DATE : __/__/____
PAGE : _____

Builder Design Pattern

Creational Design Pattern

Problem:

```
Public class Student {
        int rollNumber;  // Mandatory
        int age;
        String name;
        String fatherName;
        String MotherName;
        List<String> subjects;
        String MobileNumber;
```

// optional

```
Public Student(int rollNum, int age, String name, String
        fatherName, String MotherName, List<String>
        subjects, String MobileNum)
{
    this.rollNumber = rollNumber;
    this.age = age;
    this.name = name;
    this.fatherName = fatherName;
    this.MotherName = MotherName;
    this.subjects = subjects;
    this.mobileNumber = MobileNumber
}
}
```

DATE :__/__/__
PAGE :

P1: If the no of optional field increases the constructor will become very big and will have big constructor parameter list.

P2:- Instead of creating one large / big Constructor we can make multiple constructors as per the requirement

Ex:
```
Public Student(int rollnumber, int age)
{
    this.rollNumber = rollnumber;
    this.age = age;
}

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

Public Student(int rollNumber, int age, String name, String fatherName)
{
    this.rollNumber = rollNumber;
    this.age = age
    this.name = name;
    this.fatherName = fatherName;
}
```

Here the problem is that the number of constructor can become huge.

If we again try to create a constructor

Student (int String String)

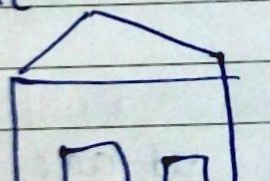
then it will through compilation error as signature of two constructors become same.

Solution of the Above Problems:
→ Builder Design Pattern.

- Builder Design Pattern.

It is a creational design Pattern or step by step object creation.

Example:



House obj
(Big House)

(House divided into steps)

⇒ Roof
window
Door
Wall

Step by step

⇒ AddRoof — (II)
Add Wall — (I)
Add Door — (III)
Add Window — (IV)
Build()
↓
Return Actual House

DATE : __/__/__
PAGE : _____

Example: StringBuilder in java

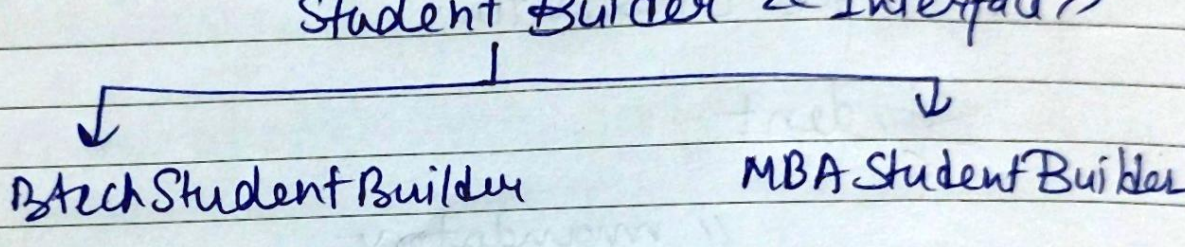
It has certain methods like append(), deleteCharAt(), delete(), etc.
And each of this object is returning a certain intermediatory builder object....

{ Intermediatory builder object is builder obj itself.

In Builder Design Pattern we are not creating a constructor with all parameters and/or with multiple constructor.

```
Student
{
        // mandatory
        // optional
      Student(Builder obj)
      {
          this.roll_no = Obj.roll_num;
      }
}
class Student
{
    int Roll;
    int age;
    String name;
    String fathername;
    String motherName;
```

```
Public Student (StudentBuilder builder)
{
    this.roll_no = builder.roll_no
    this.name = builder.name
        ;
        .
}
Public String toString()
{
    return " ";
}
};
```



- Then we have Director which will decide the sequence in which the builder setter functions will work

- Then we have client to create object of either BtechStudent or MBAStudent.

DATE :__/__/__

Layeres of Builder Design Pattern

Layer 1: Student Builder

Layer 2: BtechStudentBuilder / MBAStudentBuilder

Layer 3: Director (Setter of attributes) with Build
↑ methods

Layer 4: client
- Director → tells the sequence in which the setter methods will work
- client → It tells which object has to create

Builder Vs Decorator

1. Builder is not Dynamic	1. Decorator is a Dynamic design Pattern
2. Builder is a creational design Pattern	2. Decorator is a structural Design Pattern

Layer 1: PizzaBuilder
Layer 2: Base + cheese Base + Mushr
Layer 3: Director
Layer 4: client

What if client asked for Base + cheese + mushroom

Builder design Pattern will fail here as it is not dynamic design.