# Builder Design Pattern (Will start at 9:10PM)

Agenda

Builder Design Pattern

Problem Statement

(1) Class with a lot of attributes

```
class Student {
    name
    age
    PSP
    batch
    gradYear
    university
    email
}
```

(2) We need to validate objects of a class before they are created

Validations

1. No student with grad year < 2023
2. Valid email
3. Valid phone Number

(b) gradYear - age > 1950

OPTIONS

a.) Do validations in constructor
→ throw Exception from constructor if attributes not valid

```
Student ( _ _ _ ) {
    if (!valid ( _____ ))
        throw new Exception ( )
}
```

(b) Do validation in setter methods

---

## DO VALIDATION IN SETTER METHODS

a.) If there is a validation that involves 2 attributes
b.) setters are called after an object is created
c.) Objects are **immutable**

Mutation → Changing a class → Changing values of attributes of class

Immutable ⇒ State of class can't be change after an object has been constructed

- → Private attrs
- → no setters

---

## DO VALIDATIONS IN CONSTRUCTOR

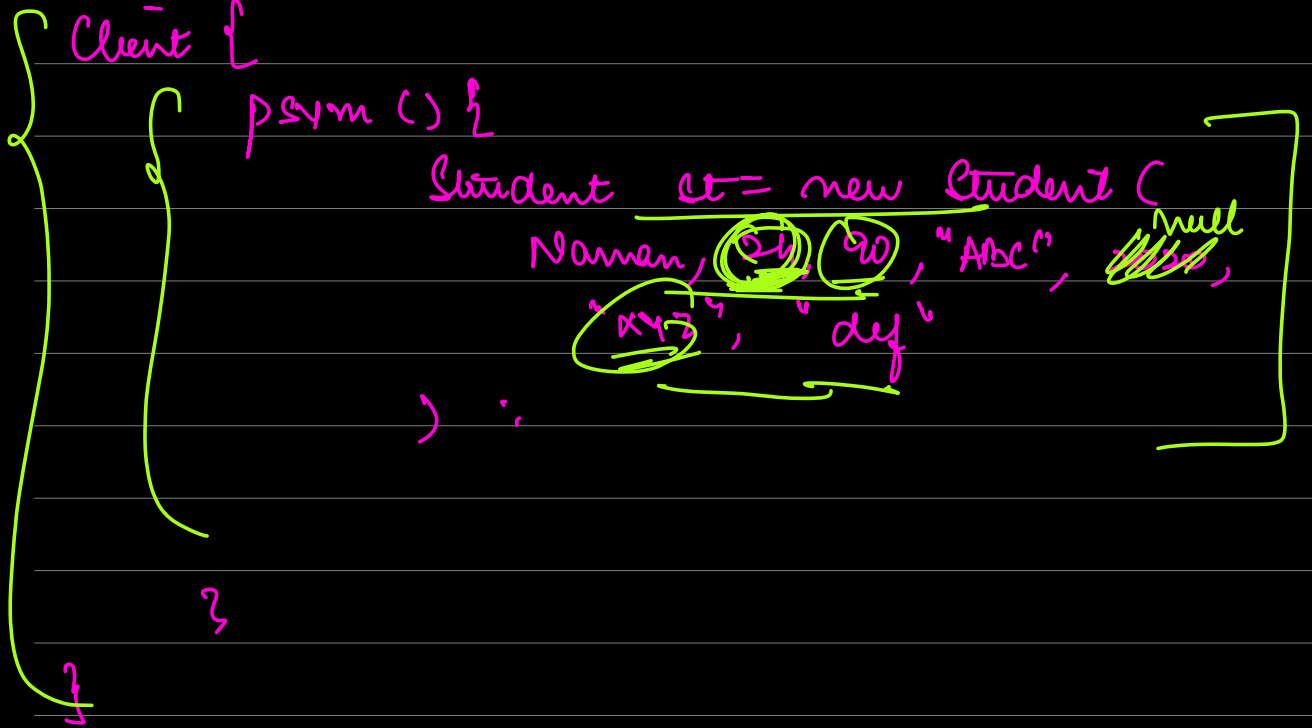
⇒ Class has too many attributes

```
class Student {
    name
    age
    psp
    batch
    gradYear
    university
    email
}
```

```
Student ( String name,
          int age,
          double psp,
          String batch,
          int gradYear,
          String univ,
          String email ){

    validate() ;

}
```




Problems

1. Potentially lead to bugs
2. Difficult to understand
3. lot of null

Student {

Student (name, email)
Student (name)
Student (name, psp, gradYear)
Student (uniName, batch)

Student (String, String)
↳ this (name, null, null, null)

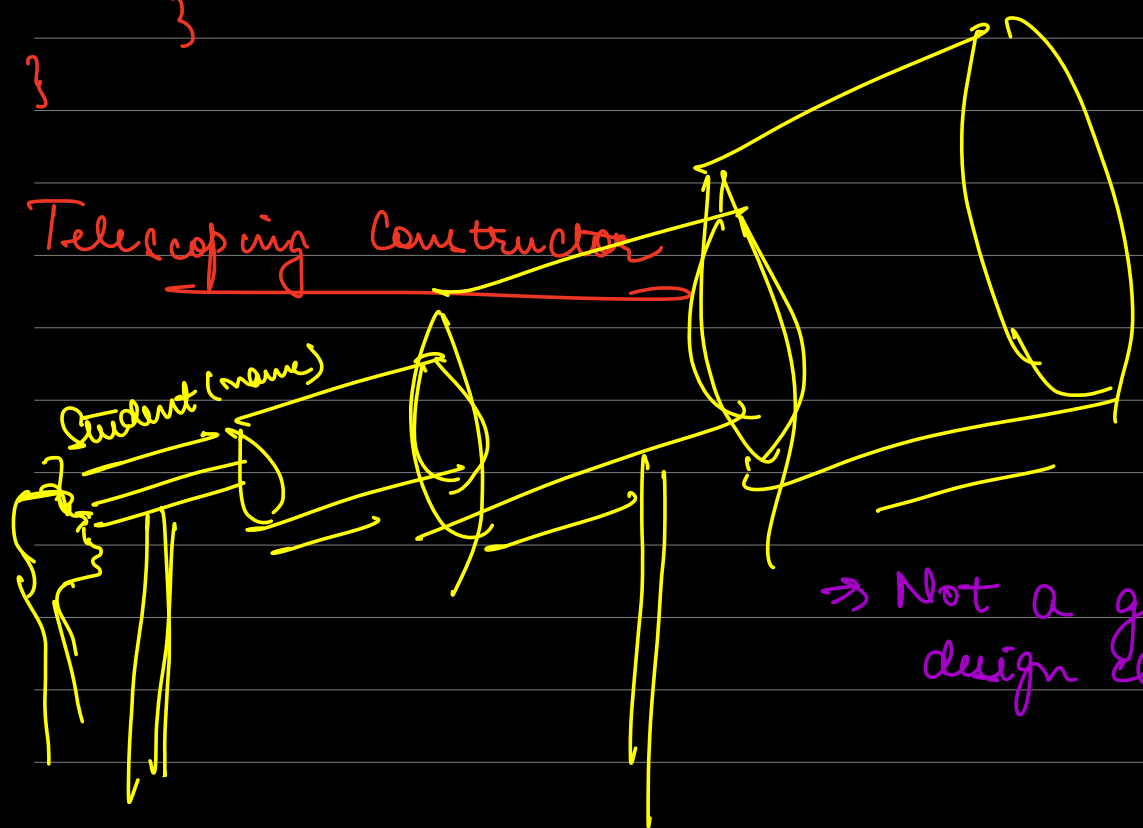
Student (String name,
int age,
double psp,
String batch;

1) Has the logic of validation
2) Has the logic

⇒ ( int gradYear, String name, String email ){ } ⇒ of setting all validate()?

```
Client {
    psvm() {
        Student st = new Student(name, email);
    }
}
```

Telescoping Constructor




⇒ Not a good design choice

Student (name)

↓

Student (name, email)

↓

Student (name, email), age)

↓

Student ( _____ )

Problem

1. Telescoping constructors are not a good choice
2. $2^n$ constructors
3. Sometimes it may not be even possible to create all combination of constructor.

Takeaways

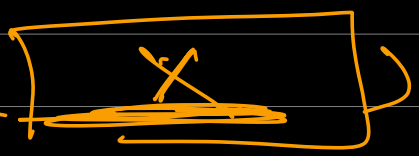
1. Whatever validation I have to do I have to do before an object is created.
2. The soln using constructor that I had till now made the code on of clients not good

→ a.) Not easy to understand what attribute stands for what
b.) Required to pass all attr even if they don't want to

LET'S THINK

We will go with a constructor
Can I change the type of param that const takes it:

① It allows to pass multiple value
② Each value should be recognizable via name

Student (  )

Name: ____

age: ____

Student ( Map < String, Object > map )

every string is an object
→ Integer
→ Boolean

map.set ("name", Naman)
map.set ("age", 18)
map.set ("university", XYZ)

Client {

psvm() {

Map<String, Object> mp = new Map();

mp.put("name", "Naman");

mp.put("age", 24);

mp.put("univ", "BML");

Student st = new Student(mp);

}

}

Problems

① Potentially lead to bugs

② Difficult to understand

③ Lot of null

Stu

→ No type safety

⇒ Will only check at runtime

↳ Typo with the name of key

class Student {

Student(Map<String, Object> mp) {

— validate

this.name = (String) mp.get("name");

this.age = (Integer) mp.get("age");

}

}

Something

a.) That has multiple values ⇒ Map

b.) Multiple values might be of diff data types ⇒ ?

c.) The name of the key of the values should be recognizable at compile time ⇒ ?

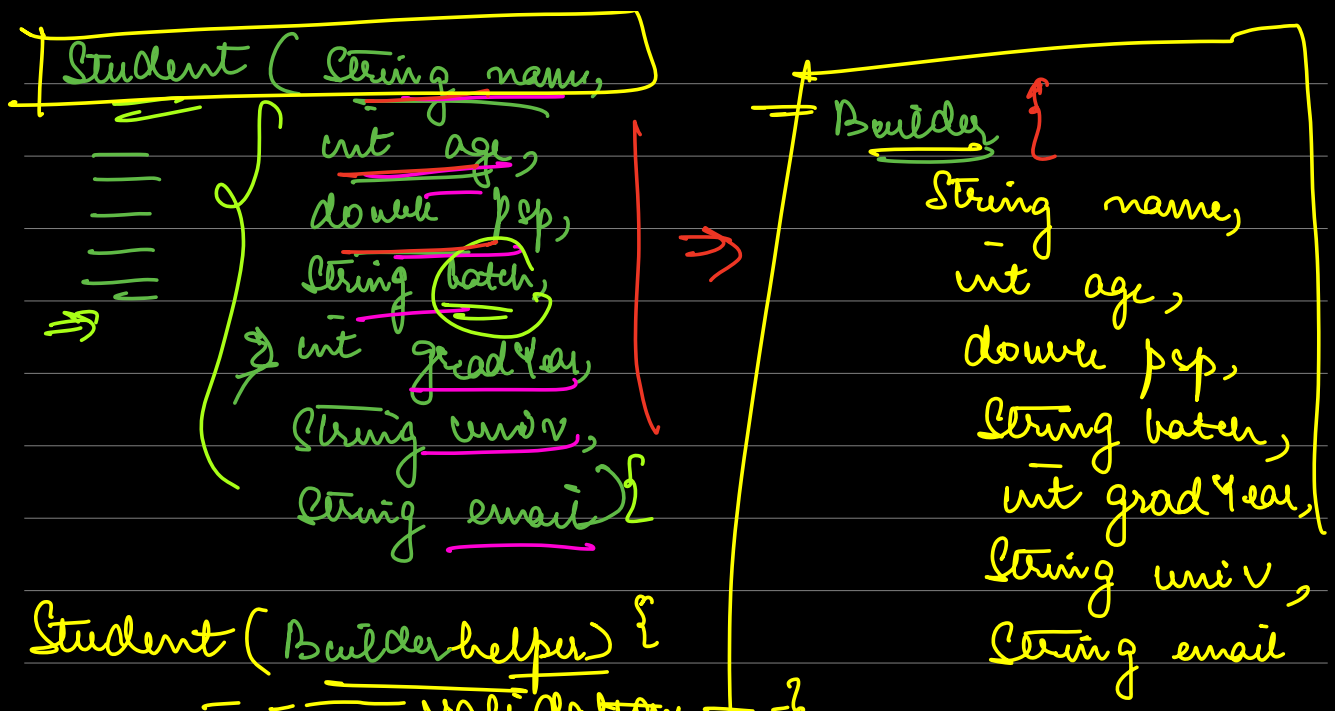
Class

Multiple Values

Diff Data Types

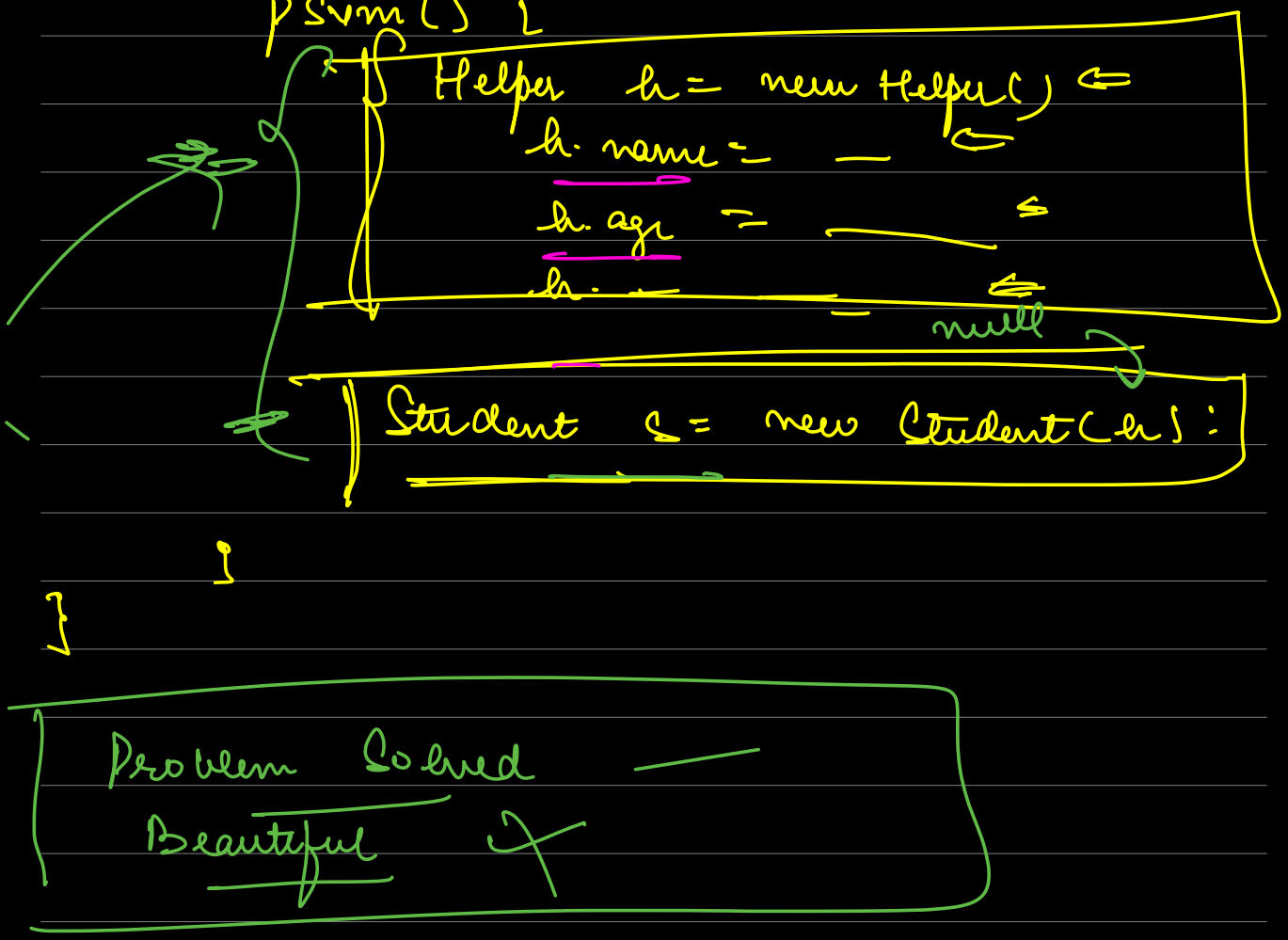
Compile Time Checking

myMap




Student (Builder helper) {
validation
this.name = helper.name;
age = age

}




Problem Solved Beautiful ✓

(1) To create an obj of Student client will need to know to create obj of Helper

(2) Client can pass null.

```
Class Student {
    String name;
    int age;
    String univ;
    String batch;
    double pcp;

    private Student (Builder b) {
        -- validation --.
        -- Setting attr --
    }

    public static Builder getBuilder() {
        return new Builder();
    }
}

Client {
    psvm() {
        => Builder b = Student.getBuilder();
           b.name = ___
           b.age = ___
        => Student s = new Student(b)
    }
}
```

class Student {

attrs - - -

⇒ private Student (Builder b) { }

public static Builder getBuilder () {}

↳ public static class Builder {

---- same attrs as Student ---

public Student build() {

return new Student(this);

}

}

}

HW:

1. Inner Class
2. Static Class

```
Client {
    psvm() {
        Builder b = Student.getBuilder()
        b.Name = ___
        b.age = ___
        b.univ = ___
        Student s = b.build()
    }
}
```

```
Student s = Student.getBuilder()
            .setName(Naman)
            .setAge( )
            .build();
```

Use Builder whenever:

1. Class with many attributes → CONSIDER
2. Immutable class with many → ALWAYS
3. Class with validation → 90%

>=4

In production :

→ Setting params in API

→ Query Builder

HW

1. Read about Builder design pattern from refactoring.guru
   → Code more in other languages
2. Implement Builder in your primary language.
3. Explore Firebase API and find why were builders used.
4. Read about static inner class

---

Singleton Design Pattern

Cons :

→ It makes testing difficult

getInstance() {}