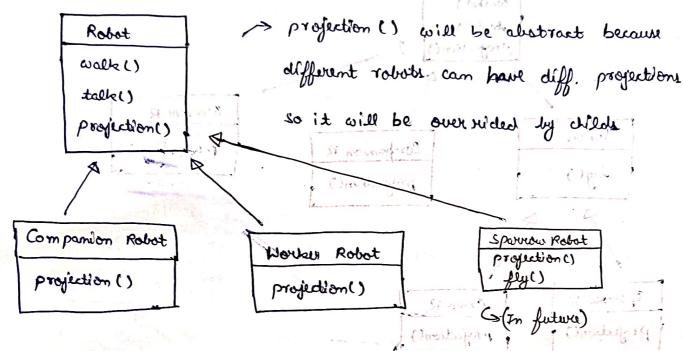
Suppose I want to make an application in which there will be robots & they can perform some simulation (walk, talk, projection)

thatter, in their entire the

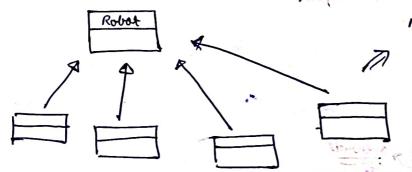
So, the basic Robot class will look like of



This degign is good but now let's say we have one more robot in future called spavnow Robot & It can fly also.

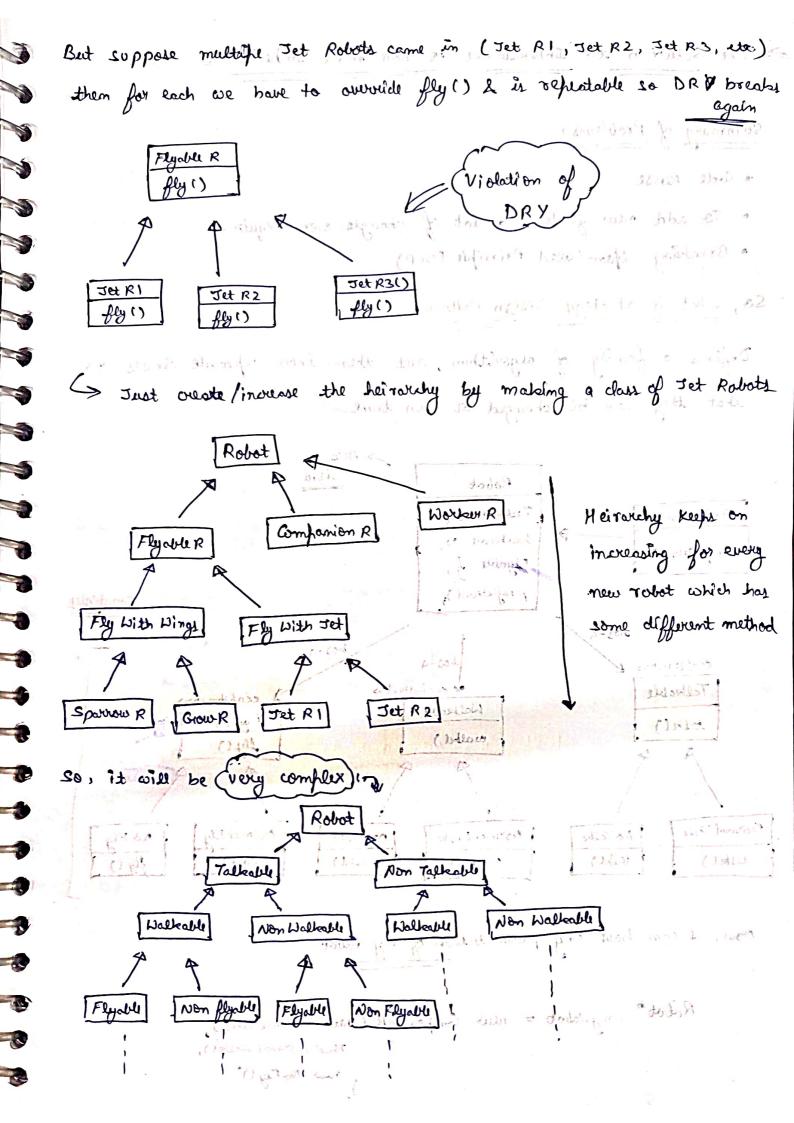
⇒ So, to implement this are will just inherit the Robot class & overwide the projection method & have a new fly method.

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 than the degign will look like in



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

And also we can't define fly () in povent because initial robots can't status is a three of the color in which the rotate be robated 2nd way 1- Invelose the herrarchy of inheritance most more gett is I the brook 12to 120% class short ett, 02 Robot projection() wolled R Flyable R projection() Companion R fly() projection() () - 10 11-15 4 Sparrow R Crow R. projection() The digina is good but now letter easy one how one more report in Box is the problem solved nowing to a touch converge belle: exected shirted in wall todon with diverting that this go the transferri 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 Did of it esoppes, reject which is meldorg a is girls the raine in like Crow Robot, Physica Robot all. so, what just make a class of Jet Robot (inherit from flyable R) and owwide the fly () principle. Flyable R fly()



> The Solution to Inheritance is not m	ore Inheritance elegges to
ido no Vari no dilettro for it & Child distriction	o of sixe on those that world
Sommary of Problems 1-	Figelili 12
· Ode Reuse () militarile	E (1892)
· To add new feature a lot of changes were required	
· Breaking Open-Closed Principle (OCP)	
So, what is Strategy Design Pattern !	(18)
Defines a family of algorithm, put the	m into separate classes so
≫ A	cts as / fall of
/ Robot	Client ?
7 11 24	,
	Since (mel) Granping
projection() Flyable f.	zá la
	19 L
projection();	Strategies 13th AVENCET
haten tamples hos-a	Strategies
haten tample hos-a kabatratos	tot de strategies
profession(); hos-a **Cabatratos** Talkable Walkable Walkable	Strategies has-a coabetraction
projection(); hos-a Talkable Accounted to the contraction of the co	Strategies has-a «abstract»
roten tour hor-a **Cabstrates & Talkable talk() A A	has a cabetraction fly ()
roten tour hor-a **Cabstrates & Talkable talk() A A	Strategies has-a «abstract»
Talkable talk() A A A A A A A A A A A A A	Strategles has-a Cabetractos fly () A A A A A A A A A A A A A
Mormal Talk No Talka	has -a <abstractor ()="" fly="" fly<="" flyphle="" no="" romal="" td=""></abstractor>
Talkable talk() A A A A A A A A A A A A A	Strategles has a cabetractor Flyphle (1) Re Normal Fly No Fly
Mormal Talk talk() No Talk talk() No Talk talk() No Talk talk()	has -a <abstractor ()="" fly="" fly<="" flyphle="" no="" romal="" td=""></abstractor>
Normal Talk talk() No Talk talk() No Talk talk() No Talk talk()	Strategies has a construction fly () Abornal Fly fly () fly () fly ()
Normal Talk talk() Ab Talk talk() Ab Talk talk() Ab Talk talk()	Strategies has a construction fly () Abornal Fly fly () fly () fly ()
Normal Talk talk() No Talk talk() No Talk talk() No Talk talk()	Strategies has a construction fly () Abornal Fly fly () fly () fly ()
Mormal Talk About Mow, I can have any permutation of my ro Robot* my Robot = new Companion R (n	Strategies has -a exabetractors fly C) Abormal Fly fly C) fly C) Abormal Fly fly C)
Mormal Talk About Mow, I can have any permutation of my ro Robot* my Robot = new Companion R (n	Abstractory Aly () Aly () Ply ()
Mormal Talk About Mow, I can have any permutation of my ro Robot* my Robot = new Companion R (n	Strategies has -a exabetractors fly C) Abormal Fly fly C) fly C) Abormal Fly fly C)

OF

M.

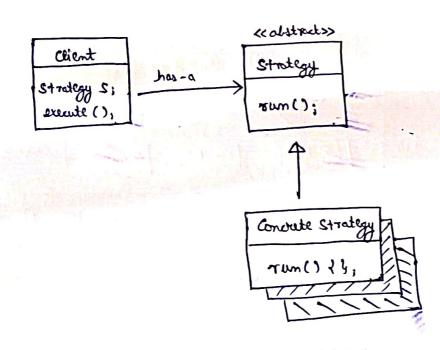
Now Robot is just delegating different methods.

So, we break Inheritance into Composition

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

But you will say that in convent 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 hos-a rely & overvide different projections from Projectable abstract class

Standard UNL Diagram



Real life Examples!

