Day 08 Design Pattern : Builder

① Builder
- Motivation
- Alternatives
- Implementation
- Out of the box implementation.

Database

```
class Database {
        String host;
        int port;
        String name;
        Long id;
        String username;
        String password;
enum ← DatabaseType type;
}
```

mysql :: | <host> : port.

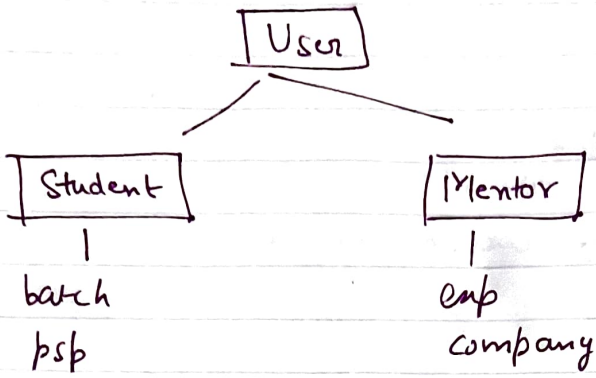
① create an instance using constructor — ctor.

```
new Database("localhost", "4453", "dev", 1, "+", "(")
```

↳ Not maintainable.

↳ Database (username, password)
- username → String
- password → String

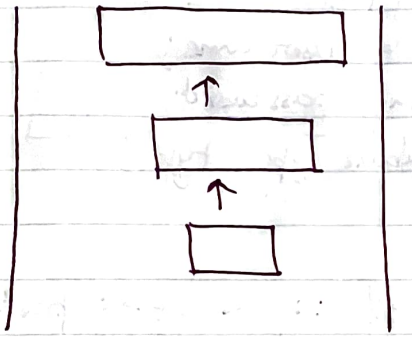
D (password, username)



② Constructors overloaded.

↳ anti - pattern.

- Telescoping pattern.



② Database (....) {

Validate ( )

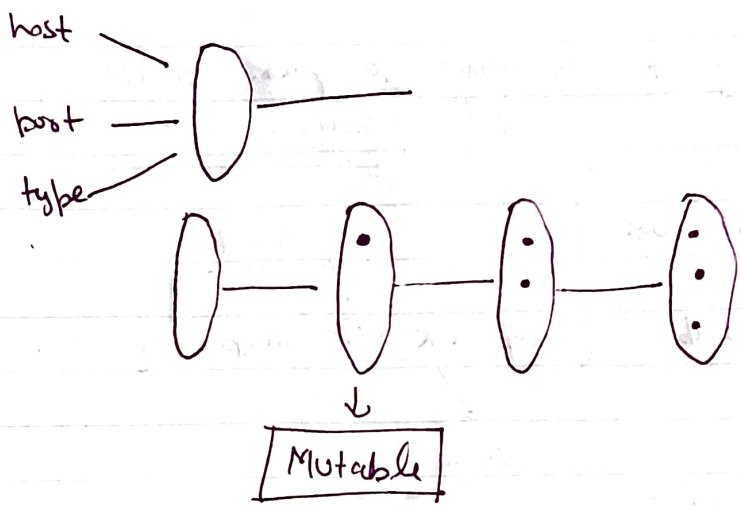
} ↓

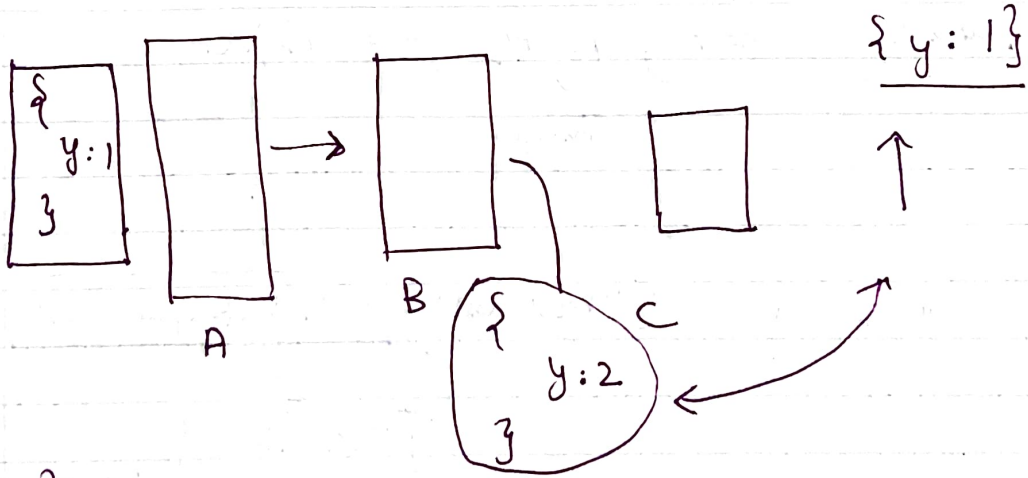
throw an error

② ① Default constructor → new Database();

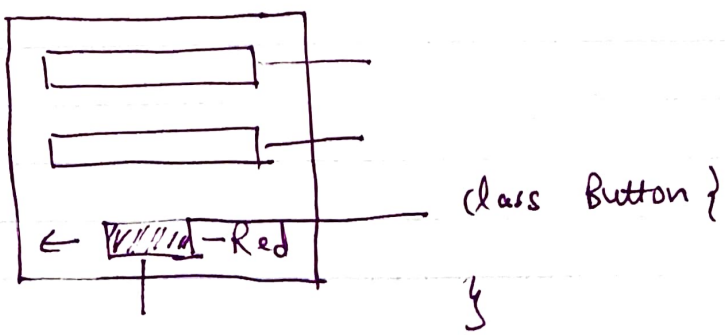
② set values using → setters

```
db.setHost("local");
db.setPort(3356);
db.setName("devdb");
```





React



Immutable → do not modify objects

① Constructor → not maintainable
→ Validation.

② default constructor + setter
→ Immutable

Controller → we have to pass a lot of value / args.

---

class Database {

Database (Map<K,V> map) { }
↑ object
↙
String

Database (String host, port) → object
↑ String
↓
Integer.

new Database ("localhost", "Vishakapatnam");
↙
Exception

+ class → Type safe.
+ Immutable → No side effects
+ Validate → before constructor

} → Builder

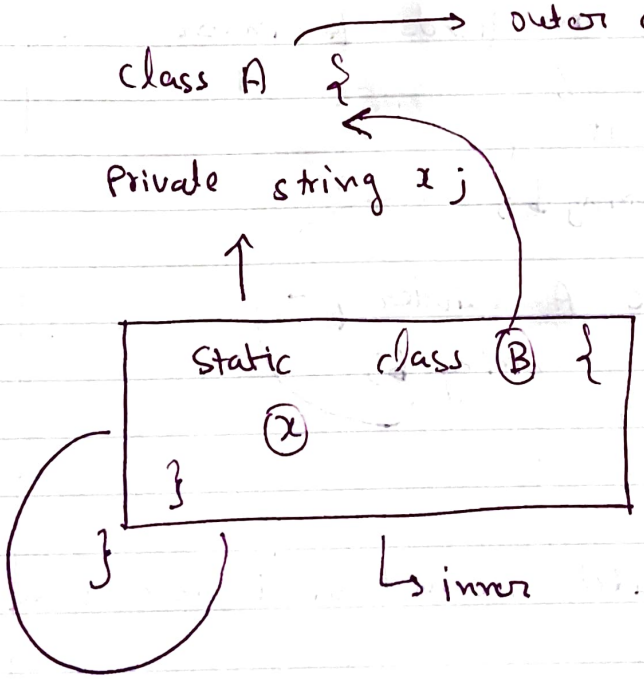
+ on demand →

Static class

↳ only nested static class.

↓

inner class



```
class Database {
    Private Database () {} ——①
    Public static class DatabaseBuilder () {
    }
}
```

Step 1 – Hide the constructor of outer.

Step 2 – Create a static inner class.

Step 3 – Copy all outer fields to inner class.

```
class A {
    string b;
}
```

⟹

```
static A Builder {
    string b;
}
```

Step 4 → Add a build method in inner class.

```
Static Builder {
    Bl;
    build () {
        A = A();
        A.b = b;
    return A;
    }
}.
```