OOPS -> object oriented programming ---> Inheritance, abstraction and polymorphism

-> While writing classes, in bigger code bases we might face some readability and maintainability challenges,

and we need to figure out some nice and innovative ways to implement our classes in a cleaner way.

↓

Design patterns

Q -> We want to create a User class, with multiple attributes like firstName, lastName, age, email, password etc. We also need to add a bunch of validation as well that might include a combination of 2 or more than 2 attributes.

```
class User {

    String firstName;
    String lastName;
    int age;
    String email;
    String password;
    ....
    ....

}
```

→ All properties have the visibility public

```
u = new User();
```

```
u.firstName = "";
```

→ This is now possible, but may be we dont want an empty firstName thats our validation.

If we keep things public we can't enforce validations.

```
class User {

    String firstName;
    String lastName;
    int age;
    String email;
    String password;
    ....
    ....

    void setFirstName(String x) {
        if(x == null or x.length() == 0) {
            throw ...
        }
        this.firstName = x;

    }

}
```

→ These are now private properties.

↓

Now no one can modify the properties outside of the class.

People won't be able to even access the property outside the class.

↓

We can introduce getter and setter functions.

```
u = new User();
```

→ Object is already created

↓

u.setFirstName("Sanket");

Because the object was created using default constructor till the time we don't manually call all the relevant setters this object is invalid as it doesn't go through any validations.

What if we put all the validation checks in the constructor call, that means we make a custom constructor where we have all the validation in place (apart from getter setter) so that if any validation fails, we will not allow object creation.

```
class User {

    String firstName;
    String lastName;
    int age;
    String email;
    String password;
    ....
    ....

    User(String firstName, String lastName, int age, String email .....) {
        if(firstName == null or firstName.length() == 0) {
            throw

        }

          ...'
            ....

    }

    void setFirstName(String x) {
        if(x == null or x.length() == 0) {
            throw ...
        }
        this.firstName = x;

    }

}
```

So many attributes passed to the constructor

Now we need to remeber the order of the params

Adding new params is also a headache

Client code will be less readable

You have to manually pass null for those params which are not reqd always during obj creation.

n -> params

2^n constructor

firstName, lastName, email

firstName, lastName, password

```
class X {
    x, y, a, b

    fun(String x, String y) {}

    fun(String a, String b) {}

}
```

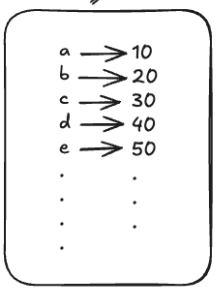
order also matters

```
constructor (a, b, c, d, e, f ....) {


}
```

(10, 20, 30, 40, 50, ...)

DS ?



should be able to
club or group values
together

no constraint on order

HashMap, unordered_map, dict, object

unordered set of key-value pairs

```
class User {

    String firstName;
    String lastName;
    int age;
    String email;
    String password;
    ....
    ....

    User(builder) {
        this.firstName = builder.firstName;
        this.age = builder.age;
        this.email = builder.email
            ...'

    }

    void setFirstName(String x) {
        if(x == null or x.length() == 0) {
            throw ...
        }
        this.firstName = x;

    }

}
```

if some how, someone creates
this key value pair based DS obj
for us, we can leverage it and make
a very clean class.

Very clean user constrcutor

Q -> User constructor is public
so someone can call it with a null

we make it private

```
class Builder {

    pvt firstName;
    pvt age;
    pvt email;
    ...


    Builder() {}


    void setAge(x) {
        if(this.x < 0) return;
        this.age = x;
```

This will contain all the properties
of our main User class

it will create a temporary object before
we create our final User.

This builder is not going to user a
bloated constructor for obj creation

Instead, we are going to use
setter based validations in the builder class.

Now becoz, our builder setter have the validation
User constructor doesn't need to have
any validation.

```
        }

    }


    b = new Builder();
    b.setFirstName("Sanket");
    b.setLastName("Singh");
    b.setAge(27);
    .....

    u = new User(b);


class Builder {

    pvt firstName;
    pvt age;
    pvt email;
    ...


    Builder() {}


    void setAge(x) {
        if(this.x < 0) return;
        this.age = x;

    }


    ....
    ....

    User build() {
        return new User(this);
    }


}


    b = new Builder();
    b.setFirstName("Sanket");
    b.setLastName("Singh");
    b.setAge(27);
    .....

    u = b.build();


class Builder {

    pvt firstName;
    pvt age;
    pvt email;
    ...


    Builder() {}


    Builder age(x) {
        if(this.x < 0) return;
        this.age = x;
        return this;

    }

    Builder lastName(y) {
        ...
        return this
    }


    ....
```

we can do chaining now

```
    ....

    User build() {
        return new User(this);
    }


}
```

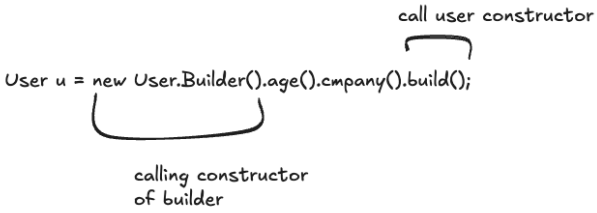
```
    b = new Builder();
    u = b
        .age(10)
        .firstName("Sanket")
        .lastName("Singh")
        .build();
```

```
class User {

    ...
    ...


    static class Builder {
        ....
    }


}
```



Builder pattern.....