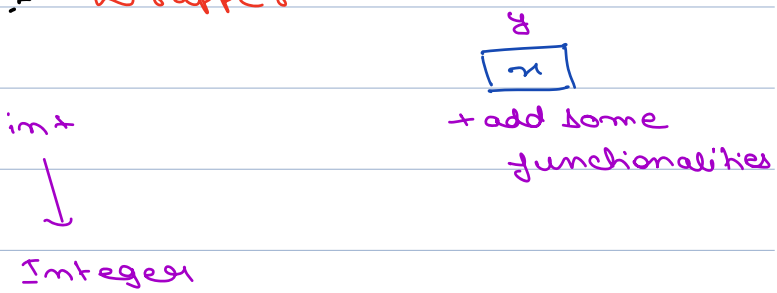
Agenda :-

—> Decorator

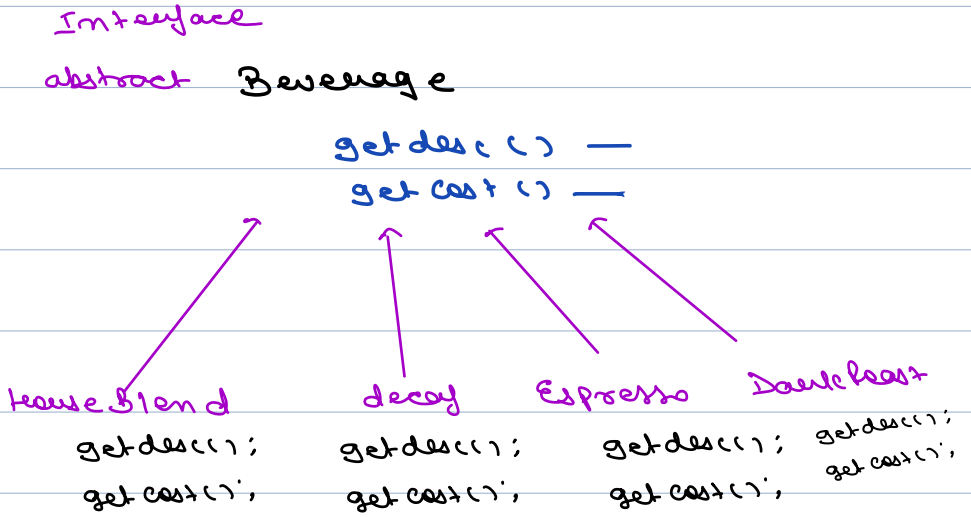
—> flyweight.

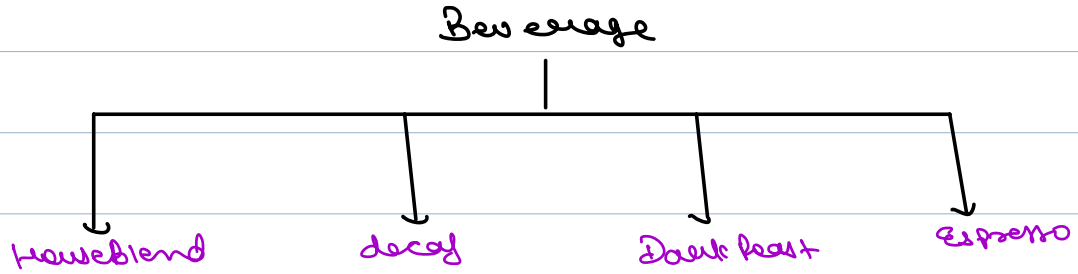
Decorator :- Wrapper



Starbucks

- Build the beverage
- description of beverage
- cost.





getDesc();
getCost();

new Decaf();
new Espresso();

DarkRoast + Milk — Milk → getCost() → getDesc();
DarkRoast + Milk + whip — whip
DarkRoast + 2Milk — soy
mocha

getDesc();
getCost();

getDesc();
getCost();

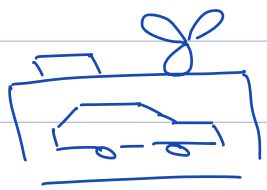
Class explosion

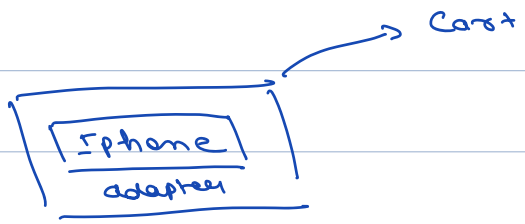
Too many classes

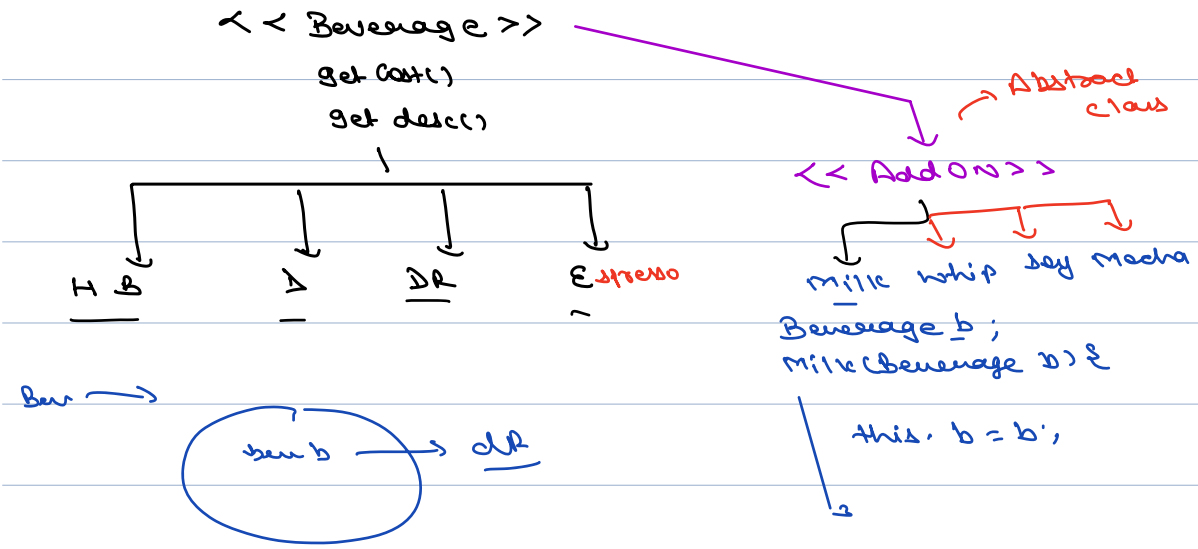
if we make subclass for

everything, → class Explosion.

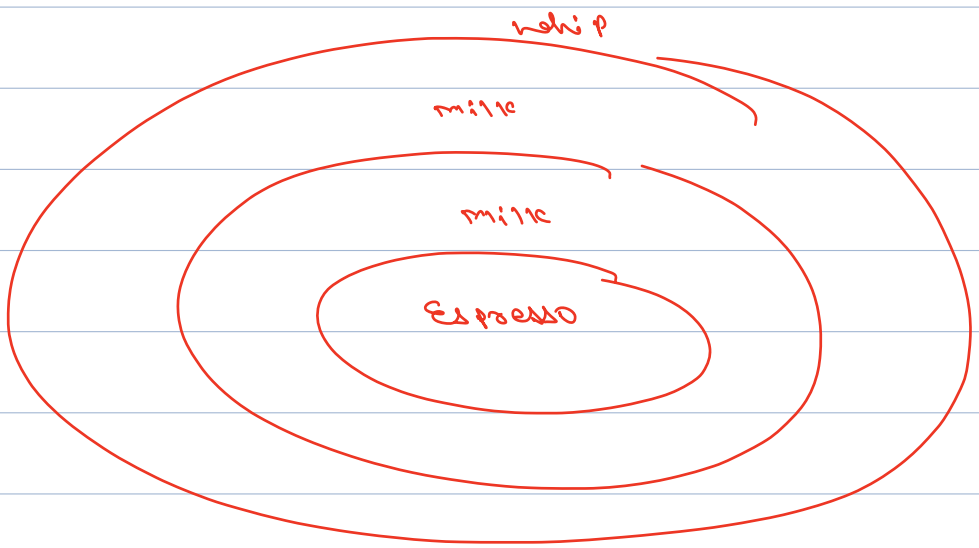
gift →

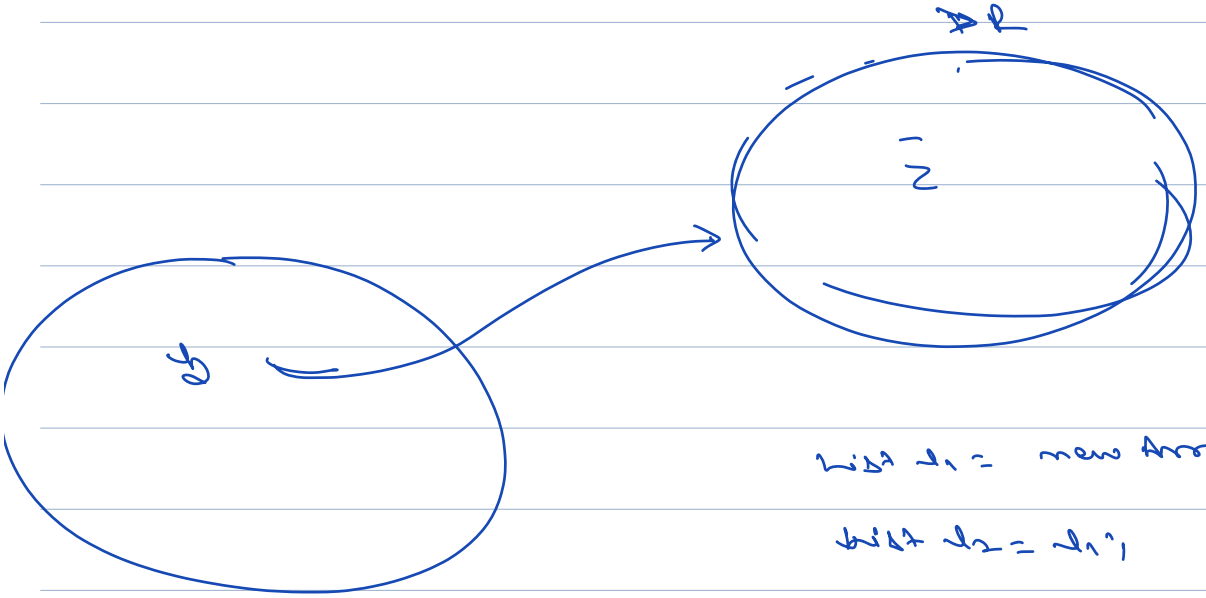




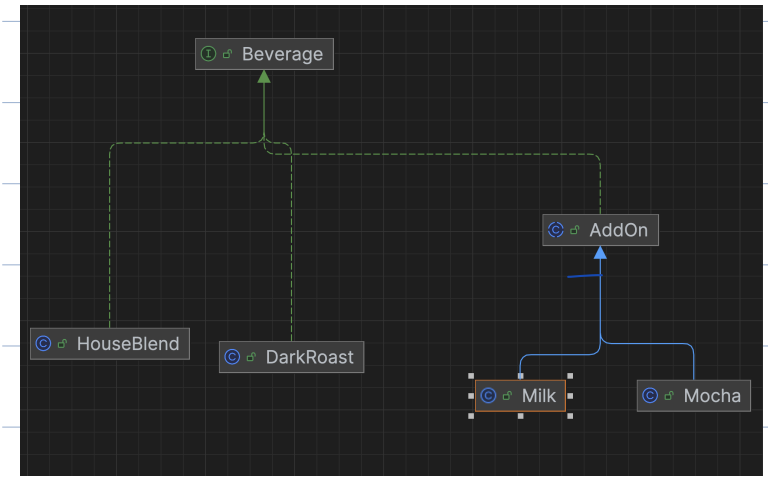



```
Beverage b = new Espresso();
b = new Milk(b);
b = new Milk(b);
b = new Whip(b);
```





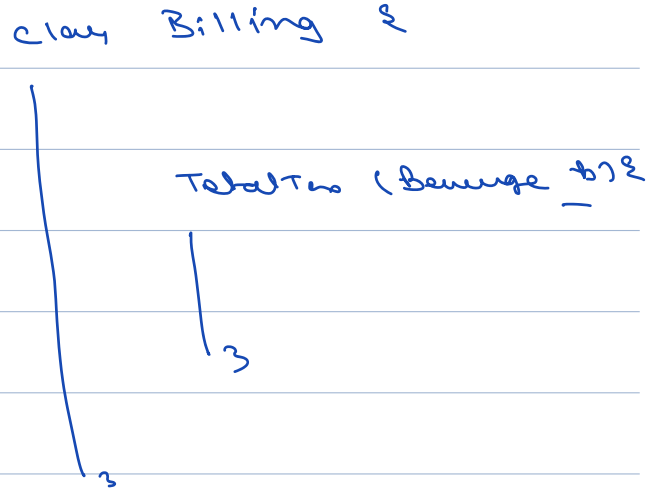
```
int[] a1 = new int[3];
int[] a2 = a1;
```

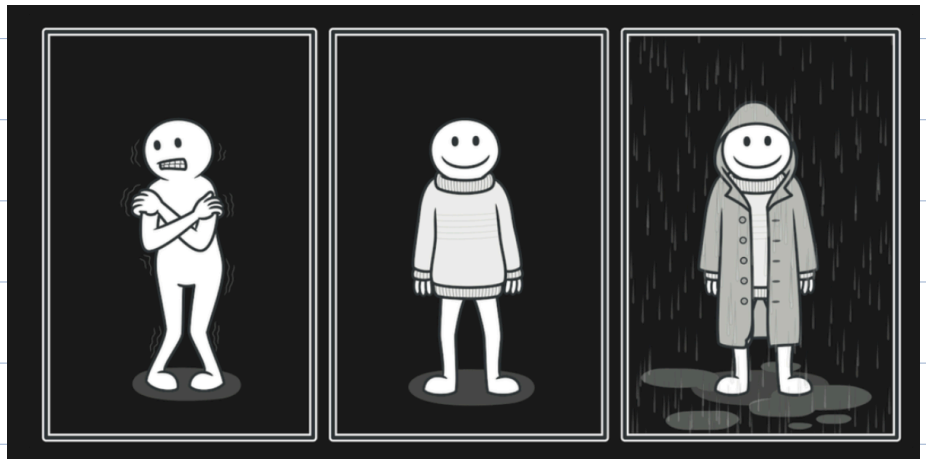


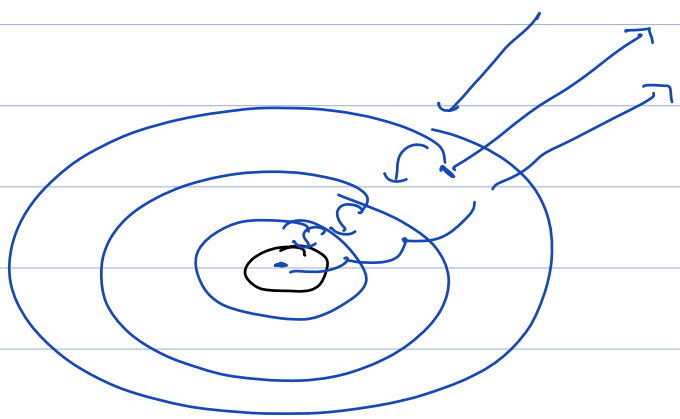

```java
public class DarkRoast implements Beverage{  1 usage  new *
    @Override  2 usages  new *
    public void getDesc() {
        System.out.println("Dark Roast : " + getCost());
    }

    @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

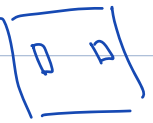






Dot net windows form

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



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

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

Java → I/P stream / output stream.

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

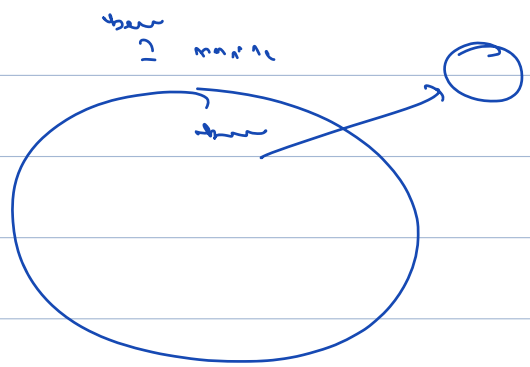
writer

writer

Logger l = new Logger();

l = new FileLogger();

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

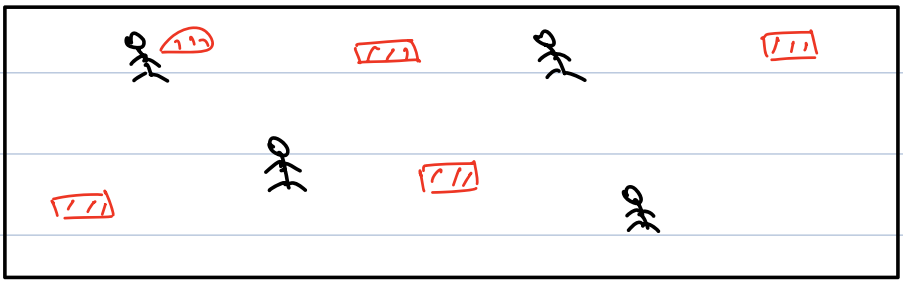


Break

10:13pm – 10:23pm

flyweight → pubg

100 players → 2 guns → 150 bullets



1 player → 300 bullets

30000 ≃ 1,00,000

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

→ 7.76 mm

→ 5.56 mm

→ 0.44 mm

→ x

→ y

1 KB + 120 bytes

1024 bytes

1.1 KB x 1,00,000

⇒ 100 MB.

i, j, k

Are all the bullets completely distinct ? → no

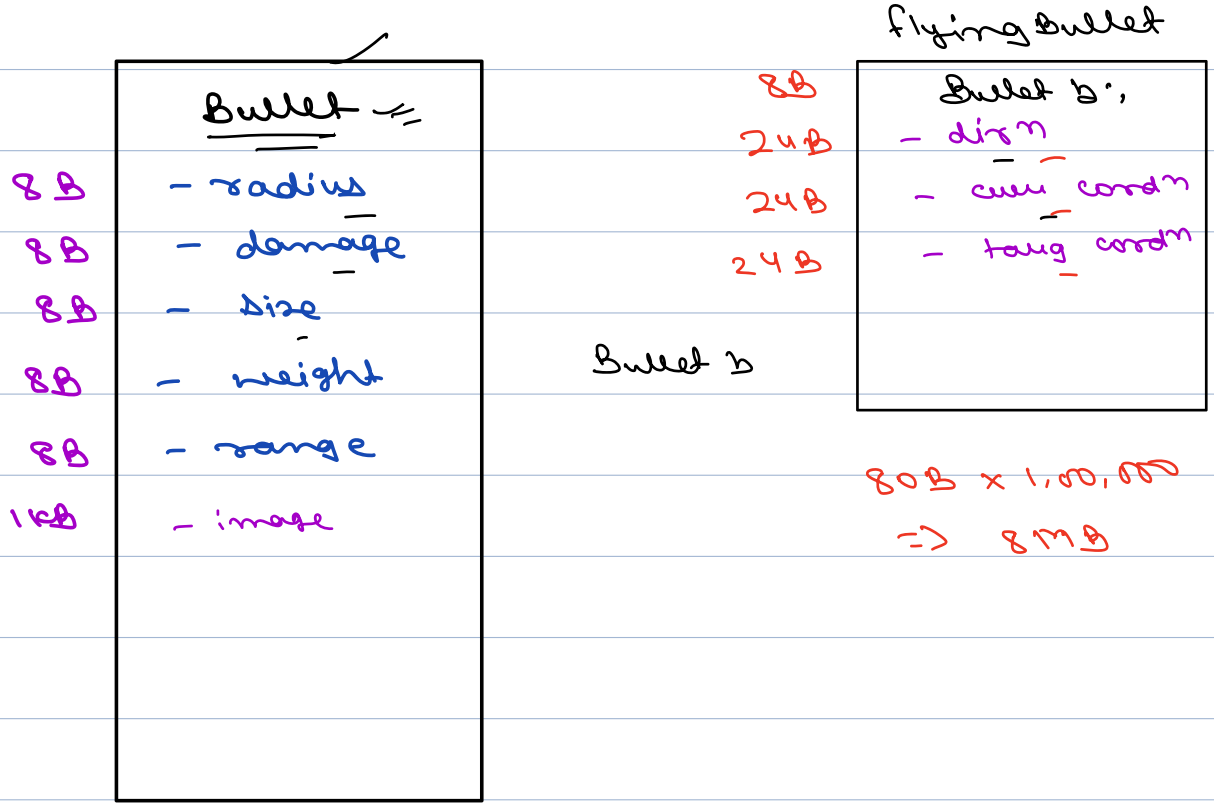
- 5 variant of bullets

Intrinsic

remains same across multiple objects

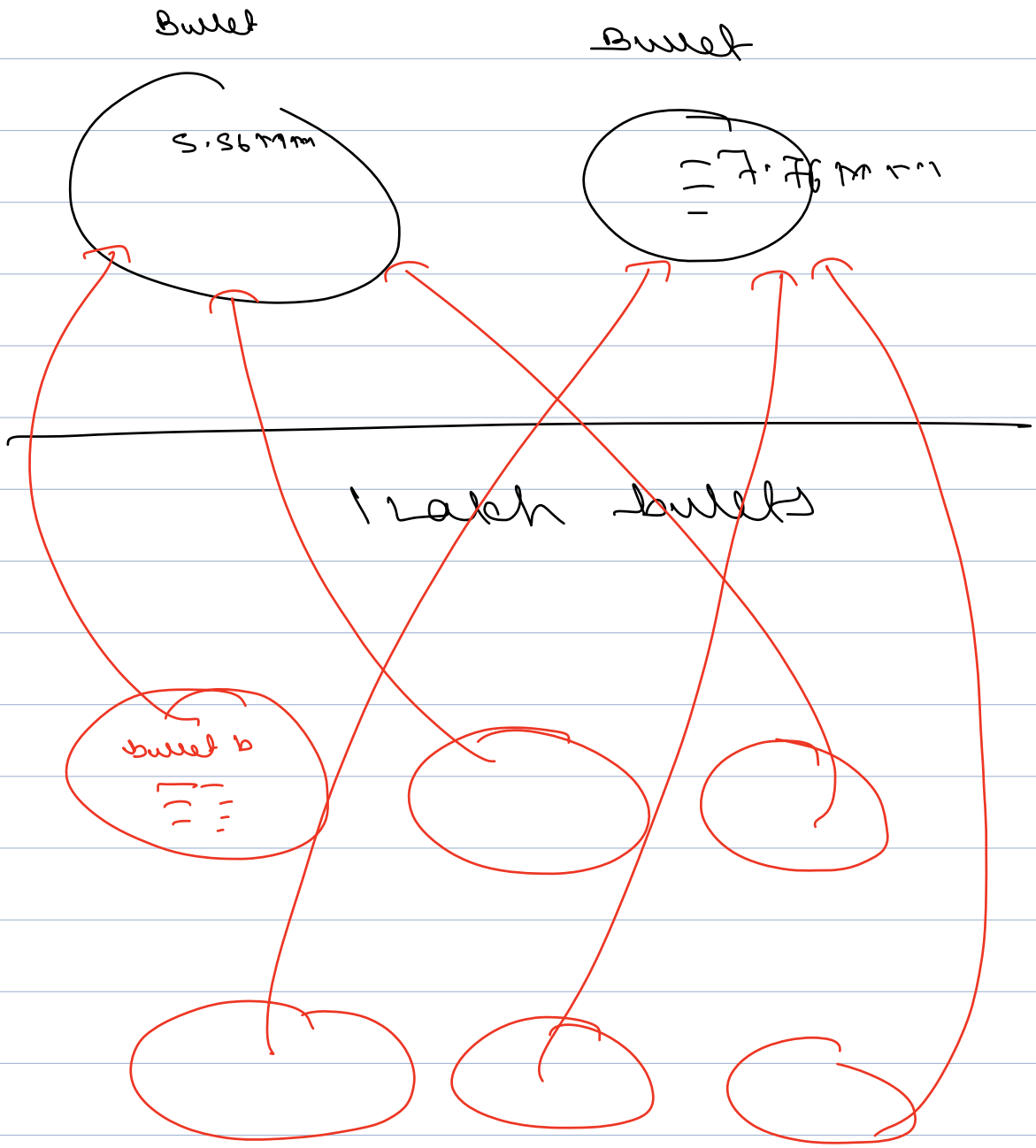
Extrinsic

value change with diff. object.



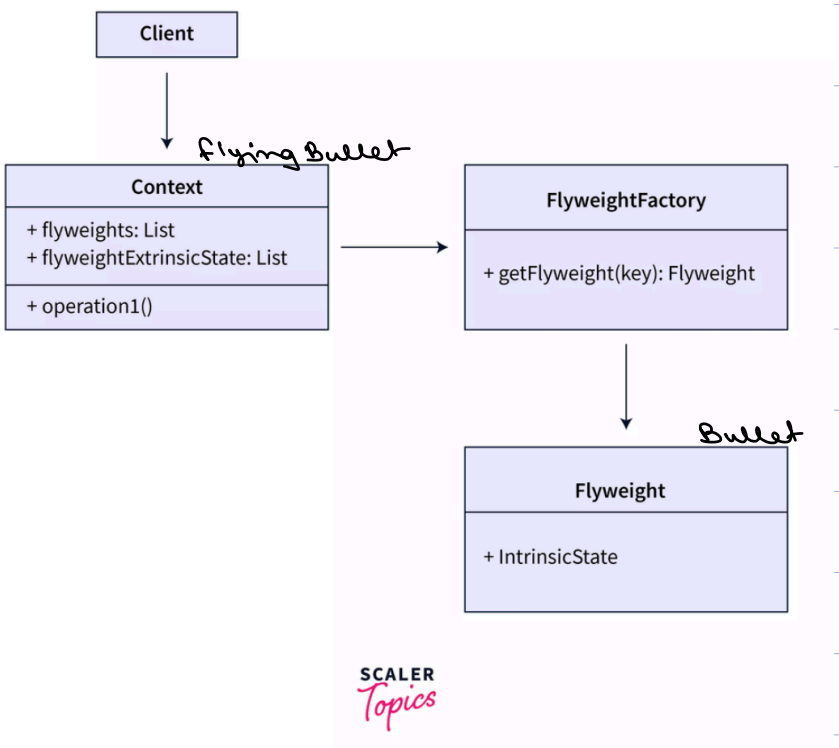
80B x 1,00,000
=> 8MB

5.56 mm → 1 object of Bullet class.



List l1 = new ArrayList<>();

List l2 = l1;


Client
FlyingBullet
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.