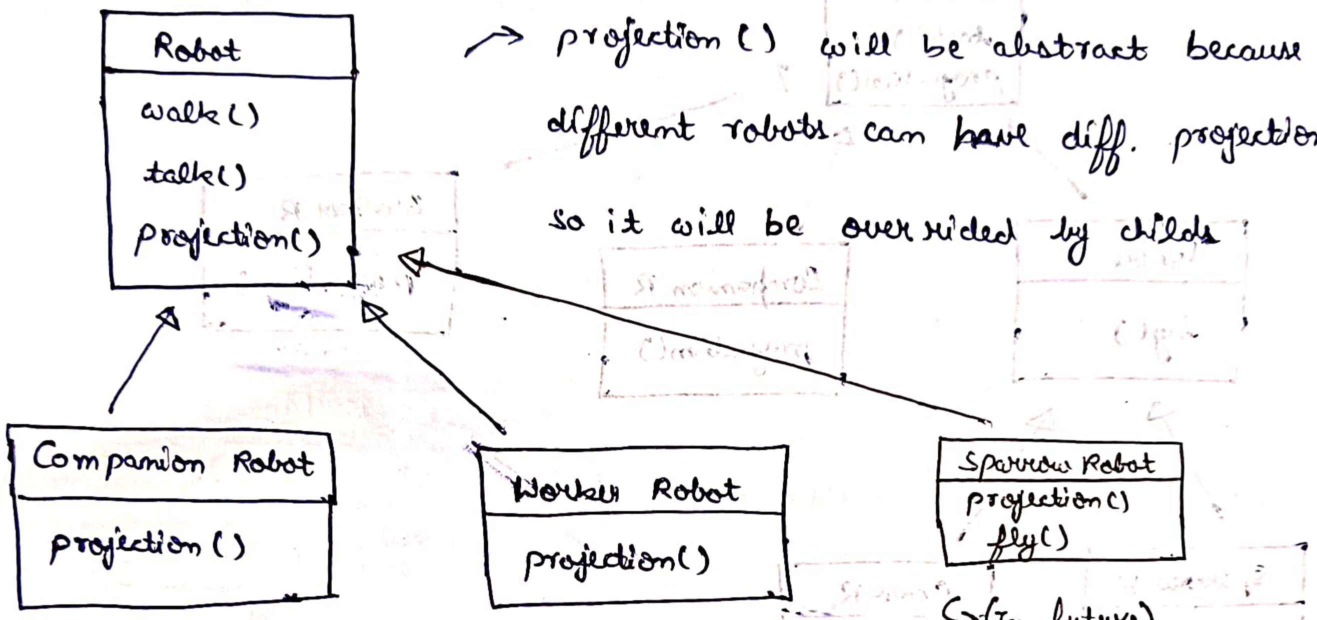
Strategy Design Pattern

Suppose I want to make an application in which there will be robots & they can perform some simulation (walk, talk, projection)

So, the basic Robot class will look like ↴

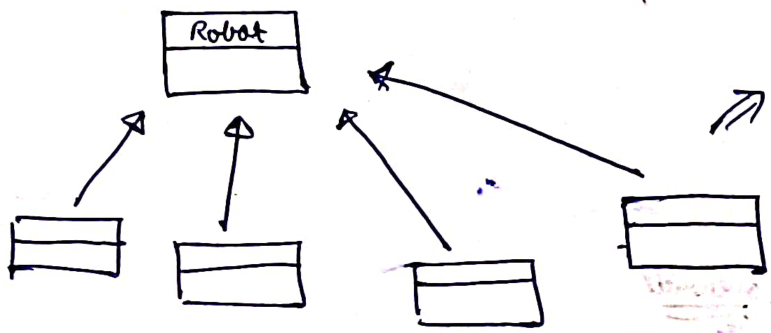


→ projection() will be abstract because different robots can have diff. projections so it will be over-rided by childs.

This design is good but now let's say we have one more robot in future called sparrow Robot & it can fly also.

⇒ So, to implement this we will just inherit the Robot class & override the projection method & have a new fly method.

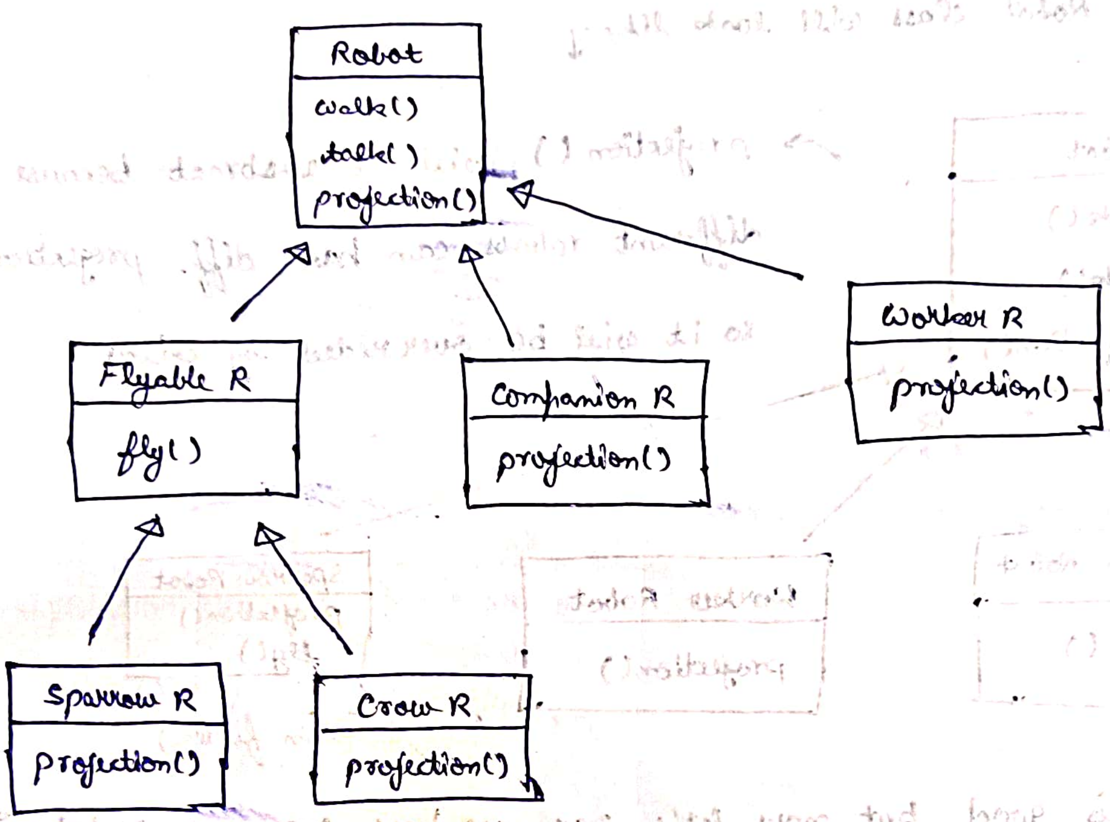
But their is a problem in this design, suppose in future more robots came in like Crow Robot, Pigeon Robot, etc. & all have fly() method then the design will look like ↴



↗ New robots have fly() method which is same so it is breaking DRY principle

And also we can't define fly() in parent because initial robots can't fly.

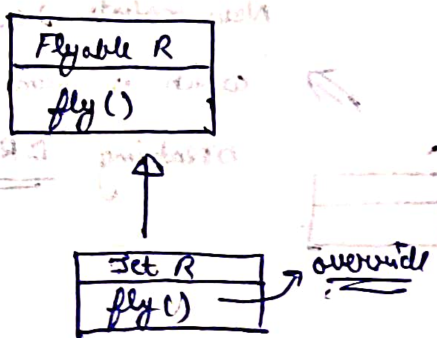
2nd way :- Increase the heirarchy of inheritance



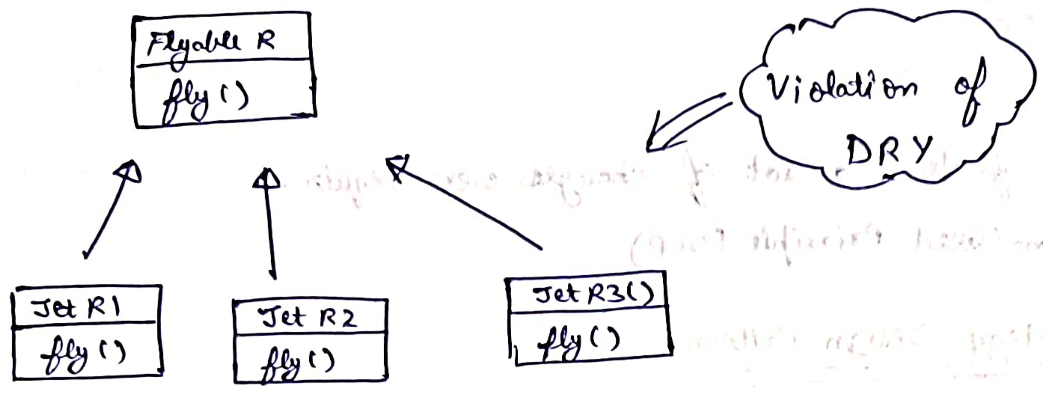
But is the problem solved now ??

Not really because let's suppose In future Jet Robot came in which can fly but not with wings as defined in flyable robot class but with Jet.

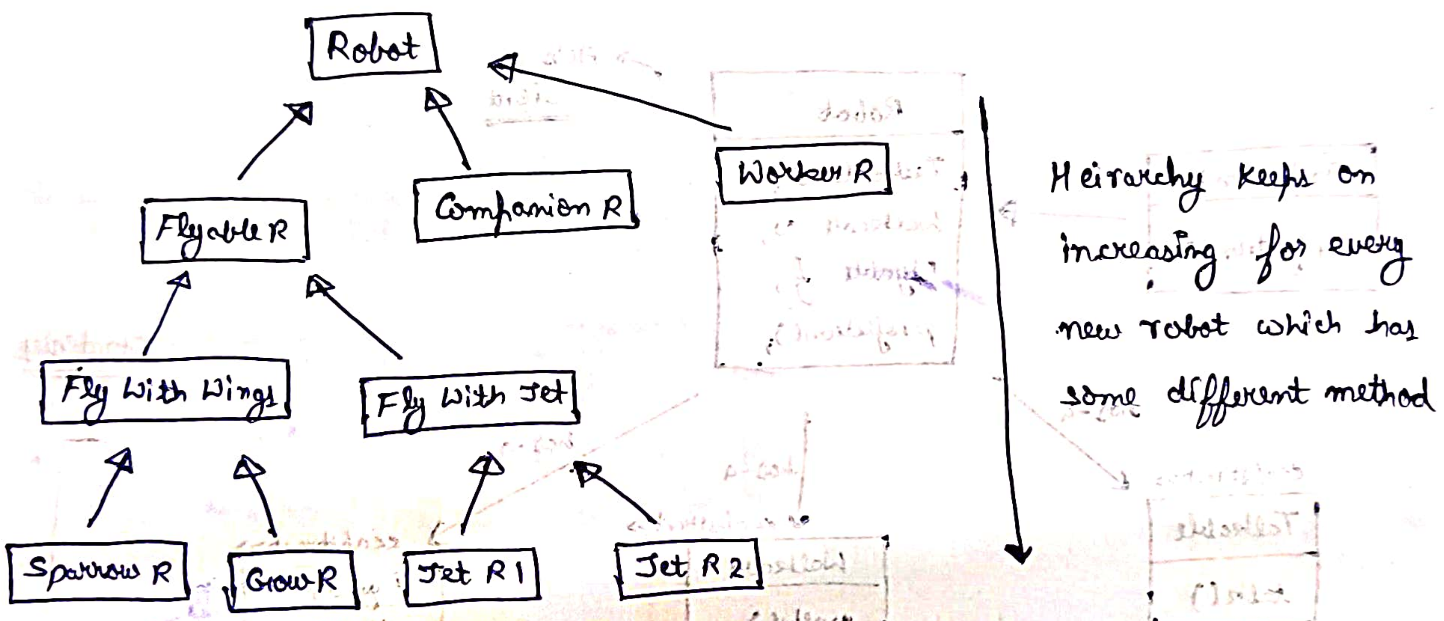
So, what just make a class of Jet Robot (inherit from flyable R) and override the fly() principle.



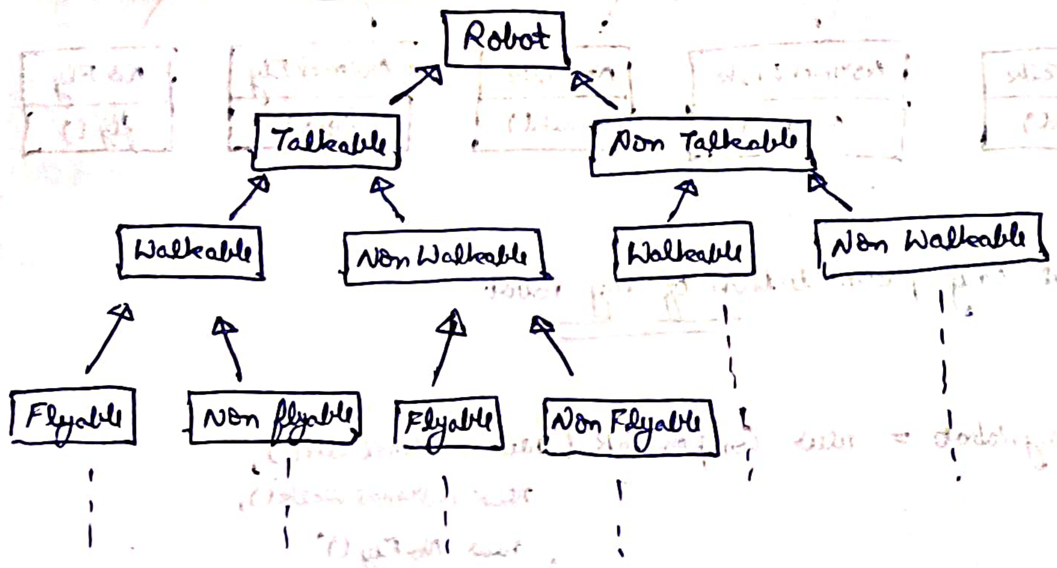
But suppose multiple Jet Robots came in (Jet R1, Jet R2, Jet R3, etc.) then for each we have to override fly() & is repeatable so DRY breaks again



↳ Just create/increase the heirarchy by making a class of Jet Robots



So, it will be very complex



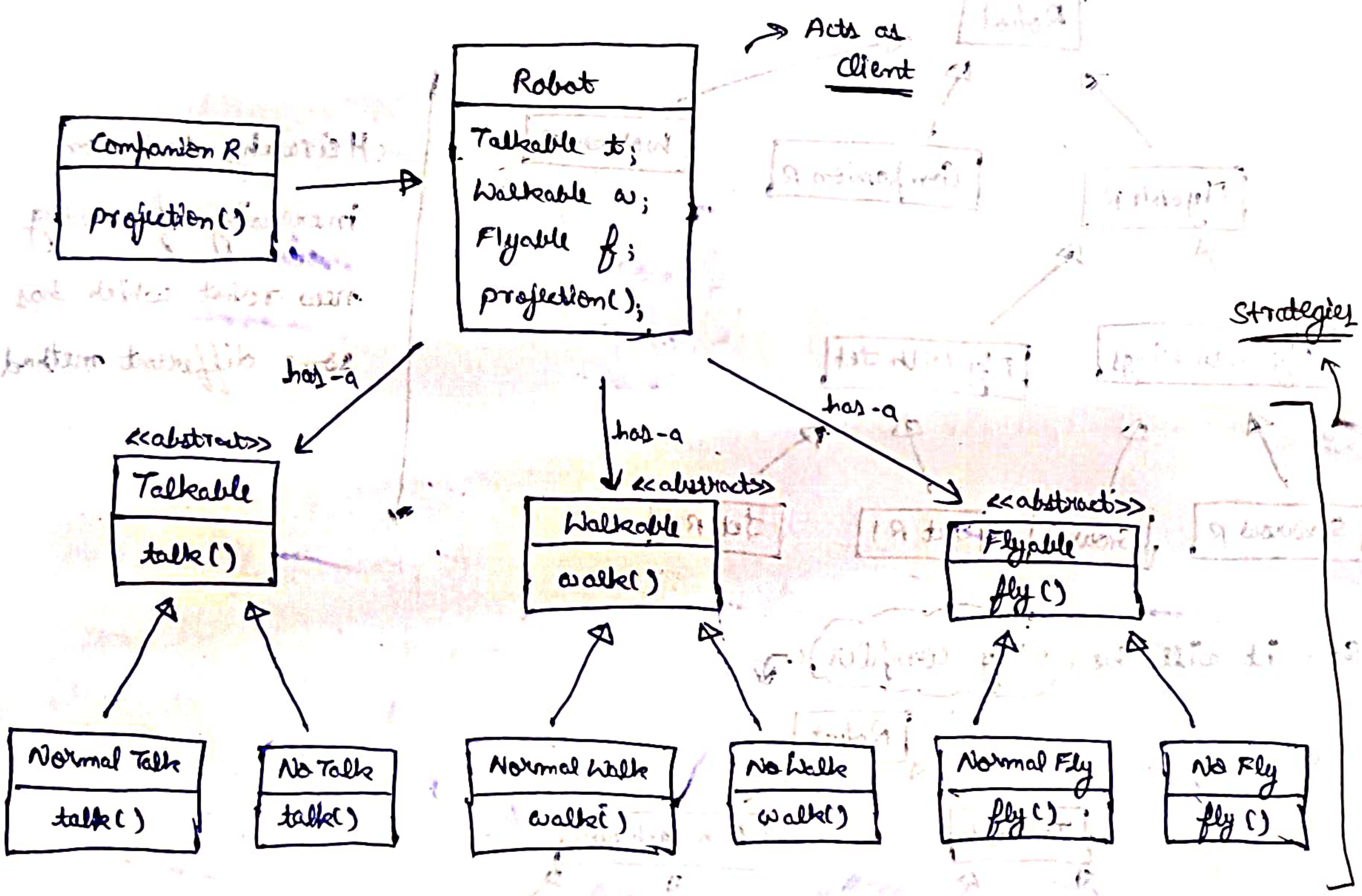
⇒ The Solution to Inheritance is not more Inheritance

Summary of Problems :-

- Code Reuse
- To add new feature a lot of changes were required
- Breaking Open-Closed Principle (OCP)

So, what is Strategy Design Pattern

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



Now, I can have any permutation of my robot

```
Robot*   myRobot = new CompanionR ( new NormalTalk(),
                                     new NormalWalk(),
                                     new NoFly()
                                   );
```

Now Robot is just delegating different methods.

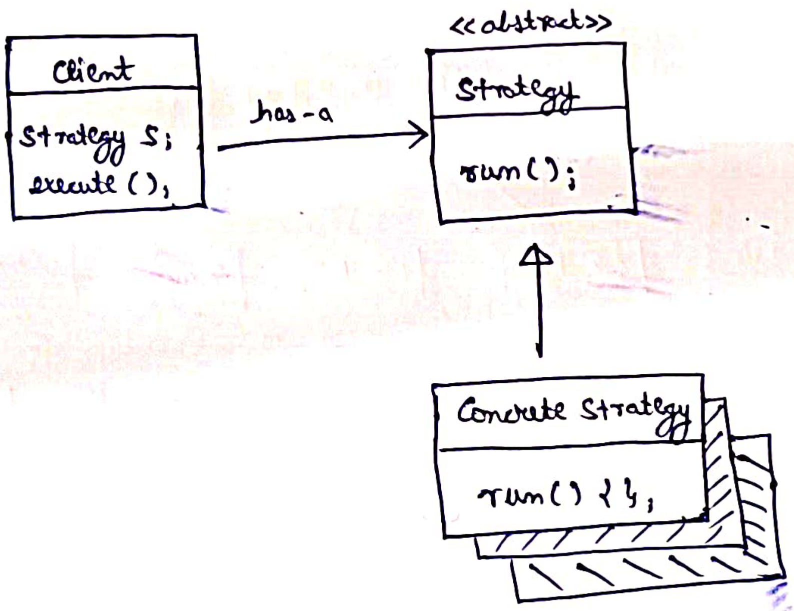
So, we break Inheritance into Composition

Now, suppose I have a robot that can fly with Jet, so just I have to, make a FlyWithJet Class & inherits from Flyable interface (abstract class)

But you will say that in current ~~day~~ design also there is Inheritance (from Robot to Companion Robot, Worker Robot, etc.) because of projection()

So, we can make new abstract class as well called projectable & will have a has-a relⁿ & override different projections from Projectable abstract class

Standard UML Diagram

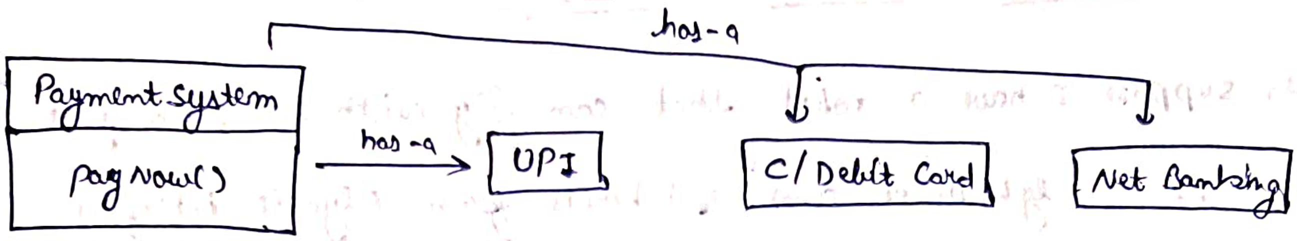


Real life Examples :-

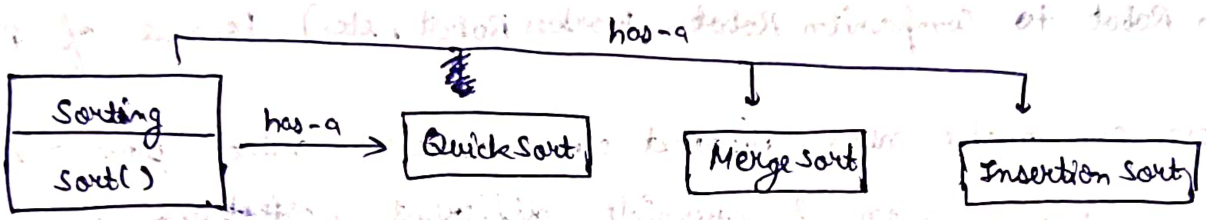
↓

① Payment System

↳ can has multiple strategies



② Sorting strategies



has-a
Sorting
sort()
has-a
QuickSort
Merge sort
Insertion Sort