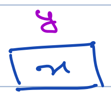
Agenda :-

-> Decorator

-> flyweight.

Decorator :- wrapper



+ add some functionalities

int

↓

Integer

Starbucks

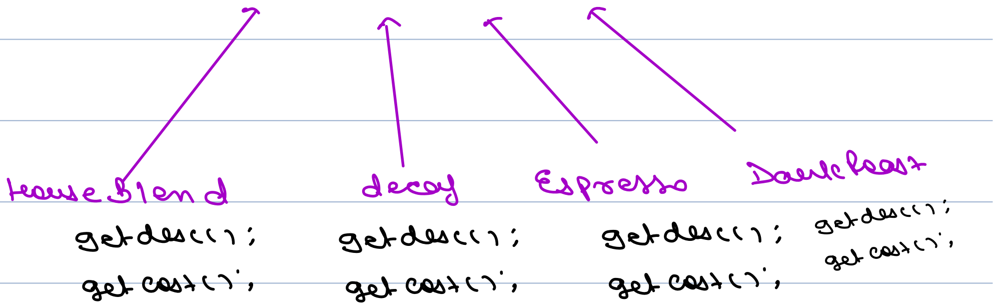
- Build the beverage
- description of beverage
- cost.

Interface

abstract Beverage

getdesc() —

get cost() —



HouseBlend

getdesc();

get cost();

decay

getdesc();

get cost();

Espresso

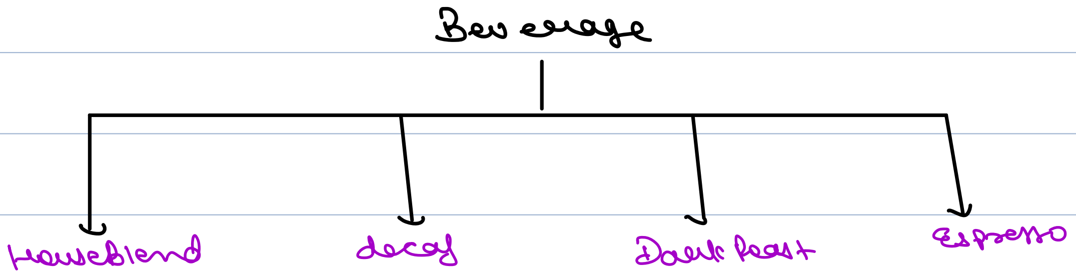
getdesc();

get cost();

Darkroast

getdesc();

get cost();



getdesc();
get cost();

new Decaf();
new Espresso();

these are add-ons .. will have extra cost

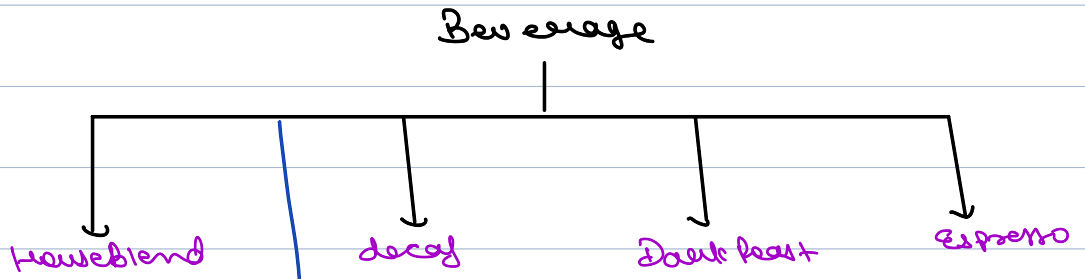
DarkRoast + Milk — milk → getCost()
→ getdesc();

DarkRoast + Milk + whip — whip

— soy

DarkRoast + 2Milk — mocha

even if we add some things it still should be same type..... for e.g. if coffee is beverage and we add extra milk it will still beverage.. so nature of object shouldn't change.



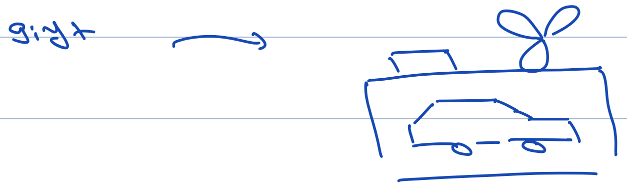
getdesc();
get cost();

HouseBlend with Milk
getdesc();
get cost();

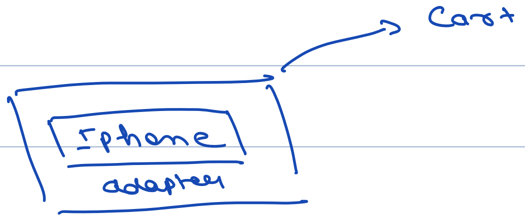
if we add extra class for every combination then

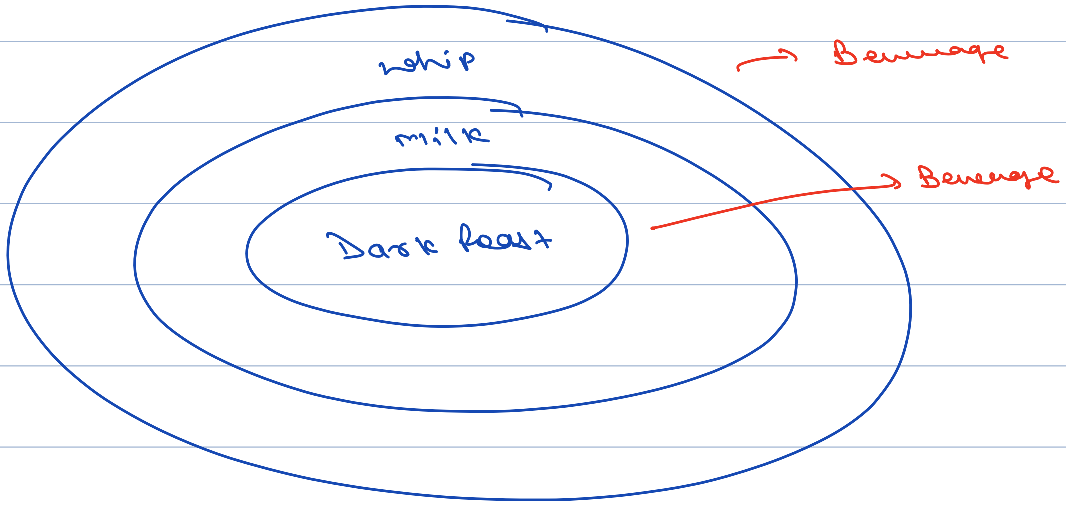
class explosion

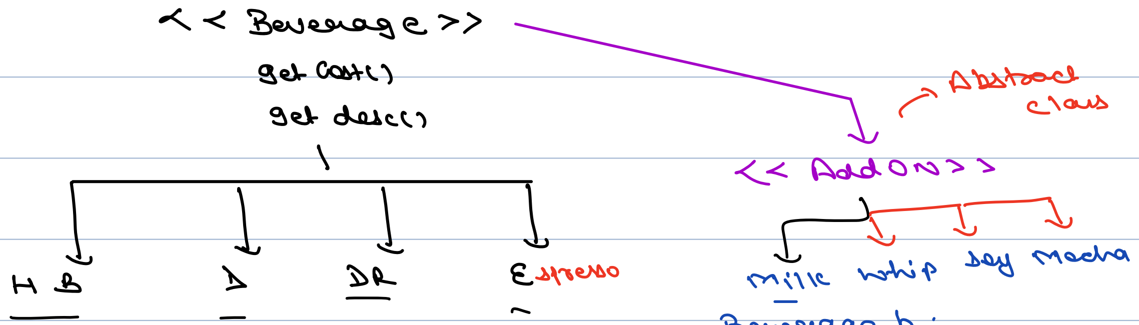
Too many classes
if we make subclass for
everything, → class Explosion.



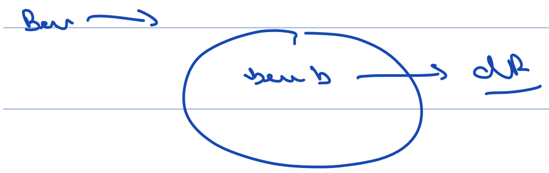
gift pr koi sticker lga diya ya flower lga diya.. its still a gift





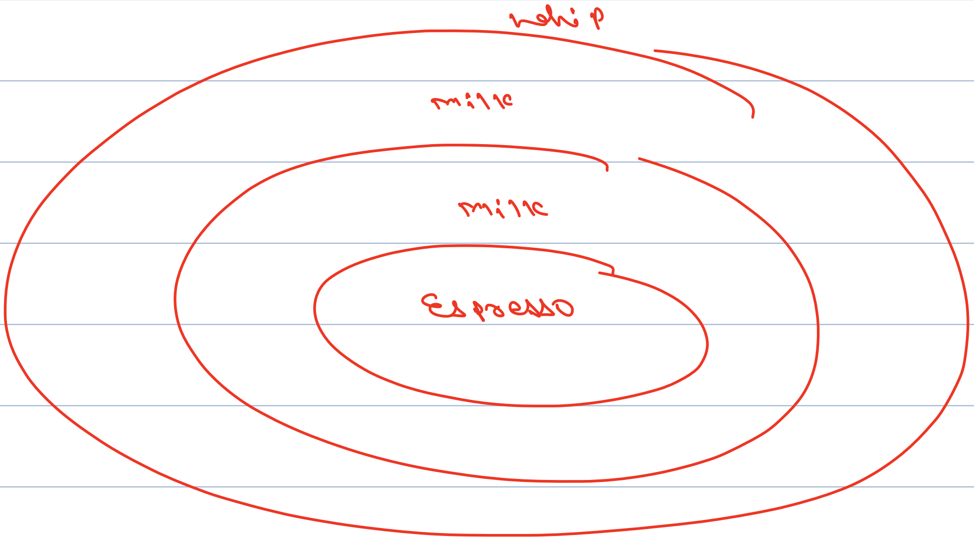


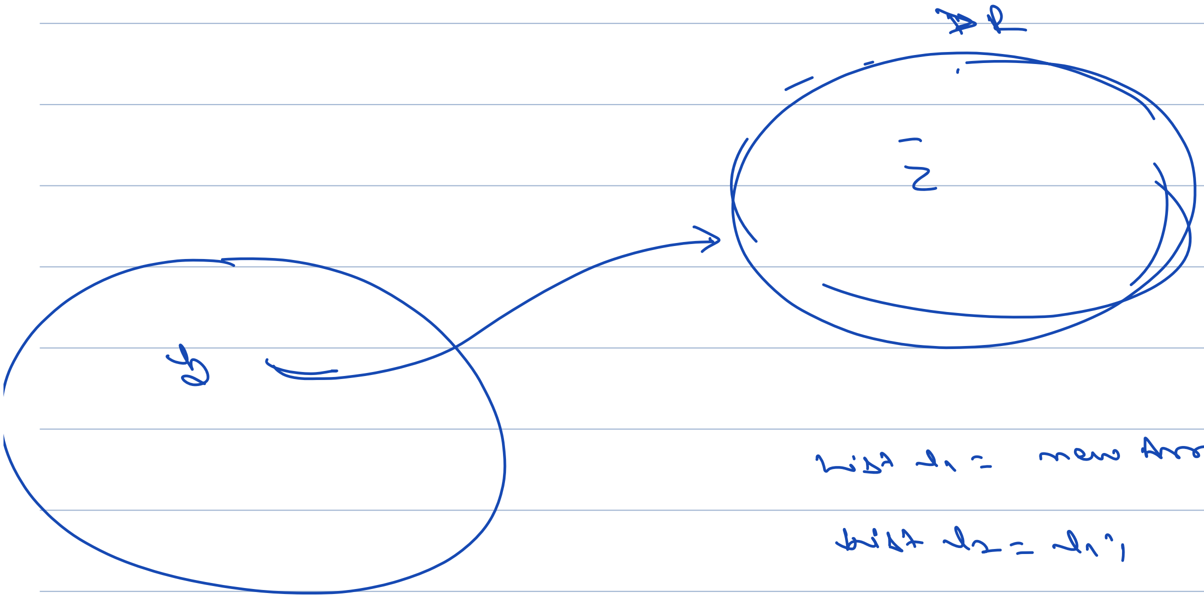
```
Beverage b ;
Milk(Beverage b) {
    this.b = b';
}
```



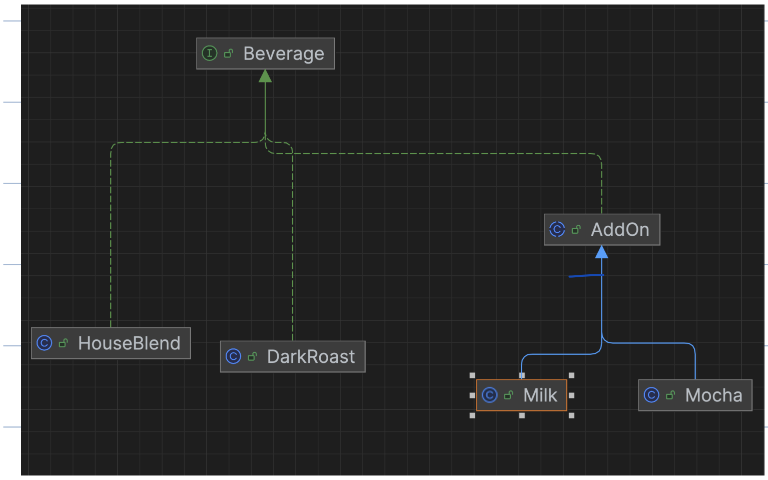
```
Beverage b = new Espresso();

          b = new Milk(b);        added milk

          b = new Milk(b);        again added milk

          b = new whip(b);        added whip
```

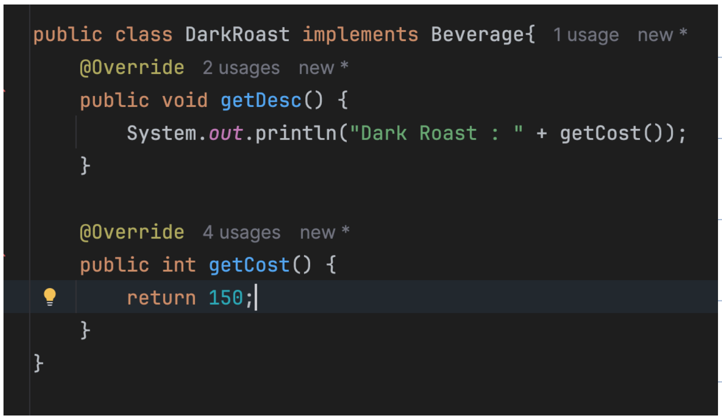
addons can be created independently.



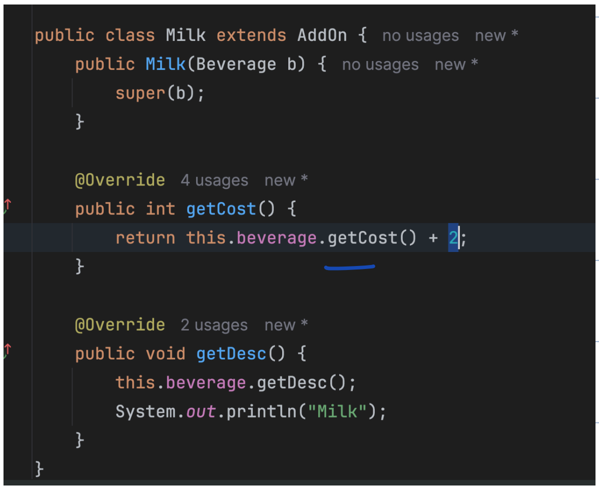


```
List l1 = new ArrayList();
List l2 = l1;
```





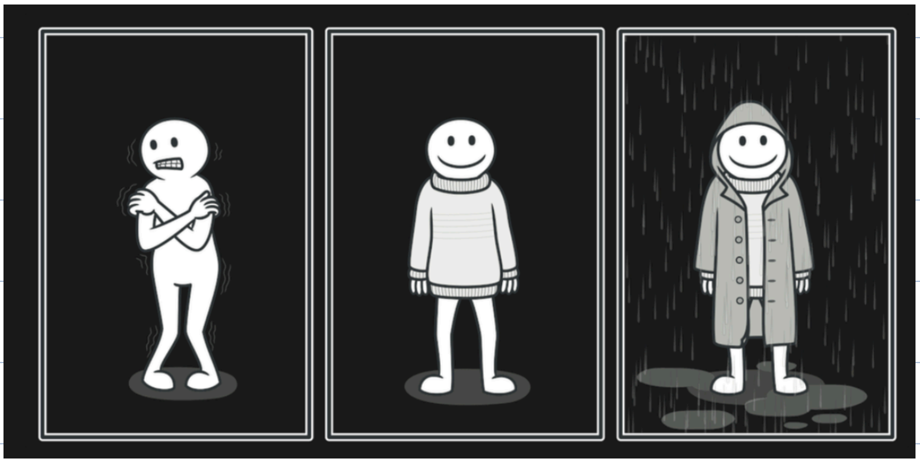
```java
public class DarkRoast implements Beverage{  1 usage   new *
    @Override  2 usages   new *
    public void getDesc() {
        System.out.println("Dark Roast : " + getCost());
    }

    @Override  4 usages   new *
    public int getCost() {
        return 150;
    }
}
```

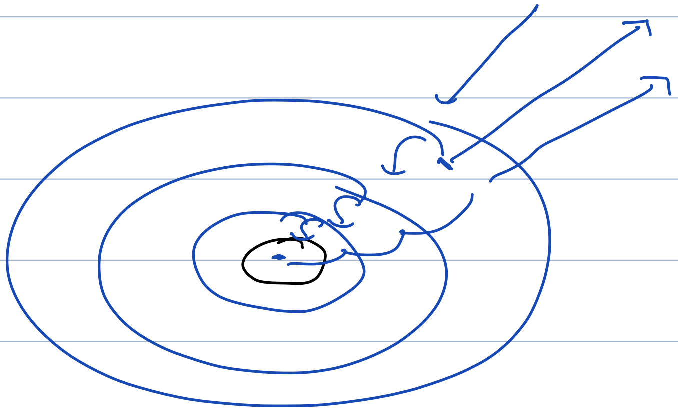


```java
public class Milk extends AddOn {  no usages   new *
    public Milk(Beverage b) {  no usages   new *
        super(b);
    }

    @Override  4 usages   new *
    public int getCost() {
        return this.beverage.getCost() + 2;
    }

    @Override  2 usages   new *
    public void getDesc() {
        this.beverage.getDesc();
        System.out.println("Milk");
    }
}
```

DB

```
class Billing {
        TotalTime (Beverage b){
        }
}
```



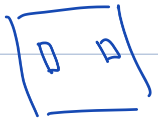


in Dot net windows form

window form created

a scroll bar is added

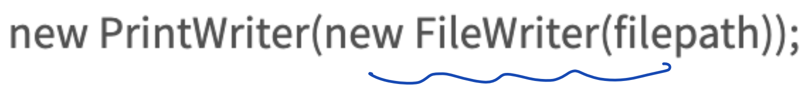
```
window w = new Window();
w = new ScrollBar(w);
```



```
Person p = new Person();
    p = new xyzClothes(p);
```

```
Button b = new Button();
    b = new BorderButton(b);
```

Java → I/p stream / output stream.



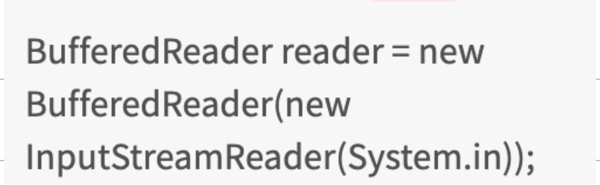
```
new PrintWriter(new FileWriter(filepath));
```

writer

writer

Logger s = new Logger();

s = new FileLogger();



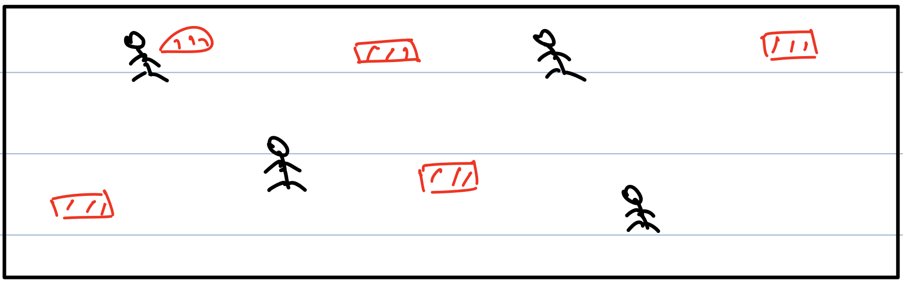
```
BufferedReader reader = new
BufferedReader(new
InputStreamReader(System.in));
```

Break:

10:13pm – 10:23pm

flyweight → Pubg

100 players → 2 guns → 150 bullets



1 player → 300 bullets

30000 ≃ 1,00,000

suppose total 1lakh bullets we can deploy in game

→ 7.76mm

→ 5.56mm

→ 0.44mm

→ x

→ y

Bullet

8B	- radius
8B	- damage
8B	- size
8B	- weight
8B	- range
24B	- dirn
24B	- curr coordn
24B	- targ coordn
1KB	- image

suppose each bullet is of 1.1 kb so 1lakh bullets will be of 100 mb of ram and we have other things also.. like guns mountains roads, cars.. so many things that will need lot of space in ram.. and that much ram we can not afford in mobile

1.1KB × 1,00,000

⇒ 100 MB.

Are all the bullets completely distinct ? → no
distinct

4 - 5 variant of Bullets

1 varient will have same type of properties

to save space dividing bullets into two different properties

Intrinsic

Remains same across multiple objects

Extrinsic

value changes with diff. object.

	Bullet =
8 B	- radius
8 B	- damage
8 B	- size
8 B	- weight
8 B	- range
1 KB	- image

only 4-5 bullet objects are required for all these fix properties(each type 1 object)

flying Bullet

	Bullet b :
8 B	
24 B	- dir^n
24 B	- curr coord^n
24 B	- targ coord^n

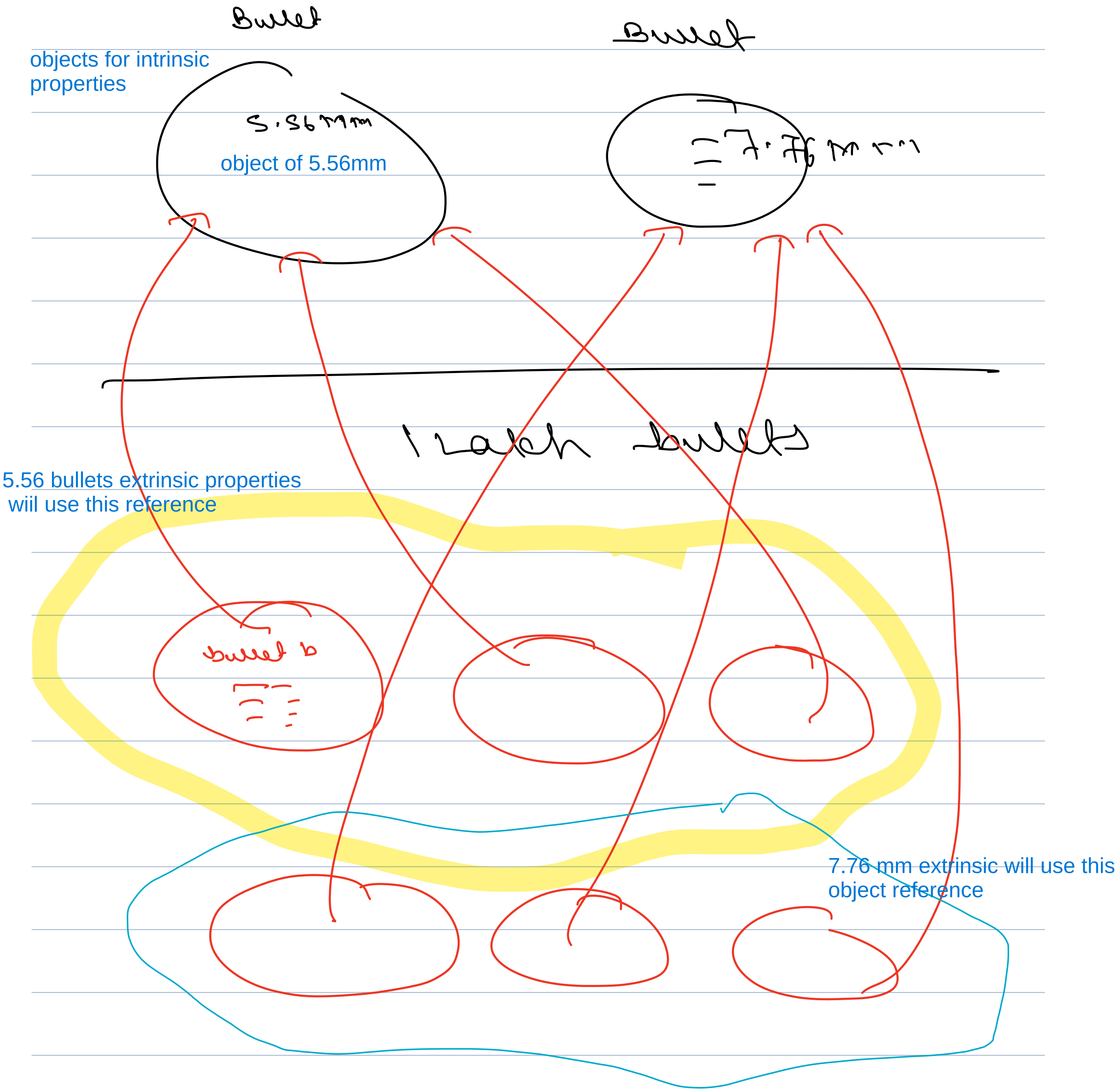
Bullet b

80 B × 1,00,000

=> 8 MB

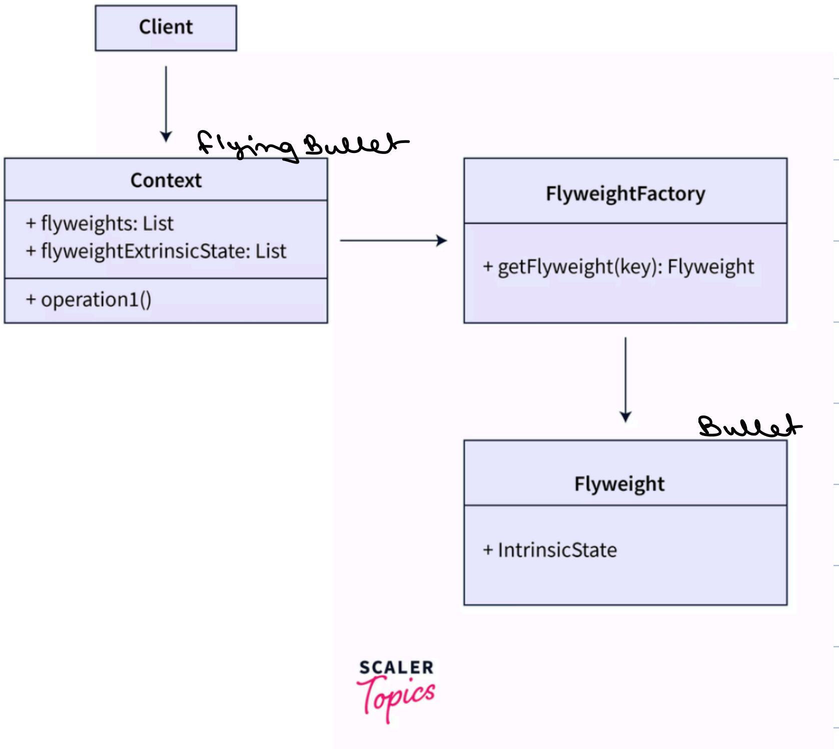
But for each bullet each object will be required for these.. because its different for each bullet. so 1lakh objects will be required.
but what we did is see below diag.

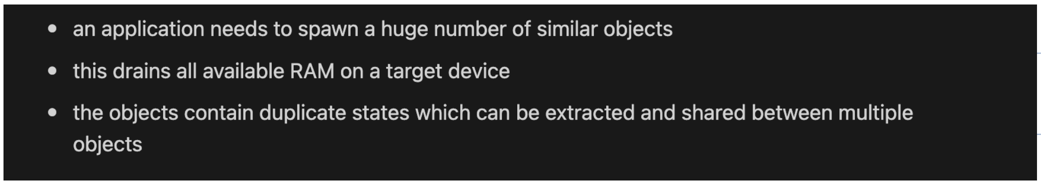
5.56 mm → 1 object of Bullet class.


Bullet
Bullet
objects for intrinsic properties
5.56mm
object of 5.56mm
7.76mm
1 Lakh bullets
5.56 bullets extrinsic properties wiil use this reference
bullet b
7.76 mm extrinsic will use this object reference

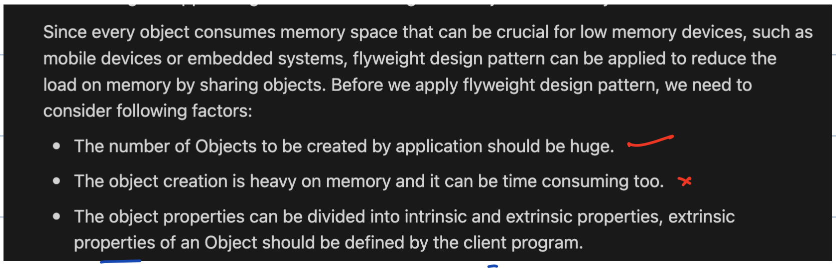
List l1 = new ArrayList();

List l2 = l1;


Client
flying Bullet
Context
+ flyweights: List
+ flyweightExtrinsicState: List
+ operation1()
FlyweightFactory
+ getFlyweight(key): Flyweight
Bullet
Flyweight
+ IntrinsicState
SCALER Topics



- an application needs to spawn a huge number of similar objects
- this drains all available RAM on a target device
- the objects contain duplicate states which can be extracted and shared between multiple objects



Since every object consumes memory space that can be crucial for low memory devices, such as mobile devices or embedded systems, flyweight design pattern can be applied to reduce the load on memory by sharing objects. Before we apply flyweight design pattern, we need to consider following factors:

- The number of Objects to be created by application should be huge.
- The object creation is heavy on memory and it can be time consuming too.
- The object properties can be divided into intrinsic and extrinsic properties, extrinsic properties of an Object should be defined by the client program.