⊛ Agenda:-

⊙ Builder Design Pattern (with variations)

⊛ Builder Design Pattern :-

Definition :-

↳ creational Design Pattern

↳ used to construct object step by step.

↳ when an object has a lot of attributes

↳ you want to ensure that object should only be created once (validations have happened.

ⓐ The Problem :-

① we have a class with a lot attributes.

```
class Student {
    - name
    - age
    - PSP
    - batch
    - id
    - uniName
    - grad year
    - phoneNumber
    - email
}
```

```
Student St = new Student();
St.setName ("Vishal");
St.age (25);
St.PSP (80);
St.batch("June 2023");
;
.
```

② we want to validate the object of the class even before creating the object

ⓐ Validations :-

① No student should be there if grad_year > 2023

① email validation

- Phone Number should be valid.
- age > 0
- ⋮

what we want ?

→ No student object should be created unless we have all the validations performed.

Q. where is an object created ?

→ new Constructor();

(Constructor)

```
class Student {
    - name
    - age
    - psp
    - batch
    - id
    - uniName
    - grad year
    - phoneNumber
    - email
    - buddyId.
}
```

```
Student ( String name,
          int age,
          double psp,
          String batch,
          int id, int buddyId
          :
        ) {
    :
    // validations
}
```

```
Client {

    psvm() {

        Student sf = new Student ("Vishal",
                                  25,
                                  80.0,
                                  "June 2023",
                                  123,
                                  :
        );

    }
}
```

- Difficult to UNDERSTAND!
- Prone to ERROR!

1. Put Null / Empty.

```
Student s = new Student(null, 0.0, null, "ABC");
```

Compilation Errors!

1. Overloading Constructor:-

↳ Instead of having 1 Constructor.
↳ will have multiple.

- name, PSP

Student (String name, String BP);
Student (String name);
Student (String name, String BP,
int gradyear);
⋮

N attributes →

1 → ✓
→ ✗

$^{N}C_{2}$ ✗

2 × 2 × 2 × 2 ... - - -

2^N ✓

10

2^N constructors !

2.

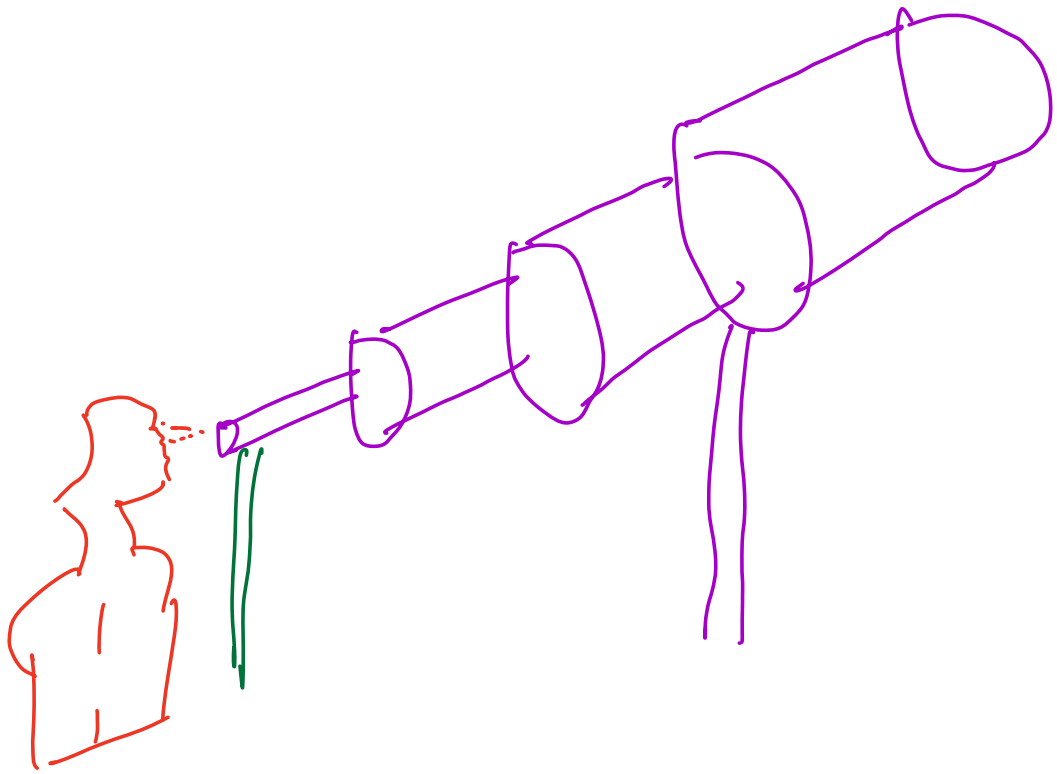
Student (String name, double PSP);
Student (String name);
Student (String name, String PSP,
int gradYear);
:
Student (String univName, double PSP);

(String, double);

Impossible!

1. Too many constructors
2. Sometimes, it is impossible to create all constructors.

Telescoping Constructors!



```
Student {

    Student (name) {
        this.name = name;
    }

    Student (name, psp) {
        this.name = name;
        this.psp = psp;
    }

}
```

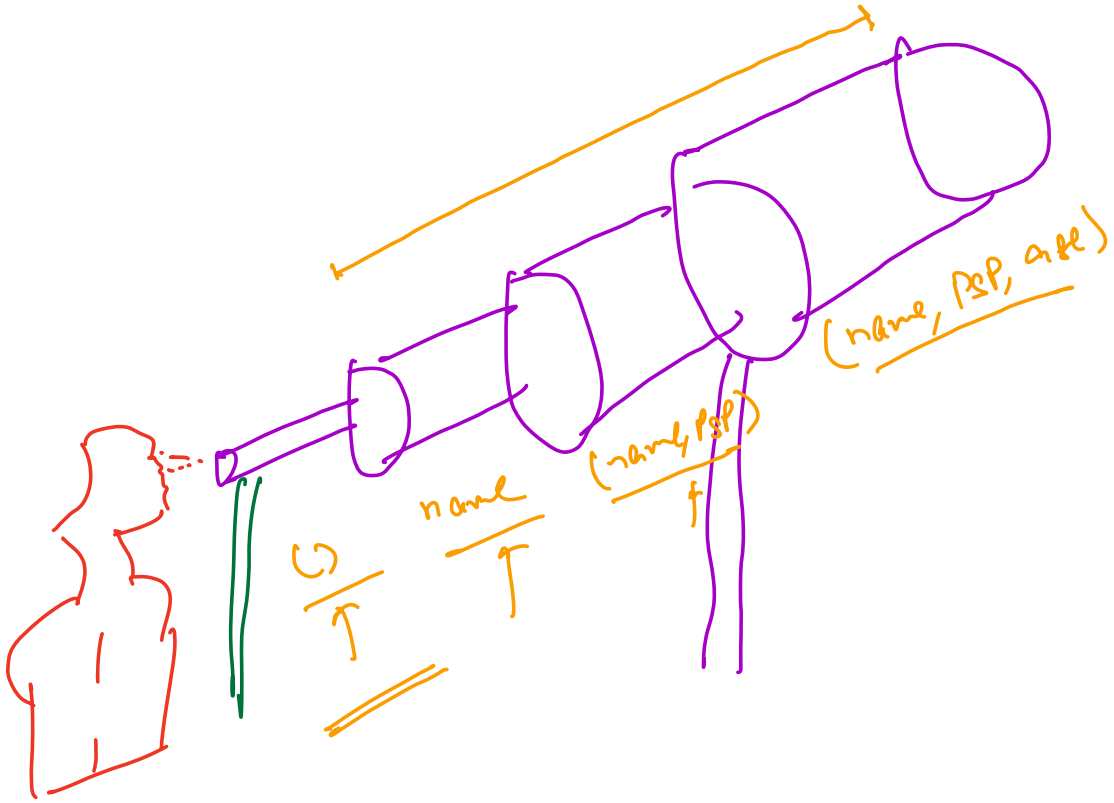
code Duplicacy

```
Student (name, psp, age) {
    this.name = name;
    this.psp = psp;
    this.age = age;
}
```

```
Student (name) {
    this.name = name;
}
```

```
Student (name, psp) {
    this (name);
    this.psp = psp;
}
```

```
Student (name, psp, gradyear) {
    this (name, psp);
    this.gradyear = gradyear;
}
```

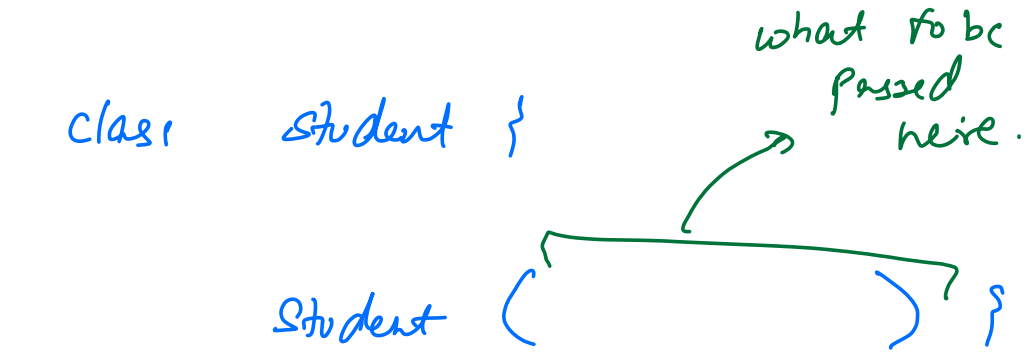


Is Telescoping constructor good?

→ NO!

Telescoping constructor should be avoided!

Constructor



Is there any DS that can hold all the variable/values with their specific Name?

Parameter List

name : Vishal
age : 25
roll : 123
:
:

Map!

name: vishal.

{ name: "Hello"
"psp": "world"
}

class Student {

Student (Map<String, Object> map){

this.name = (String) map.get("name");
this.psp = (Integer) map.get("psp");

}

}

1. Typecasting
2. Client does a "Typo".

Do we have something —

1. which is like a map.
   ↓
   it allows us to have different values within it, recognised by specific name.

2. should have compile-time checks for the key.

   map.name = "vishal" ✗

3. should have compile-time checks for data type of values.

   map.age = "Hello" ✗

⟹ Class !

```
class Helper {

    String name;
    double PSP;
    String universityName;
    int age;

};
```

helper.name = "Vishal";

helper.age = "Vishal"

Builder

```
class Student {
    - name
    - age
    - psp
    - batch
    - id
    - uniName
    - grad year
    - PhoneNumber
    - buddy id.
}
```

```
class Helper {
    - name
    - age
    - psp
    - batch
    - id
    - uniName
    - grad year
    - PhoneNumber
    - email
    - buddy id.
}
```

```
Helper helper = new Helper();

helper.setName(____);
helper.setAge(____);

Student s = new Student(helper);
```

```
Student (Builder helper) {
    // validations
    this.name = helper.name;
    this.age = helper.age;
}
```

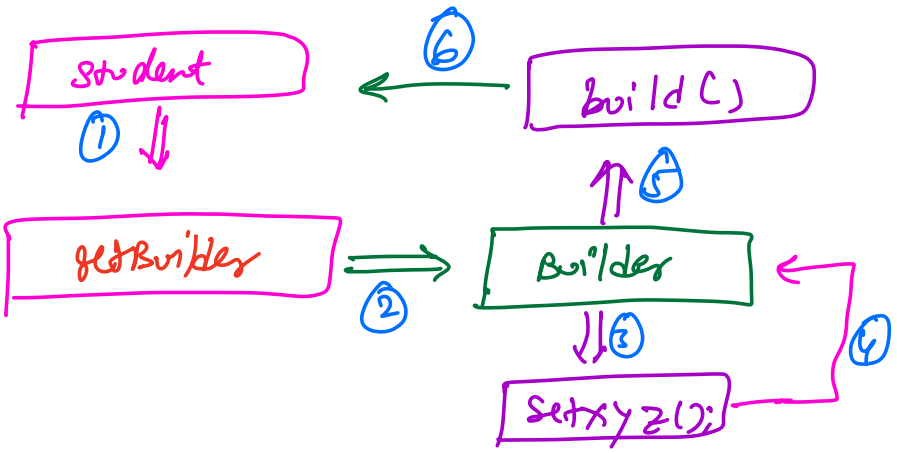
(Builder Design Pattern)!

(*) When to use Builder Design Pattern:-

1. When we have a lot of attributes.
2. Need to validate the params before creating actual object
3. Immutable class.
   - ↳ all the parameters need to be passed at the time of creation.

@ Builder

Summary :-



Assignments :-

- Practice out Builder Design Pattern.
- Read builder design Pattern from —
  - Sourcemaking ✓
  - Refactoring.guru

- Read about firebase API documentation (Builder)