Agenda

1. Builder Design Pattern
2. Prototype Design Pattern

] Creational Design Pattern

BUILDER

1. → Class with a lot of attributes

```
class Student {
    - Name
    - age
    - psp
    - batch
    - id
    - univName
    - grad Year
    - phoneNumber
}
```

⇒ `Student st = new Student();`

```
st.setName();
st.setAge();
st.setPsp();
st.setBatch();
```

2. We want to validate an object of the class before creating the object.

Validations

⇒
1. Not Student with gradYear > 2022
2. Phone Number must be valid
3. gradYear >= age + 2000

No Object of Student should be created unless all the validations have happened.

⇒ Before creation of object ⇒ Validations should happen in the constructor.

```
class Student {
    - Name
    - age
    - pop
    - batch
    - id
    - univName
    - gradYear
    - phoneNumber
    - buddyId

    Student(String name,
            int age,
            double pop,
            String batch,
            long id,
            String univName) {
        int buddyId
    }
```

Client {

psvm () {

→ Student st = new Student ("Name", 13, 29.2, "xyz", 123, "ABC");

"uniName",

}

}

→ (1) Difficult to understand code.
(2) Prone to error.

Student st = new Student (null, 0.0, "Abc", null).

Student { ↓ // N attributes

Student (String name, double psp) :

Student (String name)

Student (name, psp, grad Year)

Student (String uniName, double psp)

same signature

2^N

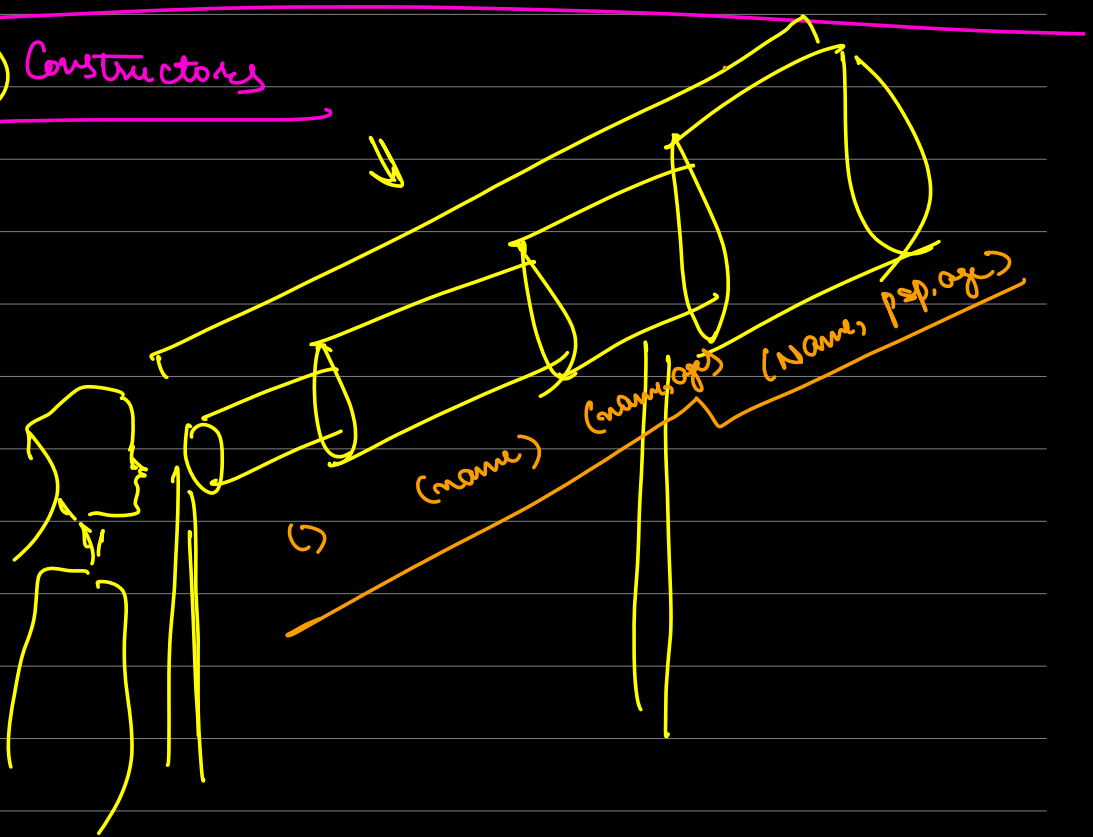
}

Problems with having multiple constructors

① Too many constructors

② Sometimes even impossible to create constructors

Telescoping Constructors




```
Student {

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

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

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

→ Telescoping constructors should be avoided

Student {

Student (one Param) {

what to pass as attributes ?

Some Data Structure that can allow to pass multiple values, and each having a specific name

Map

// validations

}

}

Parameter list

name: Naman

age: 21

pop: 19-

Univ: XYZ

Student {

"Name":
"age":
"psp": "hello"

Student (Map<String, Object> map) {

(String)

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

Runtime exception good

//validations.

}

}

Class / Struct

⇒ Something which is like a map (it allows to have diff values within it, each recognized by a diff name)

⇒ Should have compile time check for key

map.name = ✗

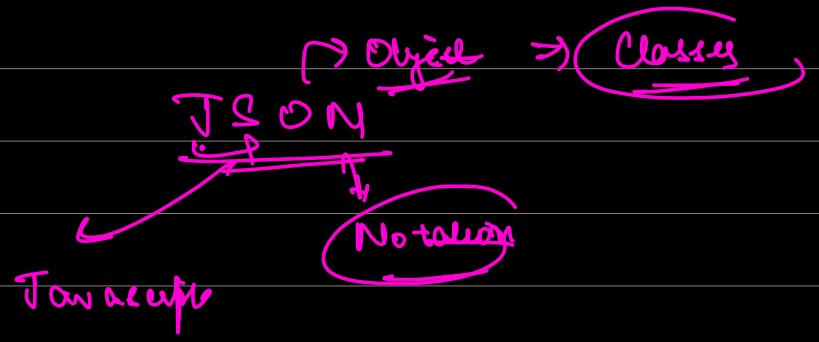
⇒ Should have compile time check for data type of values.

map.age = "Hello" ✗

map.age = Int

List ✗

HashMap ✓




```
Class Helper {
    String name;
    int age;
    double psp;
    String unitName;
}
```

helper.name = "hello" ✗

helper.age = "hi" ✗

```
class Student {
    String name;
    int age;
    double psp;
    String univName;
    String batch;
    long id;
    Integer gradYear;
    String phoneNumber;

    Student(Helper helper) {
        //validations → throw Exception
                      → else Okay
    }
}
```

```
class Builder {
    String name;
    int age;
    double psp;
    String univName;
    String batch;
    long id;
    Integer gradYear;
    String phoneNumber;
}
```

```
Helper helper = new Helper();
helper.setName(Name)
helper.setAge(>=)
helper.setPsp(89)
helper.setUnivName()
```

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

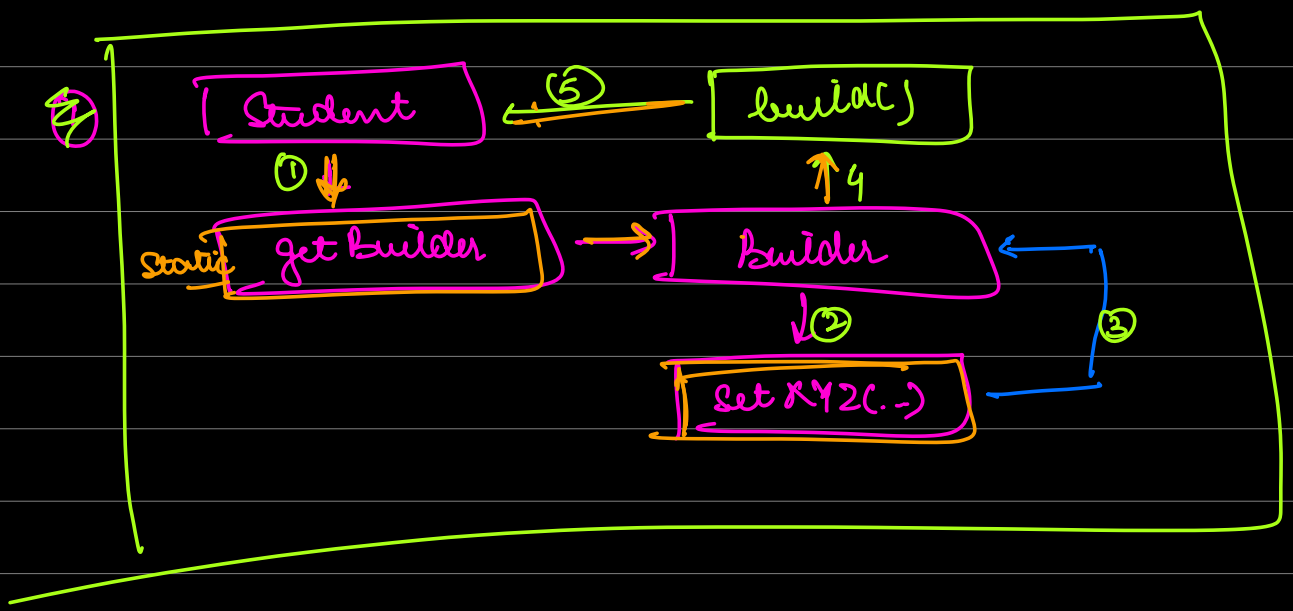
```
Student (Helper helper) {

    if (helper.gradYear > 2022)
        throw new Exception (——);
    this.gradYear = helper.gradYear;
    this.name = helper.name
}
```

Builder Design Pattern : Allows you to create an object when we have follow Constraints:

1. Too many attributes
2. Validate before creating an object.

Builder Design Pattern

1. Class with many attributes
2. Need to validate params before creating an object of that
3. Immutable Class : all params need to be passed at creation
   ⇒ Scope of error while create



Next Class

① Prototype → Class Not Required

② Factory

③

Prototype ⇒ Ruby

Decorator ⇒ Pizza

Assgn

① Implement Builder in your language

② Read about Builder

① Refactoring

② Source Making

③ Read API Ref of Fluent to see where builders are used

Disadv :

① Extra Class

Adapter

Strategy

Factory

Singleton