Today's agenda

↳ Builder design Pattern

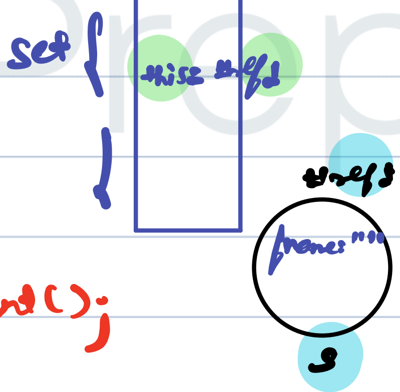
① Create an object of a class with a lot of attr.

```
class Student {
                fName
                lName
    Private     age
                weight
                CollegeYr
                Salary
                  :
                  :
}
```

create getter & setter

Sol^n :

```
//Setter method
public int set(fnam) {
        this.fname = fname;
}
```



```
Student s = new Student();
s.setfName();
```

① The attributes are immutable.

↳ if getter & setter are Public, I can always Change the value of attributes.

Soln2

↳ Create a Parametrized Cons of the class. Pass the values for each attributes.

```
class Student {                     Student ( fName, lName, age, weight,
                                        collegeyr, Salary . . . - - ) {
          fName   → 2                 this.fName = fName;
          lName   → 2                 this.lName : lName;
Private   age     → 2                       ,
          weight  → 2                       ,
          collegeyr → 2                     ,
          Salary  →                  }
            :
            :
            ,
}
```

Student s = new Student ("Subhesh", " ", 24, 80, 4,)

↳ Not understandle as well as bug Prone.

→ Not all attributes are mandatory.

↳ I will create all the Possible Combination of Constructors.

2^6 Possible Constructors

```
class Student {
    String fName → 2
    String lName → 2
    Private int age → 2
    int weight → 2
    collegeyr → 2
    Salary →
    :
    |
}
```

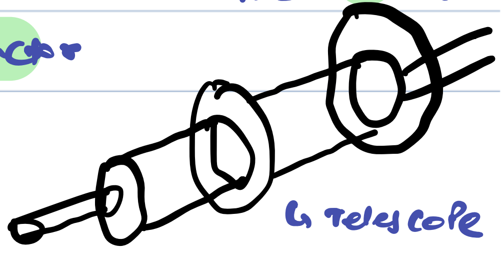
```
Student (String fName, int age){
    this.fName = fName;
    this.age = age;
}
```

```
Student (String lName, int weight){
    this.lName = lName;
    this.weight = weight;
}
```

→ Too many Possible combination of Constructors.
→ Not all Constructors may be Possible to Create.

```
Student (fName, age){ this.fName=fName
                      this.age=age }

Student (fName, age, weight){ this.fName=fName;
                              this.age=age }
                              this.weight=weight;

Student (fName, lName, age, weight){        }
                              this.fName=fName;
                              this.age=age
                              this.weight=weight;
                              this.lName=lName.
```

Code duplication?!
↓
Telescoping Constructor



```
Student (fName, age) { this.fName : fName
                       this.age : age }

Student (fName, age, weight) { this (fName, age);
                               this.weight : weight; }

Student (fName, lName, age, weight) { this (fName, age, weight);
                                      this.lName : lName; }
```

→ Ideally there should be 1 constructor

```
class Student {
        String fName
        String lName
Private int age
        int weight
            collegeyr
            Salary
            :
            :
            :
}
```

↓ any type can be placed

```
                    Map<String, Object> val
Student (              ) {
    if (val.containsKey(fName)) {
        this.fName = (String) val.get(fName);
    }
    if (val.containsKey(age)) {
        this.age = (int) val.get(age)
    }
}
```

↳ find some data structure that allows having multiple values within it.

Map<String Object>

key	value
fName	Sudhesh
age	24
weight	80
:	:

$\boxed{\text{aeg-24}}$

→ Client {

```
    map <      > {
        .Put (fName, Subhesh);
        .Put (aeg, 24);
    }
}
```

Tough to debug.

→ you might map number (in form String) to a String type

(fName, 131)

fName = "131"

↳ This issue should be caught.

v2

Final Sol^n

```
Class helper {
    String fName
    String lName
?: →    int age
        int weight
            collegeyr
             Salary
               :
               ı
}
```

```
Student (helper val){
    this.fName = val.fName;
    this.lName = val.lName;
        !
        .
        !
}
```

```
Client {
    helper h = new helper();
        h.fName = "Subhesh";
        h.age   =   24;
        h.weight =  80;

    Student s = new Student(h);
}
```

h.fName = 131??

↓
caught at compile time.

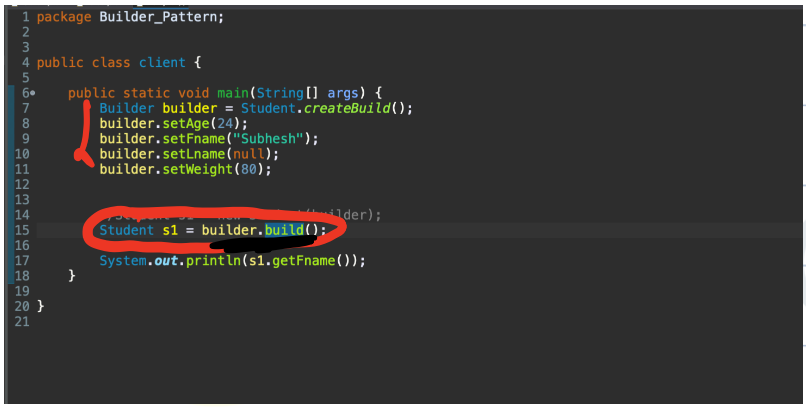
Break till 9:23 Pm

↳ The helper class doesn't need to be immutable.

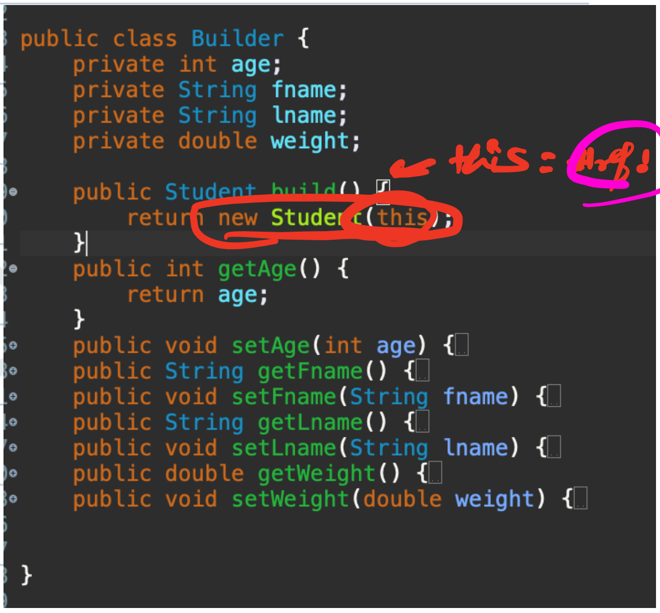
→ when to use Builder design pattern.

↳ Class with a lot of attributes.

↳ immutable attributes in class.

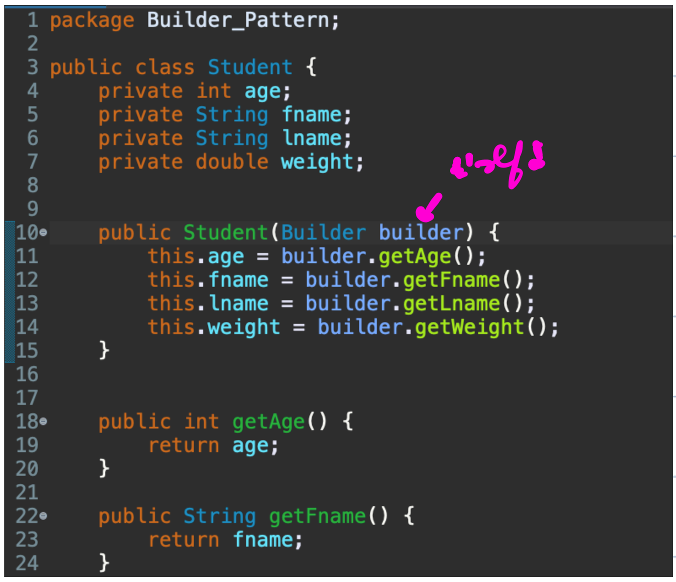


```java
package Builder_Pattern;


public class client {

    public static void main(String[] args) {
        Builder builder = Student.createBuild();
        builder.setAge(24);
        builder.setFname("Subhesh");
        builder.setLname(null);
        builder.setWeight(80);


        //Student s1 = new Student(builder);
        Student s1 = builder.build();

        System.out.println(s1.getFname());
    }

}
```



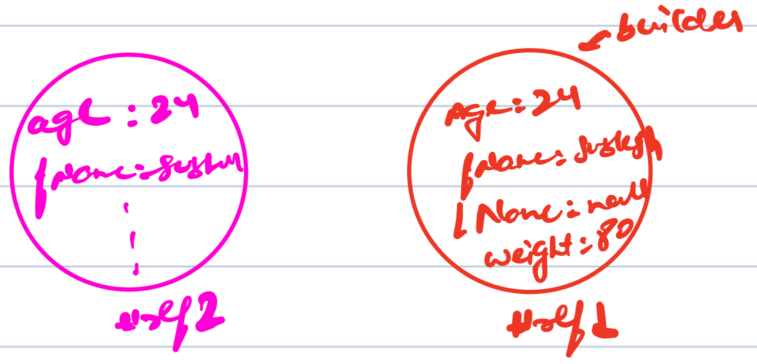
```java
public class Builder {
    private int age;
    private String fname;
    private String lname;
    private double weight;

    public Student build() {
        return new Student(this);
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
    public String getFname() {
    public void setFname(String fname) {
    public String getLname() {
    public void setLname(String lname) {
    public double getWeight() {
    public void setWeight(double weight) {


}
```

this = #ref1

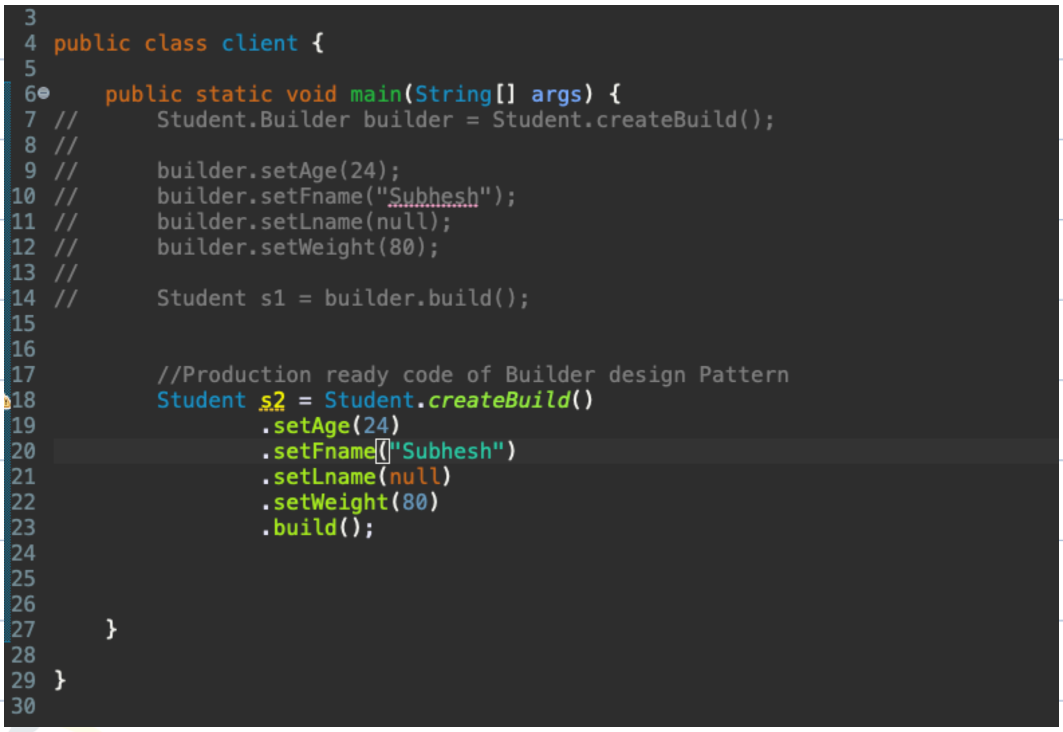


```java
package Builder_Pattern;

public class Student {
    private int age;
    private String fname;
    private String lname;
    private double weight;


    public Student(Builder builder) {
        this.age = builder.getAge();
        this.fname = builder.getFname();
        this.lname = builder.getLname();
        this.weight = builder.getWeight();
    }


    public int getAge() {
        return age;
    }

    public String getFname() {
        return fname;
    }
```

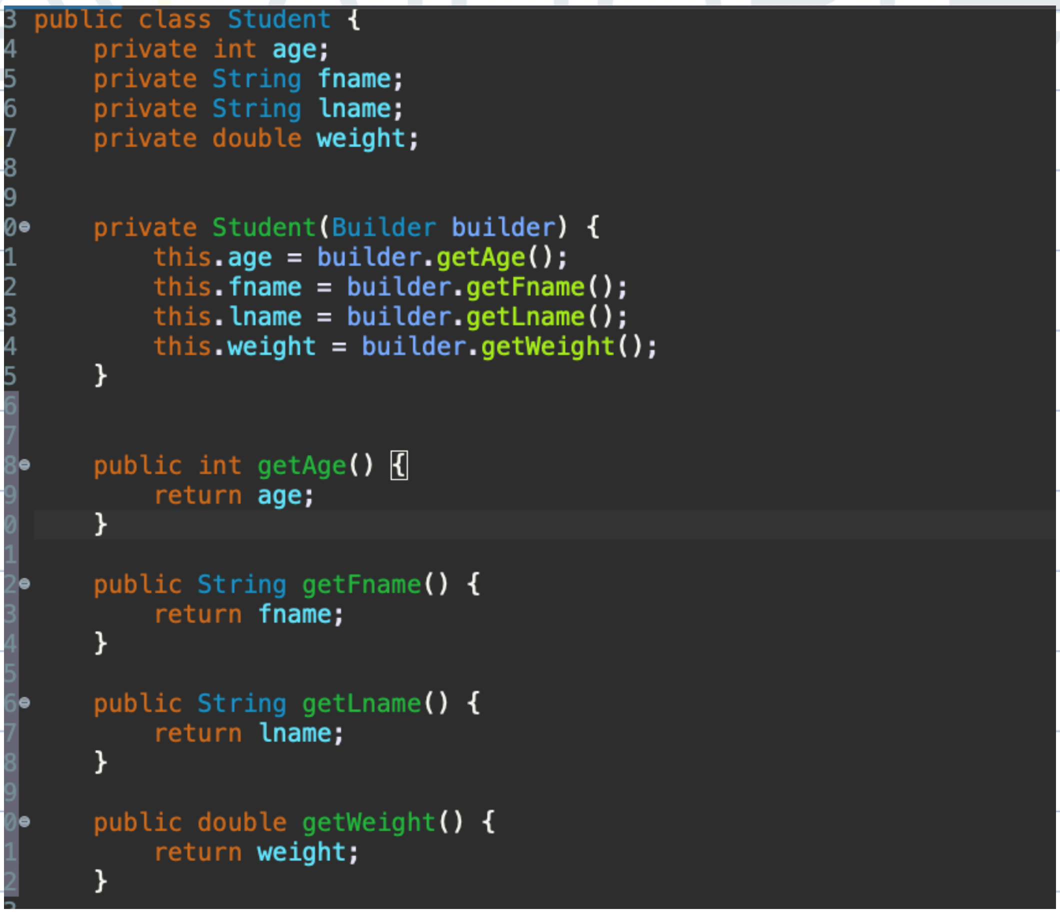
#ref1



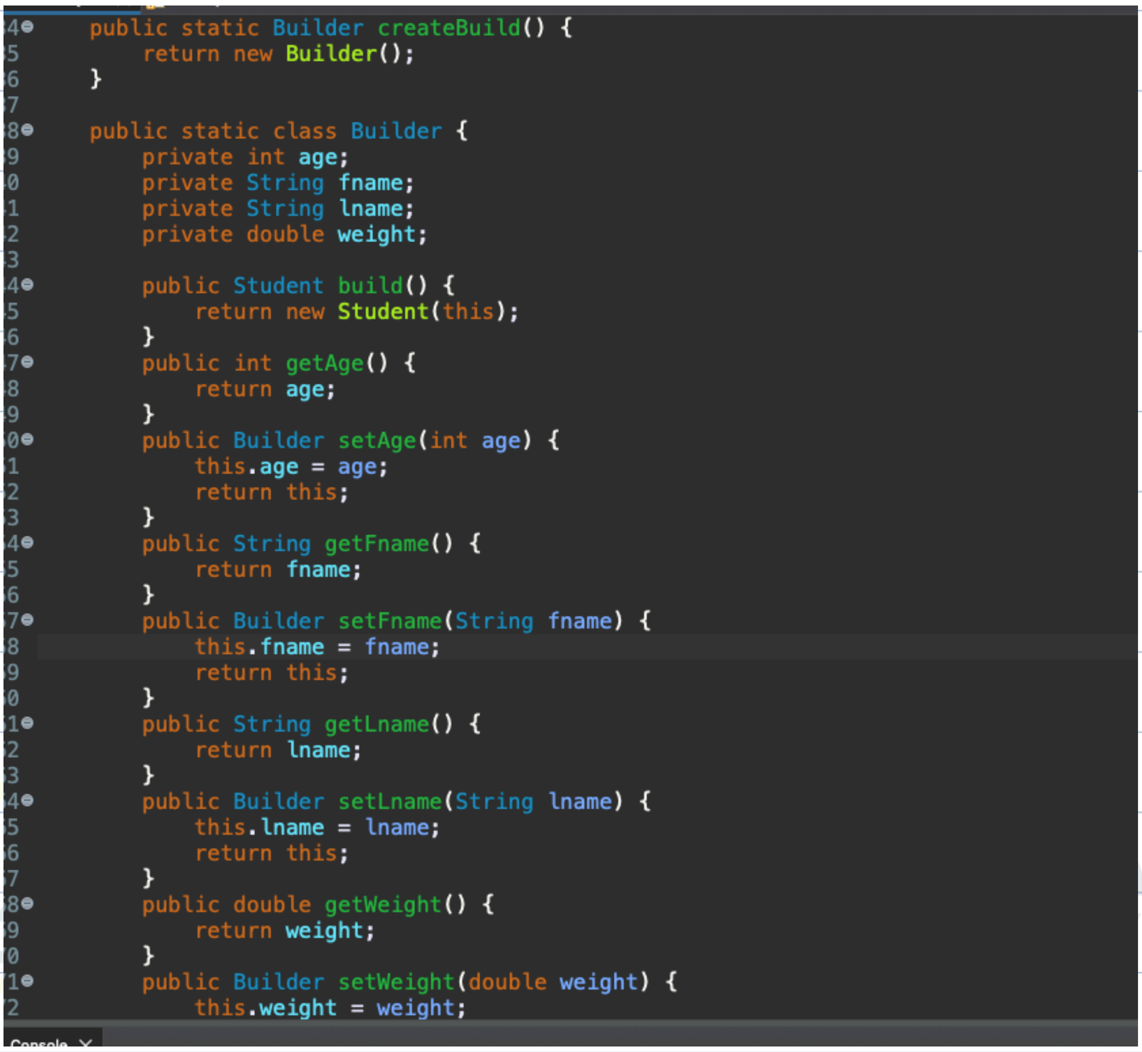
Final Soln



```java
public class client {

    public static void main(String[] args) {
//      Student.Builder builder = Student.createBuild();
//
//      builder.setAge(24);
//      builder.setFname("Subhesh");
//      builder.setLname(null);
//      builder.setWeight(80);
//
//      Student s1 = builder.build();


        //Production ready code of Builder design Pattern
        Student s2 = Student.createBuild()
                .setAge(24)
                .setFname("Subhesh")
                .setLname(null)
                .setWeight(80)
                .build();



    }

}
```



```java
public class Student {
    private int age;
    private String fname;
    private String lname;
    private double weight;


    private Student(Builder builder) {
        this.age = builder.getAge();
        this.fname = builder.getFname();
        this.lname = builder.getLname();
        this.weight = builder.getWeight();
    }


    public int getAge() {
        return age;
    }

    public String getFname() {
        return fname;
    }

    public String getLname() {
        return lname;
    }

    public double getWeight() {
        return weight;
    }
```



```
    public static Builder createBuild() {
        return new Builder();
    }

    public static class Builder {
        private int age;
        private String fname;
        private String lname;
        private double weight;

        public Student build() {
            return new Student(this);
        }
        public int getAge() {
            return age;
        }
        public Builder setAge(int age) {
            this.age = age;
            return this;
        }
        public String getFname() {
            return fname;
        }
        public Builder setFname(String fname) {
            this.fname = fname;
            return this;
        }
        public String getLname() {
            return lname;
        }
        public Builder setLname(String lname) {
            this.lname = lname;
            return this;
        }
        public double getWeight() {
            return weight;
        }
        public Builder setWeight(double weight) {
            this.weight = weight;
```