Agenda : Builder design Pattern

class starts at 9:05

⇒
```
class Student {
    String name;
    int age;
    String universityName;
    double psp;
    int rollNo;
    int gradYear;
    String address;
}
```

```
Student st = new Student();
st.setName("abc");
st.setAge(21);
st.setUniversityName("xyz");
```

1. Too many attributes
2. What if I want to validate the attributes before creating an object.

⇒ In the above example, we created the object without validating the attributes

⇒ Where should the validations happen? ⇒ Constructor

⇒ Student {

Student (String name, int age, String universityName ....) {

}

}

Client {

psvm {

Student st = new Student ("abc", 22, "xyz");

① Difficult to understand code, I have to go and look at the constructor

code to understand which attribute is getting assigned to what value.

(2) What if we dont have values for all the attributes, then we might be in a position where we have to pass null.

(3) Student st1 = new Student("abc", 21, "xyz");
Student st2 = new Student("xyz", 21, "abc");

(4) Let us say now Student has a new attribute.

Student (String name, int age,
String universityName .... , String address)

Student st = new Student("abc", 21, "xyz");
" " = new "("abc", 21, "xyz", null);

this will break

Student {

① Student (String name, double psp) { }
② Student (String name) { }
③ Student (String name, int age) { }
④ Student (String univName, int age) { }

{ ① & ③
② & ④ ✗

① Too many constructors
② Sometimes it is not even possible like ③ & ④

Student {

① Student (String name) {
this.name = name;
}

Student (String name, double psp) {
② this (name);
this.psp = psp;
}

Student (String name, double psp, String address){

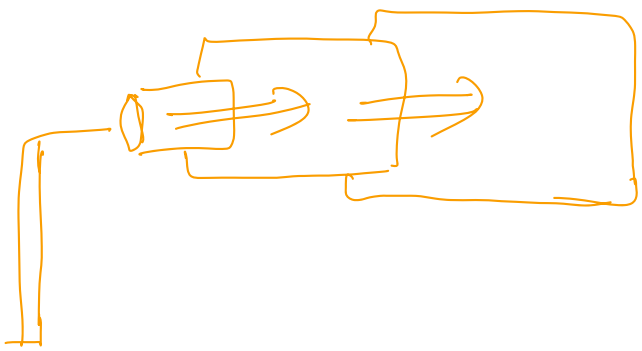
③ this (name, psp)

this.address = address;

}

}

this technique is called telescoping of Constructors.



Student (String name) {

this.name = name;

}

violation of DRY

Student (String name, double psp) {

this.name = name;

this.psp = psp;

this.address = address;

this.rollNo = rollNo;

}

Student (String name, double fsp) {

this (name, fsp, address, rollNo)

}

this.universityName = universityName;

Student {

Student (Map<String, Object>) {

}

}

Map

⇒ Now If I want to create an object of Student.

(1) I can create an object with just name;

(2) I can also create with name & fsp;

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

① What if someone passes age as "hello"? ✗

② What if theres a typo ⇒
map.put("Nam", "abc");

Let us try to think of an ideal soln.

① Something similar to a map that has named keys.

② Should have compile time check for key names
helper.Nam ⇒

③ Should have compile time check for datatype of variables.
helper.age = "Hello"

Class

```
class Helper {
    String name;
    int age;
    String universityName;
    double psp;
    int rollNo;
    int gradYear;
    String address;
}
```

```
helper.nam = "abc";
helper.name = 21;
```

let us bring back everything

```
class Student {
    String name;
    int age;
    String universityName;
    double psp;
    int rollNo;
    int gradYear;
    String address;
```

```
Student ( Helper helper ) {
    if( helper.age > 21 )
        throw new IllegalArgumentException();
    age = helper.age;
    name = helper.name;
}
```

call the validations

```
Client {
    psvm {
        Helper helper = new Helper();
        helper.setAge(21);
        helper.setName("abc");
        Student st = new Student(helper);
    }
}
```

⇒ Helper class is helping in building an object of Student class and this is know as Builder Design Pattern.

⇒ Def^n of Builder

↳ (1) Too many attributes
(2) Validations before creating an object.

```
(3) Student (String name, double psp){
        (4) this(name);
        (5) this.psp = psp;
    }
```

⇒

```
(1) Student (String name, double psp, String address){
        (2) this(name, psp) x [this.name=name; this.psp=psp;
        (6) this.address = address;
    }
```

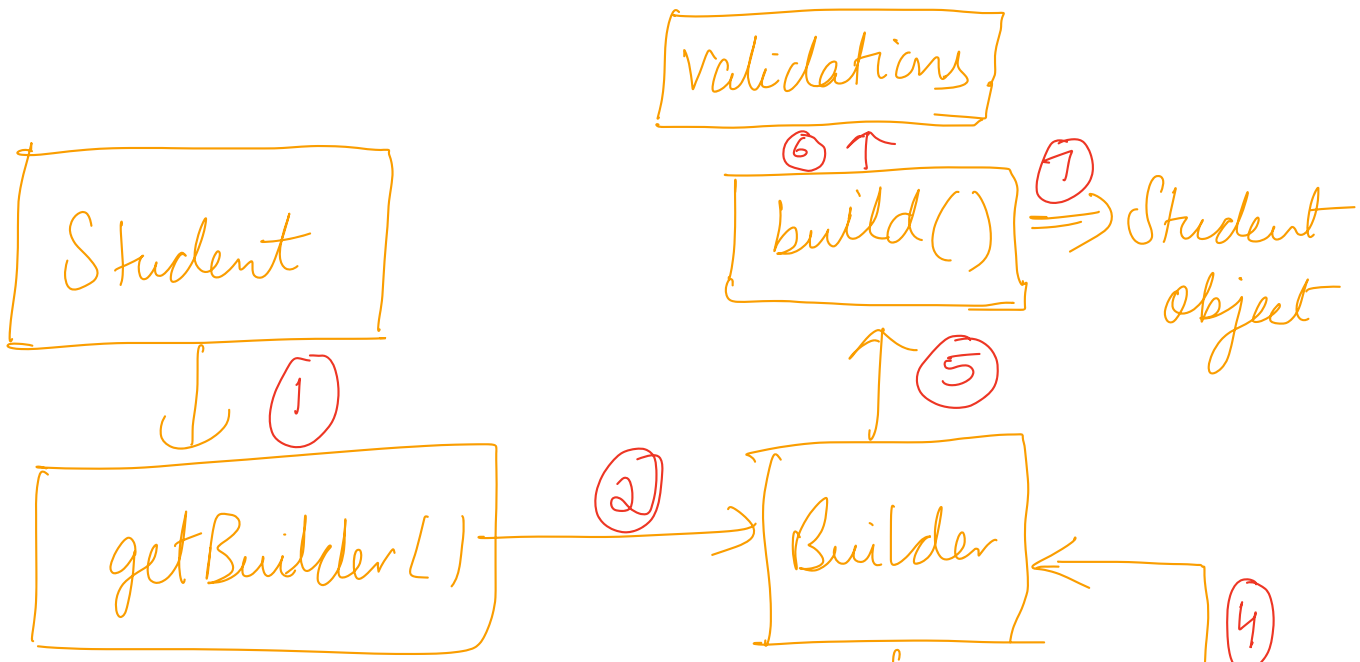
```
Student st = new Student("abc", 90.0, "xyz");
```

Summary of Builder

(1) Class with many attrs, even though if there are no validations, consider using Builder design pattern

(2) Need to validate params before object creation

(3) Immutable class ⇒ [Once the object is created, you cannot alter the attributes.] so a lot of attributes are passed in object creation, so scope of error increases.



Refactoring. guru
SourceMaking

Assignment : Implement it in your preferred language.