BUILDER. → where it is most useful :-

use case :-

1) Class with a lot of attributes.

```
Class Student {
    - Name
    - age
    - batch
    - Rep
    - univName
    - gradYear
    - PhoneNo
}
```

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

2) We want to validate the Student object before its creation.

Validations

→ gradYear <= 2023

→ Phone no. should be valid.

→ ___

→ ___

No Student object should be created if any of the validation fails.

Class Student {

- Name
- age
- batch
- Psp
- univName
- gradYear
- PhoneNo

Student (String name, int age, String batch,
double psp, String univName, ...) {

if (age > 20) {
throw ——

}

if (Phone is invalid) {
throw ——

}

this.name = name
this.age = age

}

}

PSVM () {

Student st = new Student ("Raj", 20, "Eve", 90.0, ------);

→ Prone to Errors.
→ Difficult to understand.
→ Difficult to read

}

Class Student { → If we have to set just few attributes instead of all then for 'n' attributes it might have to write 2^n constructor.

- Name
- age
- batch
- Rep
- univName
- gradYear
- PhoneNo

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

Student (String name, int age)
    this.name = name;
    this.age = age;
}

(1)

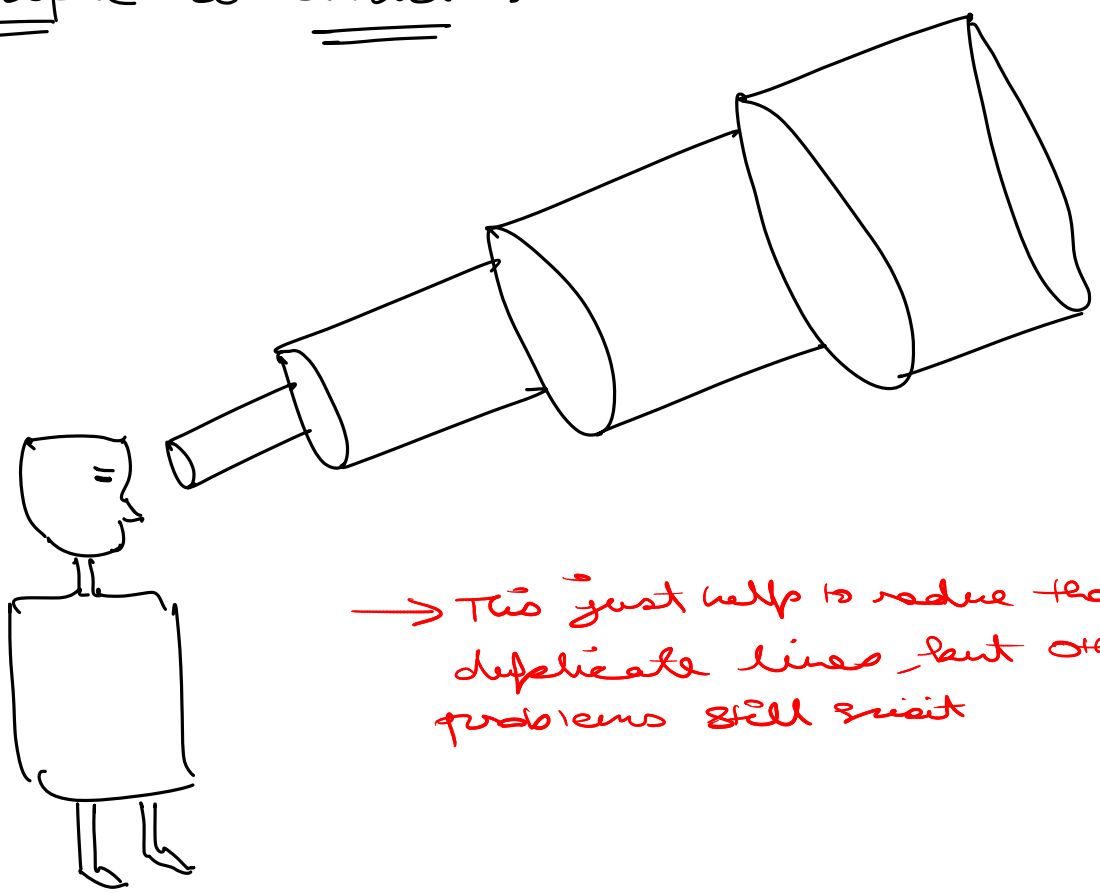
⇒ N attrs. : # of Constructors = 2^4

Student (String univ, int age){

}

② Some of the constructors might not be feasible as well because of same signature.

Telescopic Constructors.



→ This just helps to reduce the duplicate lines but other problems still exist

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

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

```
Student (name, age, batch)
    this(name, age)
    this.batch = batch
}
```

Usecase

Student (<Data Structure>)

↓

Some DS that can allow us to pass multiple attributes of different datatypes.

"name" : ——
"age" : ——
"batch" : ——
——
——

Map

Map<String, Object>

Class Student {
- Name
- age
- batch
- Rep
- univName
- gradYear
- PhoneNo

```
Student(Map<String, Object> map){
    this.name = (String)map.get("name");
    this.age = (Integer)map.get("age");
}
```

map.put("name", —);
map.put("age", "Scaler");
while setting the issue can occur;

"Scaler"

Runtime Exception.

Issues

1) Typo in keys.
2) Type mismatch issue.

⇒ What we need ?

Something like map, that allows to store different values each with a different name. Also it should provide a compile time check over the values & keys.

ds.nama ×

ds.age = "—";

Class Helper {

- Name
- age
- batch
- Rep
- univName
- gradYear
- PhoneNo

}

Student {

- Name
- age
- batch
- Rep
- univName
- gradYear
- PhoneNo

Student (Helper helper) {

//Validations

this.name = helper.name

this.age = helper.age

}

}

psvm() {

It is just like Map

Helper helper = new Helper();

helper.setName(—);

helper.setAge(—); => Compile Time Check.

helper.setBatch(——);

{ helper.name = ——

Student st = new Student(helper);

}

(*) → Demo of Builder Pattern :-

→ ctrl + N => To create getter & setters

→ To create the object of a particular class we will send the code in that particular class to creat the object

→ But student class we don't have any idea of the helper class like how helper object should be created then should be some way student class should tell how should we creat the helper object

(4) → when class is present in other class then it is called as Inner class.

https://www.baeldung.com/java-nested-classes