* Modular design
* Object oriented design
  * information hiding.
  * data encapsulation
  * user-defined data types.
  * inheritance
  * polymorphism
  * . .

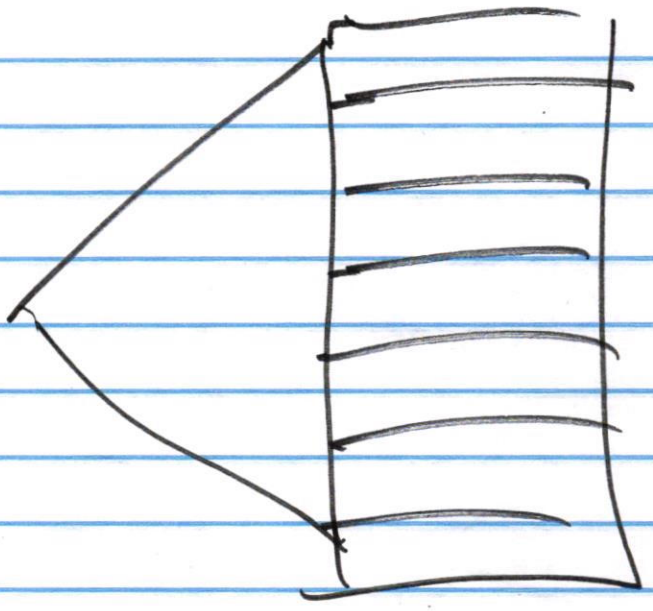
User-defined data type

graph
stack
tree

```
Stack    S1, S2;

S1.push(x)
y = S1.pop()
```

implementation



class name

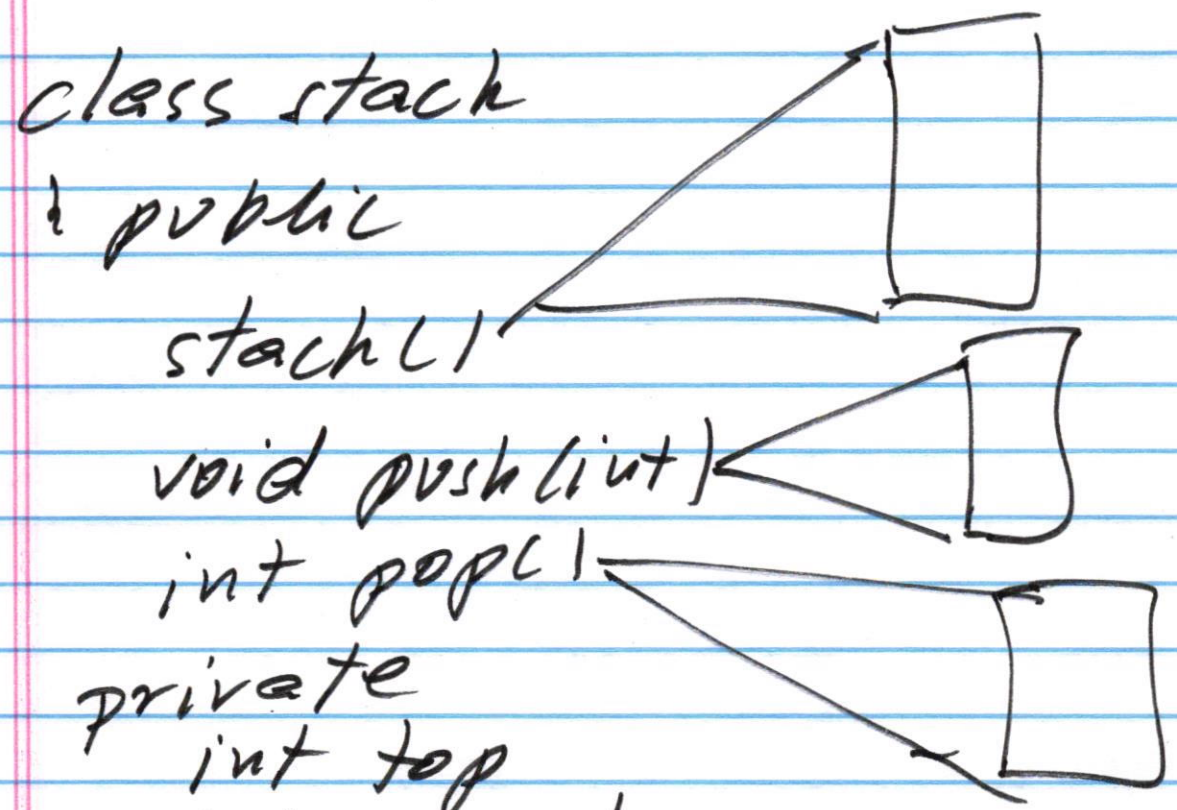
{ public:

operations

private:

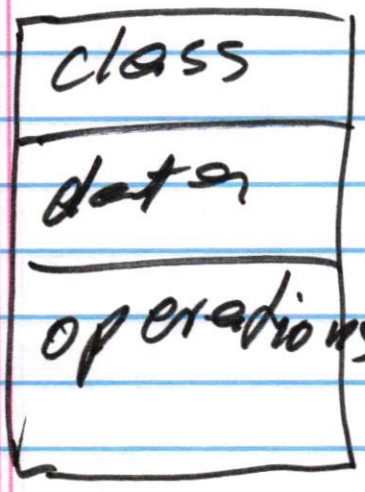
data

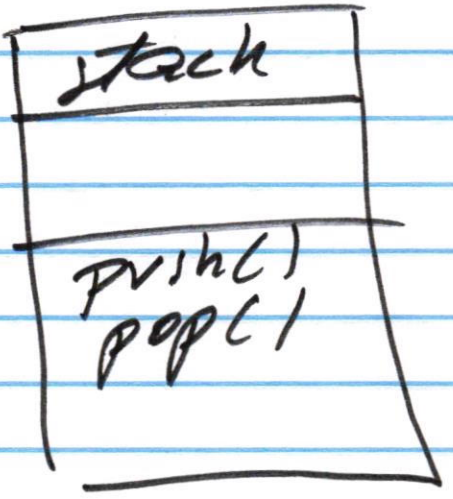
};



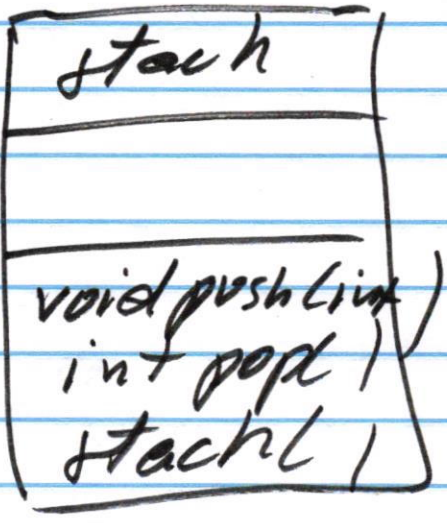

```
stack s1, s2, s3;

s2.push(x)
y = s2.pop()
```





detailed design



Object-oriented design

system structure

⇓

a set of classes

+

relationships between classes

⇓

class diagram

relationships between classes

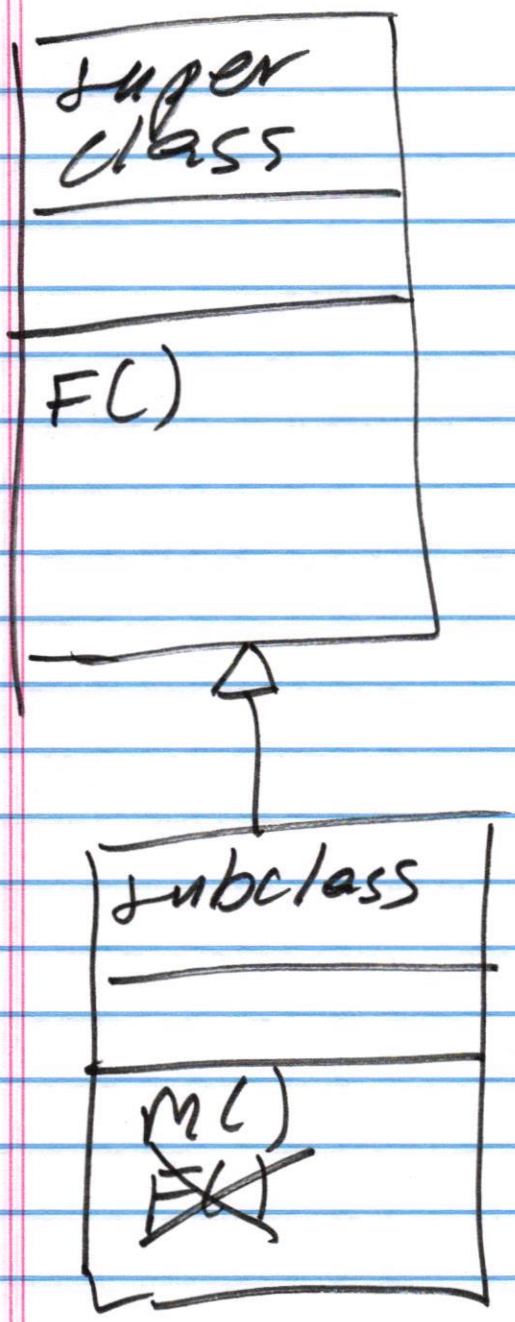
- inheritance
- aggregation
- association

Inheritance

1. superclass
2. subclass

subclass "inherits"
* operations
* data structures

from superclass



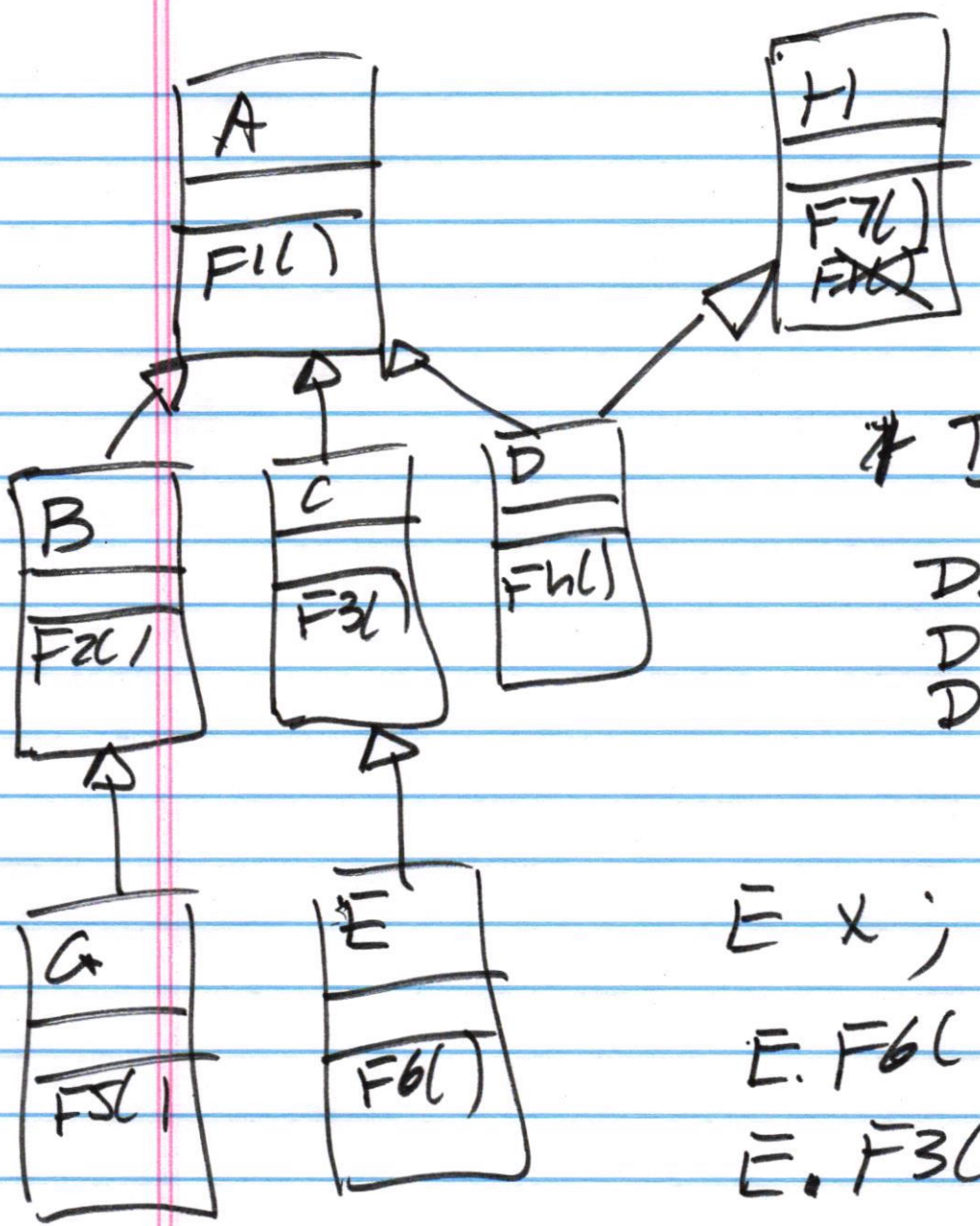
superclass A

subclass B

A. F()

B. M()

B. F()



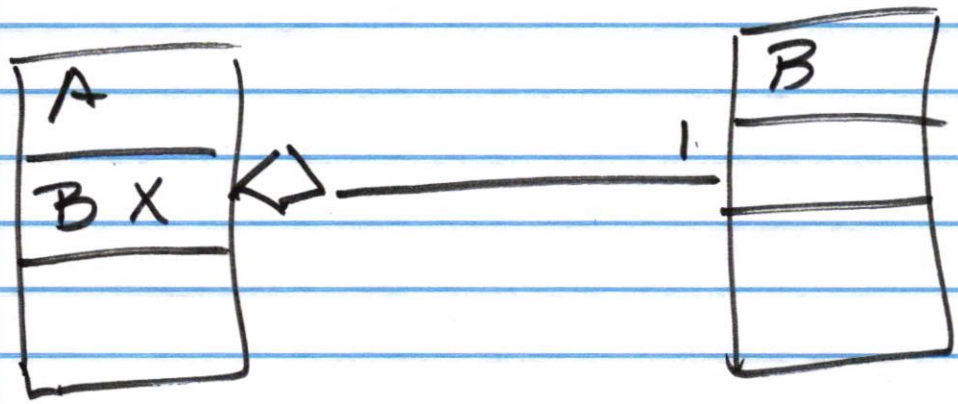

```
*D y;
D.F4();
D.F1();
D.F7();
```

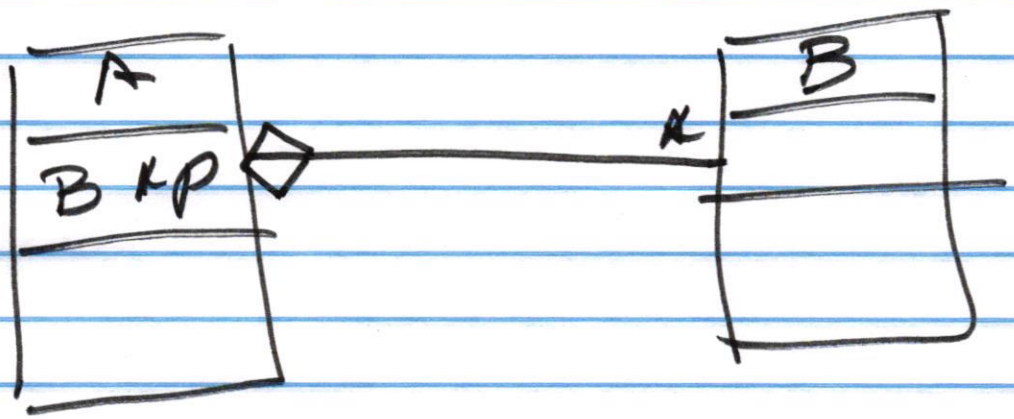
```
E x;
E.F6()
E.F3()
E.F1()
```

Aggregation

"has" relationship

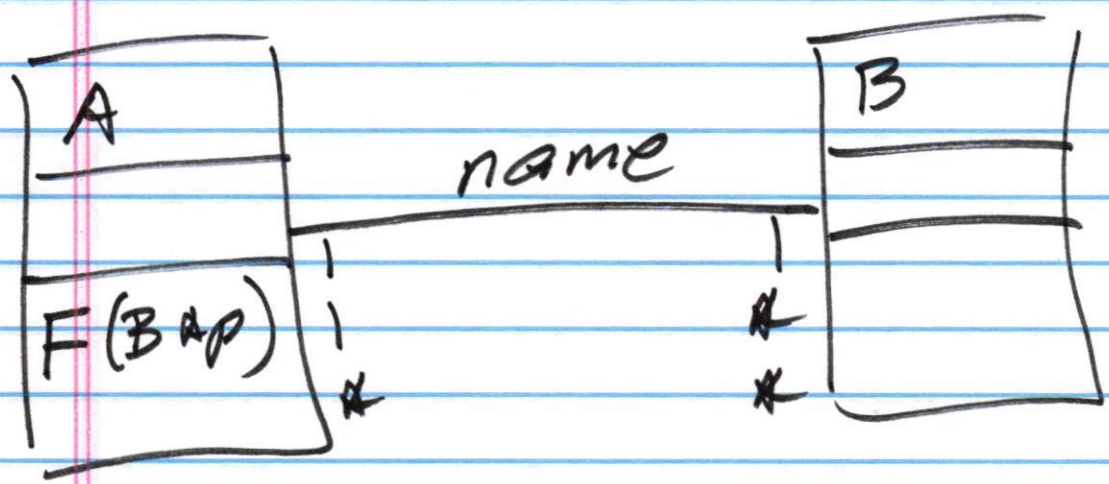
One class "consists of" one or more objects of another class.

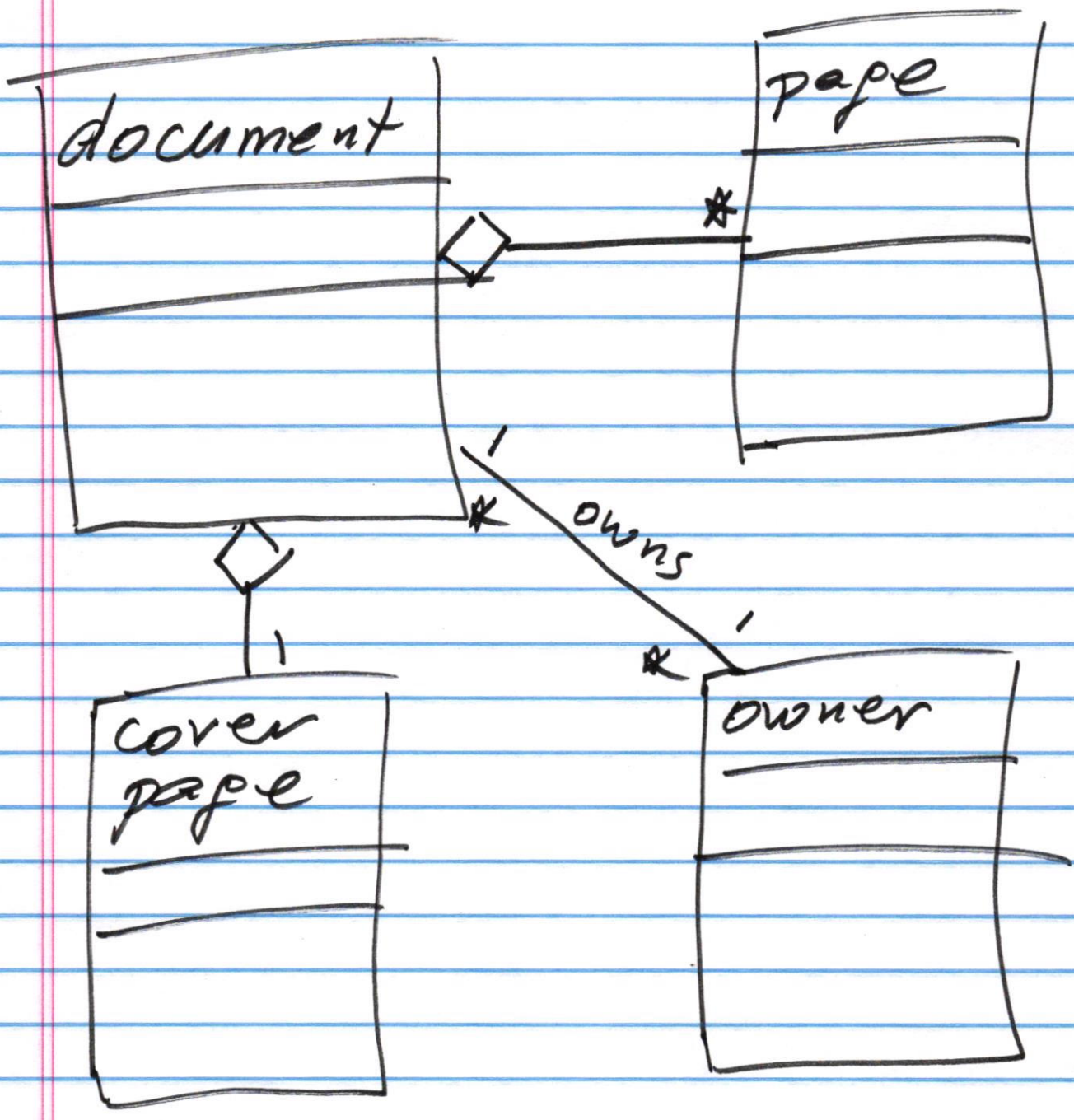




Association

"anything else" relationship




document
page
owns
cover page
owner
1
*
1
*
1
*
1
*

Design patterns

Reuse: source code ✗

Motivation: reuse the design solutions.

Design pattern

a set of rules describing how to solve / design certain design problem

Low-Level design patterns

a book on algorithms.

⇓

design patterns

⇓

low level

Design problem:

a list of names.

sort names in ascending order

⇓

sorting algorithms.

Design pattern

1. Name
2. Problem
   * description of the design problem
   * description of situation when it can be used
3. solution
   * description of components.
   * relationships between components
   * responsibilities of components.
   * collaborations.
4. Consequences
   * results.
   * advantages/disadvantages
   * trade-offs

Design patterns

High-level design

object-oriented design

Types of design patterns

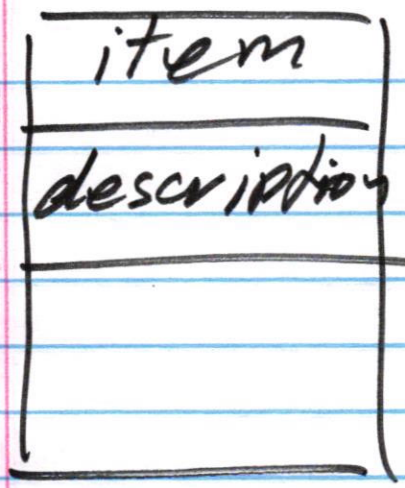
* domain independent
* domain specific

Description of an item pattern

Problem:

* objects contain a description
* many objects may have the same description
* it is likely that in the future, the description will change

all objects that have the old description must have the new description



descriptions:

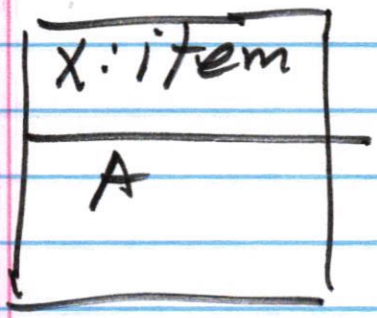
$A \rightarrow A'$

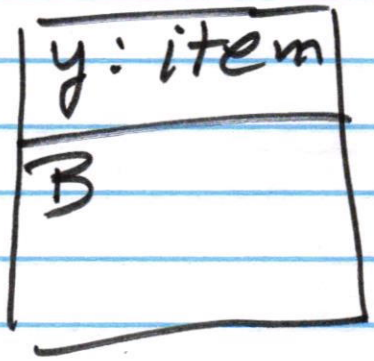
B

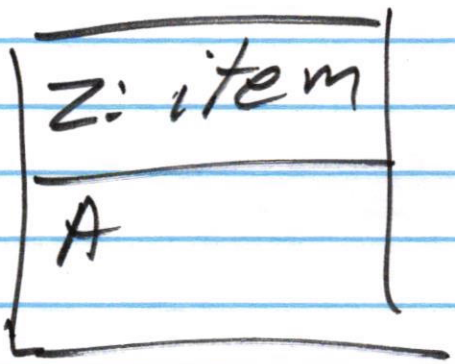
C

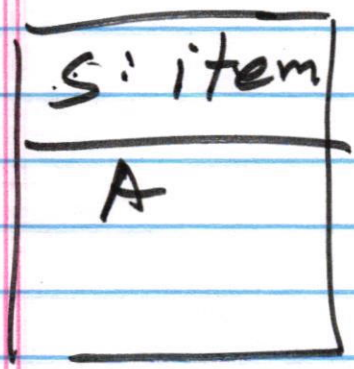
---

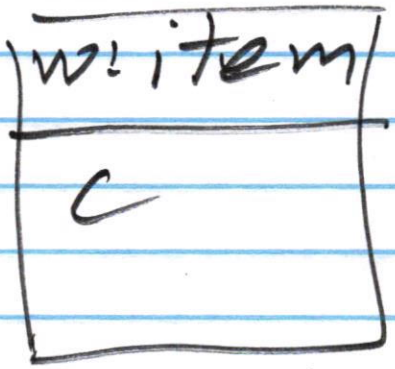
objects







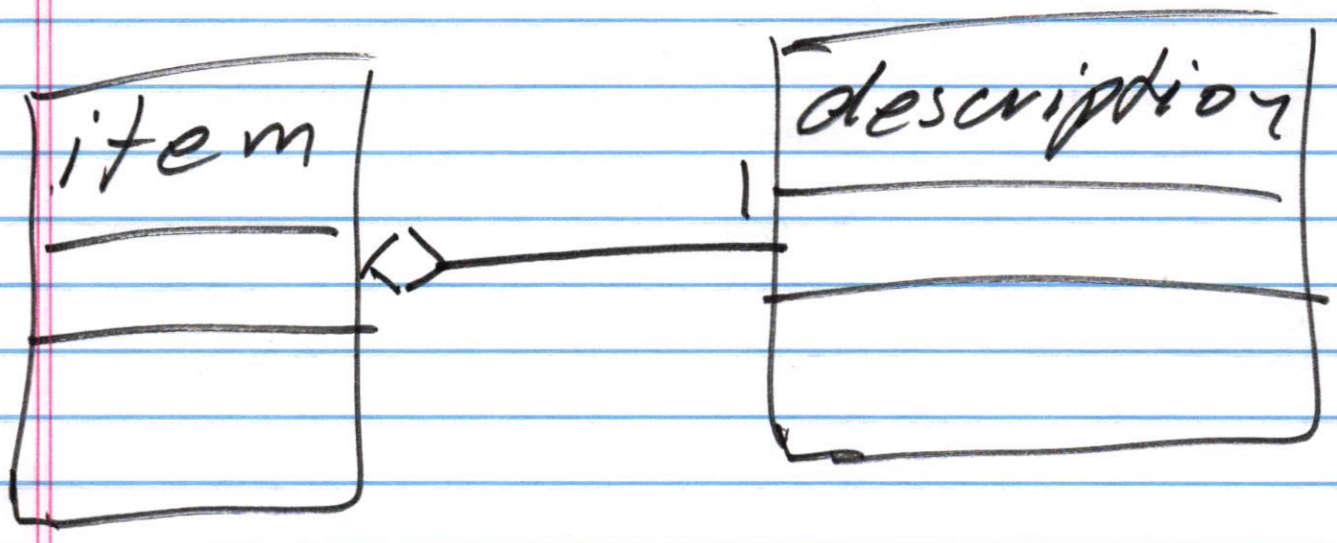


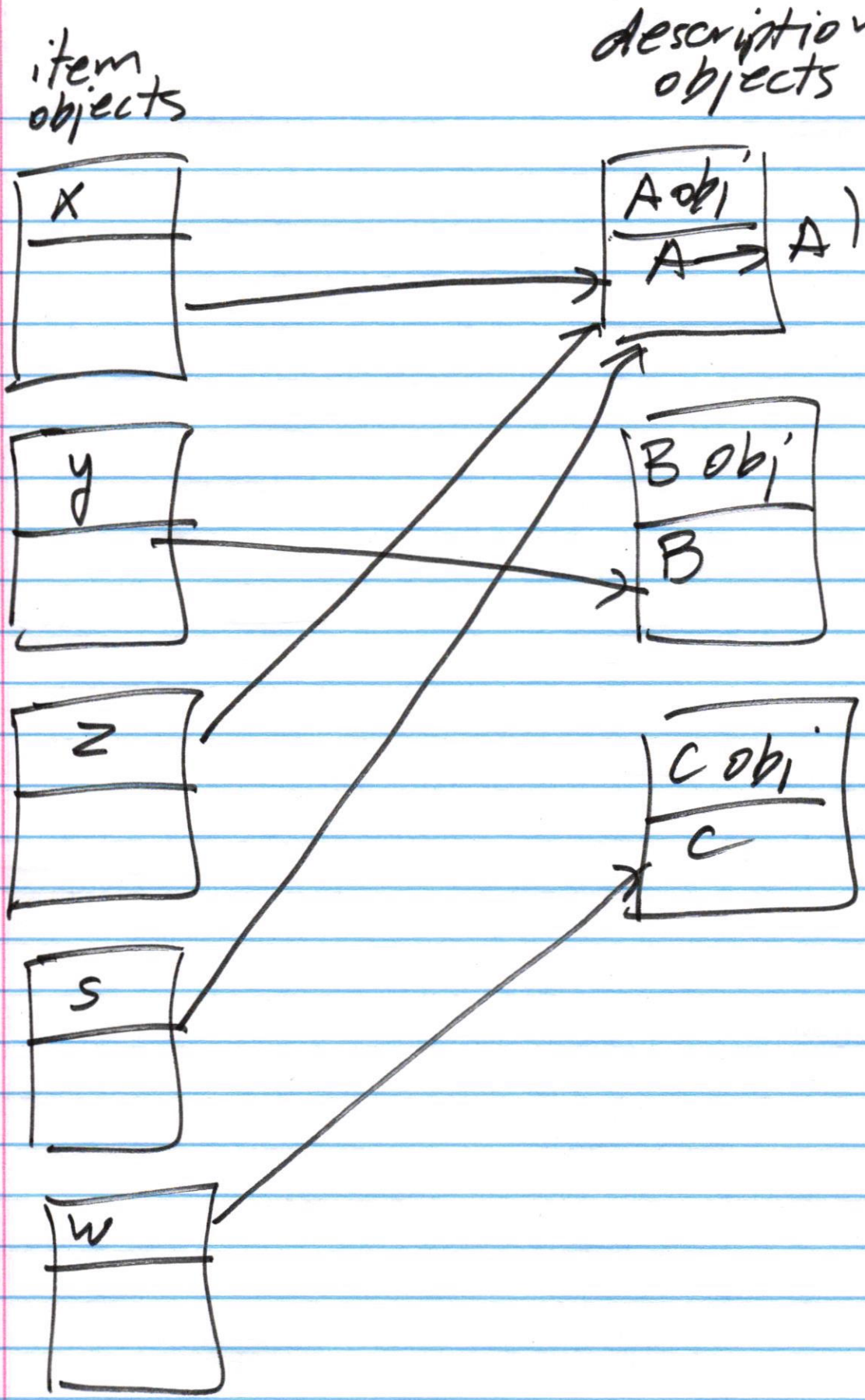


bad design

solution

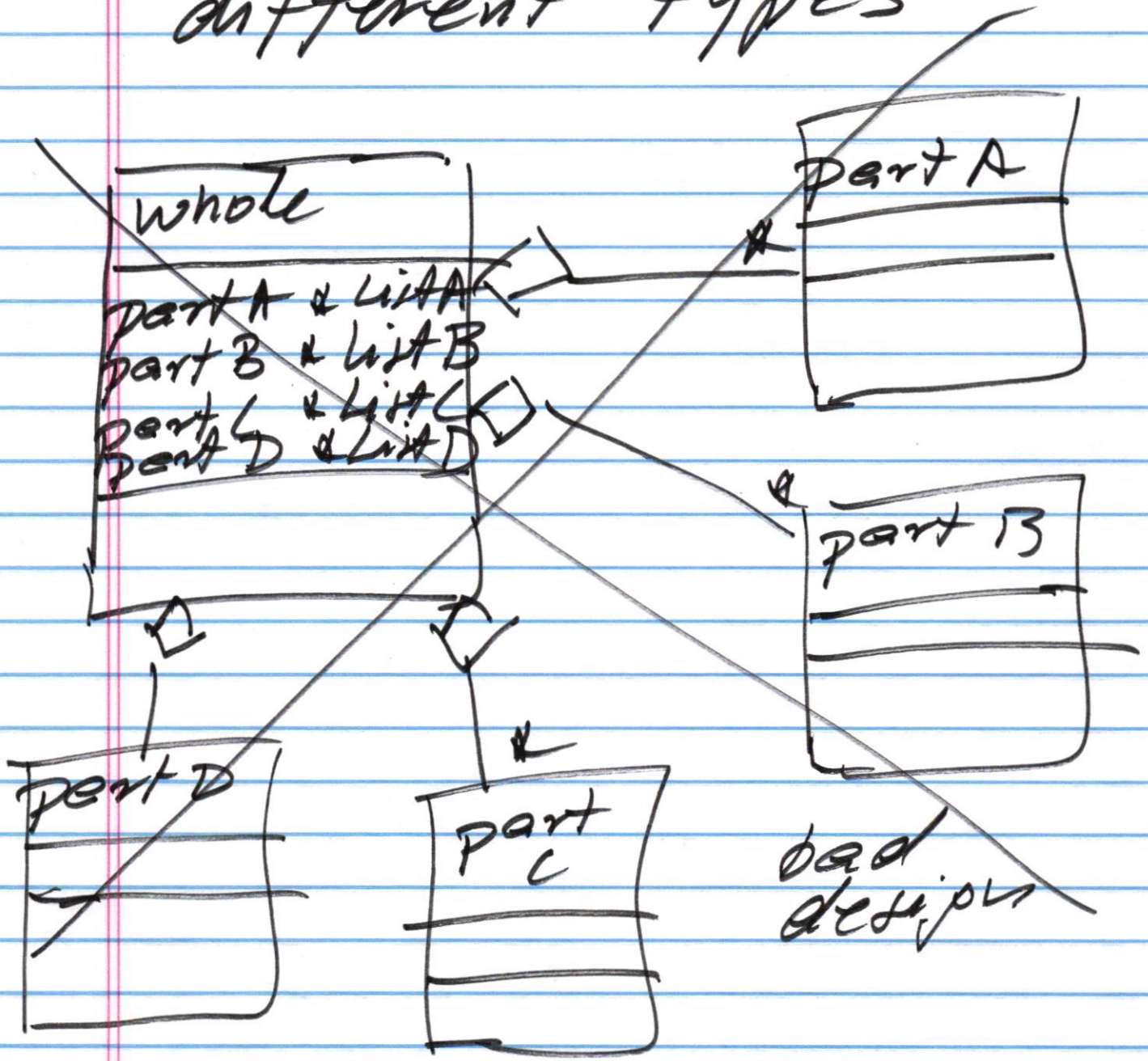
create a new class with description

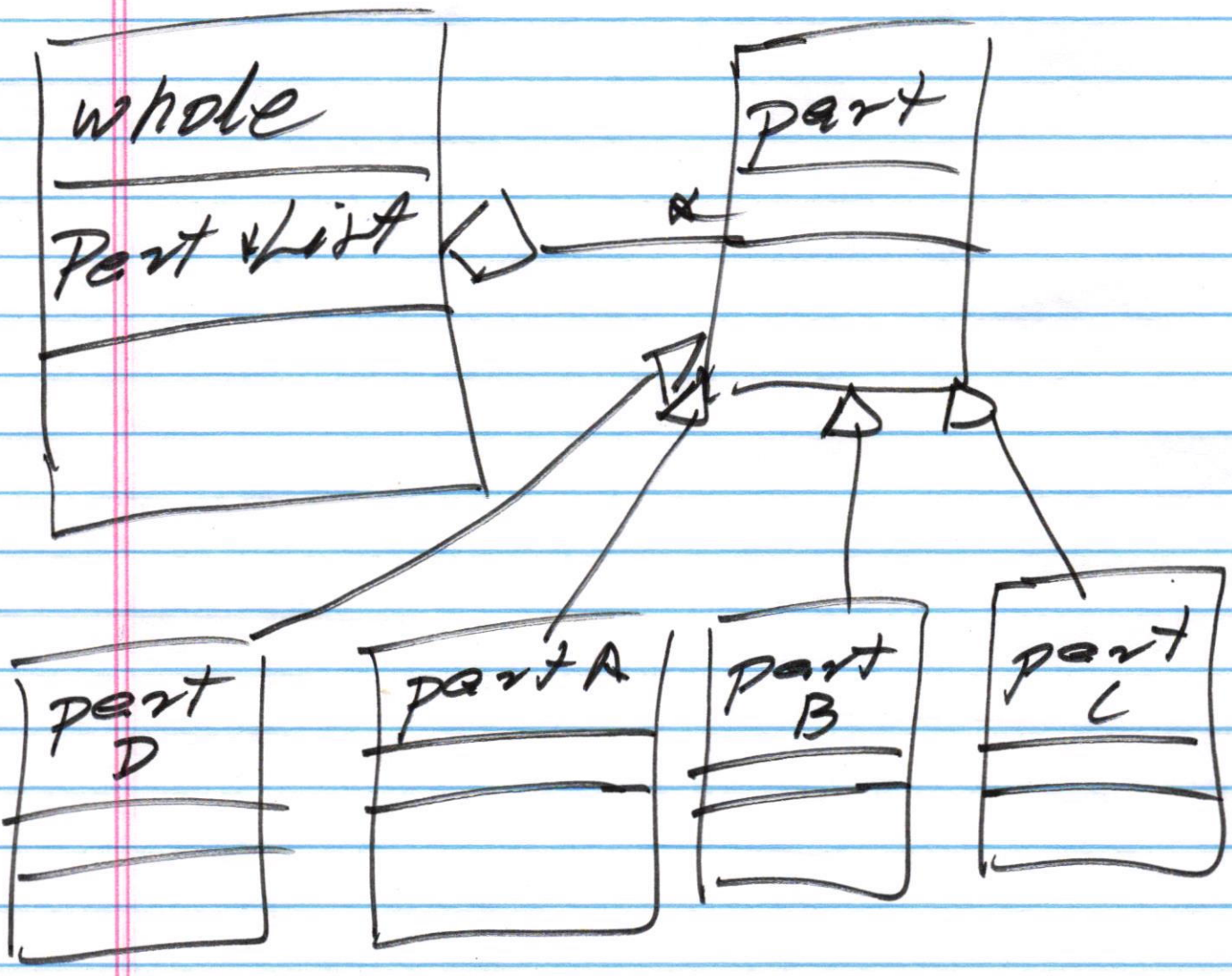



item objects
description objects
x
y
z
s
w
A obj
A
A)
B obj
B
C obj
C

whole-part pattern

an object (whole) consists of parts of different types

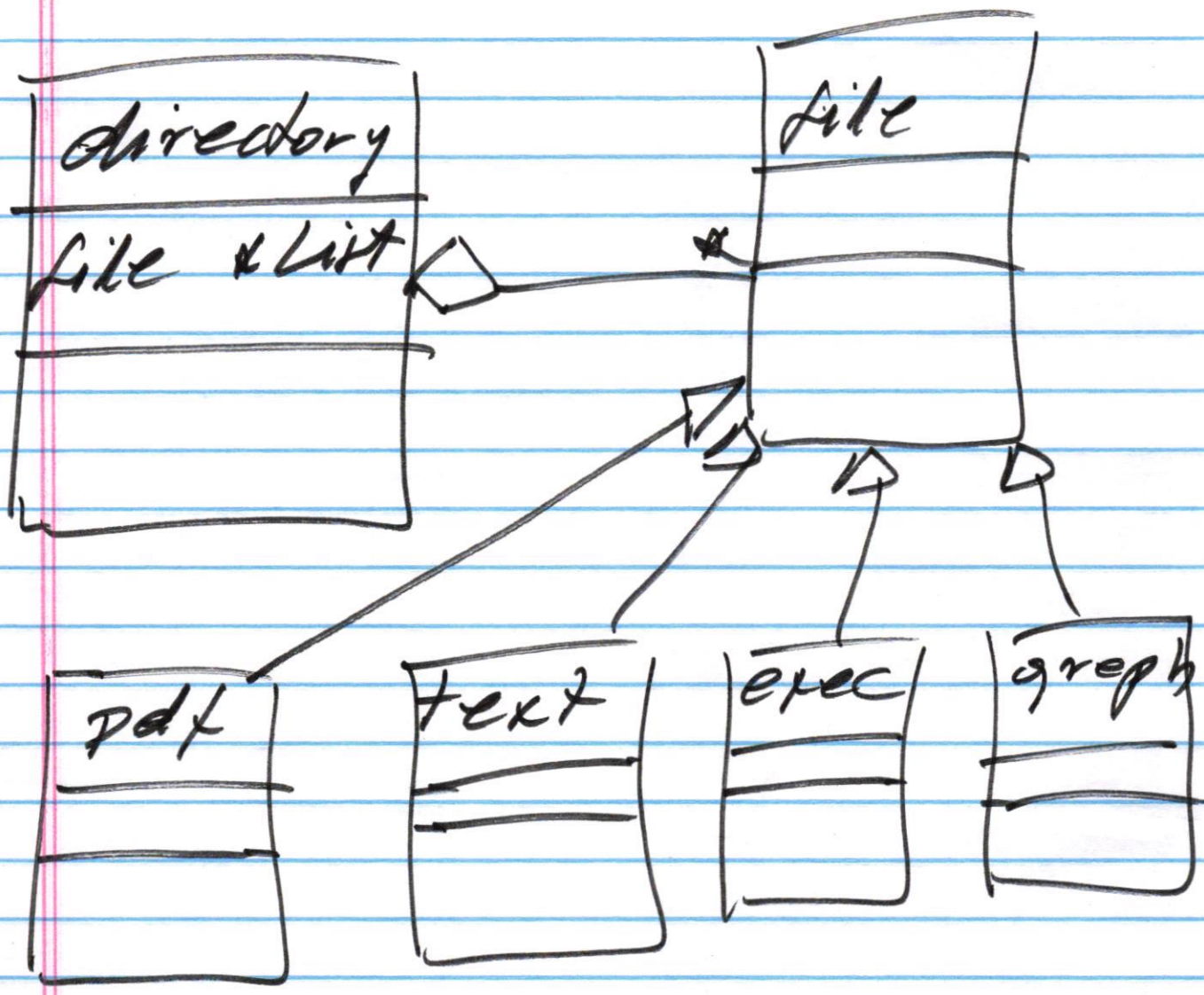



whole
Part List
part
part D
part A
part B
part C

File system

1. directory ⇒ files

2. file → text
   → exec
   → graph.


directory
file *List
file
pdf
text
exec
graph

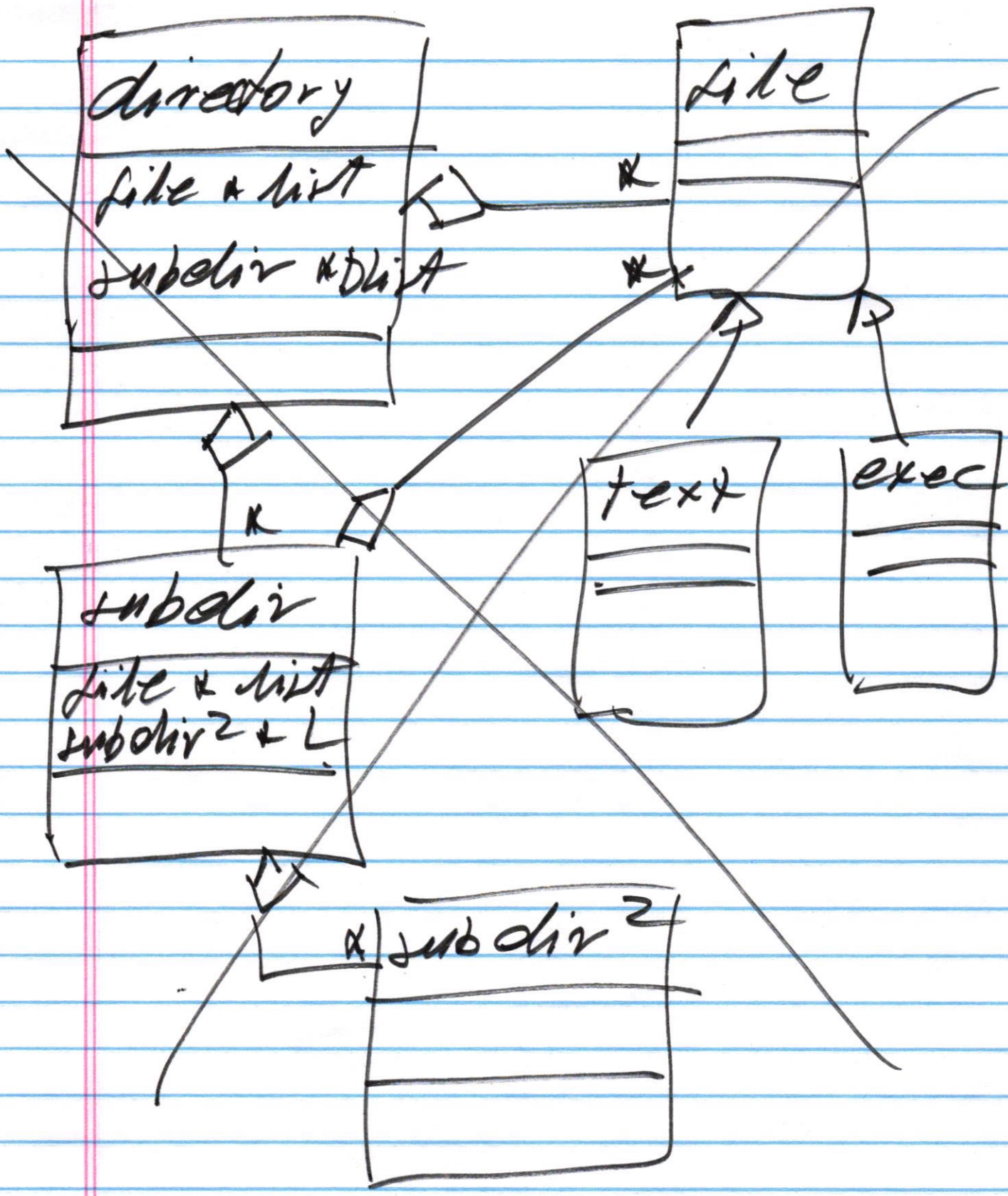
Problem

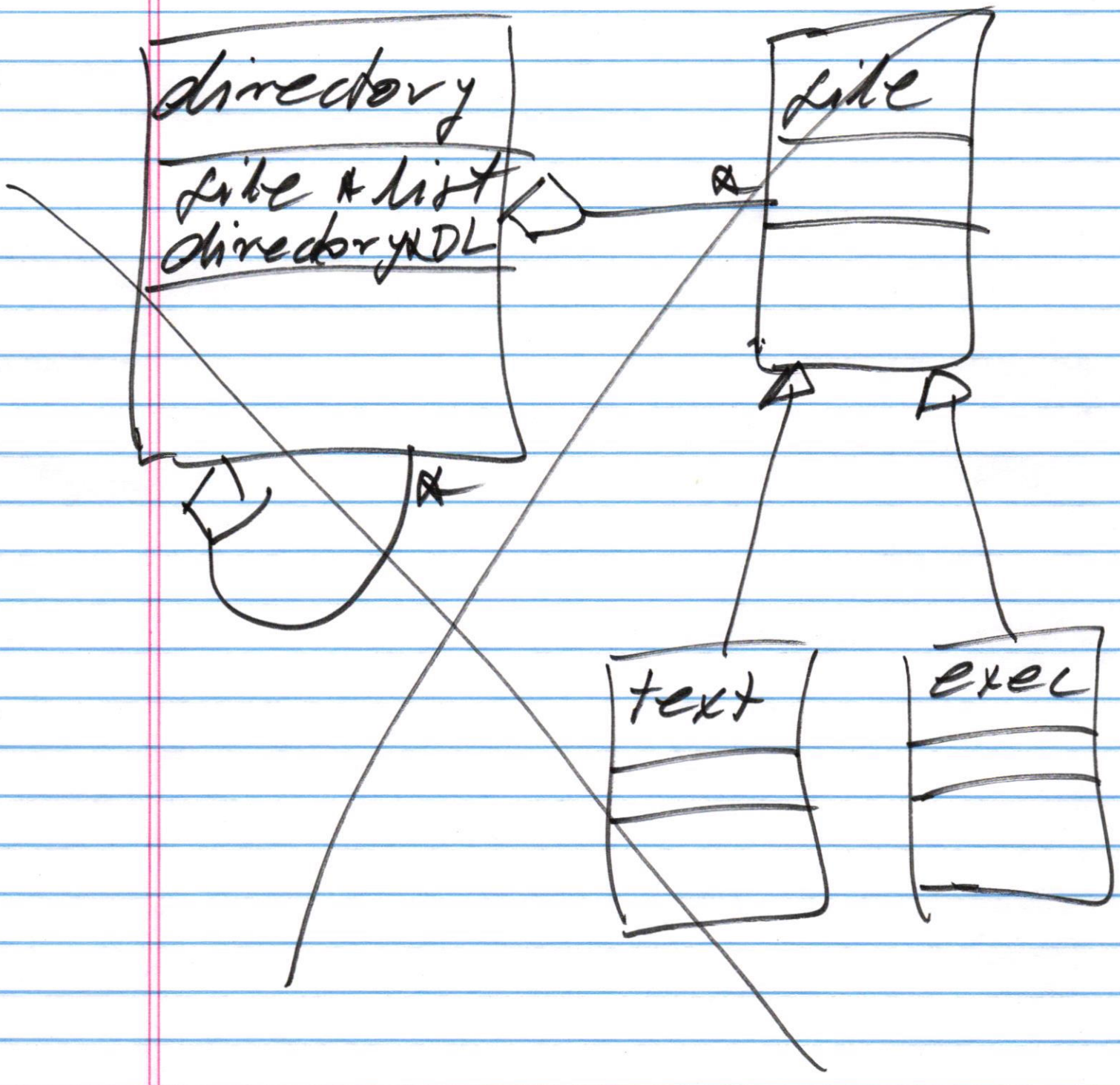
a container consists of many parts of different types including its own type.

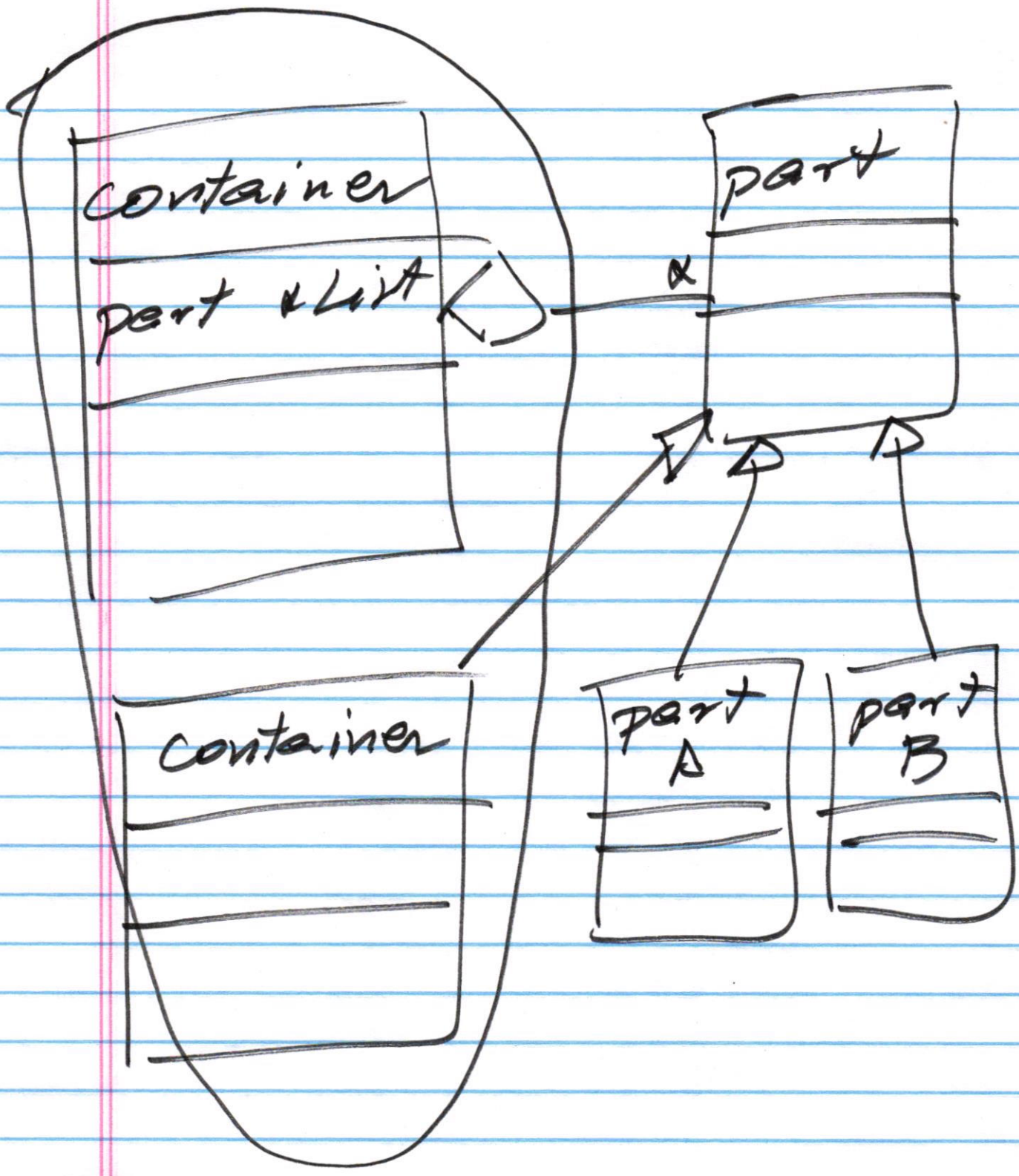
File system

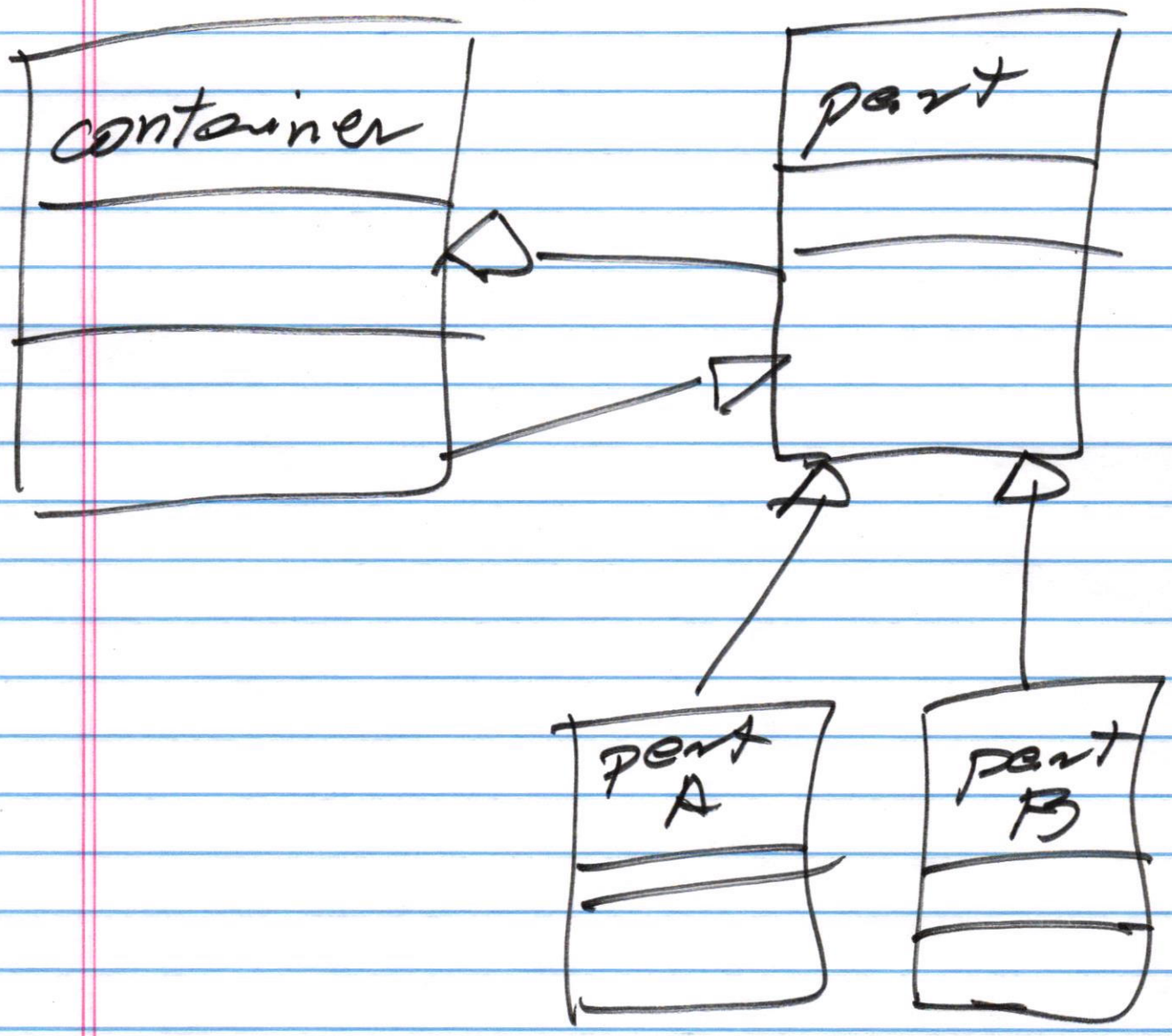
(1) Directory <
- file
- directory

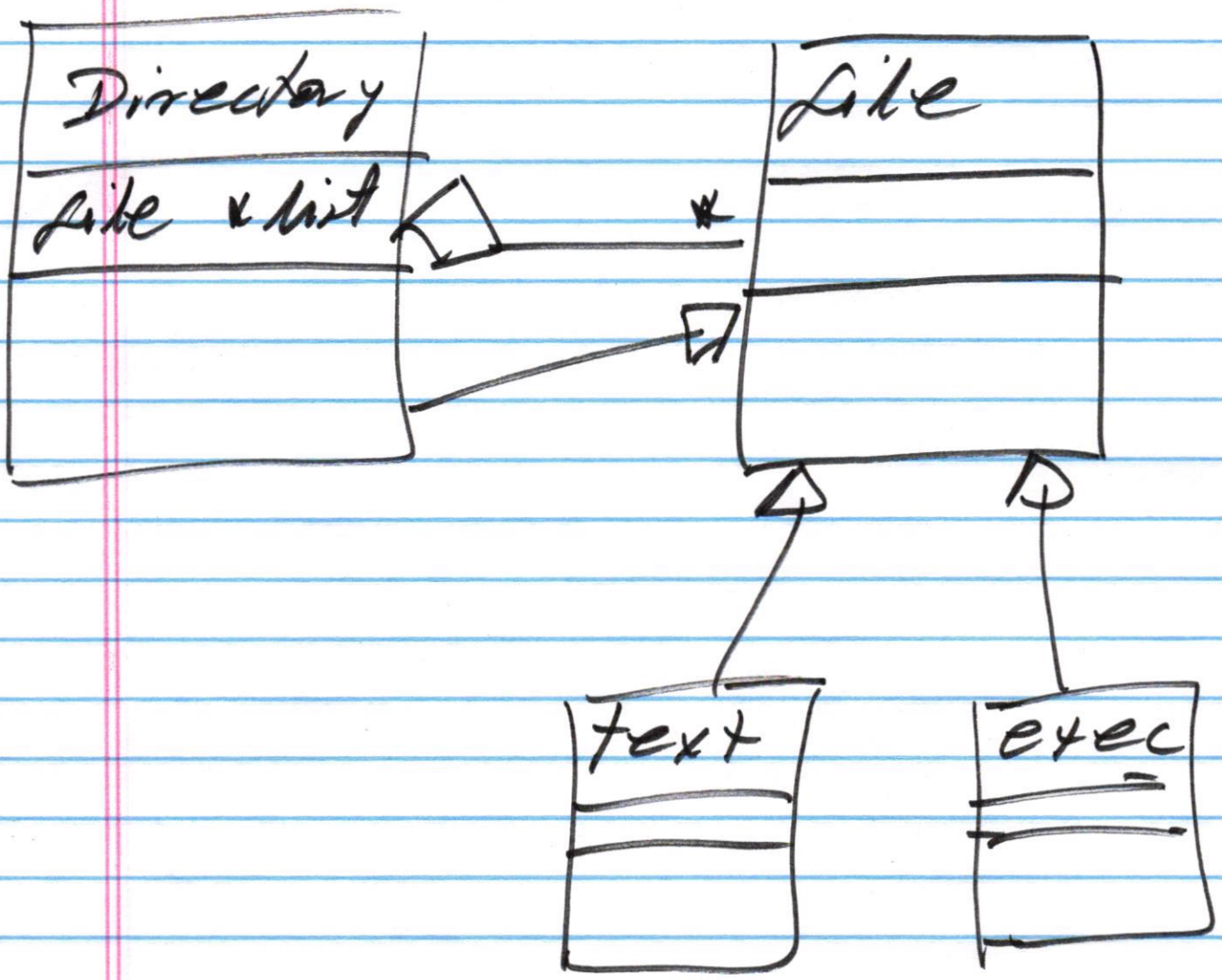
(2) file <
- text
- exec
- graph.

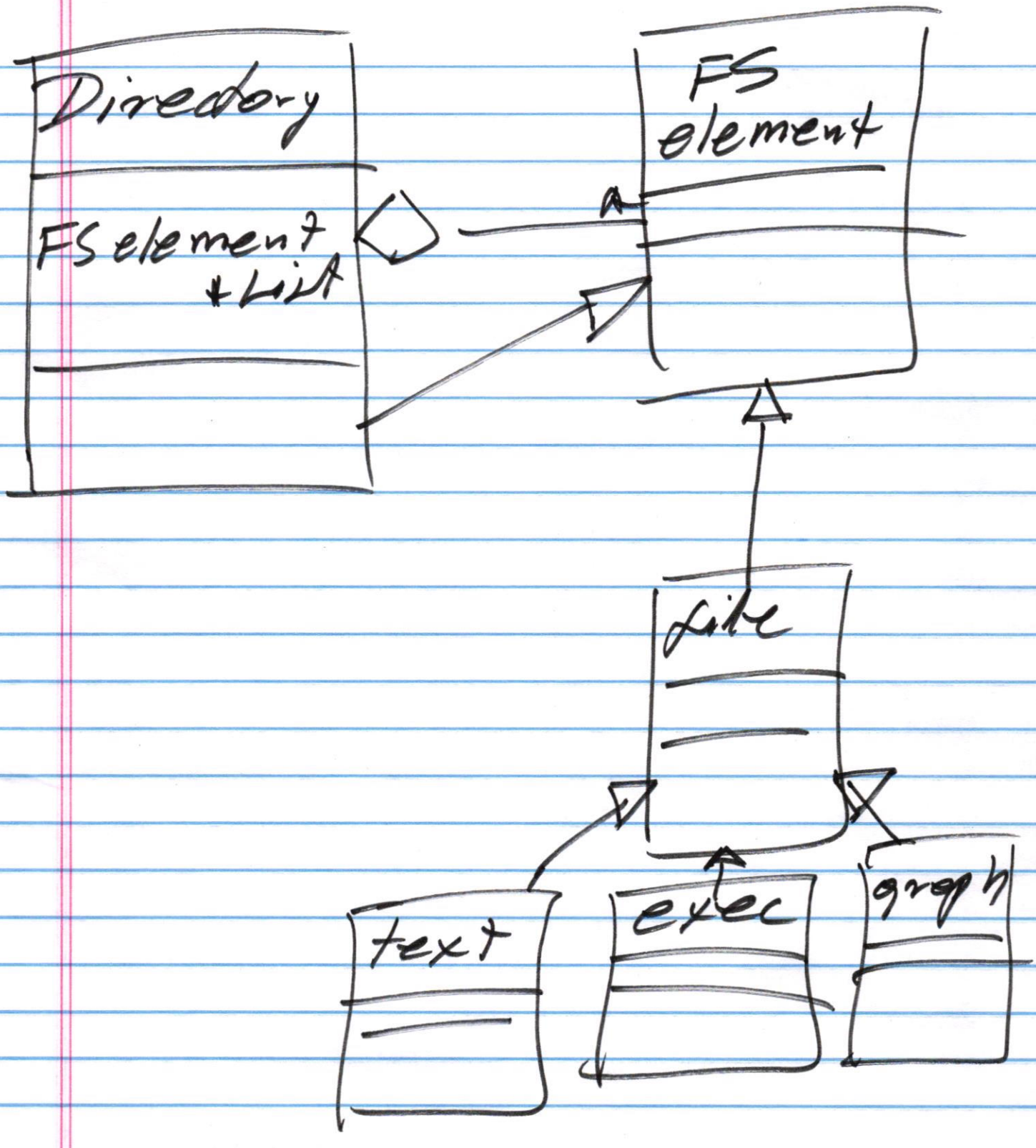

directory
file * list
subdir * DirA
file
text
exec
subdir
file * list
subdir2 * L
subdir2


directory
file * list
directoryNDL
file
text
exec


container
part *List
part
container
part A
part B


container
part
part A
part B


Directory
file * list
*
File
Text
exec


Directory
FSelement
*List
FS
element
List
text
exec
graph

polymorphism

using the same name to get different services

procedural languages

F(int) ⟶ F(x)

F(int, int) ⟶ F(x, y)

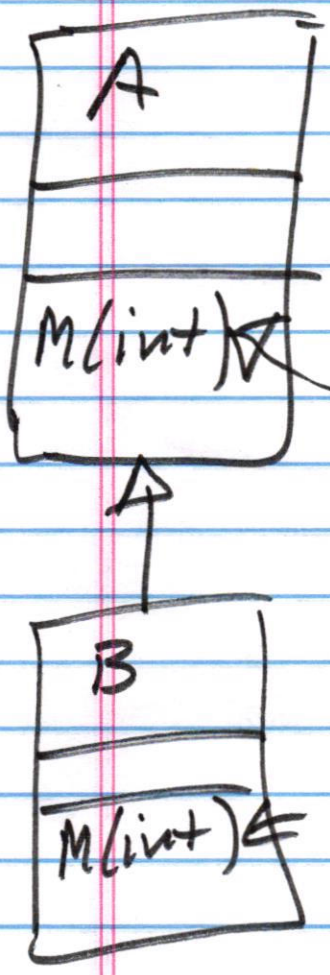
F(int, string) ⟶ F(x, s)

F(string, int) ⟶ F(s, x)

F(int, int) syntax error

use the same name but signatures are different

polymorphism related to inheritance



static objects

A a

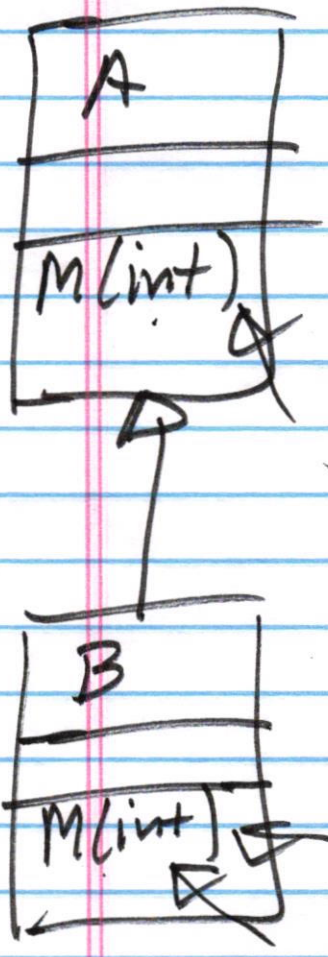
B b

a.M(x)

b.M(x)

The same name
+
the same signature.

Dynamic objects



A *pa

B *pb

pa = new A

pb = new B

① pa → M(x)

pb → M(x)

pa = pb

① pa → M(x)