4 Builder Design Pattern

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

In simple words:
- Buid an object step by step.
- Avoid huge constructors.
- make object creation readable and safe.

Why Builder:

→ Telescopic constructors.

Eg: User user = new User(
"Antony", "ajoy@gmail.com", 25, "India", "Developer",
true, false, true);

→ hard to read
→ error prone (wrong parameter order).
→ many optional fields.

Builder solves the problem of complex object construction with many optional parameters.

Core idea of Builder:-

Instead of passing everything at once.

we
- Build the object incrementally.
- call only what we need.
- create final obj explicitly.

```
Class User{
    Private String name;
    Private String email;
    Private int age;
    Private Sting country;

    Private User( UserBuilder builder){
        this.name = builder.name;
        this.email = builder.email;
        this.age = builder.age;
        this.country = builder.country;
    }
}
//Builder class.
Class UserBuilder{
    String name;
    String email;
    int age;
    Sting country;
    UserBuilder setName (String name){
        this.name = name; return this;
    }
    UserBuilder setEmail (String email){
        this.email = email; return this;
    }
    UserBuilder setAge (~~String~~ int age){
        this.age = age return this;
    }
    UserBuilder setCountry (String country){
        this.country = country; return this;
    }
    User build(){
        return new User(this);
    }
}
```

Client code

```
User user = new UserBuilder()
            .setName("Antony")
            .setAge(25)
            .build();
```

It is readable, safe, Flexible.

Eg:- Burger Analogy.

- choose bun.
- Add patty
- Add cheese.
- Add sauce
- Finally → build burger

Factory → decides which obj to create.
Builder → decides how to build an obj

Builder patterns simplifies the creation of complex objects by constructing them step by step.