⇒ **Builder**

1. Class with lot of attributes.

```
Class Student {
      name
      age
      batch
      ~~psp~~
      gradYear
      id
      univName
      phoneNo
}
```

```
Student st = new Student();
st.setName(——);
st.setAge(—);
st.setBatch(—);
—
—
—
—
```

2. We want to validate the object before it gets created.

Validations.

1. Phone no. should be valid.
2. GradYear <= 2022

No object should be created if any of the validation is getting violated.

```
Student st = new Student();

Class Student {
    name
    age
    batch
    psp
    gradYear
    id
    univName
    phoneNo
    Student(String name, int age, String batch,
            double psp, int gradYear,
            int id, String univ, String phone){
        ____        //Validations
        ____
        ____
        ____
    }
}
```

Client

```
Student st = new Student("Vijaya", 23,
                 "June 22", 84.0, . . . . . );
```

→ Prone to errors.

→ Difficult to understand.

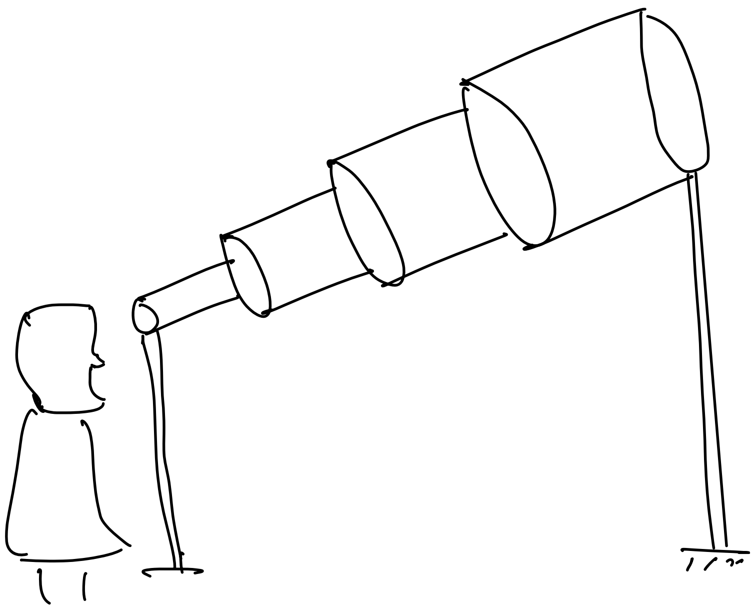
```
Student {

        Student( String name, int age ){ — }
$2^N$   Student( String name, double psp ){ — }
        Student( String name, String univName ){ — }
        ==
        Student( String univName, int age ){ — }
}
```

→ Too many constructors.

→ Sometimes its impossible to create some constructors
   (due to same method signature)

Telescoping Constructors.



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

⇒ Telescoping constructors should be avoided.

```
# Student {
    Student (Oneparam) {
        ---
        ---
        ---
    }
}
```

↑ Some DS that allows us to Store values for a particular field.

⇒ {
  "name": ____
  "age" : ____
  "batch": ____
  ≡
}

⇒ HashMap

⇒ Student {
    Student (Map < String , Object > map) {
        this. name = (String) map.get("name");
        this. age = $\underbrace{\text{(Intega) map . get ("age");}}_{\text{Runtime Exception.}}$
        ____
        ____
        ____
    }
}

⇒ map. put ("nama" , "Vijaya") : Typo

⇒ map. put ("age" , "hello");

⇒ Something like map (that allows us to store different values with type check at compile time).

map · nama ✗

map · age = "hello" ✗

⇒ Class Builder

name
age
batch
~~psp~~
gradYear
id
univName
phoneNo

}

helper · nama ✗

helper · age = "hello"; ✗

⇒ Helper helper = new Helper();

```
helper · setName ( — );
helper · setAge ( — );
___
___
___
___
```

Student st = new Student(helper);

```
Student {
    name
    age
    batch
    psp
    gradYear
    id
    univName
    phoneNo
    Student(Helper helper) {
        // Validations ✓
        this.name = helper.name;
        this.age = helper.age;
    }
}
```

8:14 am.