AGENDA

1.) Prototype design Pattern

2.) Registry design Pattern

start by 9:05 PM IST

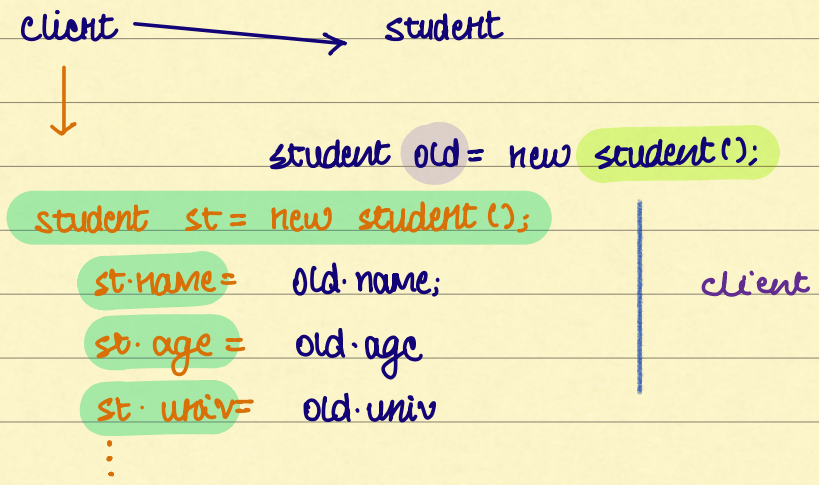
*) PROTOTYPE D.P.

Problem- Given an Object, we Need to create copy of the same.

Ways:

I) Naive way -

copy elements one by one



*) PROBLEMS :

1.) forces client to know all details of class.
(all attributes should be accessible)
→ Not always possible

2.) if some Private attributes → would not be visible on client.

3.)
**

student original = new student();
is intelligent = new intelligentstudent();

if we have child of student:

```
intelligentstudent extends student {
        iq;
}
```

In client: we would need if-else checks before creating object

•) client cannot automatically determine –
new student() OR new intelligentstudent();

II:> COPY CONSTRUCTOR

```
Student scopy = new student(original)
```

↓

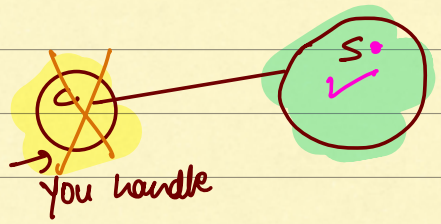
same problem.

original → student OR intelligentstudent ?

client code:

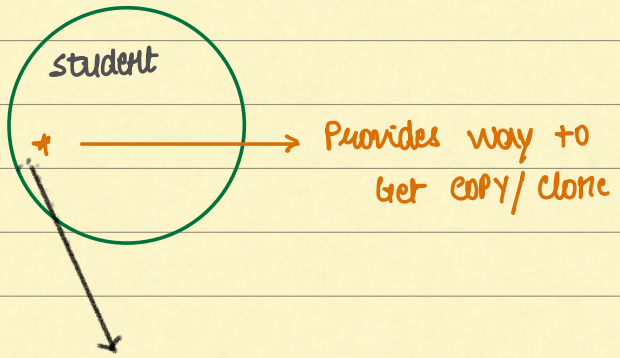
```
if (org instance of student)
    copy = new student();
else if (org instance of intstudent)
    copy = new intelligentstudent();
```

VIOLATES → OCP / SRP.



Learning: creating copy of any Object → client Not Good way.

Ideal solution :

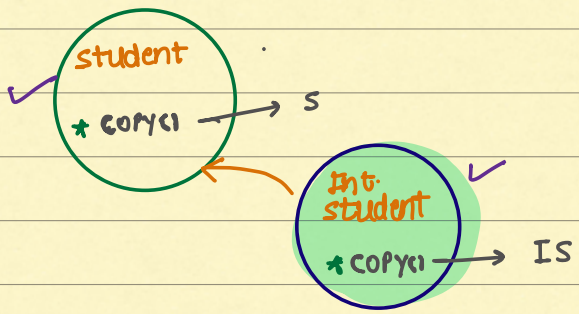


student scopy = org.copy(); (✓) better way.

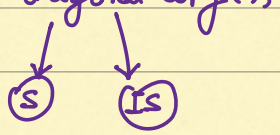
Advantages :

① No tight coupling b/w client & Object
② No violation of OCP

HOW WOULD CLICK WORK:



student s =
original.copy();



student org = new student();
OR

Student org = new IntStudent();

Student scopy = org.copy();

*) IMPLEMENTATION:

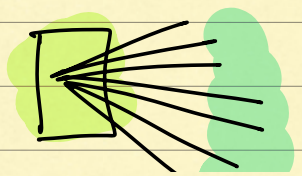
```
class Student {

    copy(Student s) {
        // copy attributes
        return std;
    }
```

```
class IntelligentStudent extends Student {

    IS
    copy() {        (overrides)
        ----
        ----
    }
}
```

What if child class doesn't override copy() ?



Definition:

Prototype - reference / B.P.

Eg: classmate Notebook

WHY special ??

1. Last Page
2. front Page

} special

work for classmate client → you are writing code for s/w.

10k copies (Print)

Each copy → have New Object (A4 size) / 120

Notebook

- Pg count ✓
- price ✓
- type ✓
- size ✓
- brand ✓
- front Pg cover ✗
- back Pg cover ✗

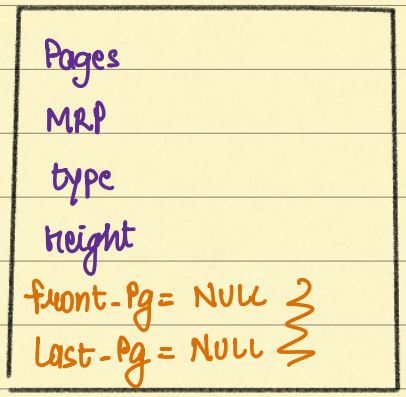
90 % | 10 %

Now → we want to create 10k Notebooks of A4 size / 120 Pg / 70 INR.

Attributes that Remain same →

A1: you create & set all values by self

A2: copy prototype & update unique values



create a Prototype

=> When code Runs → you Just create copy

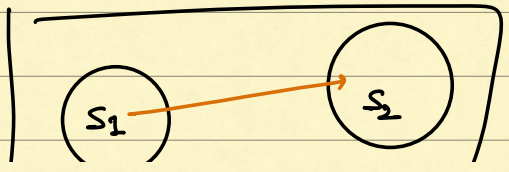
=> Append extra attributes

DONE !!

PROTOTYPE WORKS ON Problem statement:

whenever we want to create Multiple objects of same type + creating them is heavey Operation

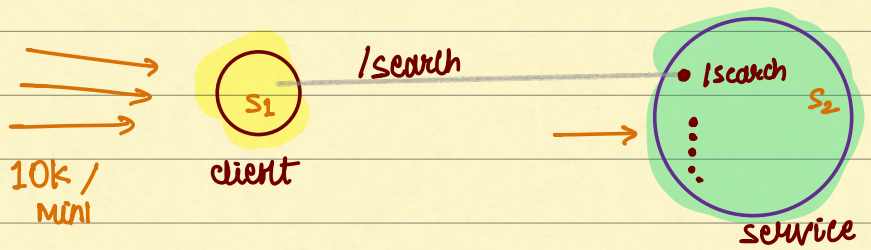
↳ Get a copy of already created object



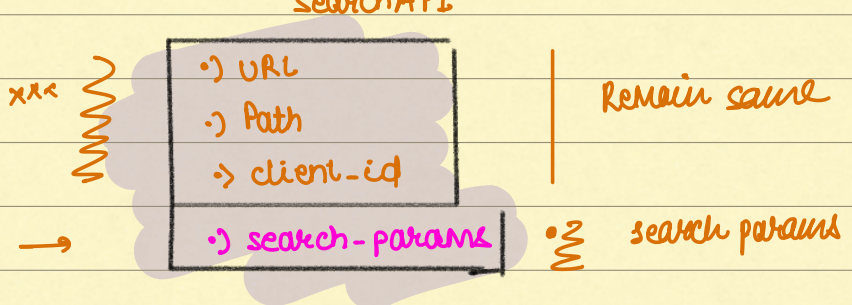
*) PRACTICAL USECASE:

searchQuery

- Many APIs in your service
- one of those is search API /search (GET)



In client → you always create object of searchAPI



Now; client calling service → (2 possible ways)

(A1) ① create same object again & again → call

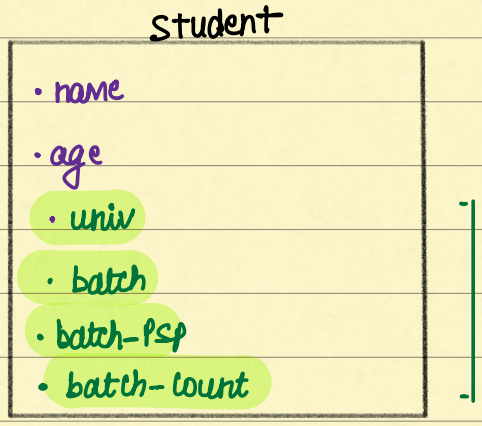
(A2) ② Get copy of already created object ⊕ Append data → call

(Assignments in college)

original should Not be updated.

Another Example:

Scaler Application



→ we want to create 1000 objects.

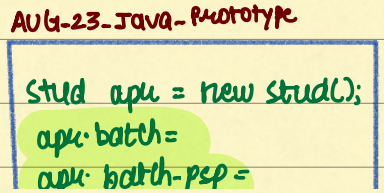
SIMPLE:

-> Get copy

-> Append extra value

NOW: where this ORIGINAL object is stored ??

Assume object is stored in some file of code called REGISTRY



→ sets all values

apr .univ =

beforehand.

ON CLIENT:

```
Student x= registry.get(AUG-23-Java).copy()
x.age = 21
x.name = "yosh"
```

Advantage:

1.) Object creation code re-used

2.) code more cleaner

Q.) Can I have more than 1 classes added in Registry ??

*) STEPS:

1.) declare a method copy() / clone()
in class

2.) This method creates copy of object
(child MUST override)

3.) store Prototypes in Registry

REGISTRY:

Generally a class which has
Mapping for Object to tag(name)

```
class studentRegistry {


  void register(key, student)
 student get(key)

 }
```

data structure ??

```
class studentRegistry {
    Map<String, student> m = new...

    void register(key, student)
    student get(key)

}
```

CLIENT FLOW:

1.) Get Registry by Name
2.) call clone()
3.) Append extra attributes

1.) Prototype

2.) Registry

DEMO