what is good software design

cost of changing is minimum.

software are not written
they are always rewritten

writing 1 line get 1 reward
deleting 1 line should get 2 rewards

code review and design review

simple or avoid complexity

familiar

tribal knowledge

Java concurrency

Python concurrency

C++ concurrency

class

cohesive

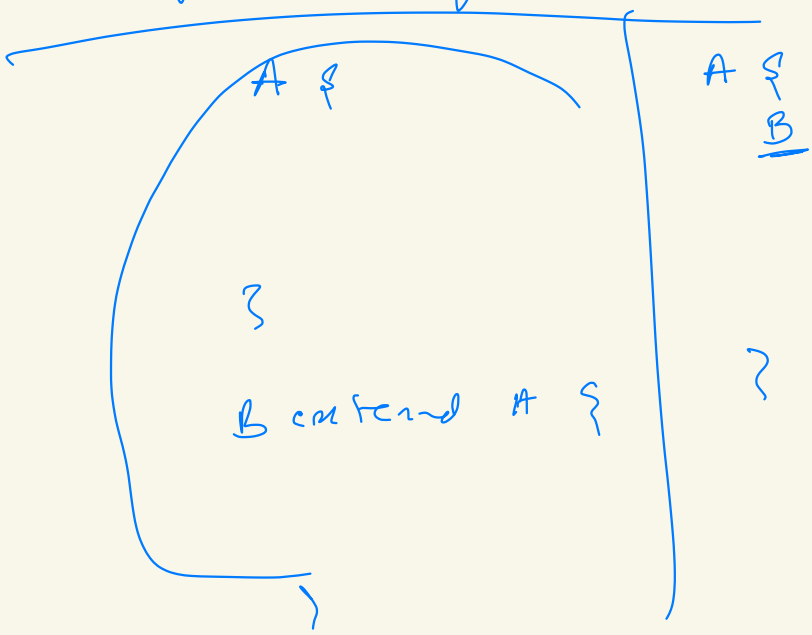
Principle of cohesion

A code should be narrow, focused and does only one thing

coupling

$M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4$

Inheritance is the worst form of coupling



high cohesion and low coupling

Why do we write bad code ?

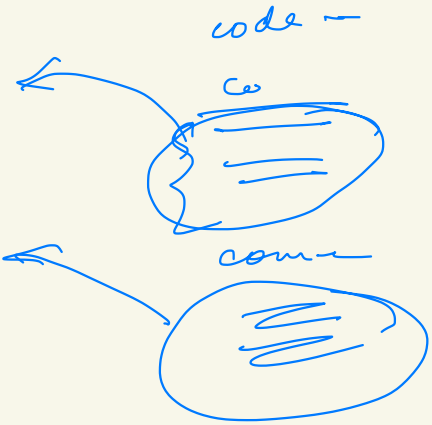
We wants move fast

Long fun method

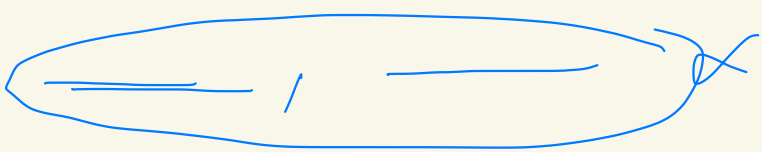
```
fun() {
    ...
    ...
    ...
}
```

Why long methods are bad

1. hard to reuse — Dry
2. lack cohesion and high coupl
3. comments
4. optimize
5. hard to debug



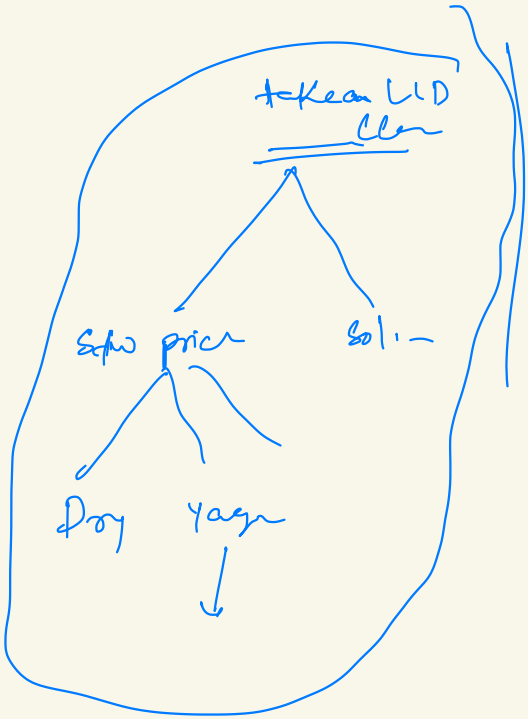
How small function should



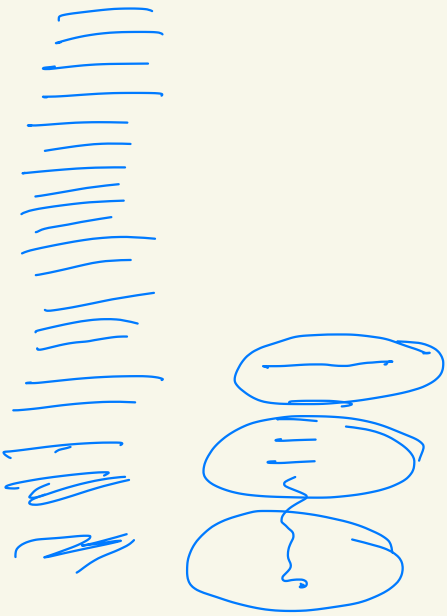
10
15
20

5 1

SLAP — single level of abstraction



attend Birthday par



function

SOLID Principles

S → Single resp priciples

O → O C P → Open closed principle

L → Liskov's Sub principle LSP

I → Interface segregation pr ISP

D → Dependency Inversion pr DIP

Single Responsibility principle

```
class OrderService {

    createOrder(Order order) {

        validateOrder(order);

        double total = calculateTotalPrice(Order);

        order.setTotal(total);

        try {
            db.save(Order);
        } catch (SQLExc
        ;
        emailService.sendOrderConf(ord
    }
```

class OrderValidator {
validate

class OrderRepo {

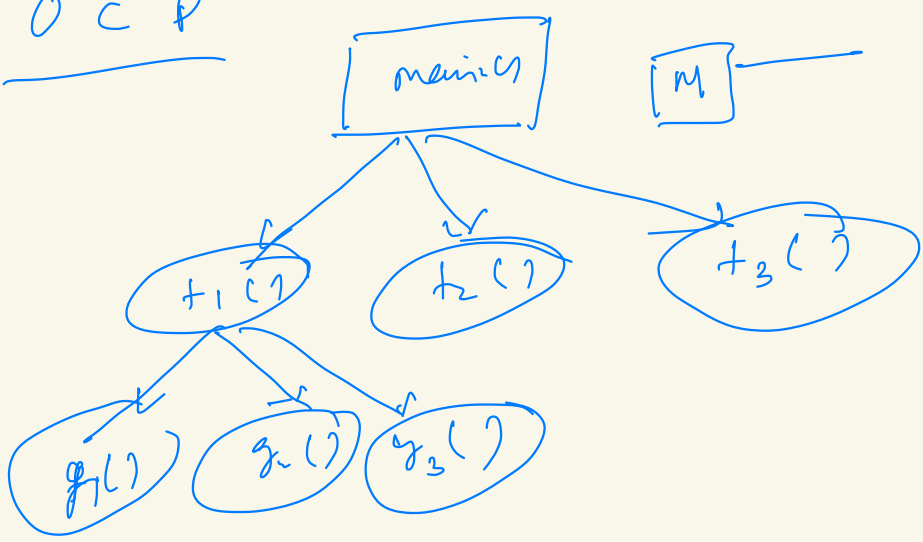
}

class EmailSe

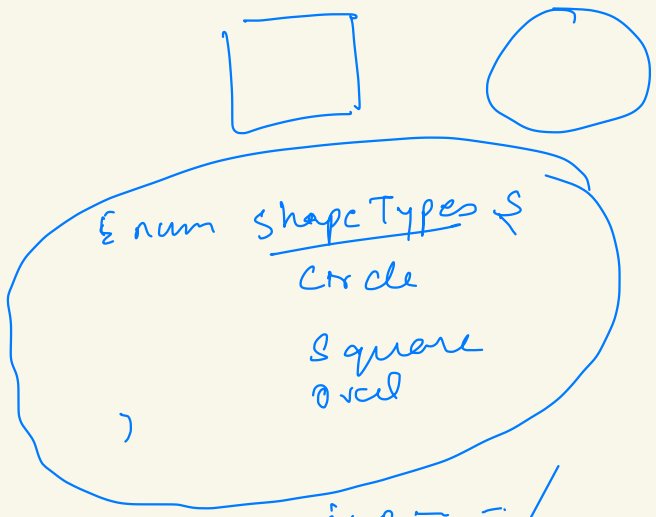
order
{

}

OCP



shapes.com



enum ShapeTypes {
Circle
Square
Oval
}

imp — ;

class Square {
sh sq
draw Square() {
}

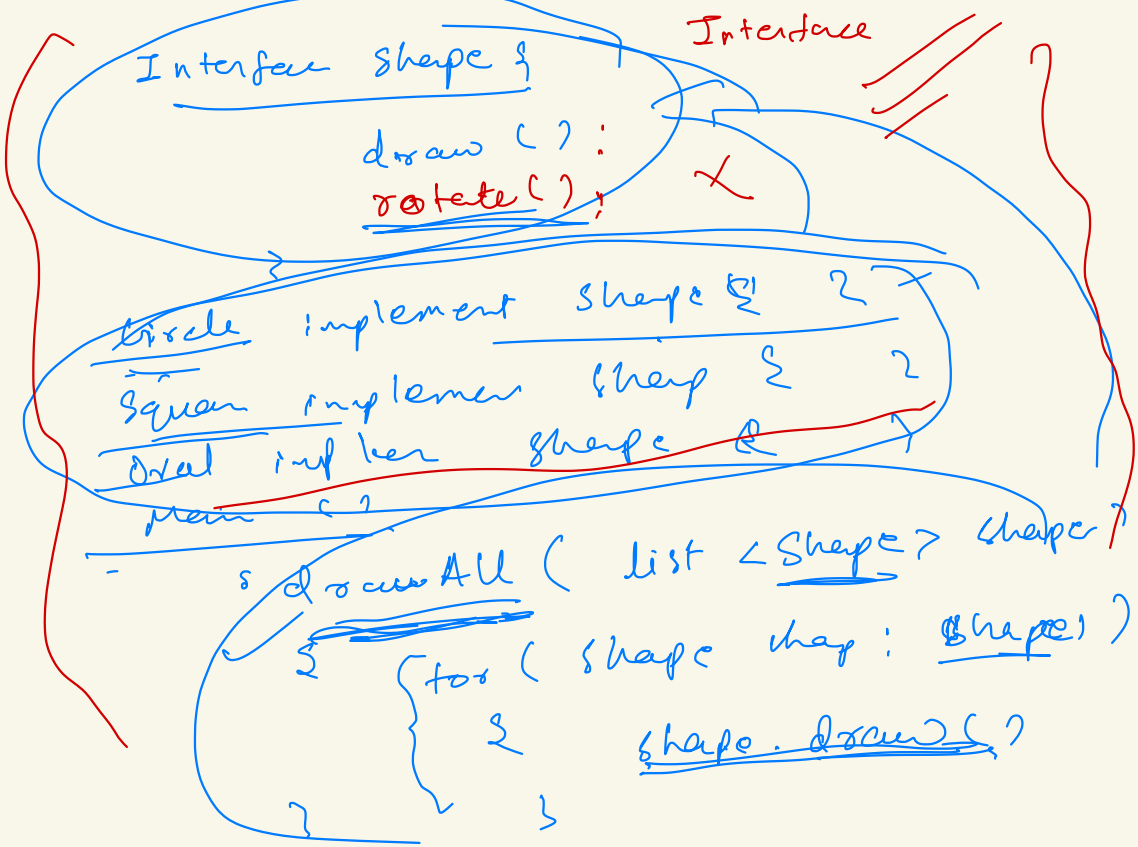
→ draw All shapes

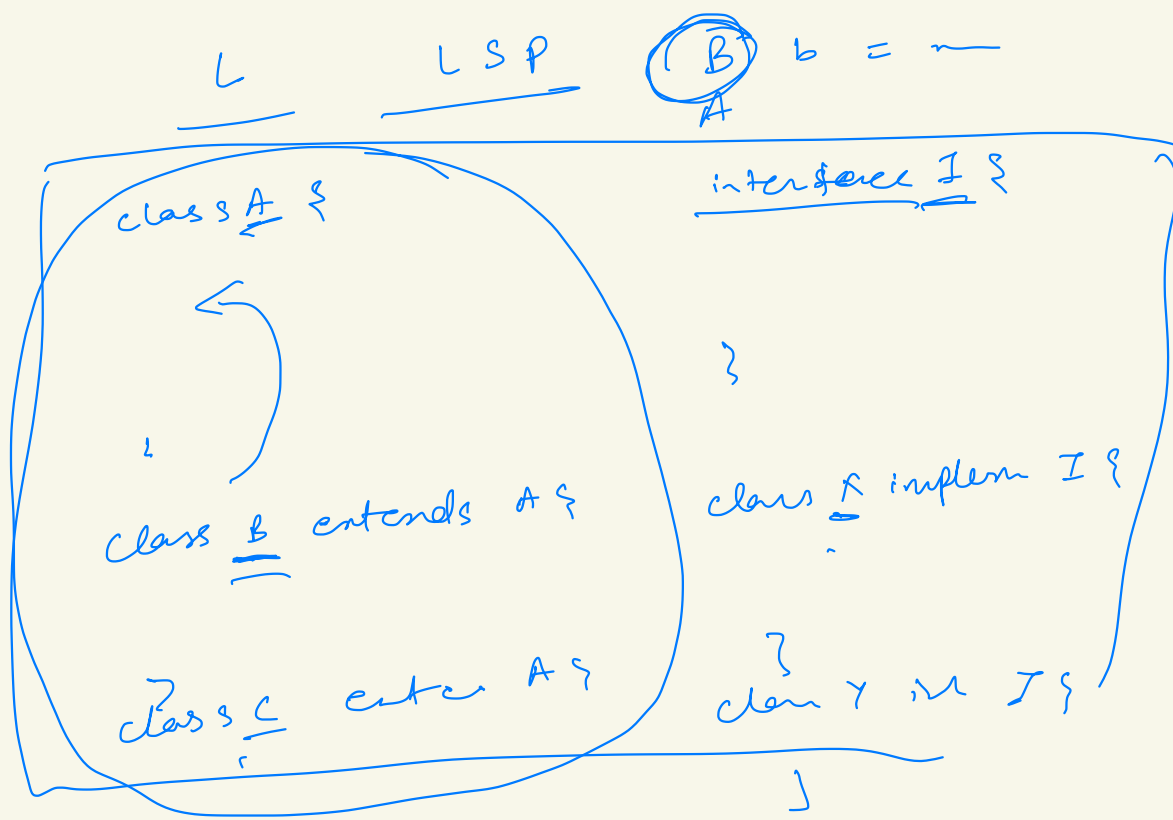
import shapes

class Circle {
shapeTypes st =
draw Circle() {
}
}

import circle
Main { import square;

drawAllShapes (List<Object> shapes)
{
for (Object shape : shapes)
{
if (shape instanceof Circle)
{
((Circle) shape.draw();
} else if
}
}


Interface
Interface Shape {
draw();
rotate();
}
Circle implement Shape {
}
Square implement Shape {
}
Oval implement Shape {
}
drawAll(List<Shape> shape)
{
for(Shape shap : Shape)
{
shape.draw();
}
}


L
LSP
B
A
b = n
class A {
interface I {
}
class B extends A {
class X implem I {
class C extends A {
class Y im I {
}

class RotatableShape{

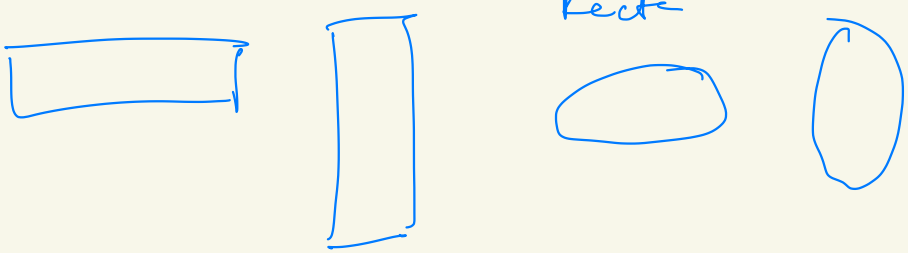
rotate()

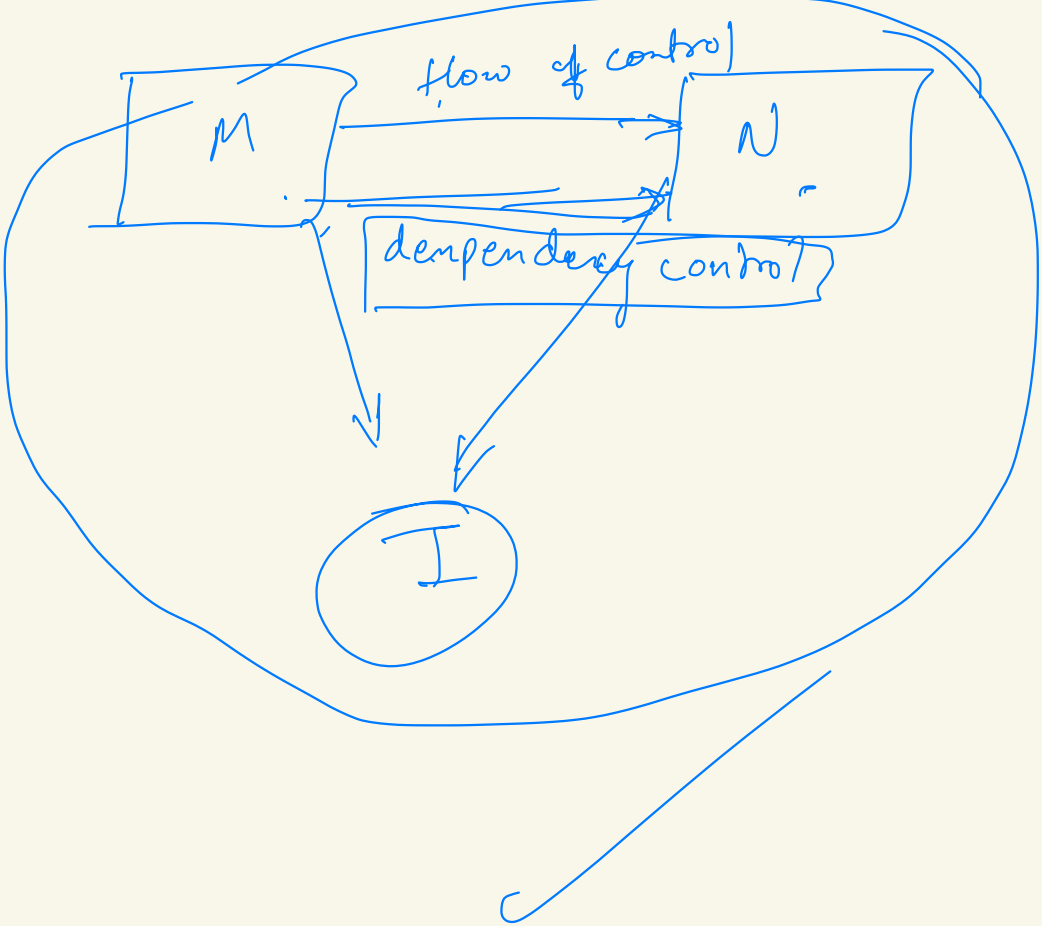
}

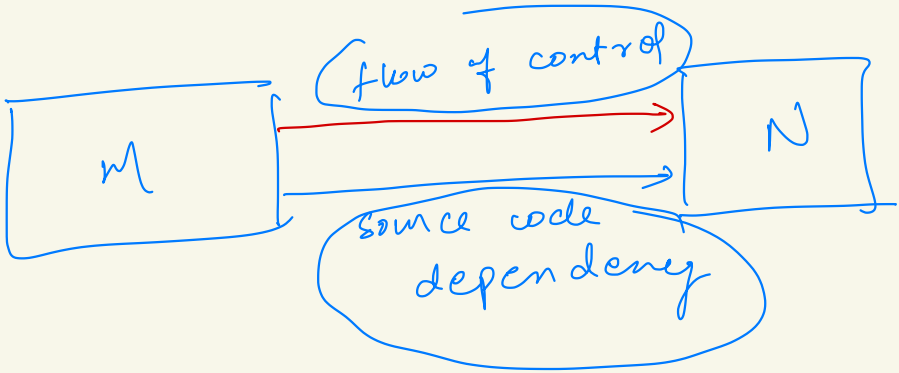
Square implements Shape,

Circle implements Shape

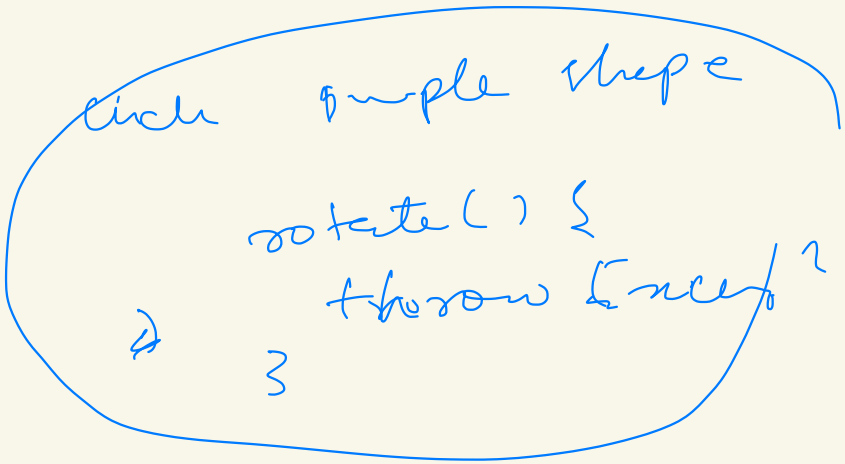
Oval implements Rotatable, Shape, Resizable

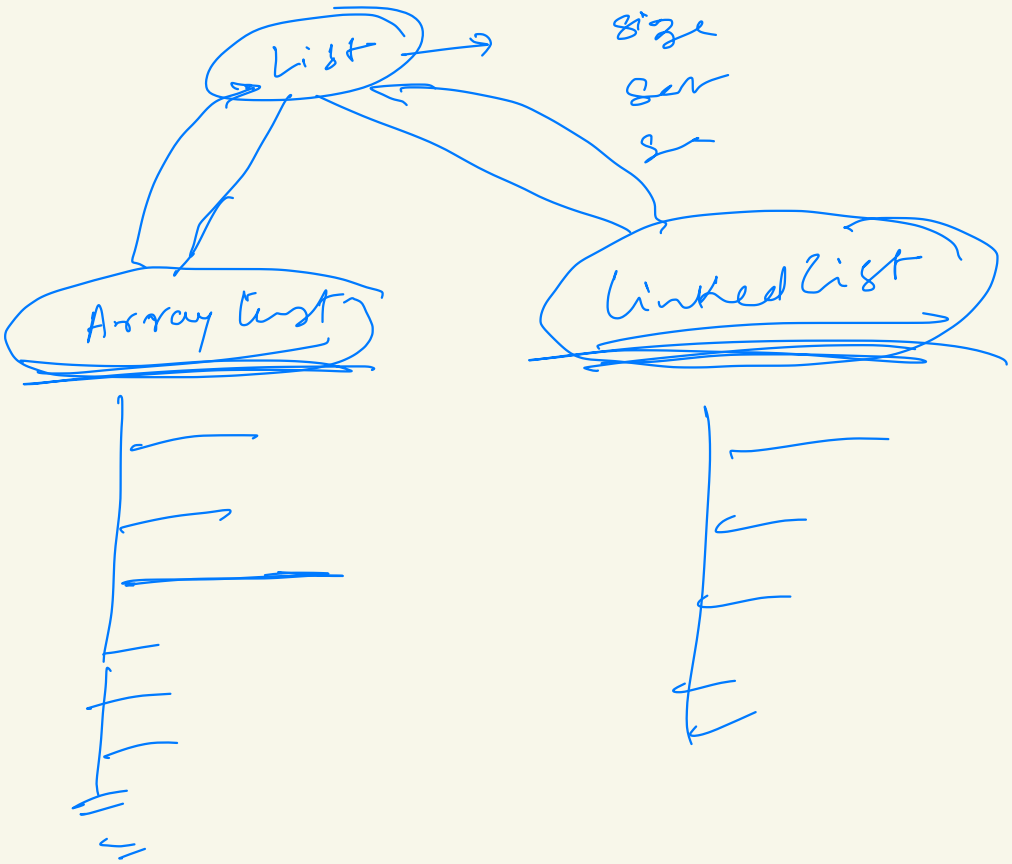



flow of control
M
N
dependency control
I


flow of control
M
N
source code
dependency

```
interface Shape {
    draw()
    rotate()
}
```




List
size
ser
g
ArrayList
LinkedList