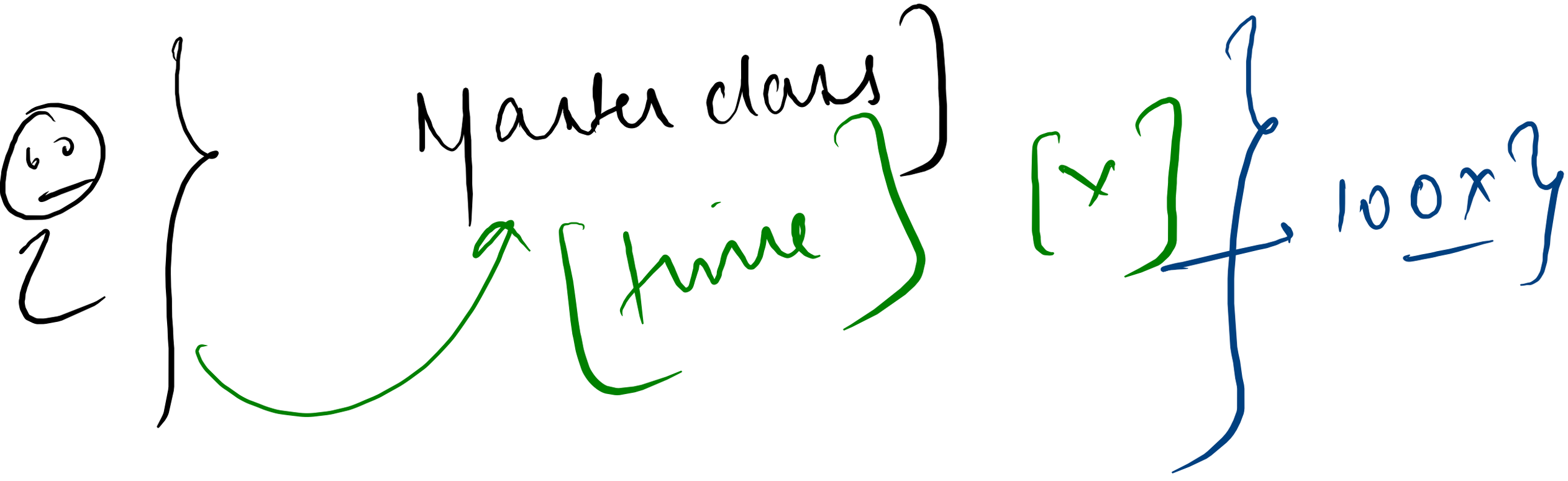

Many days
time
x
100x

Java | Spring framework

Aware of Java

topic

what

why ?

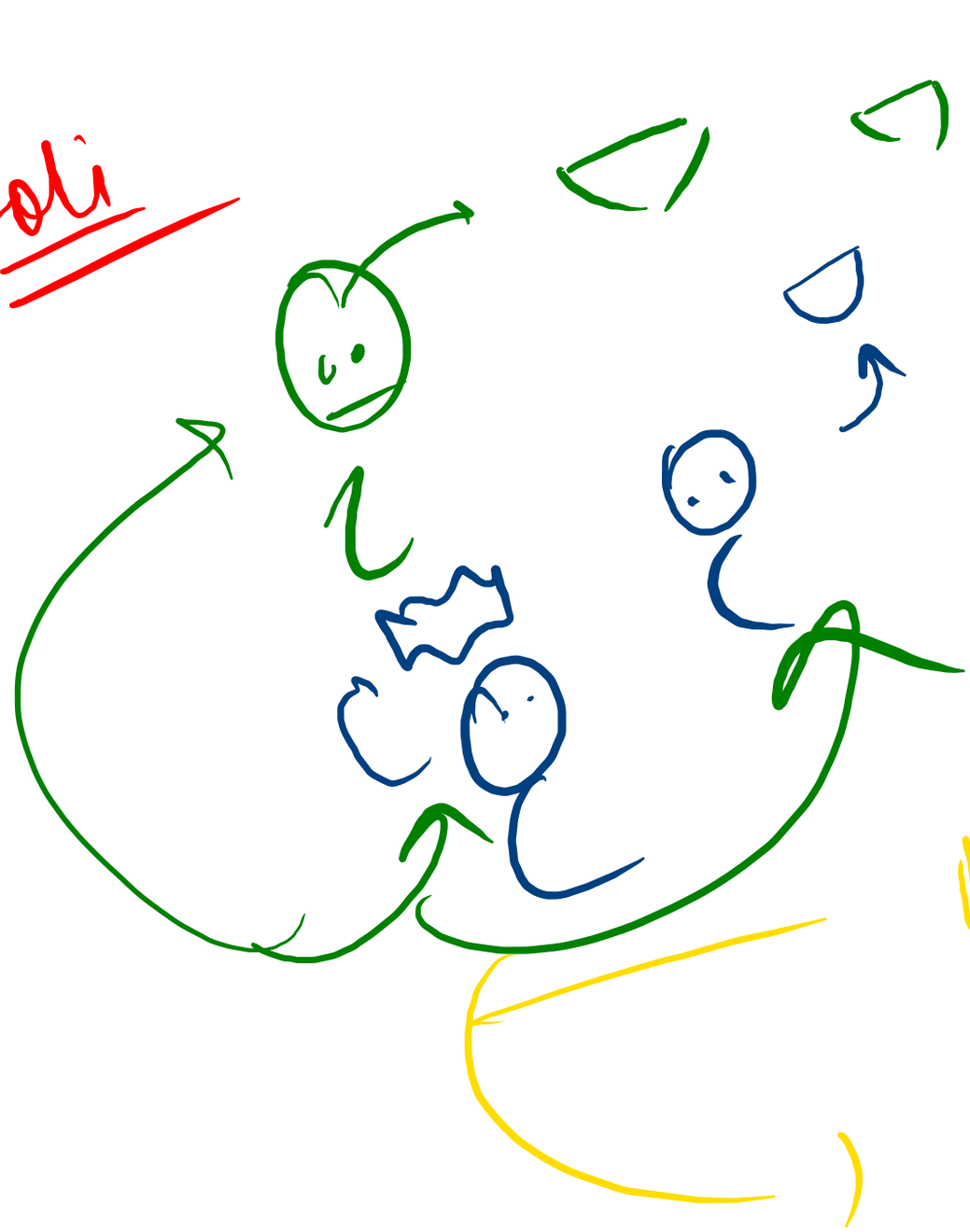
How

Spring

What is Spring ?

Framework

Holi



Framework

Market

Agriculture

Company

Project

→ Minimum

→ collaborate

Quality limit

// → boilerplate code

company

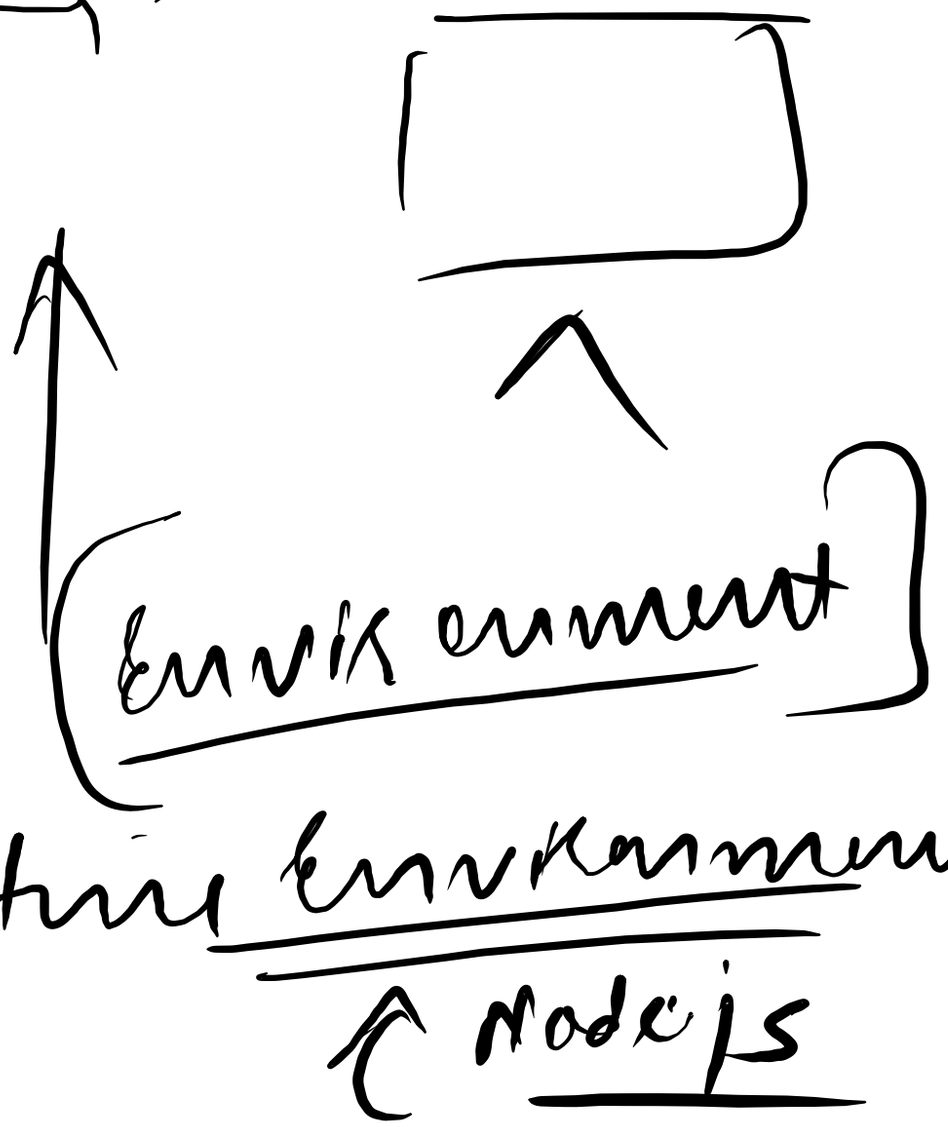
Java — Spring

JS — Express

Python — Django

React

(JS) → Programming Language



Java } → JRE

Nodejs {JS }

Java

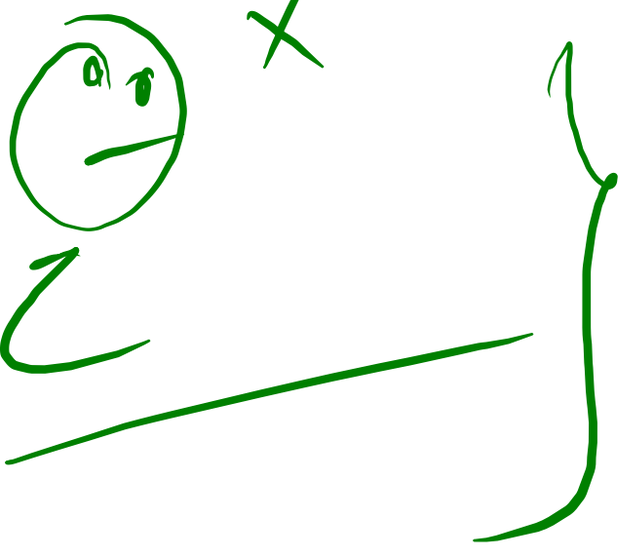
Spring → framework

company

Fb

Ci

Why Spring

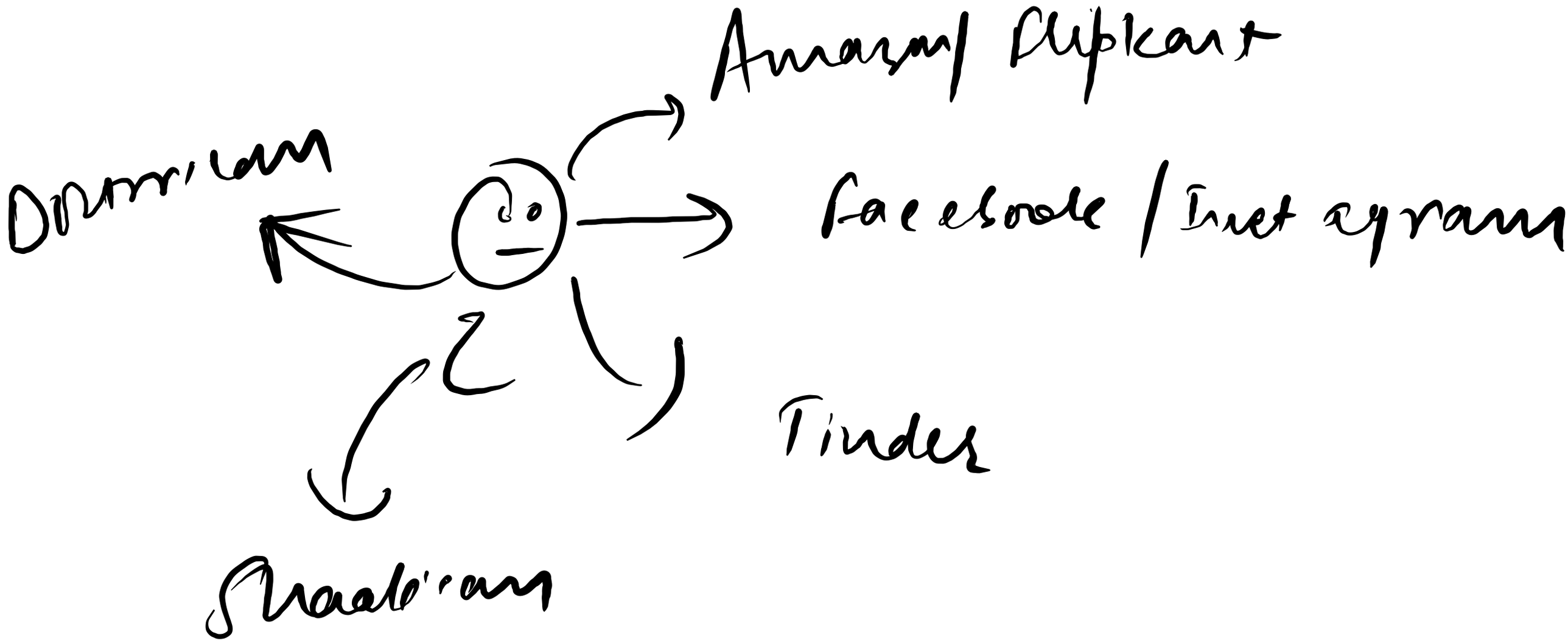


Loose

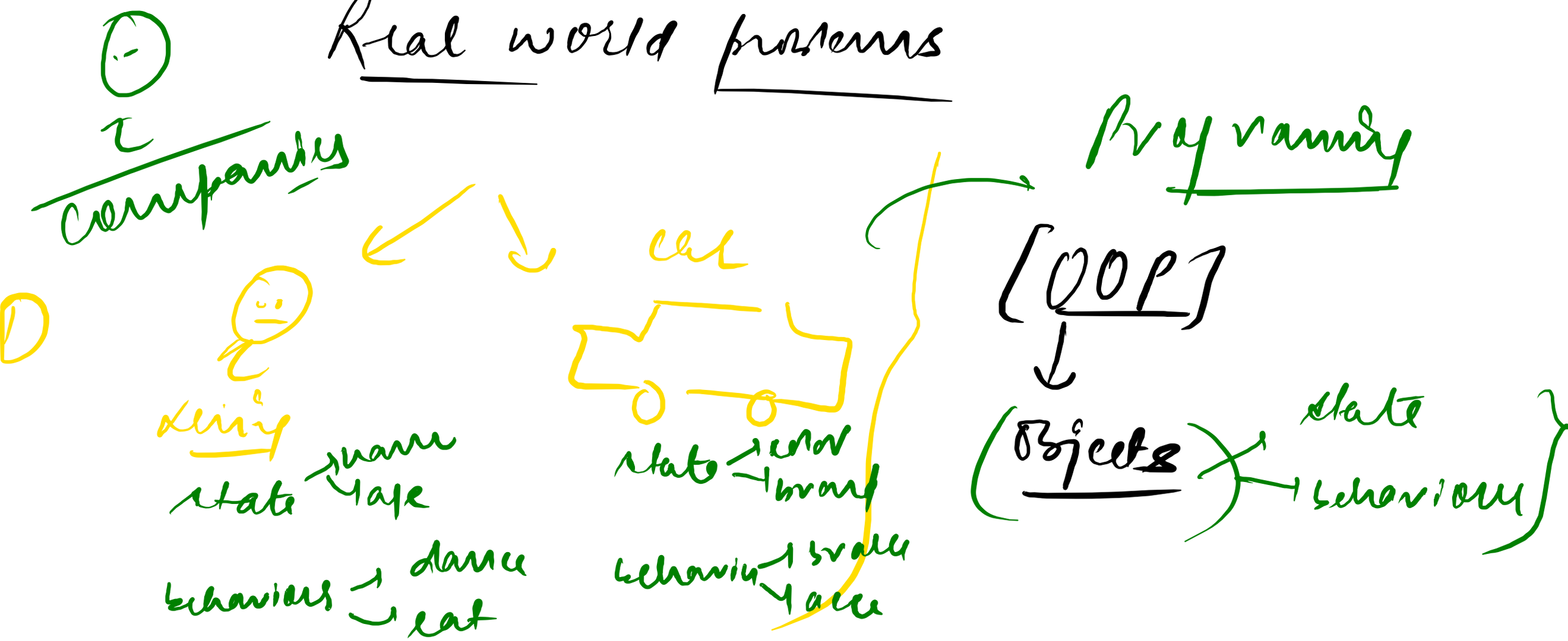
Low / coupling

A to D } (, Loose coupling }

} Relationship but flexibility]


Amazon/ Flipkart
Facebook / Instagram
Tinder

Real world problems



(9) Relationships

Programming

App

① Objects

② Relationship

Relationships

→ good
→ bad

} Depends !!!

↳ important !

↳ yes !!!

Make friend

Relationship

changes

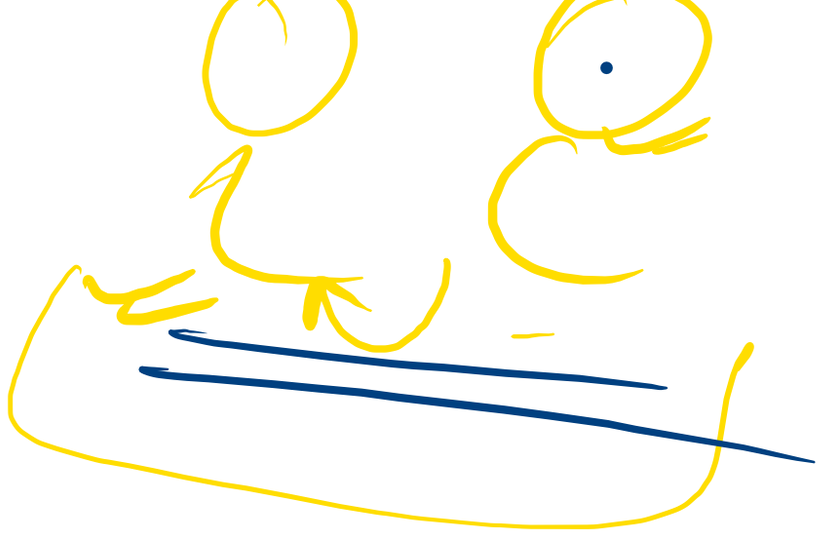
Future

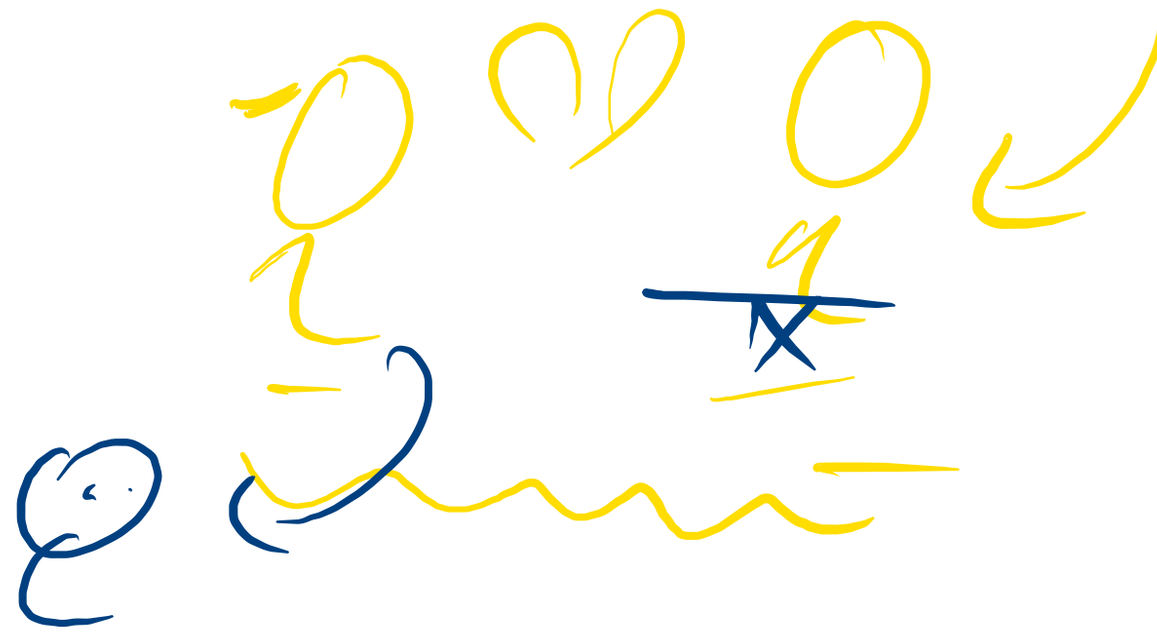
changes are easy

Tightly coupled

Loosely coupled

(Married





Application to send Message

↓

(App.java) → (MessageService)

↳ send message()


```java
package com.vishwa;


import com.vishwa.services.TextMessageService;


public class Main {
    public static void main(String[] args) {


        TextMessageService textMessageService = new
TextMessageService();
        textMessageService.sendMessage("Hello Students");
    }
}
```

Tightly coupling

```java
package com.vishwa.services;


public class TextMessageService {


    public void sendMessage(String message){
        System.out.println("Sending message : " + message);
    }


}
```

Open close principle → SOLID

→ closed for modification

Open for extension

→ [App]

Easily coupled

Tightly coupled code

→ Loosely coupled code

↓ Factory Design Pattern

Creational design pattern

```java
package com.vishwa.looselyCoupled;

import com.vishwa.looselyCoupled.services.MessageService;

public class App {

    public static void main(String[] args) {

        MessageServiceFactory messageServiceFactory = new MessageServiceFactory();
        MessageService messageService =
messageServiceFactory.getMessageService("wattsApp");
    }
}
```

```java
public class MessageServiceFactory {

    public MessageService getMessageService(String serviceType){

        if(serviceType.equals("text")){
            return new TextMessageService();
        }else if( serviceType.equals("wattsApp")){
            return  new WattsAppMessageService();
        }else{
            throw new RuntimeException("No such message service
exists");
        }
    }
}
```

Loosely coupled App

↓ Client

↑ runs

Main

Spring

Industries

App

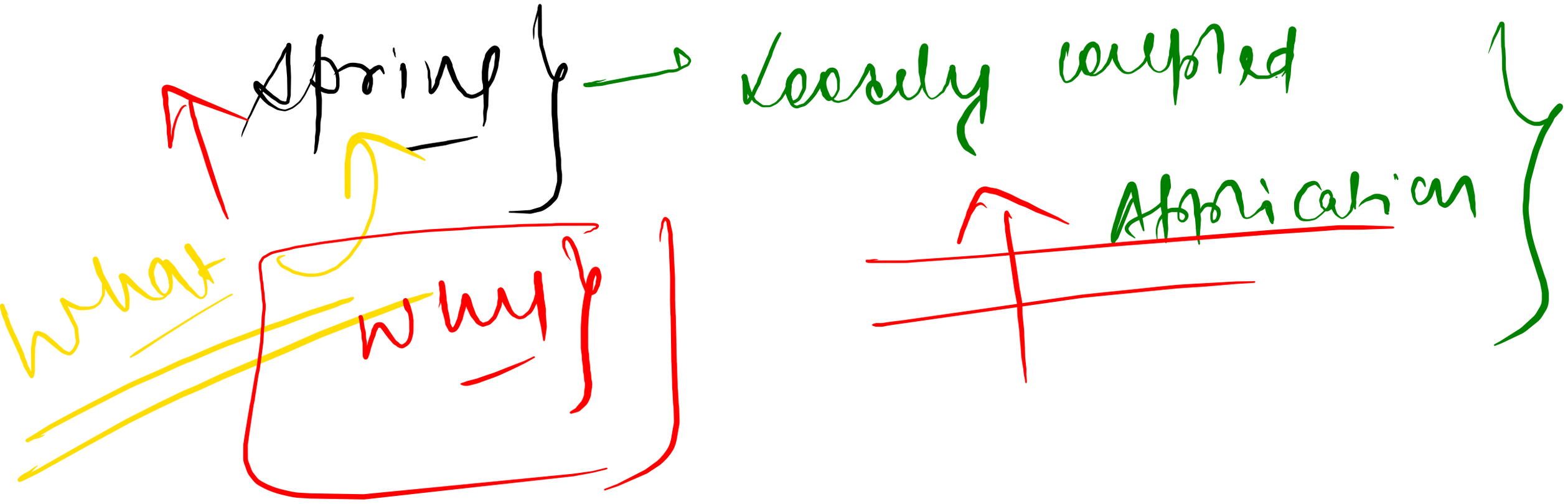
Loosely coupled app

Factory design pattern

Less coding

Too much effort

Too much time


Spring
Loosely coupled
Application
What
Why

Spring → loose coupling

internals → IoC | Inversion of control

DI | Dependency injection

IoC

DI

Design principle

Design Pattern

GOF

factory

creating

Behav.

Struc

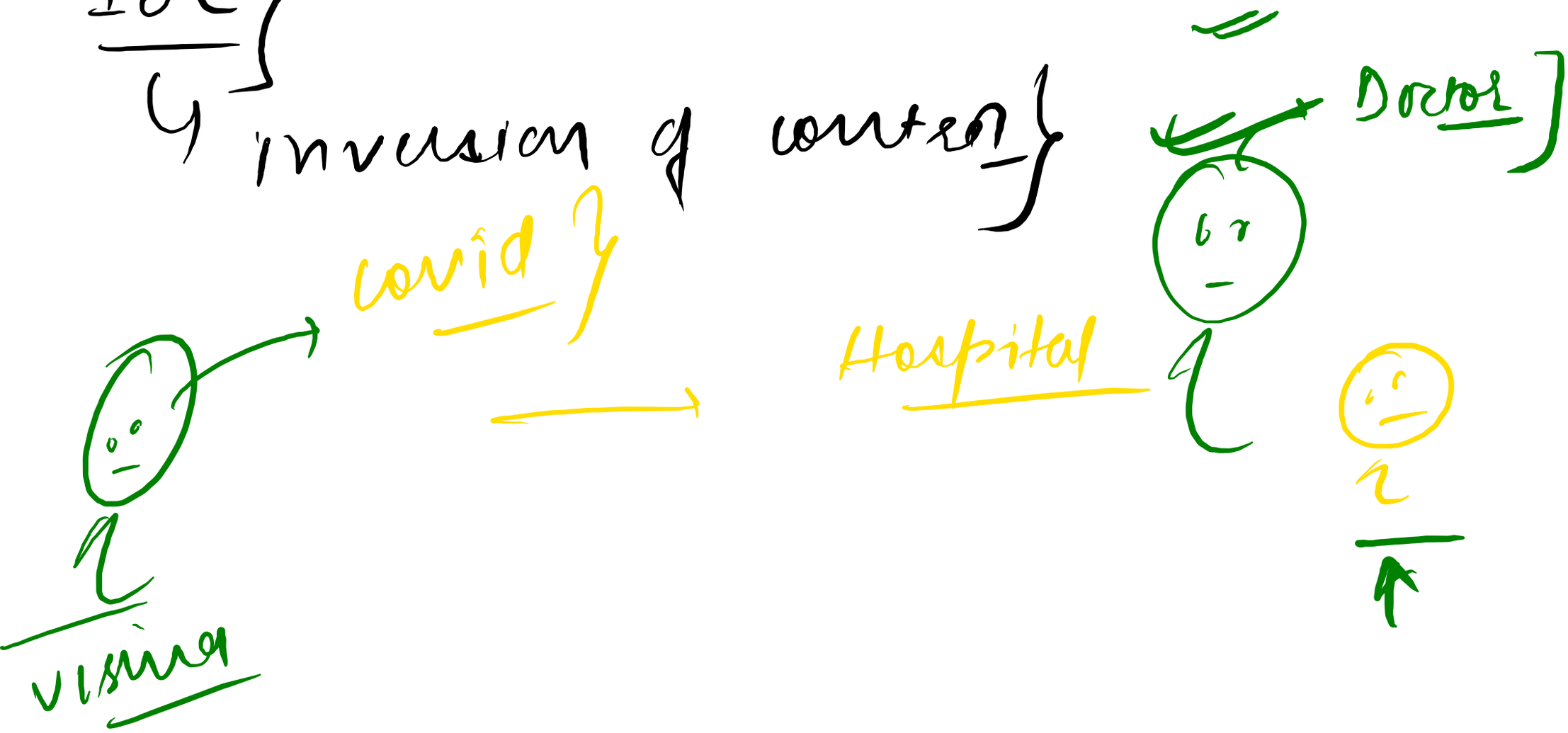
OO

How

So UI

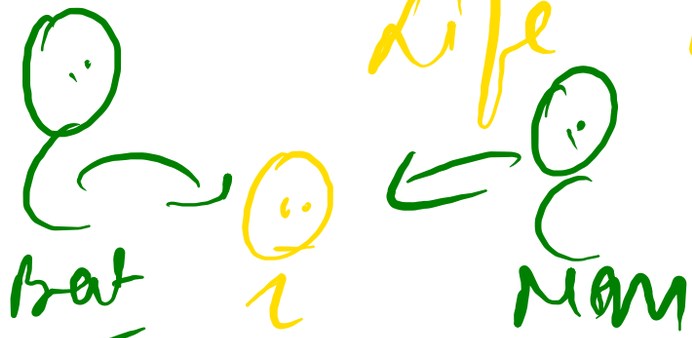
IoC

↳ inversion of control



Dependency Injection

Life when I was a kid !!!



(Dependency Injection)

grow up

Spring

Spring → Project

↳ all the dependencies, I will take care

(Spring)

class Person {

↳ Car car

class Car

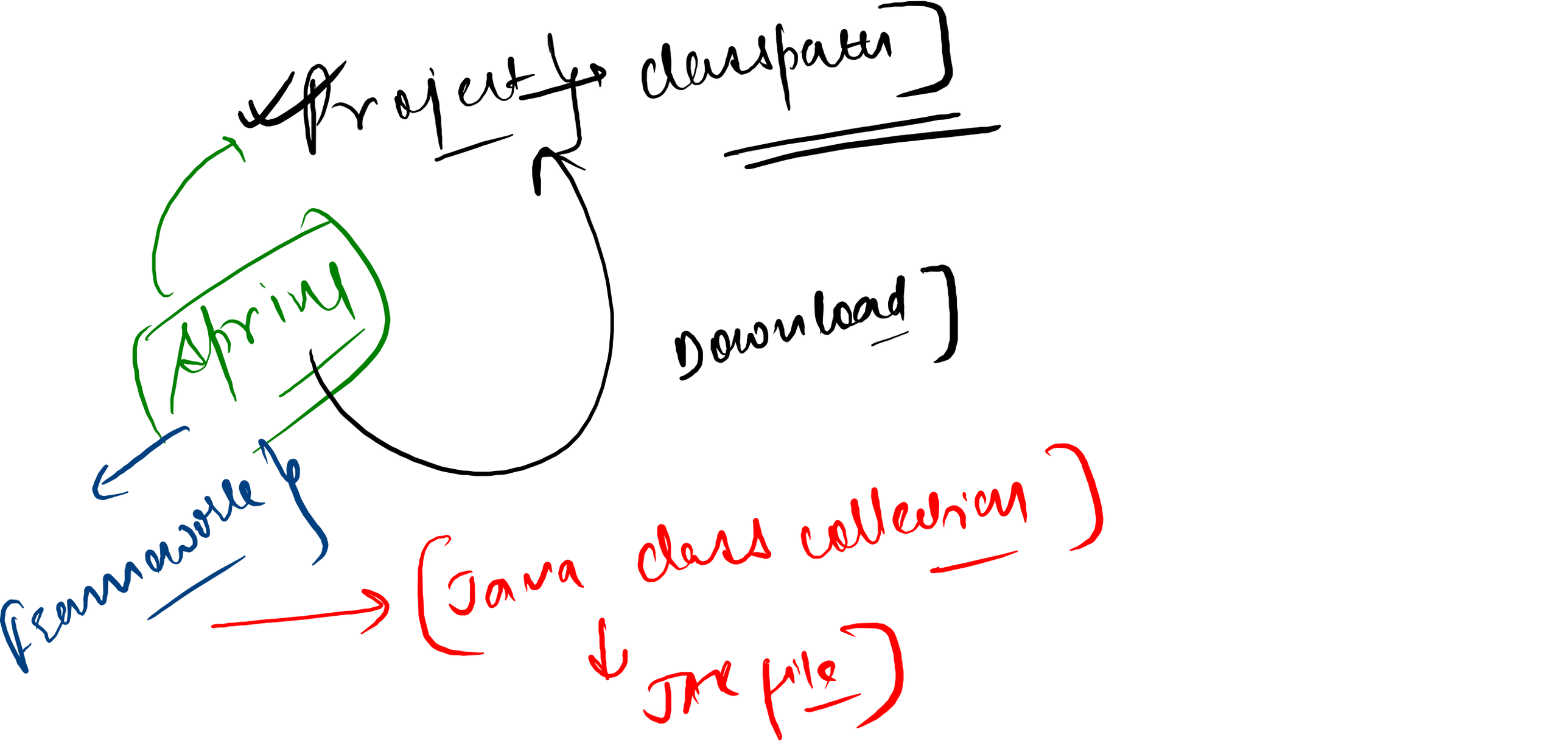
```java
public class App {


    public static void main(String[] args) {
        //This class needs the TextMessageService object to send message


        // It depends on TextMessageService object


        // Who will provide this ?

    }
```


Project
classpath
Spring
Download
Resource
Java class collection

→ spring related jar files are available

↓

① spring-core → Apache commons logging

② spring-beans

③ spring-context

④ spring-aop

(App)

(Spring → add all the external jars]

we have added Spring JARs

Project

(added)

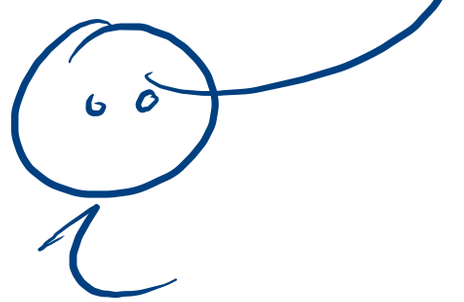
structure provided

① XML files

② Java code / configuration

③ Annotations

web.xml



Spring → Project]

↓

(Annotations) ↗ use it to start using Spring]

@ Component]

↓ Before a class)

Project

Run

Spring

Spring container / IoC container

@component
C1

C2

@component
C3

C4

obj of C1

obj of C3

Beans

Objects

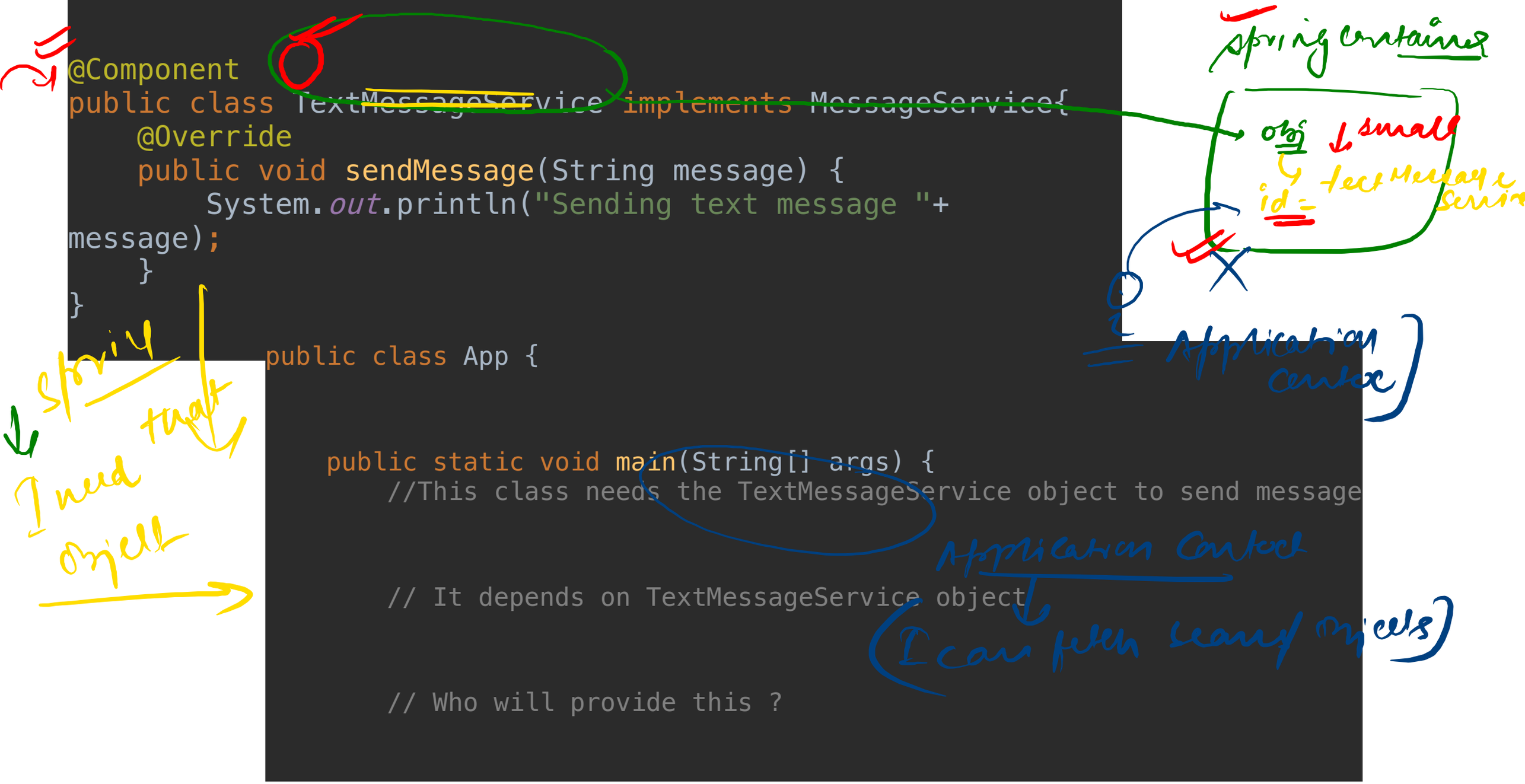
```java
@Component
public class TextMessageService implements MessageService{
    @Override
    public void sendMessage(String message) {
        System.out.println("Sending text message "+
message);
    }
}
```

```java
public class App {

    public static void main(String[] args) {
        //This class needs the TextMessageService object to send message

        // It depends on TextMessageService object

        // Who will provide this ?
```



```java
public class App {



    public static void main(String[] args) {
        //This class needs the TextMessageService object to send message


        // It depends on TextMessageService object


        // Who will provide this ?
        ApplicationContext applicationContext = new AnnotationConfigApplicationContext("com.vishwa.learnSpring");
        MessageService messageService = (TextMessageService)applicationContext.getBean("textMessageService");
        messageService.sendMessage("Hello Students");
```

Spring → Framework

Open close Principle

Loosely coupled

factory design pattern (singleton)

Spring

spring } dependencies]

① XML

② Java code }

Annotations } → ③ component
↑ class
before]

Project

@Component

C1

C2

C3

@Component

C4

C5

Spring

Object

id

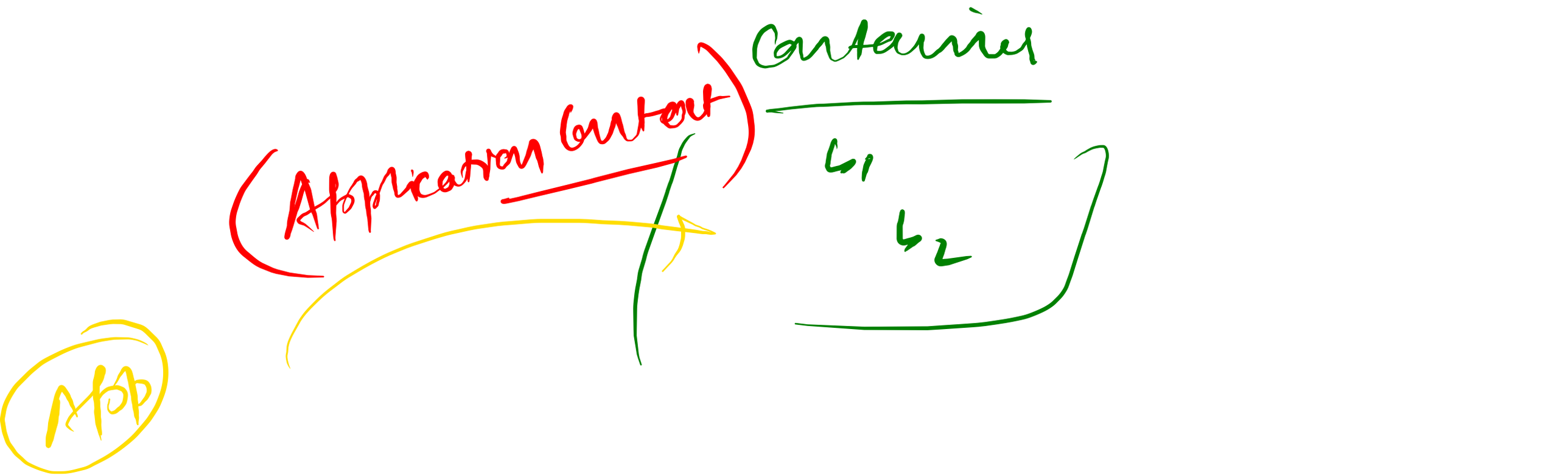
same as the name of the class

Spring container / IOC container

C1

C4

beans

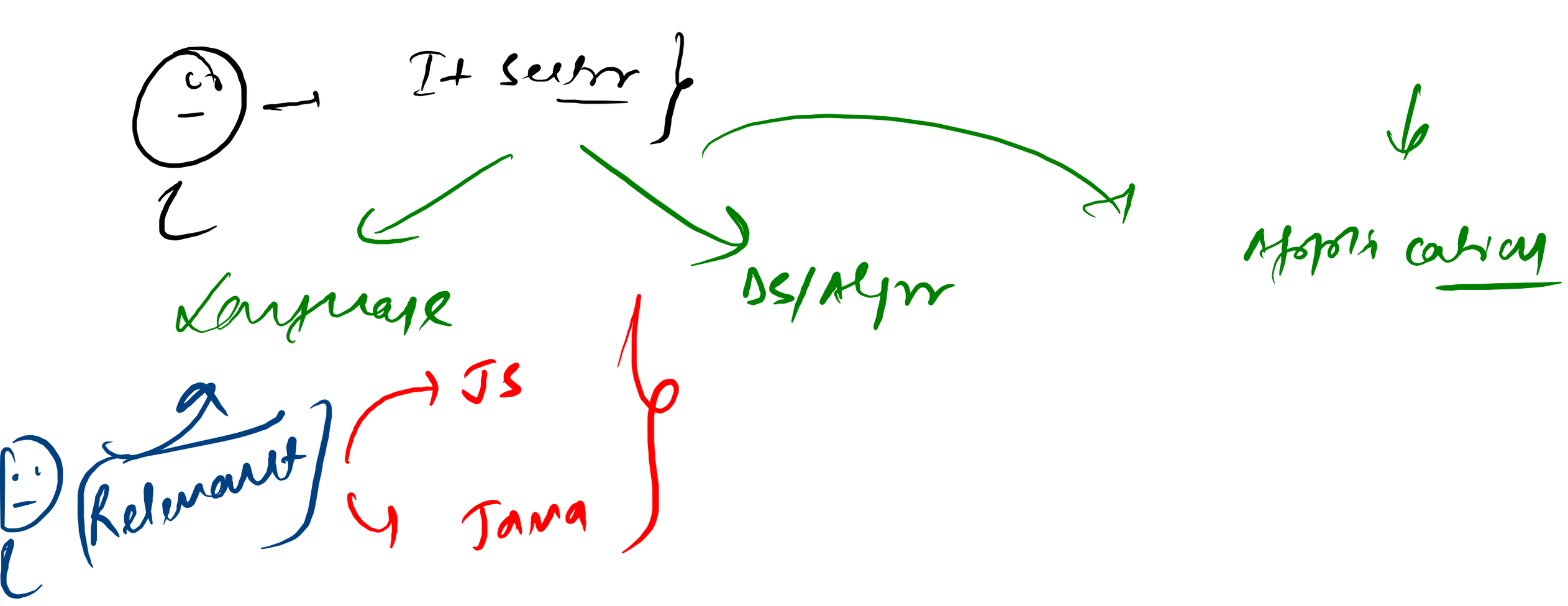

Application Context
Container
L1
L2
App

Real world

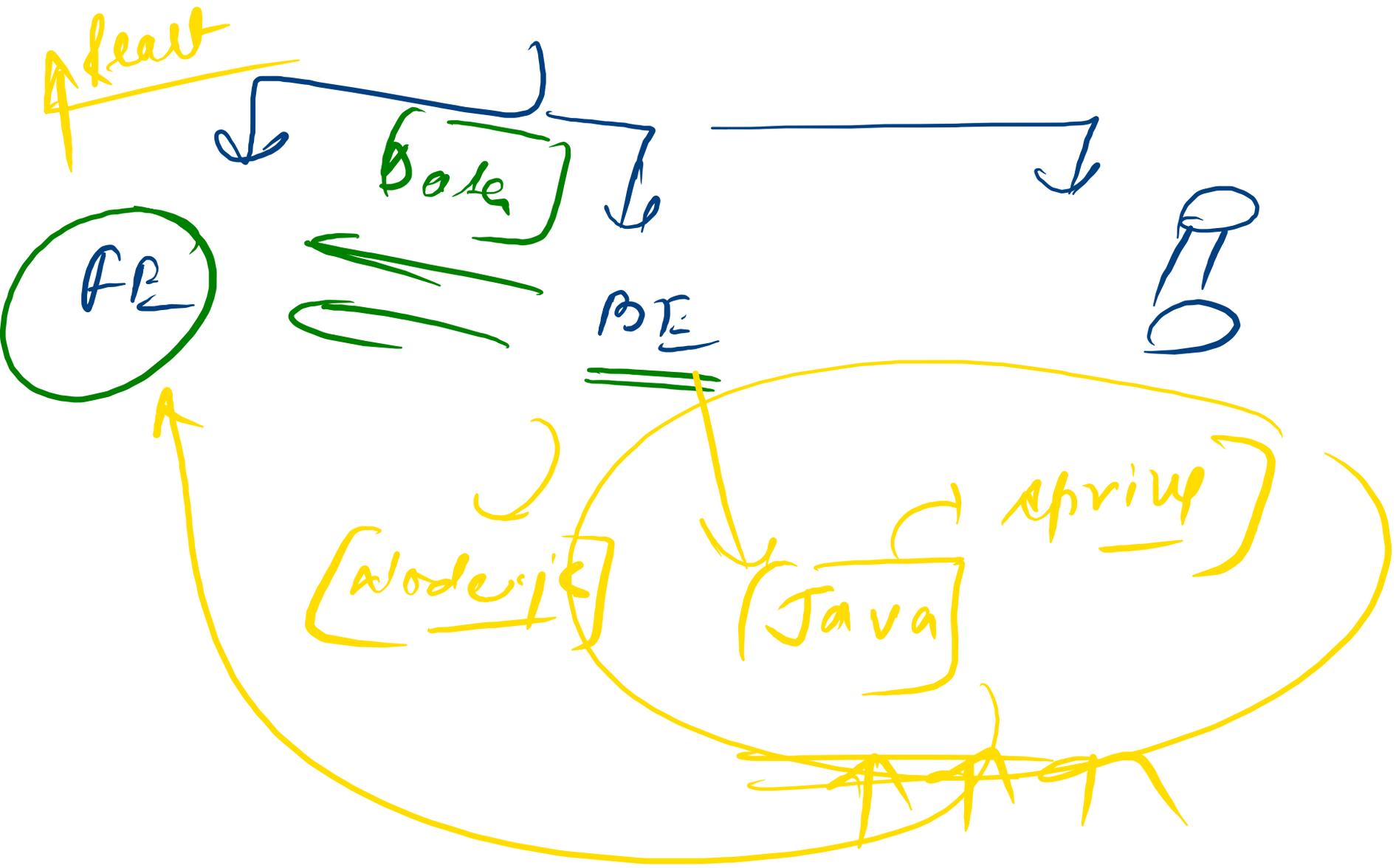
Real analysis

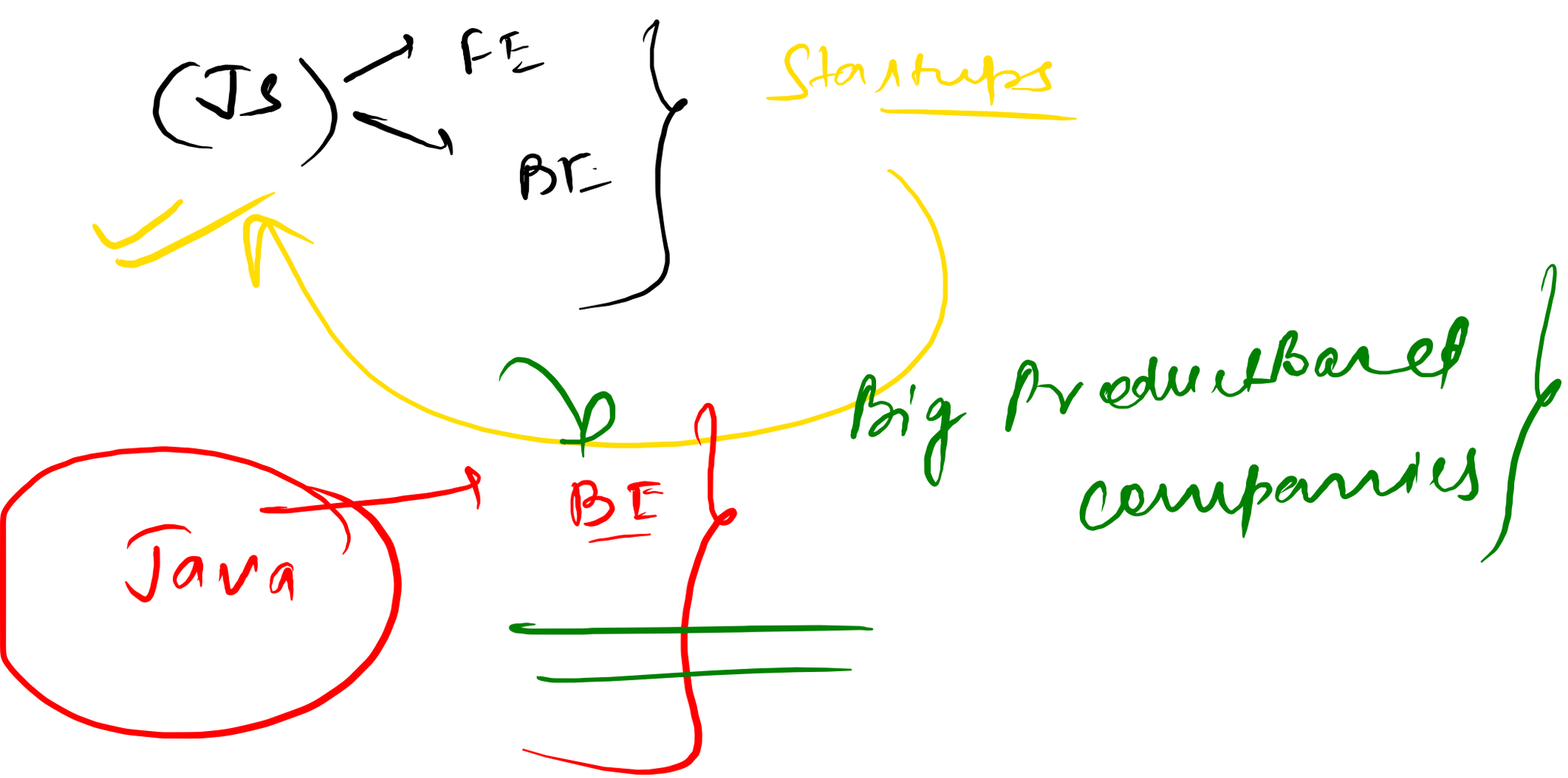
Loosely coupled

Tightly coupled


I+ sector
Language
DS/Algm
Application
Relevant
JS
Java

Full Stack




(JS)
FE
BE
Startups
Java
BE
Big Product Based companies