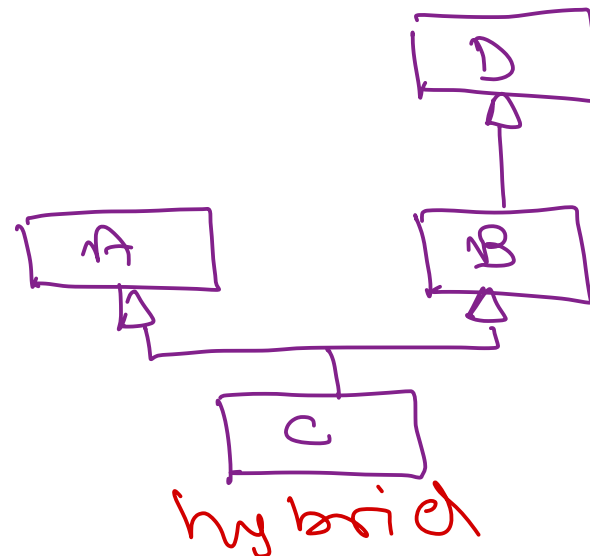
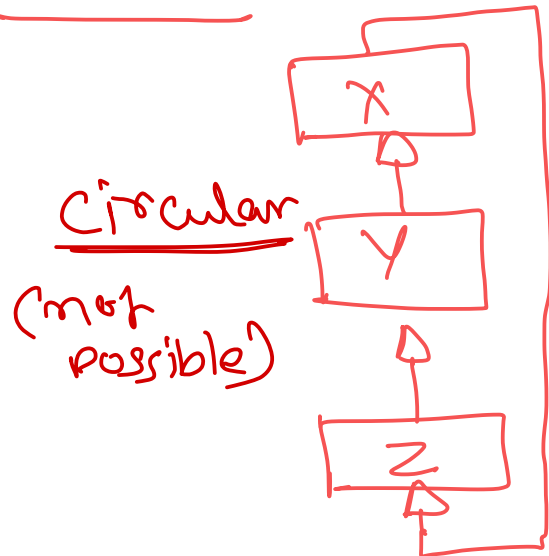
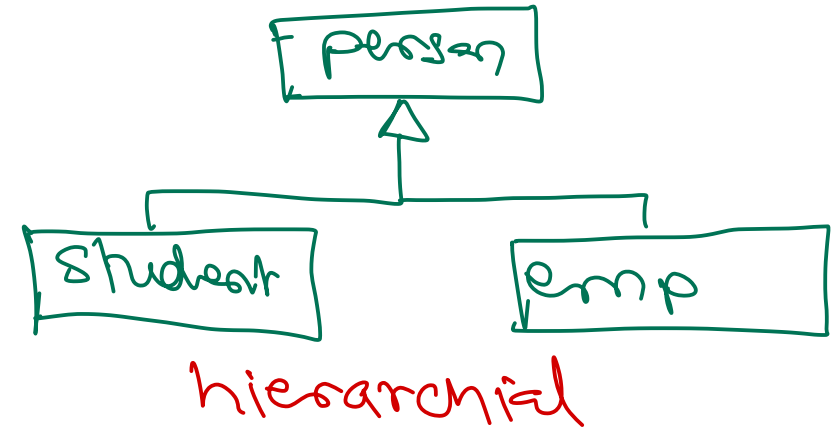
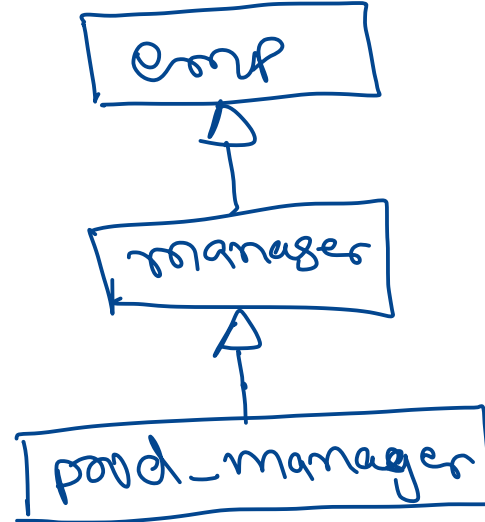
by default base class constructor is called.



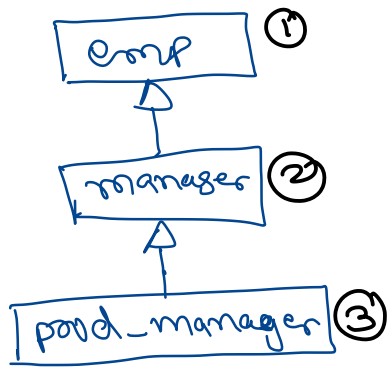
# Types of inheritance

multi-level

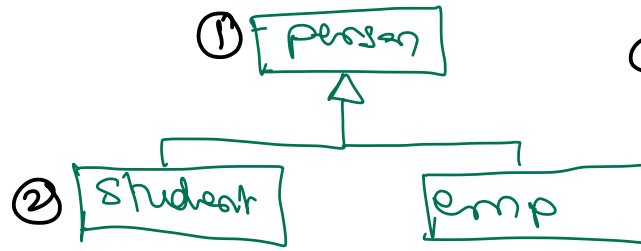
- Single
- Multiple
- Multi-level
- Hierarchical
- Hybrid
- Circular







prod\_manager pm;



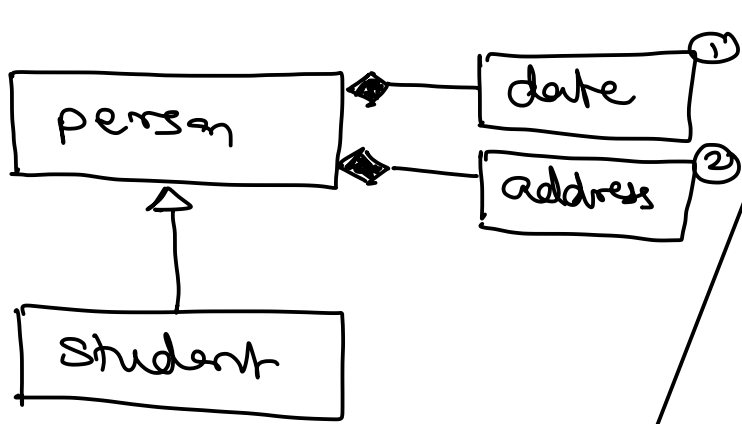
Student s1;



Sales\_mgr sm;

```

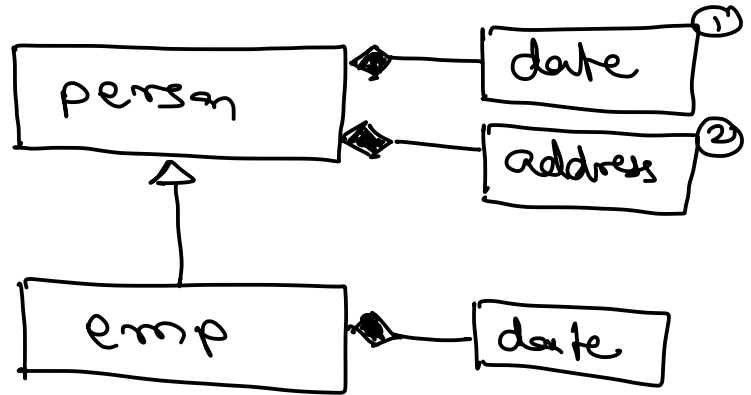
class Sales_mgr :
    public Salesman, ①
    public manager ②
}
//
};
  
```



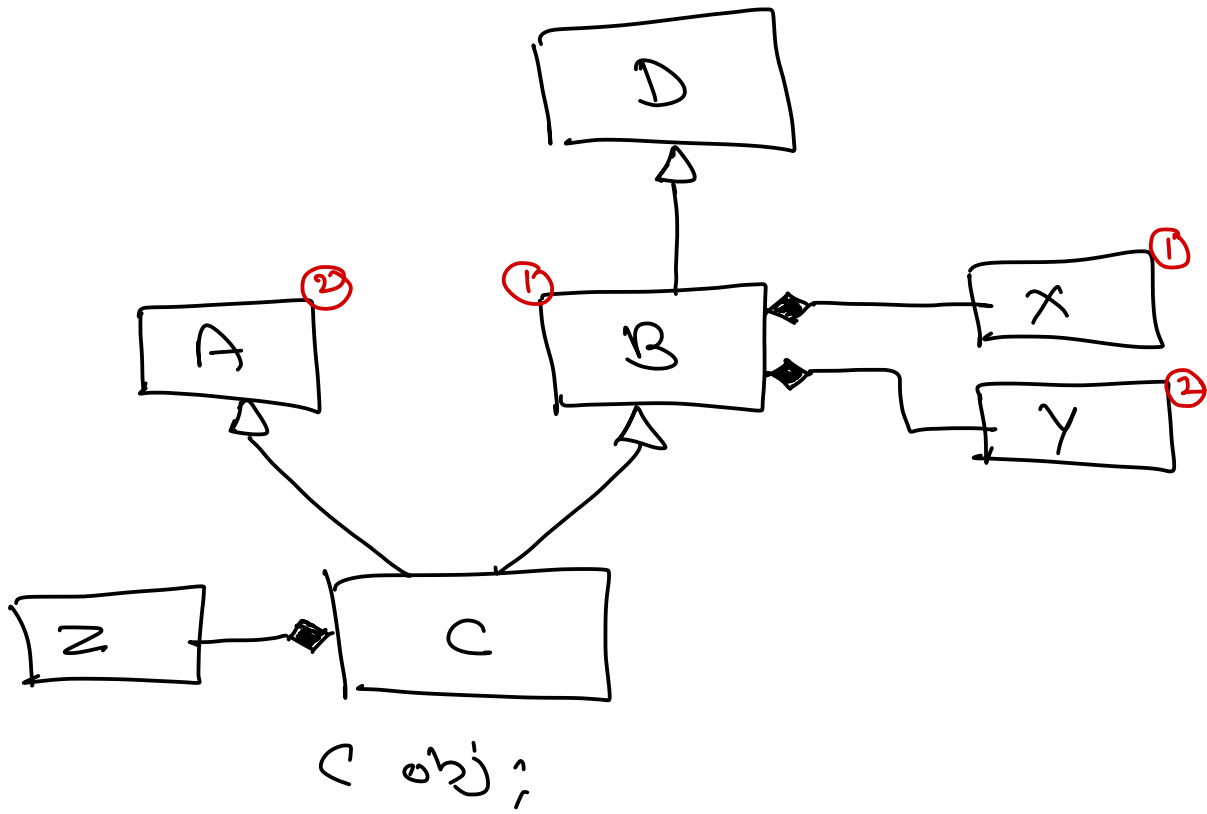
Student s1;

- ① date
- ② address
- ③ person
- ④ student

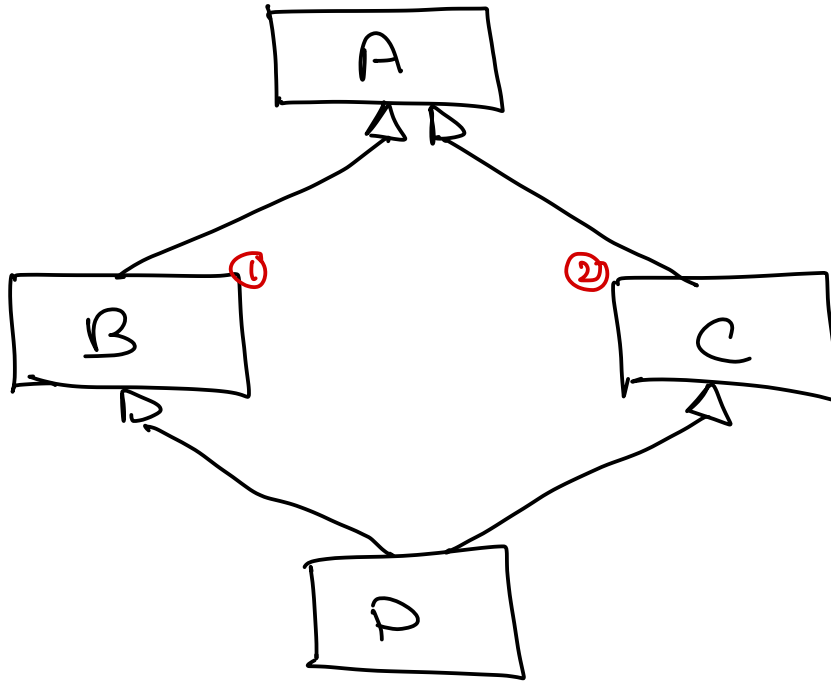
- ① date (birth)
- ② address
- ③ person
- ④ date (joining)
- ⑤ emp



emp e1;



C N D B Y X D



D obj;

A  
B  
A  
C  
D