Homework #1 is posted

Exam #1

Wednesday, October 1

at 5:00 pm

Closed books and notes

Coverage is posted

COVERAGE FOR EXAM #1

CS 586; Fall 2025

Exam #1 will be held on **Wednesday, October 1, 2025,** at **5:00 p.m.**
Location: 111 Stuart Building
The exam is a **closed-book and notes** exam.

Coverage for the exam:

- Modular design, information hiding, coupling, and cohesion.
- Object-oriented concepts: abstract data type, data encapsulation, inheritance, polymorphism, static and dynamic binding.
- Object-oriented design. Relationships between classes: aggregation, association, and inheritance. Class model. Sequence diagrams.
- OO design patterns: item description, whole-part, observer, state, proxy, and adapter patterns. [Textbook: Sections 3.1, 3.2; Section 3.4 (pp. 263-275); Section 3.6 (pp.339-343); Handout #1, class notes]

Sources:

- Textbook: F. Buschmann, et. al., Pattern-oriented software architecture, vol. I, John Wiley & Sons.
- Handout #1
- Class Notes

Abstract factory pattern

Problem:

Objects need to be created in some coordinated order

Application needs to be de-coupled from the problem of creating these objects!!

drivers — print
drivers — display

Display

LRDD: Low Resolution Display Driver

HRDD: High Resolution Display Driver

Print

LRPD: Low Resolution Print Driver

HRPD: High Resolution Print Driver

Low Capacity Machine

LRPD
LRDD

High Capacity Machine.

HRDD
HRPB

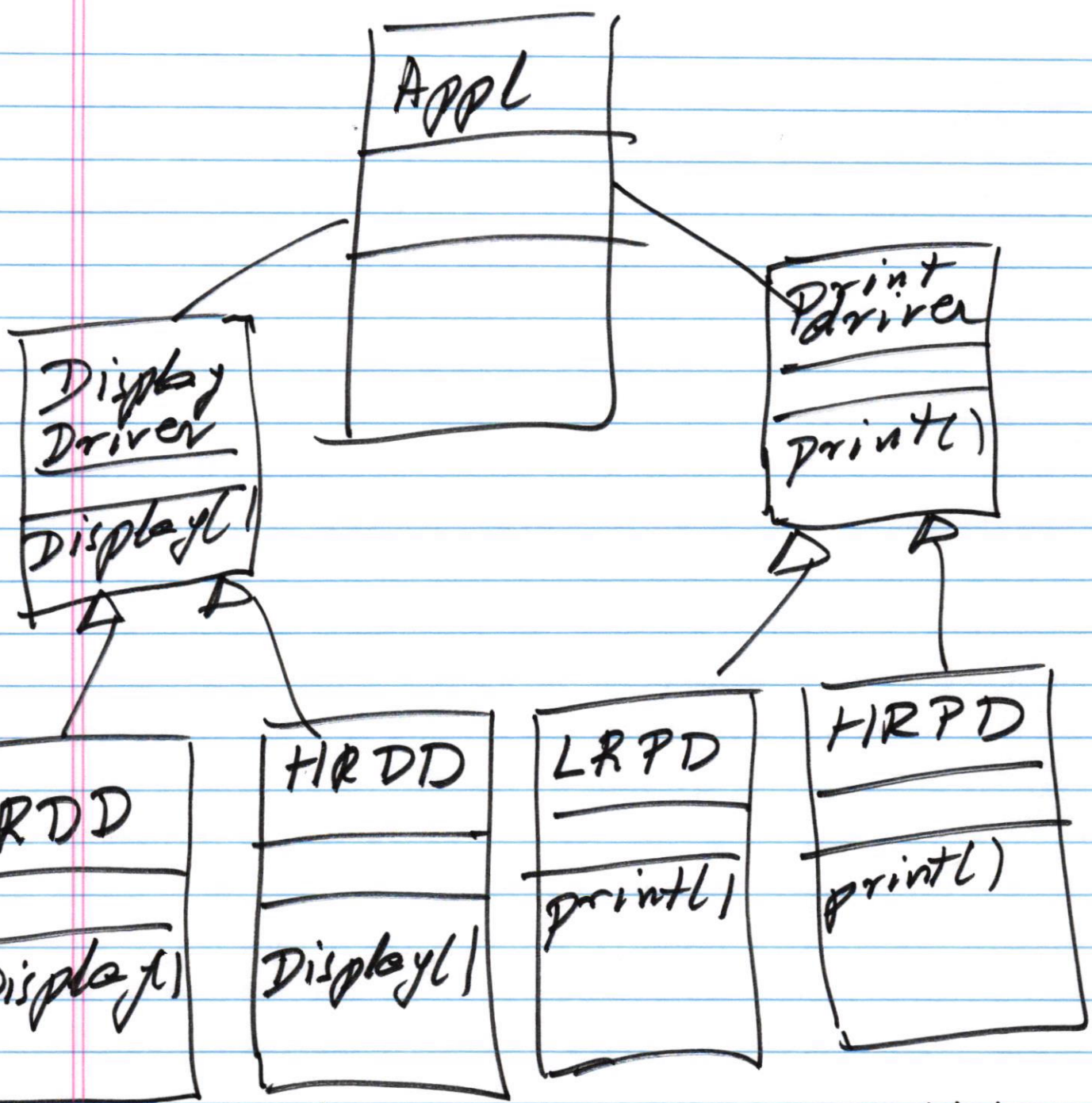
Midrange Machine

HRDD
LRPD

Display : 10 types

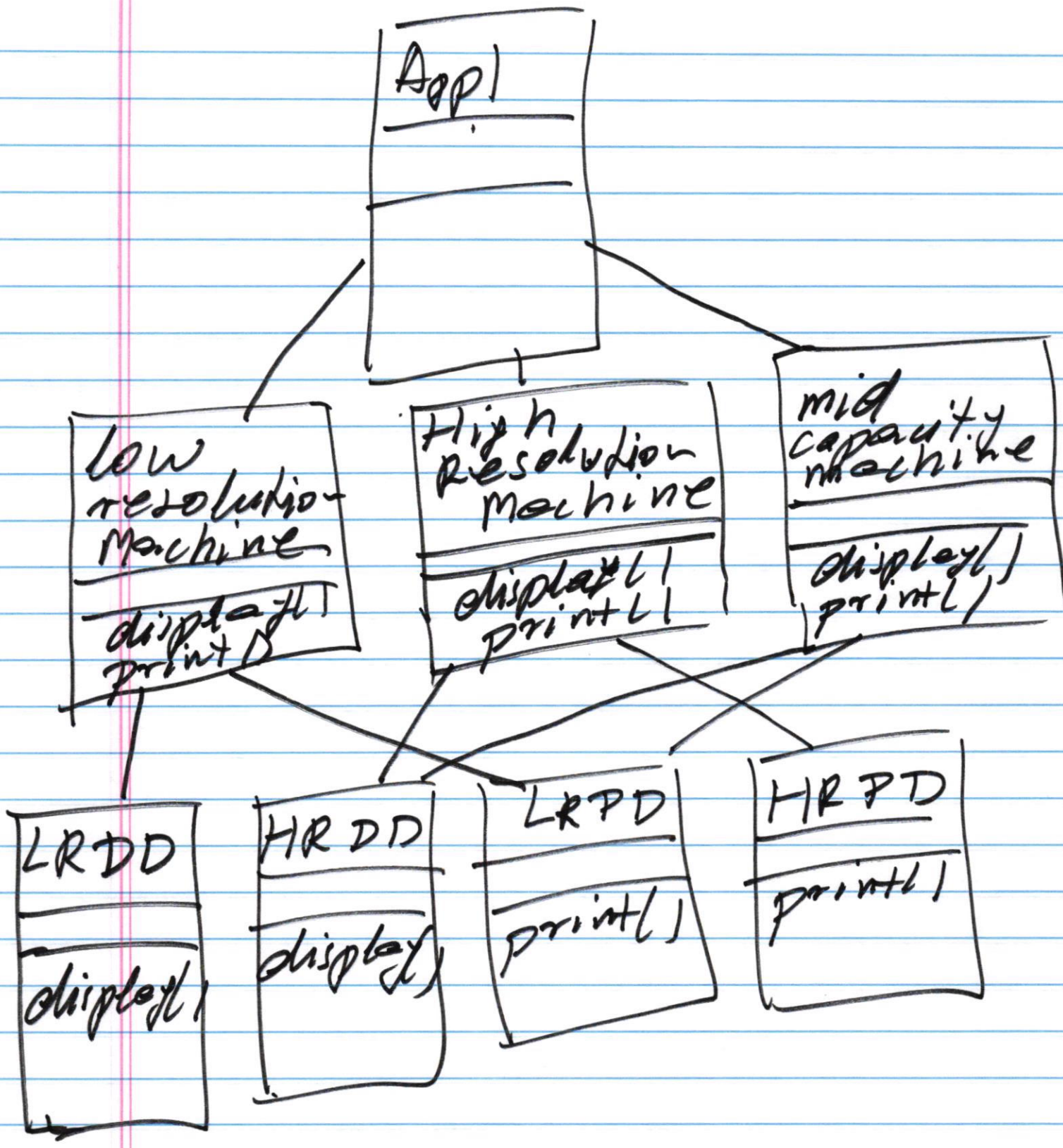
Print : 10 - 11 —

10 · 10 = 100 configurations

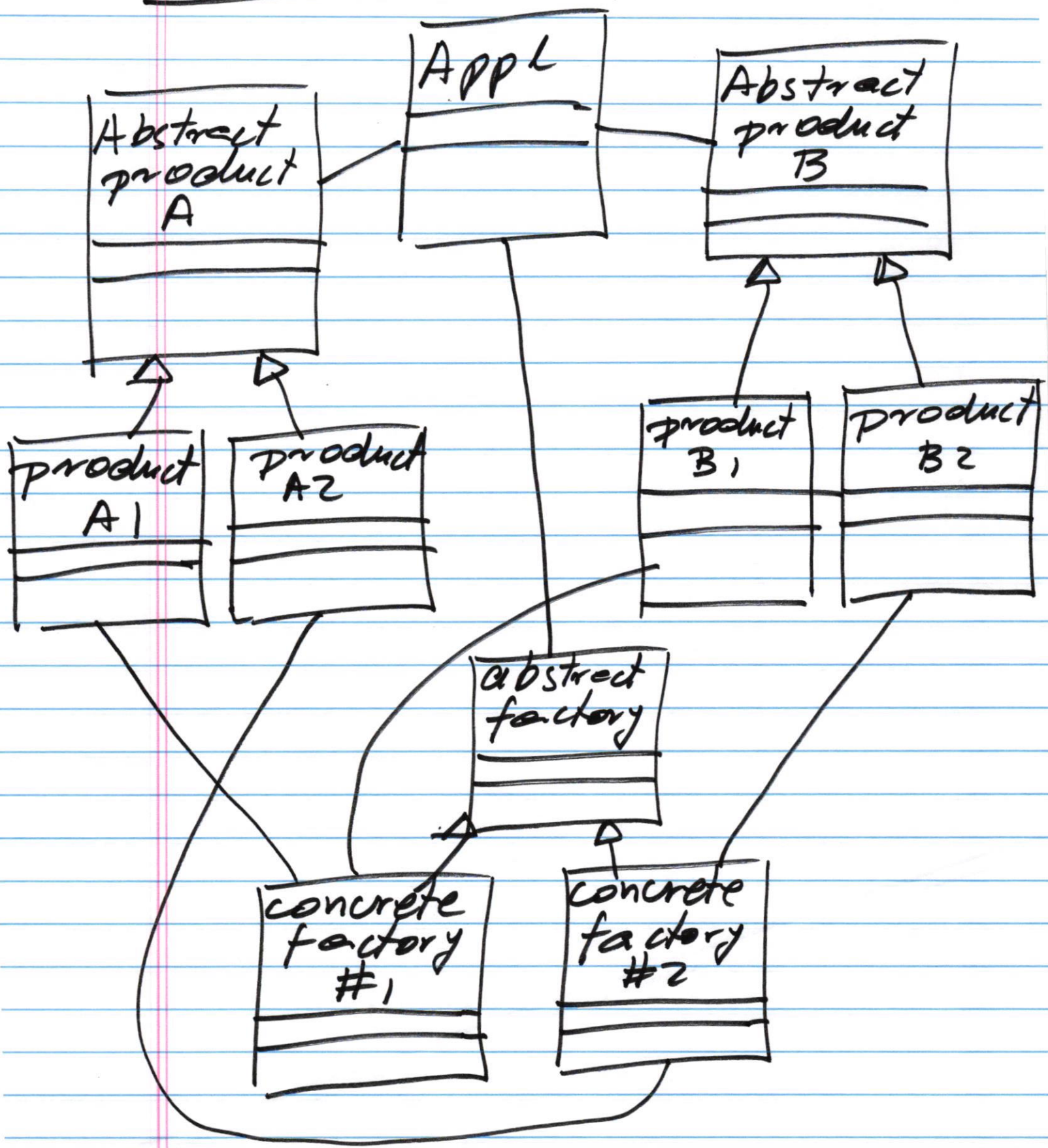


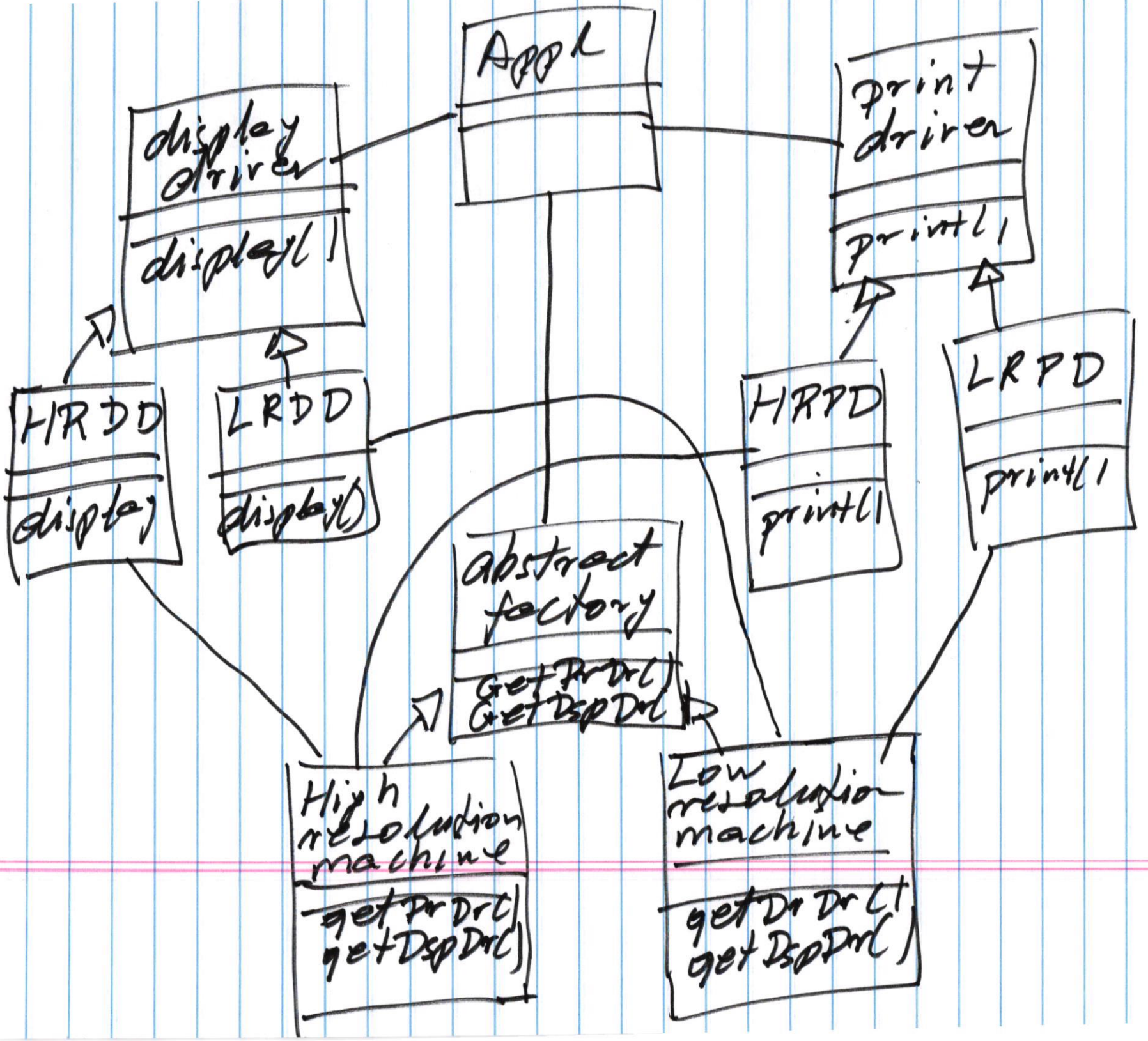
Application is responsible for creating "correct" drivers for different configurations!!

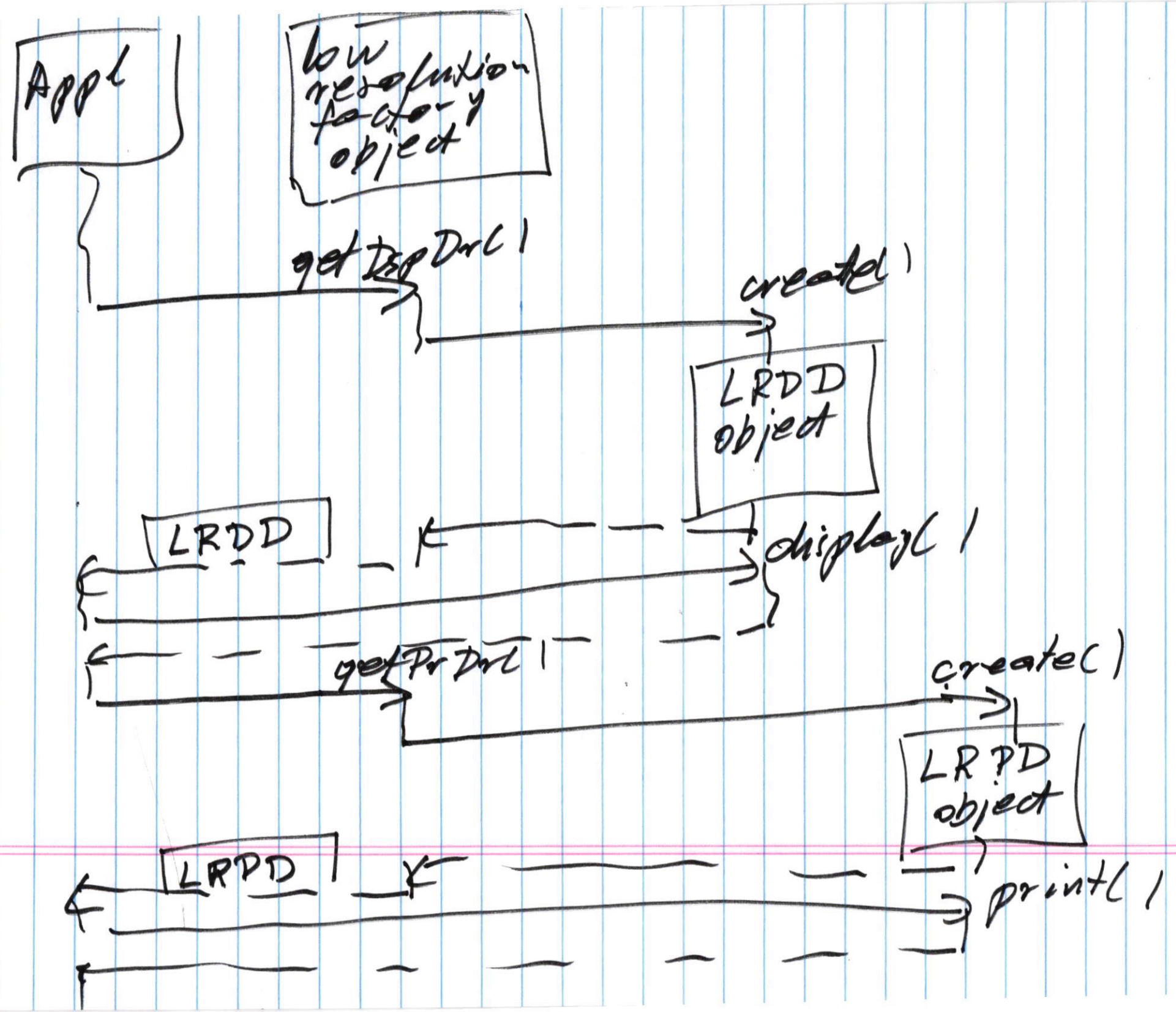
Bad design!!

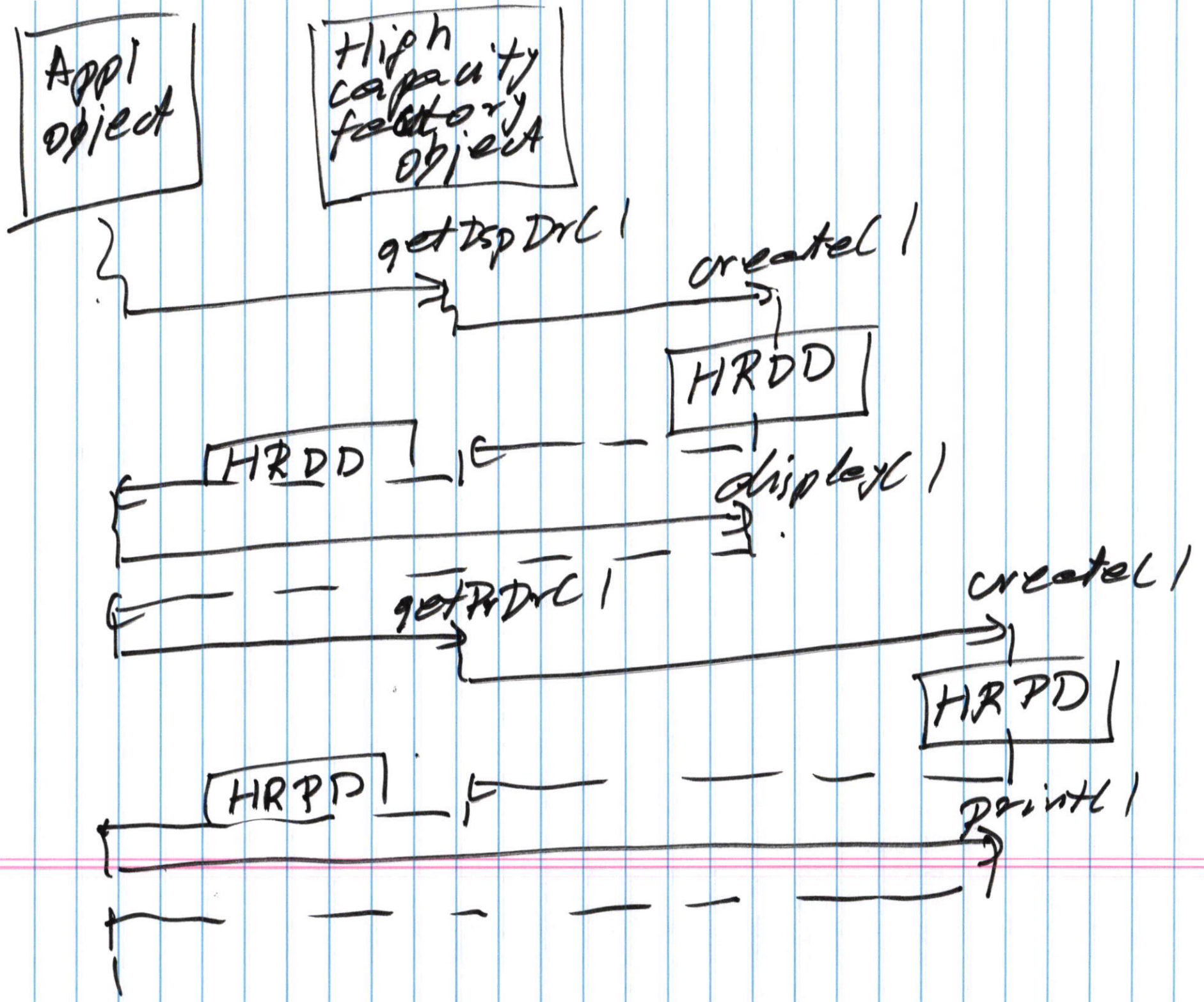

Appl
low resolution Machine
display()
print()
High Resolution machine
display()
print()
mid capacity machine
display()
print()
LRDD
display()
HRDD
display()
LRPD
print()
HRPD
print()

Abstract Factory Pattern



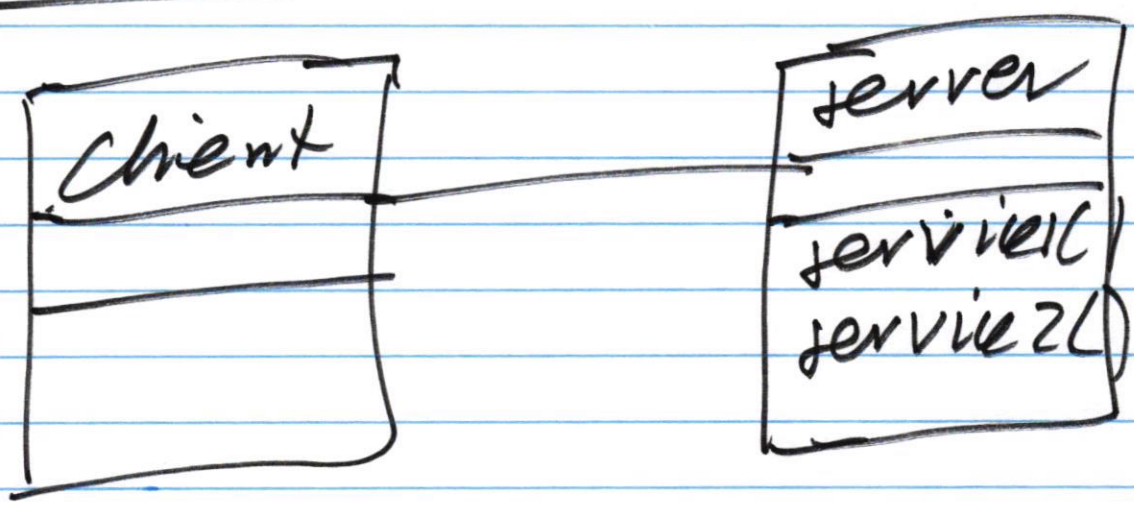

Appl
display driver
displayl()
print driver
printl()
HRDD
display()
LRDD
display()
HRPD
printl()
LRPD
printl()
abstract factory
GetPrDrC()
GetDspDrC()
High resolution machine
getPrDrC()
getDspDrC()
Low resolution machine
getDrDrCt
getDspDrC()


Appl
low resolution factory object
getDspDrC()
create()
LRDD object
LRDD
display()
getPrDrC()
create()
LRPD object
LRPD
printC()


Appl object
High capacity factory object
getDspDrC()
createC()
HRDD
HRDD
displayC()
getPrDrC()
createC()
HRPD
HRPD
PrintC()

Proxy pattern

Problem



clients directly access the server

Direct access to the server is not desired!!

solution

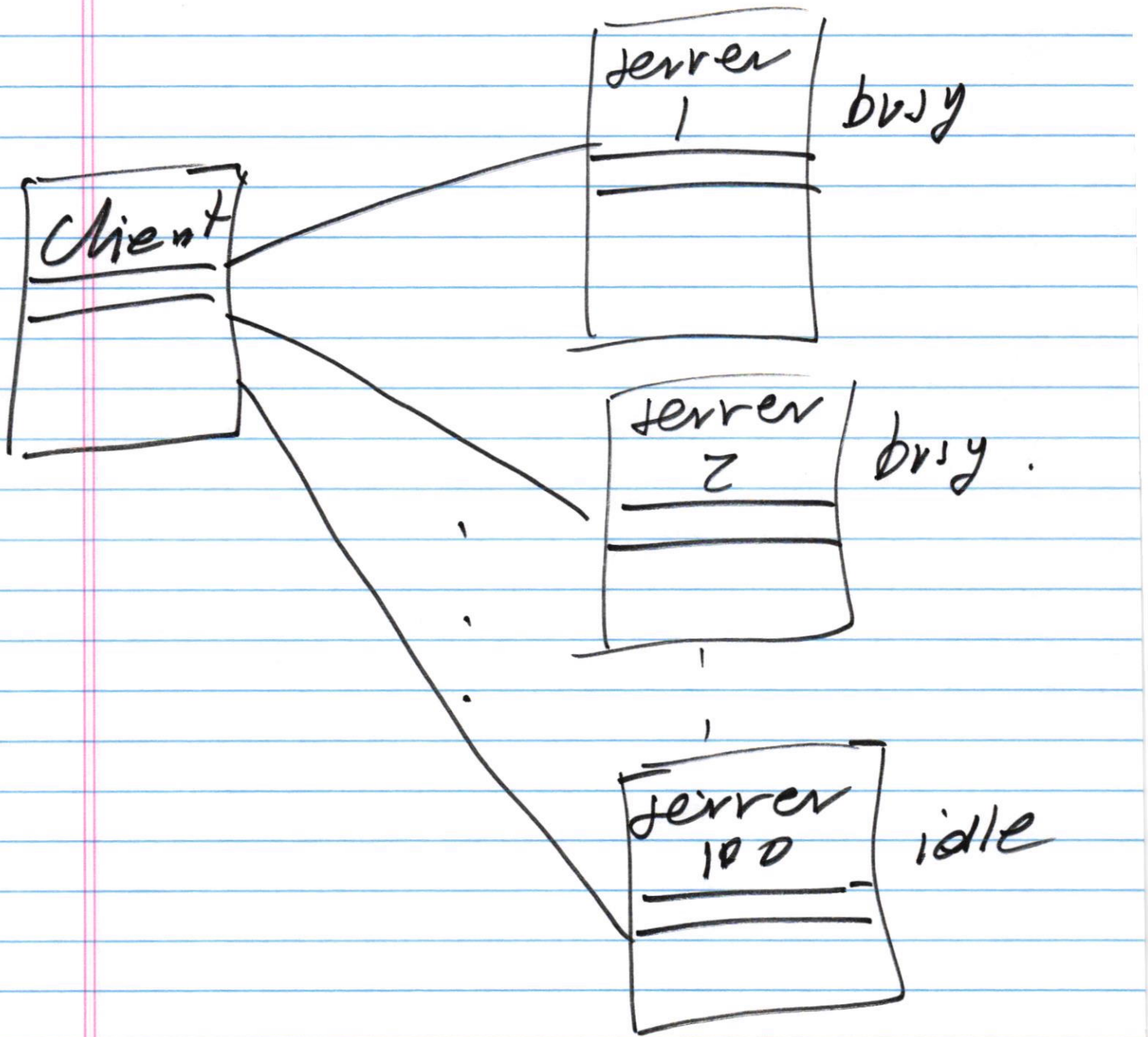
"introduce proxy that represents the server

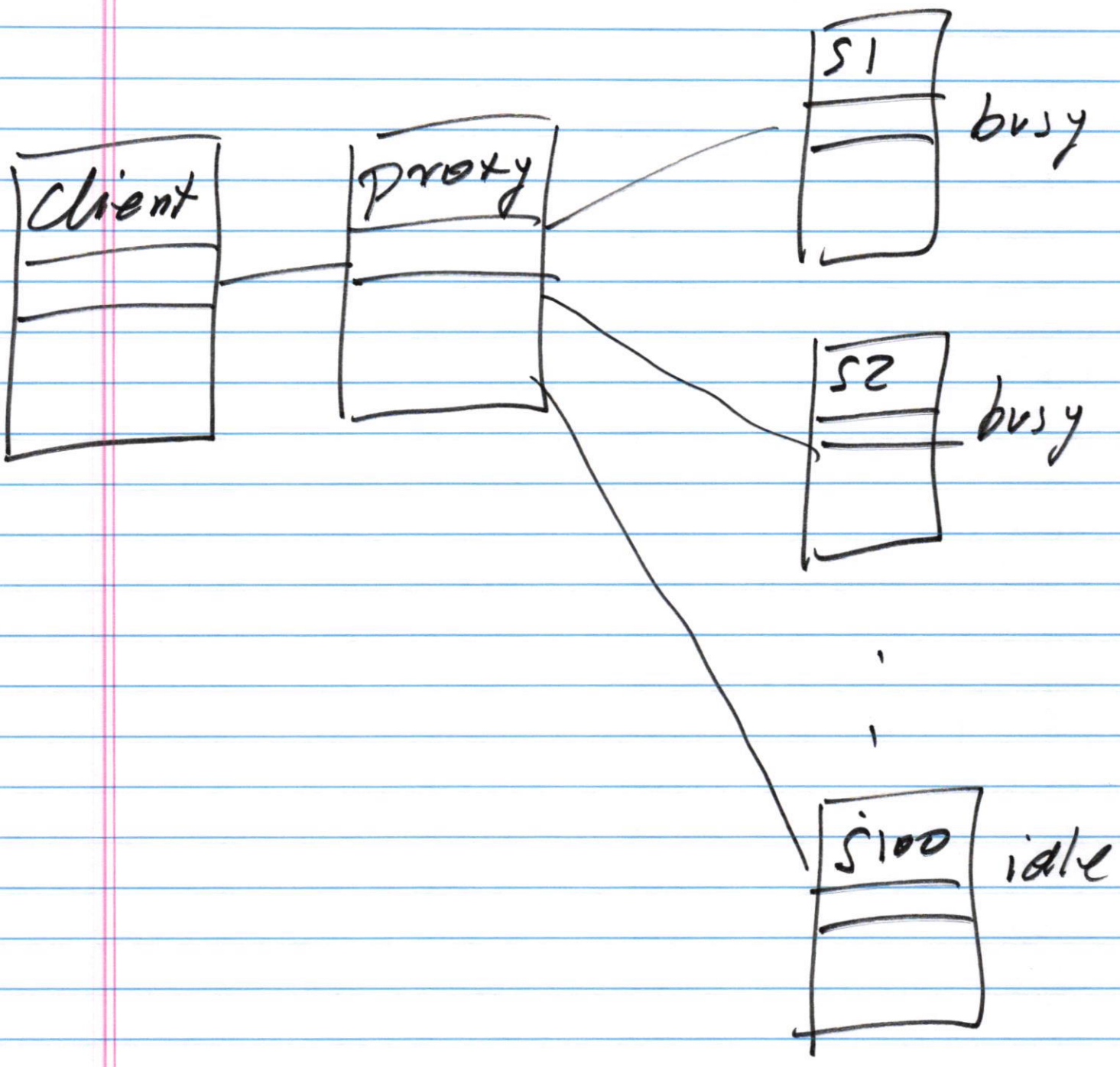
Responsibility

"coordinate" interactions between a client and servers

Reasons

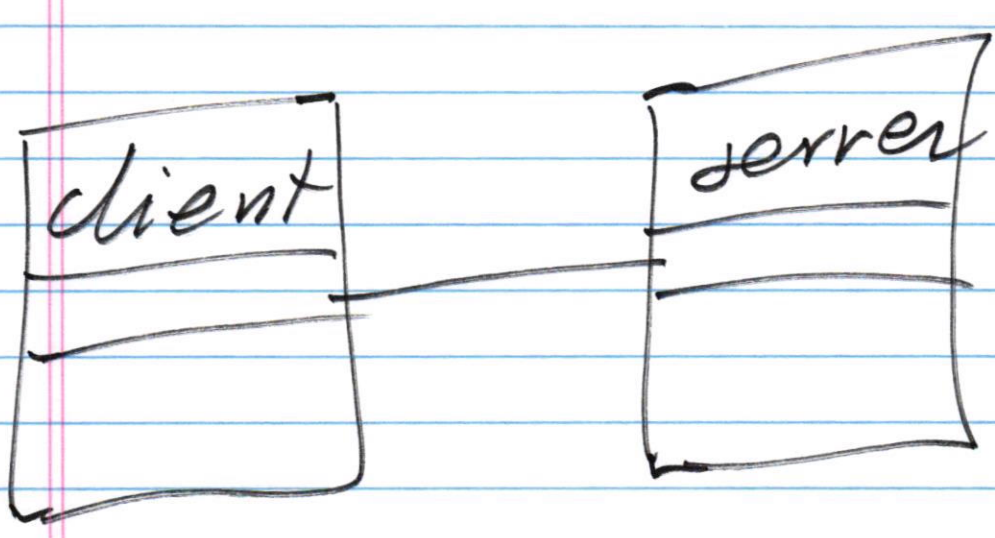
to enhance efficiency for a client.



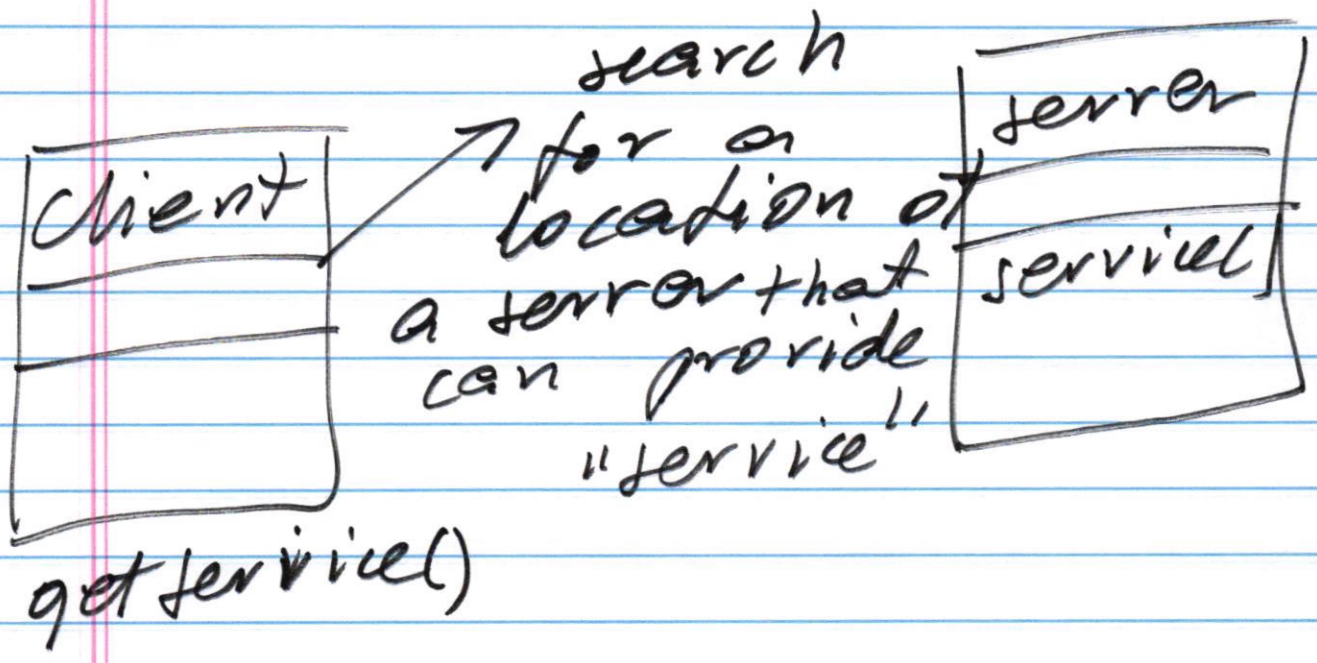

client
proxy
S1
busy
S2
busy
S100
idle

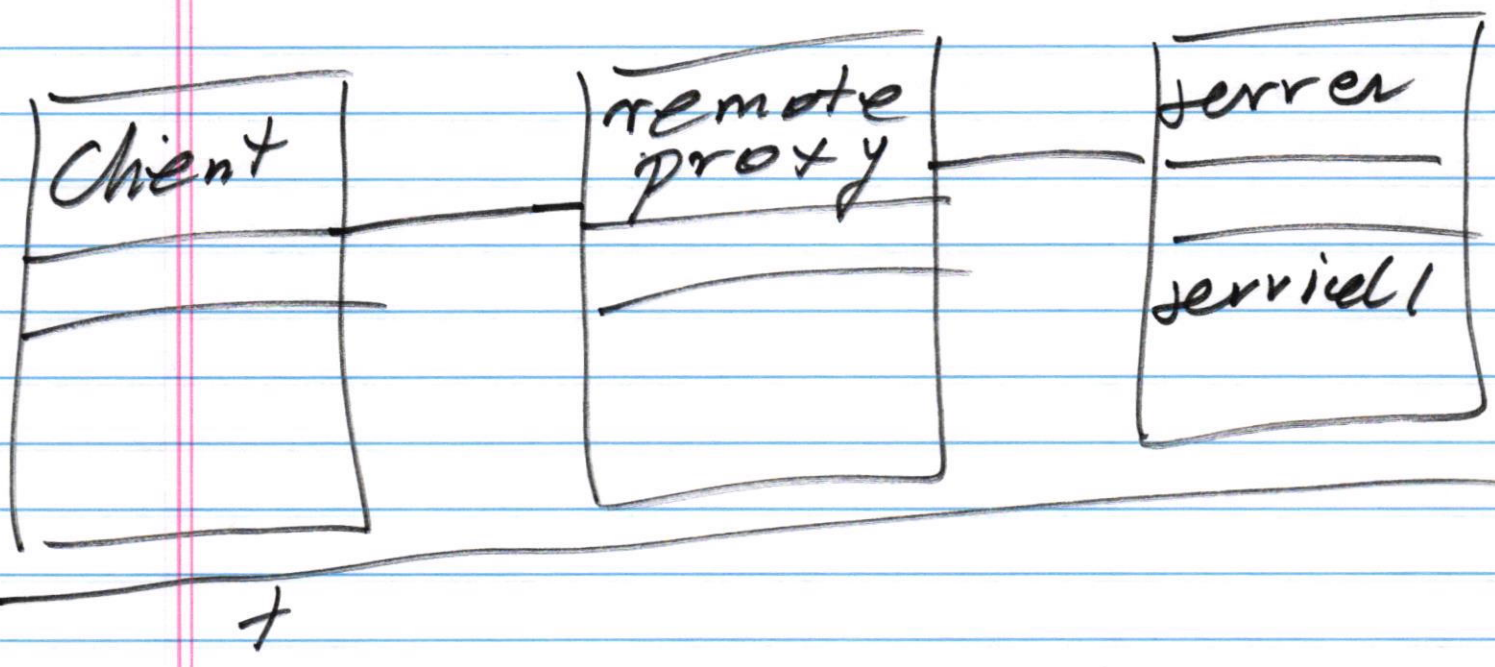
"protection" proxy.

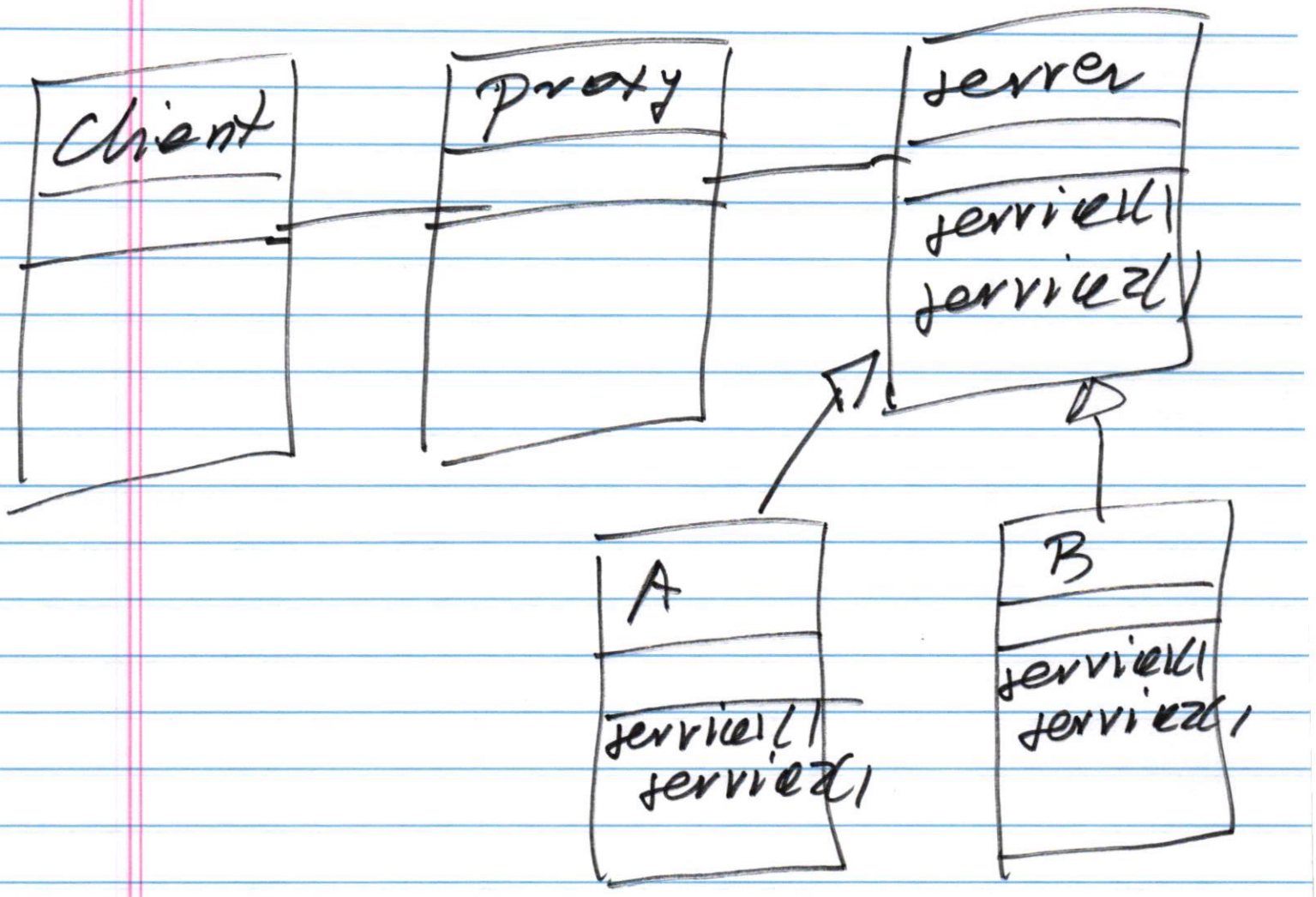
to protect server from unauthorized access



"remote" proxy






Client
Proxy
server
service1()
service2()
A
service1()
service2()
B
service1()
service2()

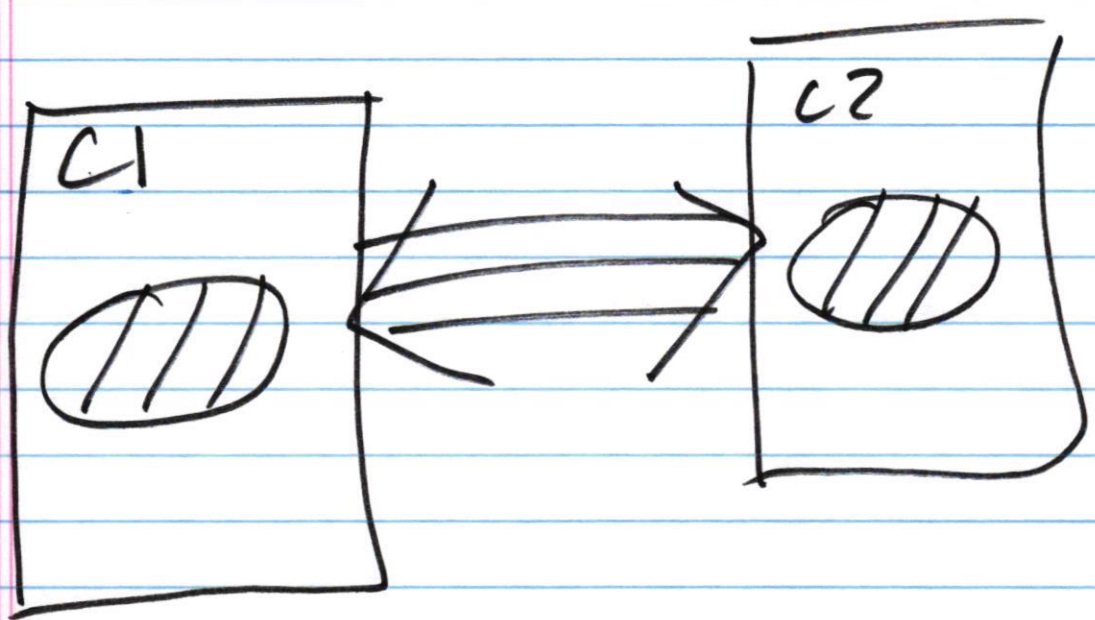
There are many other types of proxies.

check the textbook!!!

de-coupling patterns

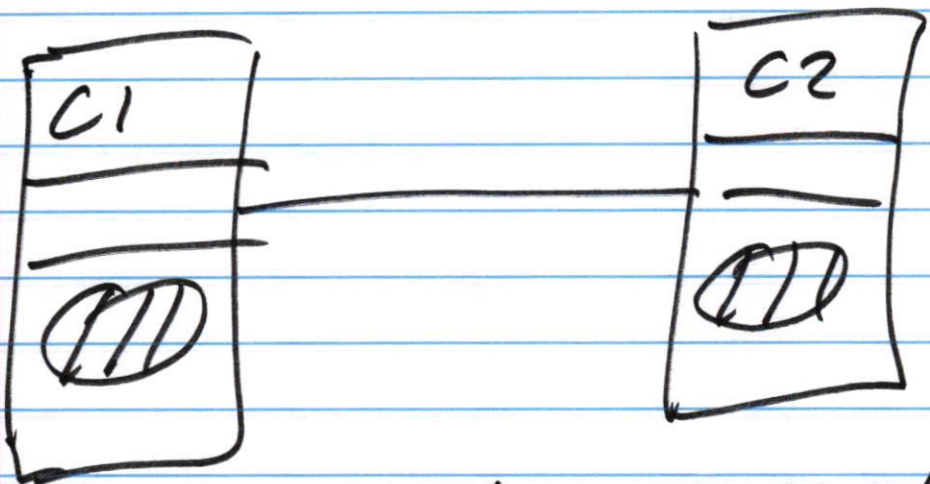
Problem

Two components are strongly coupled

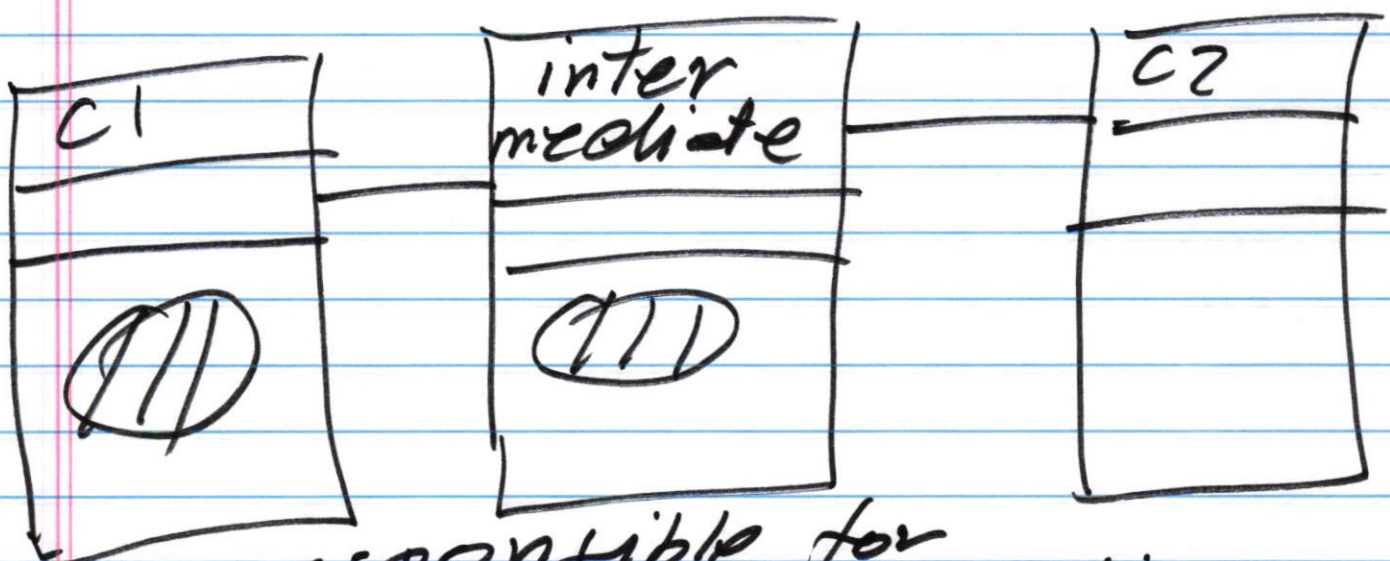


How to weaken the coupling between these components?

introduce an intermediate component



strongly coupled



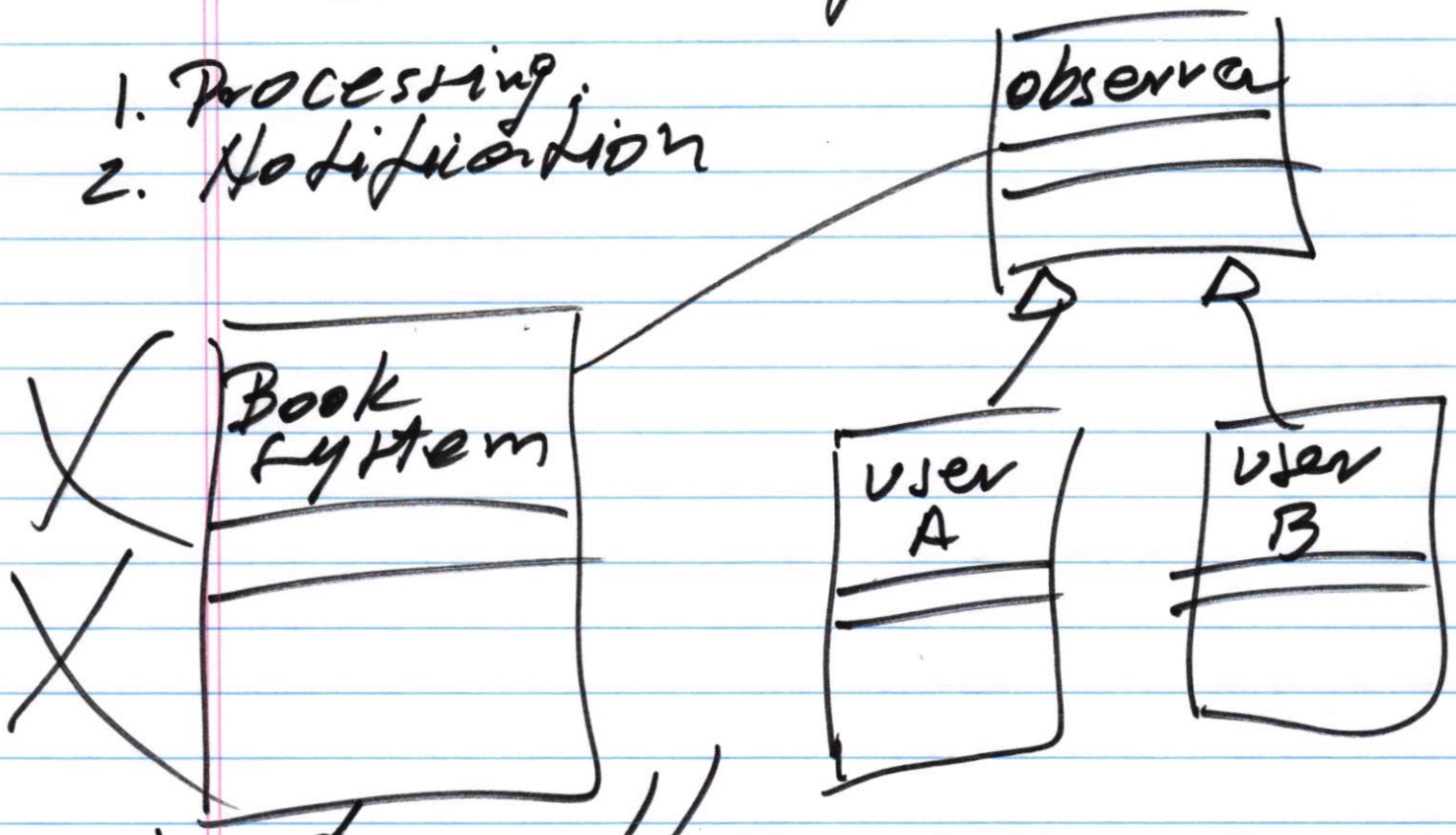
responsible for "weakening" coupling between C1 and C2

Homework #1

Problem #1

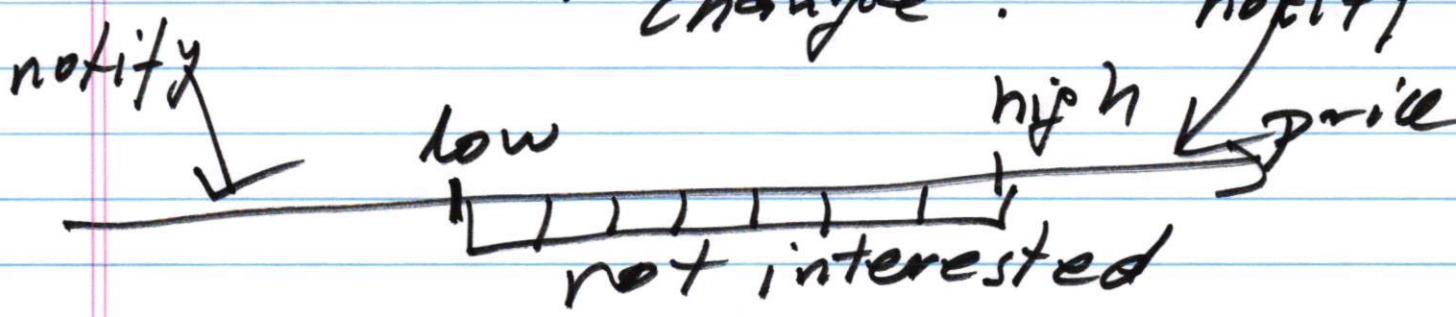
Observer pattern

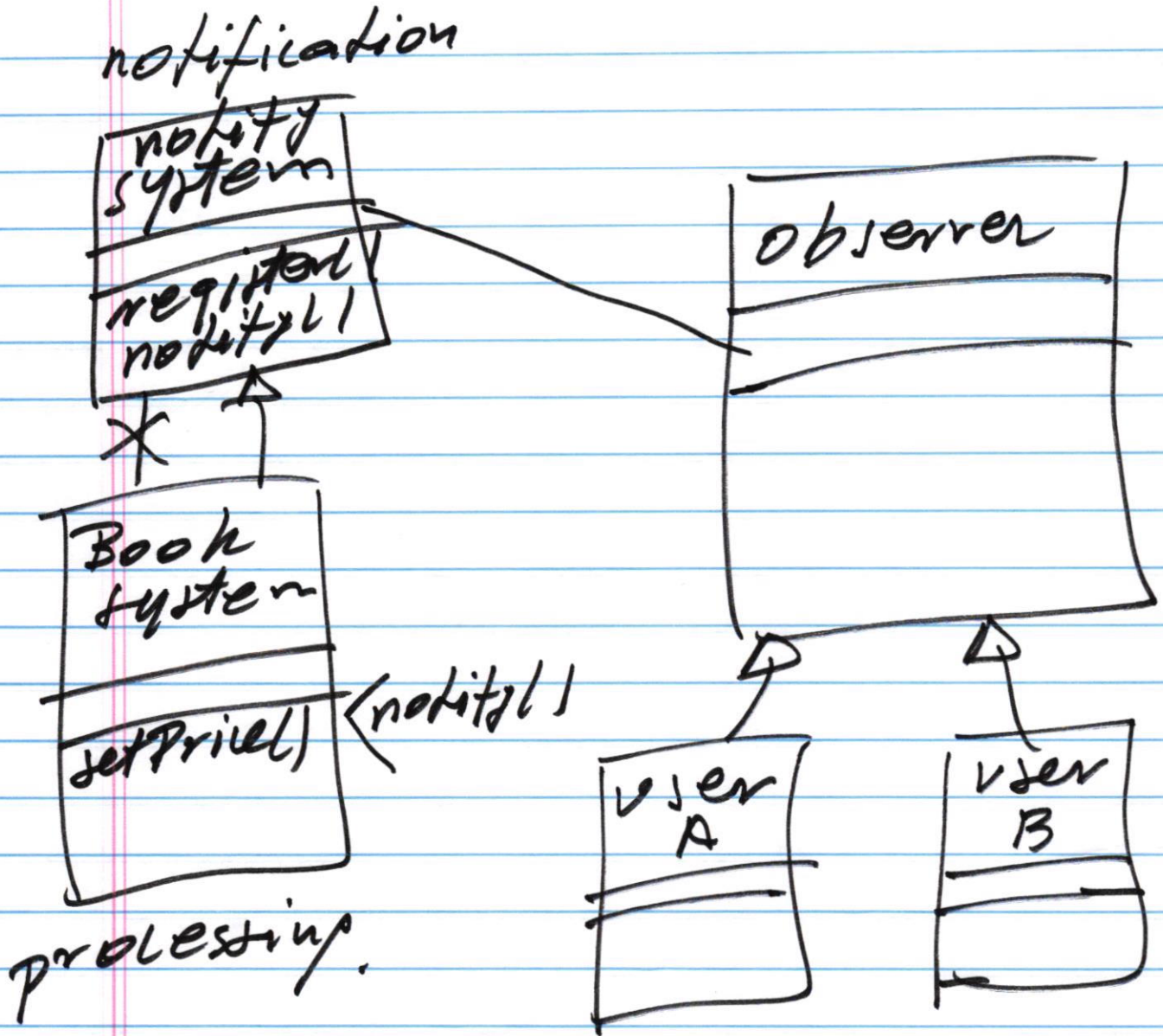
1. Processing.
2. Notification



bad Cohesion !!

out-of-range price change !

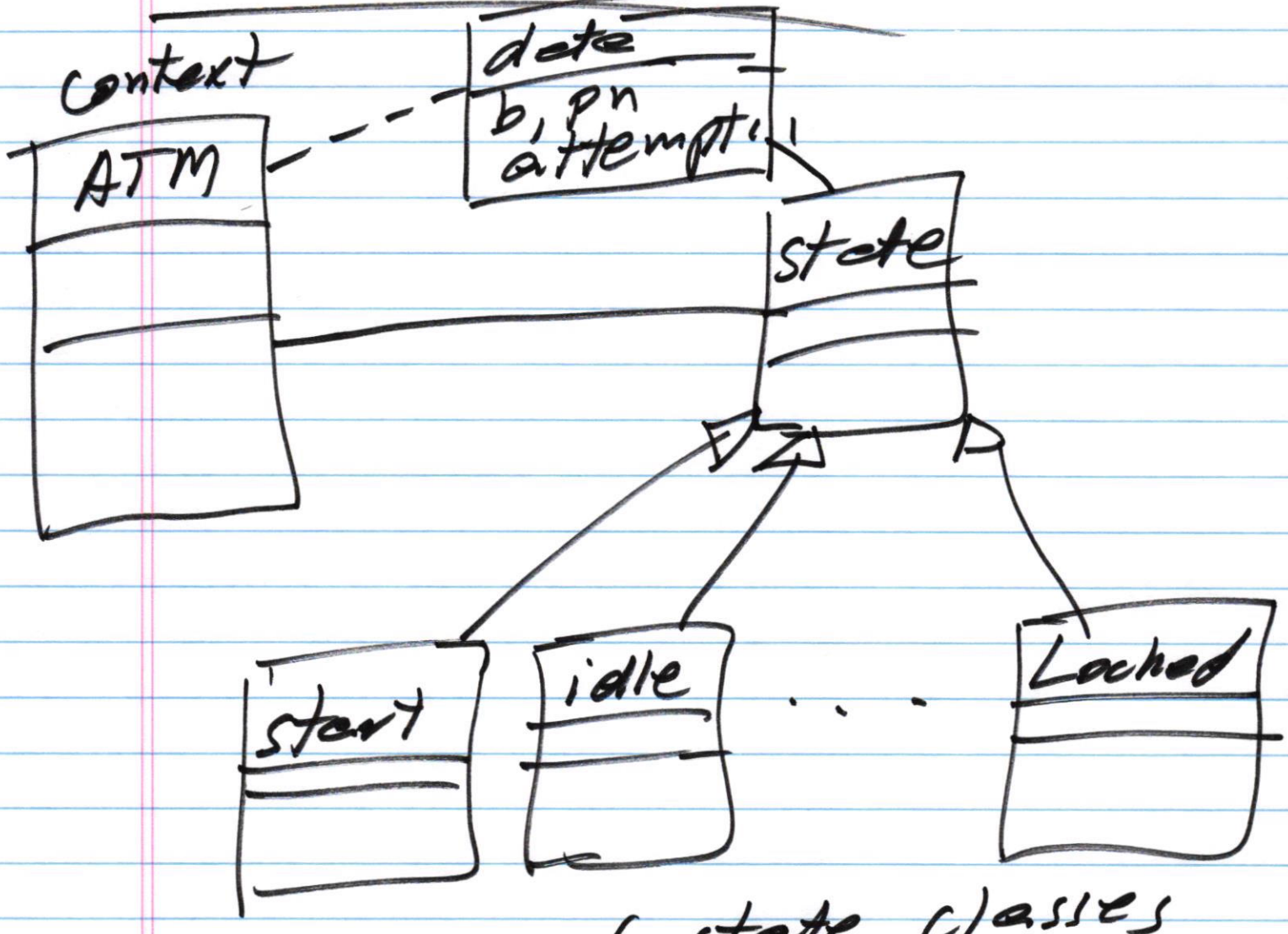



notification
notify system
register notify()
observer
Book system
setPrice()
notify()
user A
user B
protesting.

Problem #2

State Pattern

ATM component



6 state classes

(a) centralized

* ATM changes states
* state classes perform actions.

(b) de-centralized

ATM invokes operations on state classes

state classes
* change state
* perform actions.

HOMEWORK ASSIGNMENT #1

CS 586; Fall 2025
Due Date: **September 18, 2025**
Late homework: 50% off
After **September 23,** the homework assignment will not be accepted.

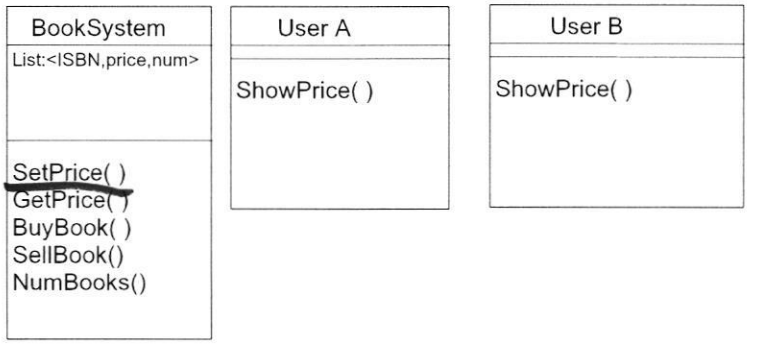
This is an **individual** assignment. **Identical or similar** solutions will be penalized.

Submission: All homework assignments must be submitted on Canvas. The submission **must** be as one PDF file (otherwise, a 10% penalty will be applied).

PROBLEM #1 (40 points)

In the system, there exists a class *BookSystem* which keeps track of prices of books in the Book Market. This class supports the following operations: *SetPrice(price,ISBN)*, *GetPrice(ISBN)*, *BuyBook(ISBN)*, *SellBook(ISBN),* and *NumBooks(ISBN)*. The *SetPrice(price,ISBN)* operation sets a new *price* for the book uniquely identified by *ISBN*. The *GetPrice(ISBN)* operation returns the current price of the book identified by *ISBN*. The *BuyBook(ISBN)* operation is used to buy a book identified by *ISBN*. The *SellBook(ISBN)* operation is used to sell a book identified by *ISBN*. The operation *NumBooks(ISBN)* returns the number of copies of a book identified by ISBN that are available in the system. Notice that each book is uniquely identified by *ISBN*.

In addition, there exist user components in the system (e.g., *UserA*, *UserB*, etc.) that are interested in watching the changes in book prices, especially, they are interested in watching the out-of-range book price changes. Specifically, interested users may register with the system to be notified when the price of the book of interest falls outside of the specified price range. During registration, the user needs to provide the boundaries (*lowprice*, *highprice*) for the price range for the specific book, where *lowprice* is the lower book price and *highprice* is the upper book price of the price range. At any time, users may un-register when they are not interested in watching the out-of-range book price changes of a specific book. Each time the price of a book changes, the system notifies all registered users (for which the new book price is outside of the specified price range) about the out-of-range book price change. Notice that if the book price change is within the specified price range for a given user, this user is not notified about this price change.



Design the system using the **Observer pattern**. Provide a class diagram for the system that should include classes *BookSystem*, *UserA,* and *UserB* (if necessary, introduce new classes and operations). In your design, it should be easy to introduce new types of user components (e.g., *UserC*) that are interested in observing the changing prices of books. Notice that the components in your design should be **decoupled** as much as possible. In addition, components should have **high cohesion**.

In your solution:

a. Provide a class diagram for the system. For each class, list all operations with parameters and specify them using **pseudo-code**. In addition, for each class, provide its attributes/data structures. Make the necessary assumptions for your design.

b. Provide two **sequence diagrams** showing:
   - How components *UserA* and *UserB* register to be notified about the out-of-range book price change.
   - How the system notifies the registered user components about the out-of-range book price change.

PROBLEM #2 (60 points)

The ATM component supports the following operations:

```
create()                  // ATM is created
card (int x, string y)    //ATM card is inserted where x is a balance and y is a pin #
pin (string x)            // provides pin #
deposit (int d);          // deposit amount d
withdraw (int w);         // withdraw amount w
balance ();               // display the current balance
lock(string x)            // lock the ATM, where x is a pin #
unlock(string x)          // unlock the ATM, where x is pin #
exit()                    // exit from the ATM
```

A simplified EFSM model for the *ATM* component is shown on the next page.

Design the system using the **State design pattern**. Provide two solutions:
- **a decentralized** version of the State pattern
- **a centralized** version of the State pattern

Notice that the components in your design should be **decoupled** as much as possible. In addition, components should have high **cohesion**.

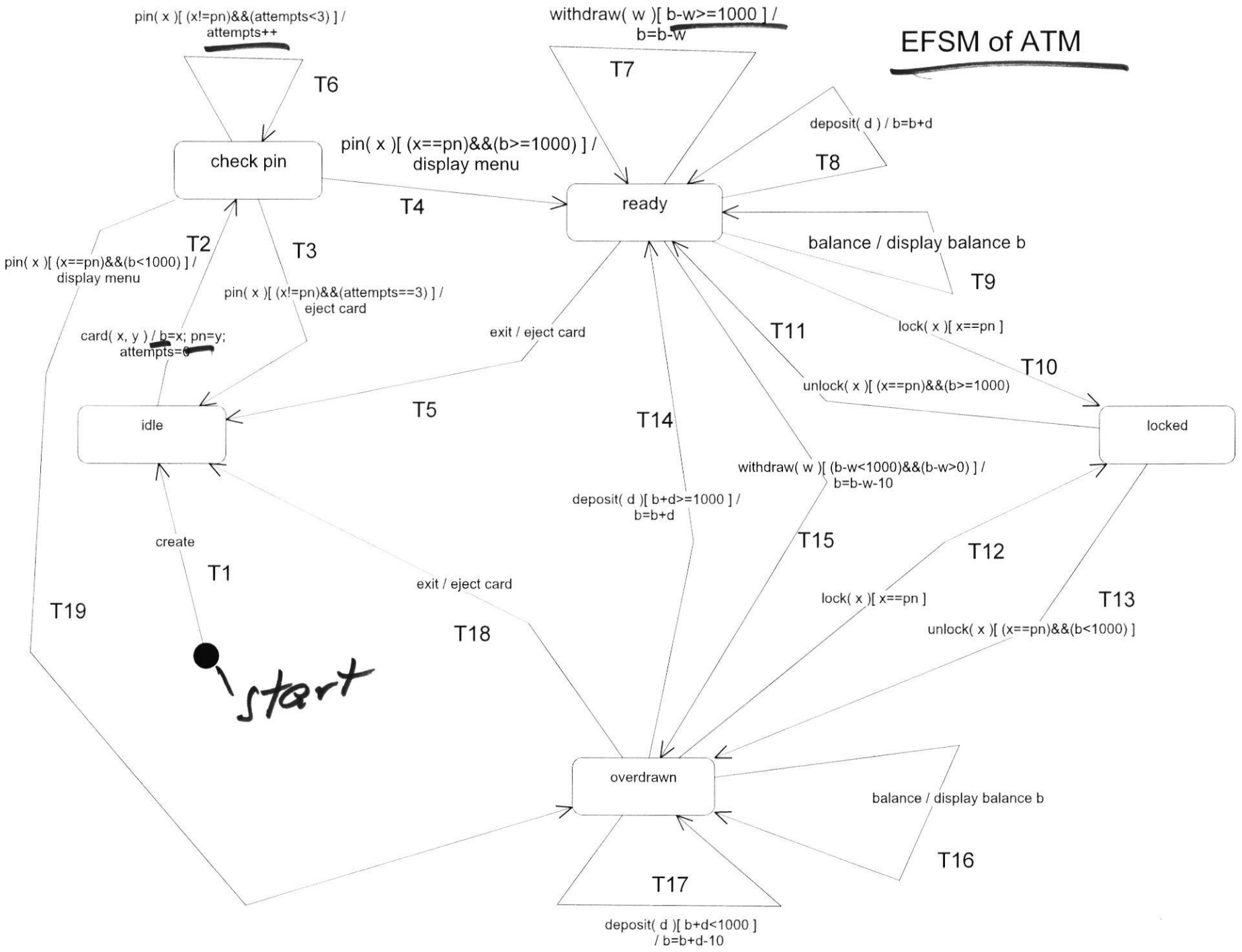
For each solution:

a. Provide a class diagram for the system. For each class, list all operations with parameters and specify them using **pseudo-code**. In addition, for each class, provide its attributes and data structures. Make the necessary assumptions for your design.

b. Provide a **sequence diagram** for the following operation sequence:

   create(), card(1100, "xyz"), pin("xyz"), deposit(300), withdraw(500), exit()

   When the EFSM model is "executed" on this sequence of operations, the following sequence of transitions is traversed/executed: T_1, T_2, T_4, T_8, T_{15}, T_{18}


EFSM of ATM
pin(x)[(x!=pn)&&(attempts<3)] /
attempts++
T6
withdraw(w)[b-w>=1000] /
b=b-w
T7
deposit(d) / b=b+d
T8
check pin
pin(x)[(x==pn)&&(b>=1000)] /
display menu
T4
ready
T2
T3
pin(x)[(x==pn)&&(b<1000)] /
display menu
pin(x)[(x!=pn)&&(attempts==3)] /
eject card
balance / display balance b
T9
card(x, y) / b=x; pn=y;
attempts=0
exit / eject card
T11
lock(x)[x==pn]
T10
unlock(x)[(x==pn)&&(b>=1000)
idle
T5
T14
locked
withdraw(w)[(b-w<1000)&&(b-w>0)] /
b=b-w-10
deposit(d)[b+d>=1000] /
b=b+d
T15
T12
create
T1
exit / eject card
lock(x)[x==pn]
T13
T19
T18
unlock(x)[(x==pn)&&(b<1000)]
start
overdrawn
balance / display balance b
T16
T17
deposit(d)[b+d<1000]
/ b=b+d-10