Lecture 8: Design Patterns

Page No.
Date

What is Design Patterns and why do we use them?

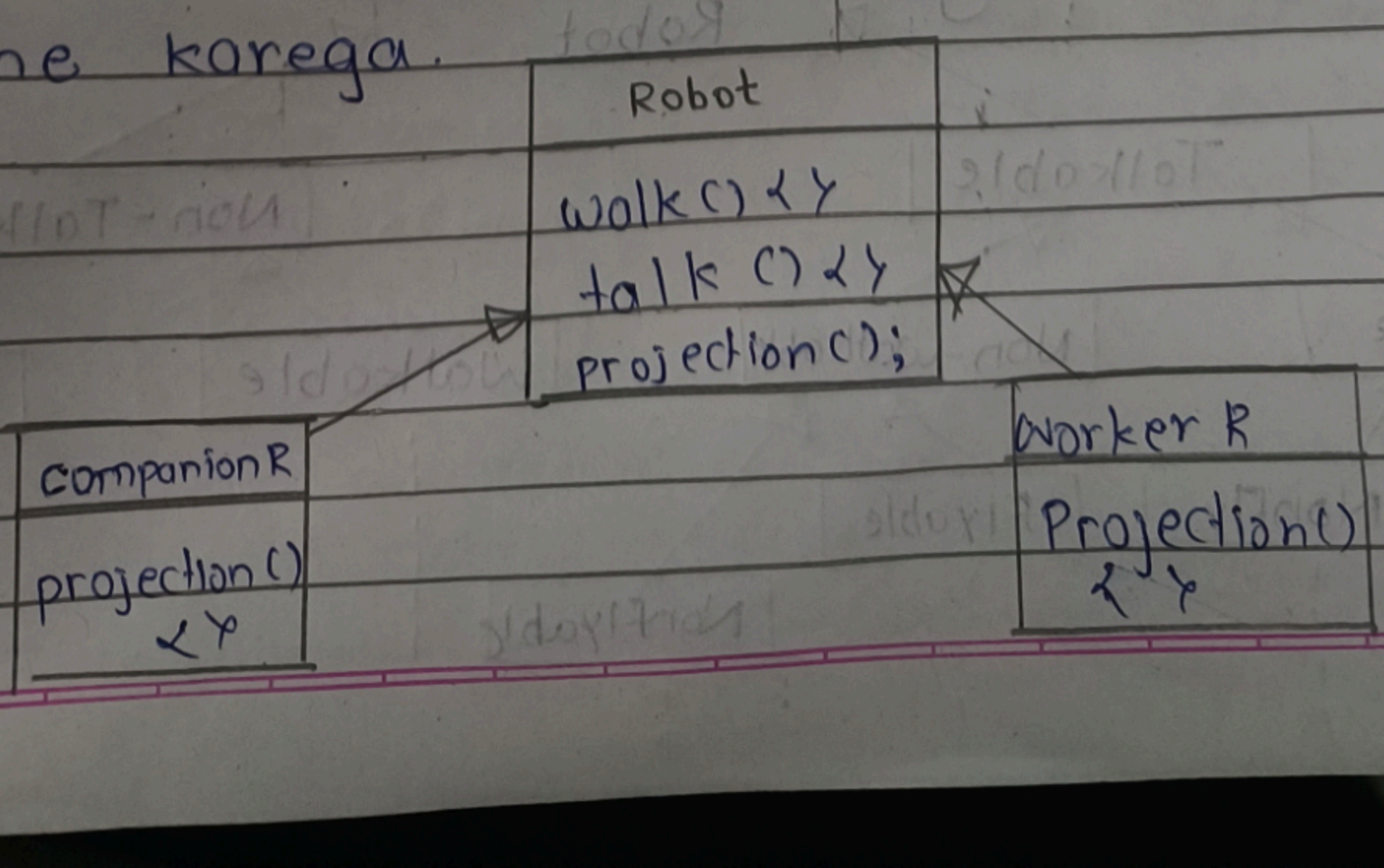
- Ab hum koi nayi application jab develop kar rahe hai toh humme kuch problems aa skte aur yeh problems humare pehle bhi bahoot logo ko aye hai toh unhone woh pattern de diya ki aise problems ko iss tarah tackle karo.

⟹ Change is only constant ... (even in IRL)

- Inshort What all design patterns will suggest?
Humare application ~~ke~~ mein two parts hote - ek static (joh change nahi hota) aur ek dynamic (joh baar baar change hota hai). toh hume bass in hisso ko humesha alag rkhna hai.
⇒ Isolation of static.

Design Pattern 1: Strategy Design Pattern

Let's take example:
Ab hum ek robot class bana rhe sbka apna style/projection() hoga, walk(), talk() methods honge. walk() & talk() method common honge ∴ main class mein define honge aur projection() ko har koi robot child class apne tarah se define karega.



* This method will help you to hate Inheritance!

Page No.
Date

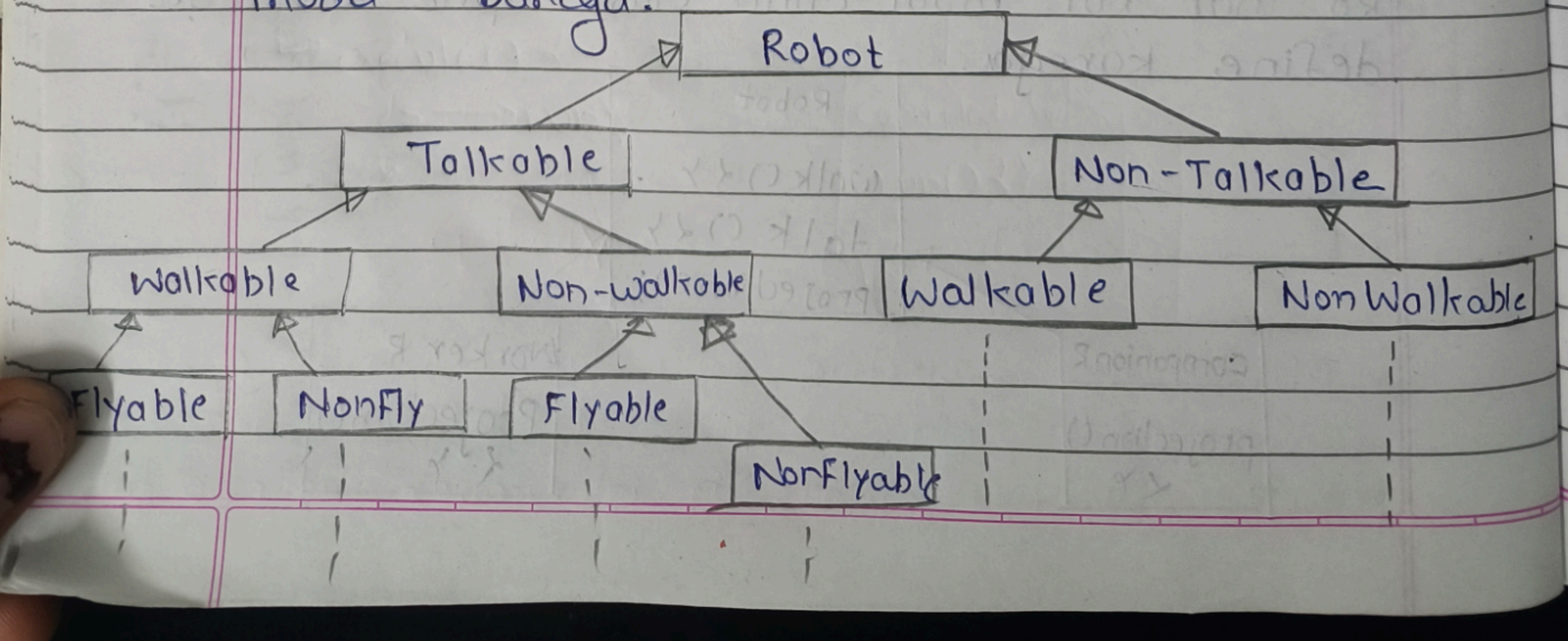
→ Ab problem start hoga jaise jaise alag features yaa features ke child feature wale robot aana start hoge

→ Jaise ek sparrowR joh fly() kar skta hai then aur multiple Robots aaye joh bhi fly() kar skte. Ab har baar unn robots ke liye mujhe same code yaane fly() ka code repeat krna hoga which breaks principle DRY - Don't Repeat Yourself

→ Ab ek approach ayegi main class mein add krdo par joh robots fly nhi kr skte unka toh woh class narrow/ blocking code likhna pdega like ki fly() nhi kr skta robot

→ Another approach could be implement inheritance jaise humne (Account - FD) wale mein kiya tha

→ But problem is yeh ek hi feature nhi hai suppose koi Jet ke madat se udta ho robot; yaa koi alag tarah se walk/talk krta ho toh iska solution agar inheritance se kiya toh koch aisa confusing sa model banega.



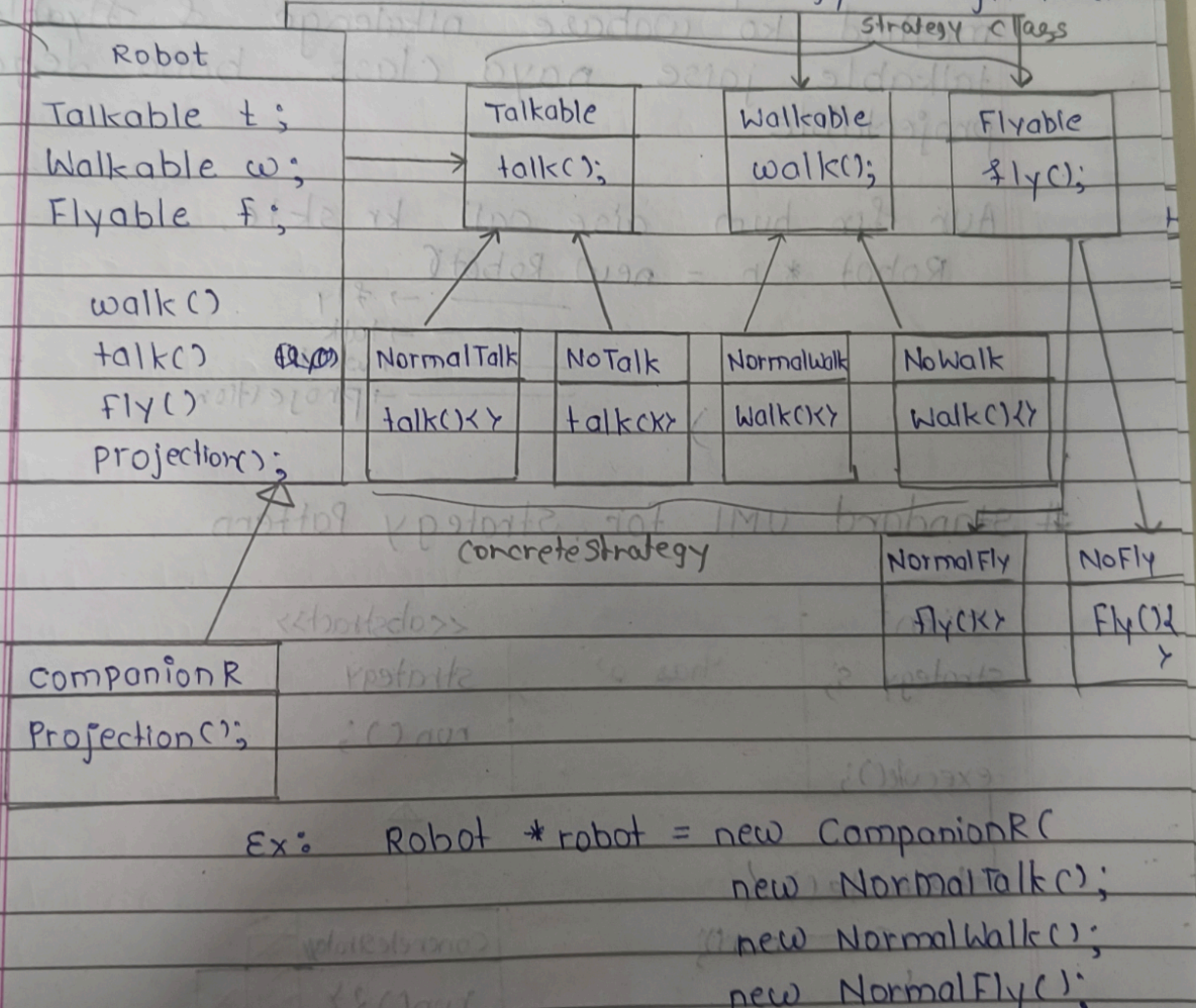
Favour Composition over Inheritance

Page No.
Date

Inshort these problems we faced with Inheritance

- Code Reuse
- To add new feature a lot of changes were required
- Breaking OCP

Solution: Using Composition ⇒ Strategy Design Patter



Ex: Robot *robot = new CompanionR(
new NormalTalk();
new NormalWalk();
new NormalFly();
);

Strategy Design Pattern
↳ Defines a family of algorithm, put them into separate classes so that they can be changed at run time

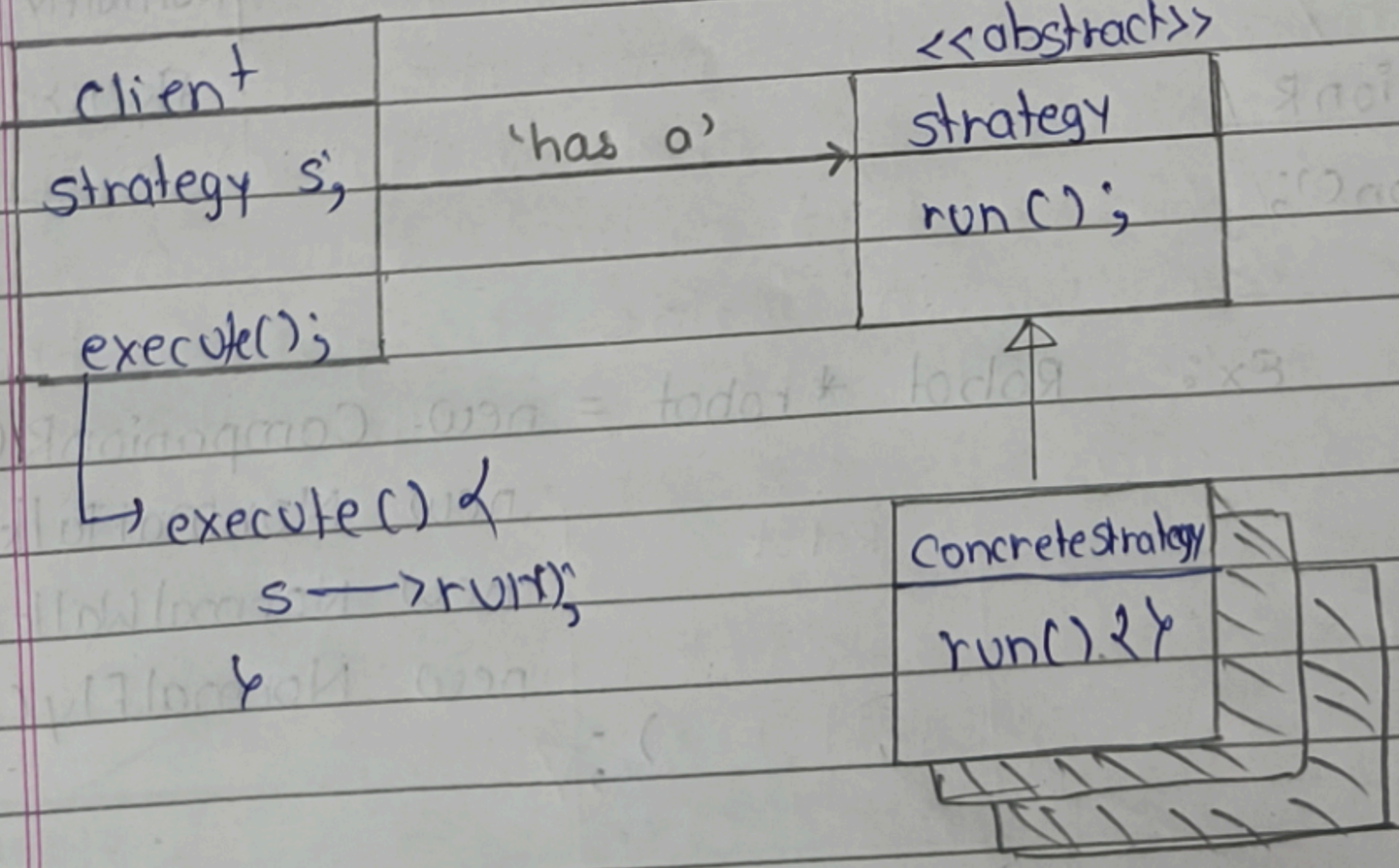
- Now in optimized way agar hume nayi Flying method banani hai so just add one more class in Flyable (no brainer :).

Question:
Ab humare jab robots ban rhe fir bhi hum inheritance use kar rahe hai toh hum woh bhi optimize kar skte projection ke method ko wahase nikalenge & flyable/ talkable jaise naya class bana dege projectable

Aur fir hum aise call kr skte.

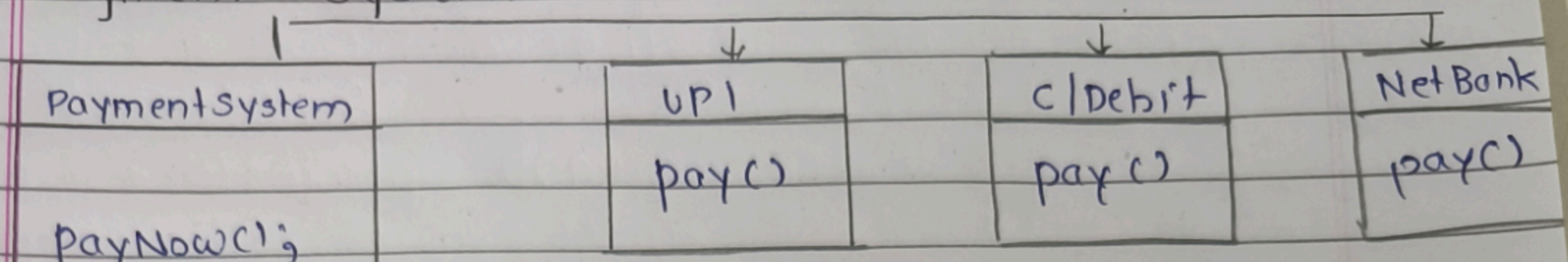
```
Robot *r = new Robot(
                    → fly
                    → talk
                    → walk
                    → projection
)
```

Standard UML for Strategy Pattern

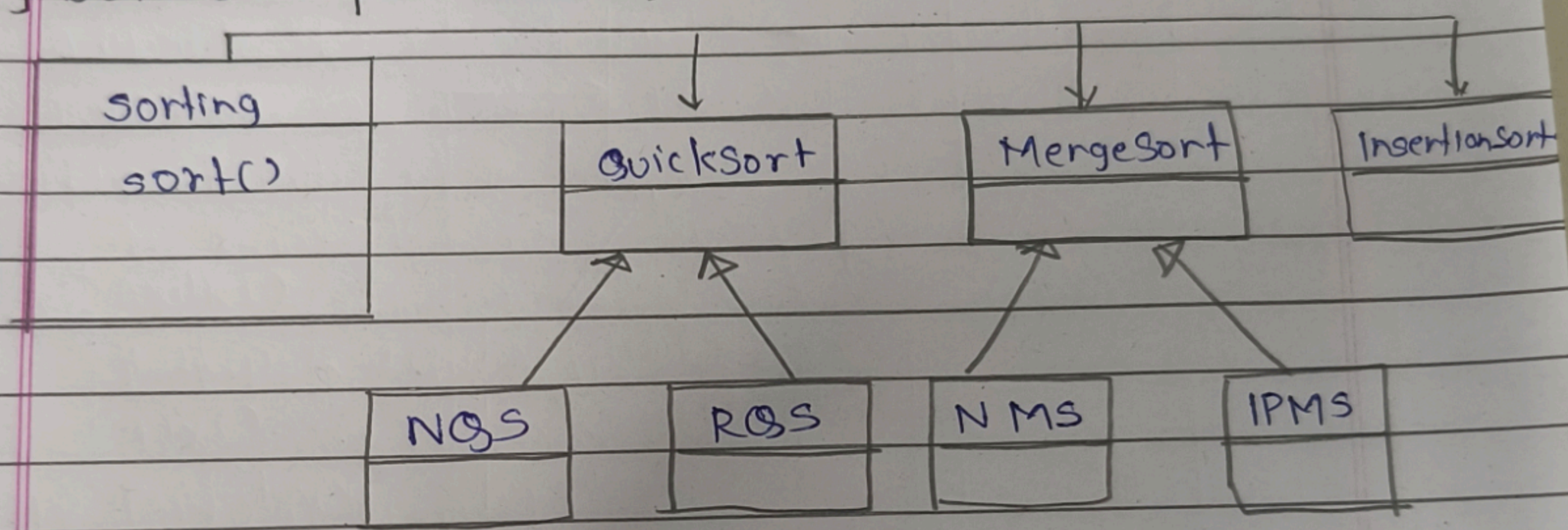


Real-life example

1] Payment system.



2] DSA Example sorting



Conclusion

1. Encapsulate what varies & keep it separate from what remains same

2. Solution to inheritance is not more inheritance

3. Composition should be favoured over inheritance

4. Code to interface & not to ~~coneredu~~ concretion

5. DRY - Do Not Yourself