Agenda

1. Prototype Design Pattern
2. Factory Design Pattern

} CDP

Factory Method | Abstract Factory | Practical Factory

Prototype Design Pattern (Registry Design Pattern)

Problem Statement

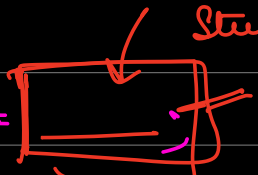
→ Given an object of a class

→ We need to create a copy of that object

(Creating a new obj with exact same attributes as original)

```
Client {
    psvm() {
        Student st = new Student();
        Student stCopy = //get copy of st
    }
}
```

Approach 1 Client

Student original = 

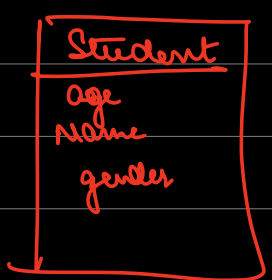
Student copy = new Student();

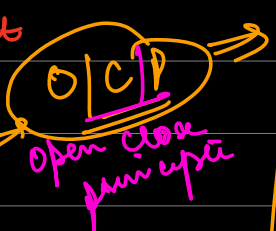
```
copy.name = original.name
copy.age = original.age
copy.email = original.email
```

~~copy = original;~~

Cons

1. Client will need to know all implementation details of Student
   → Tightly coupled.
2. Student might have private attributes that client will not be able to get





Client

```
Student original = ___;
Student copy;

if (typeof original == Student)
    copy = new Student
else if (typeof == Intelligent)
    copy = new Intelligen
```

Copy constructors

Will this work

```
Student {
    Student (Student or) {
    }
    Student () {}
}
```

```
Client
Student or = ____;
Student copy =
    new Student(or);
```

intelligent Student

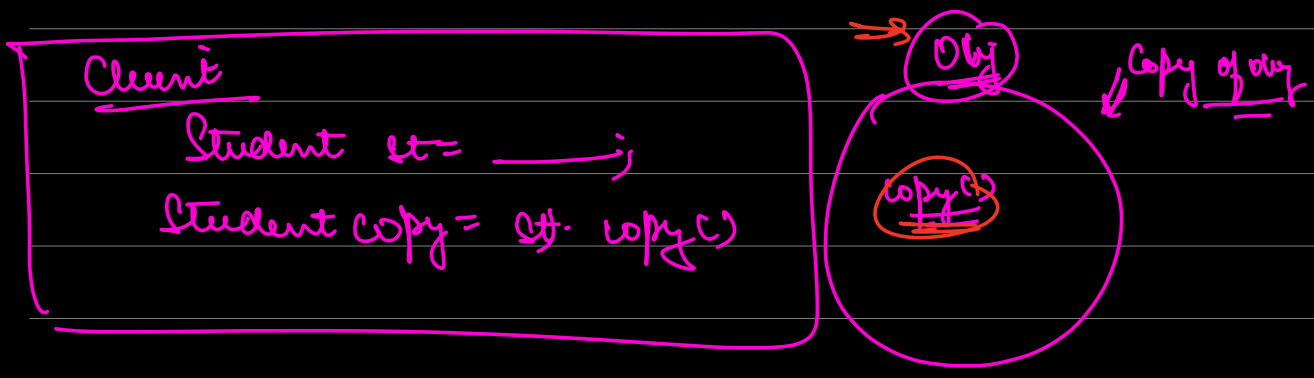
Client

```
if (original instance of Student) {
    Student copy = new Student (original)
} else if (original instance of Intel
```

```
St Copy = St
St.Name = "Name"
```

→ If a client wants to create copy of an object, having the logic to create copy within the client is prone to errors.

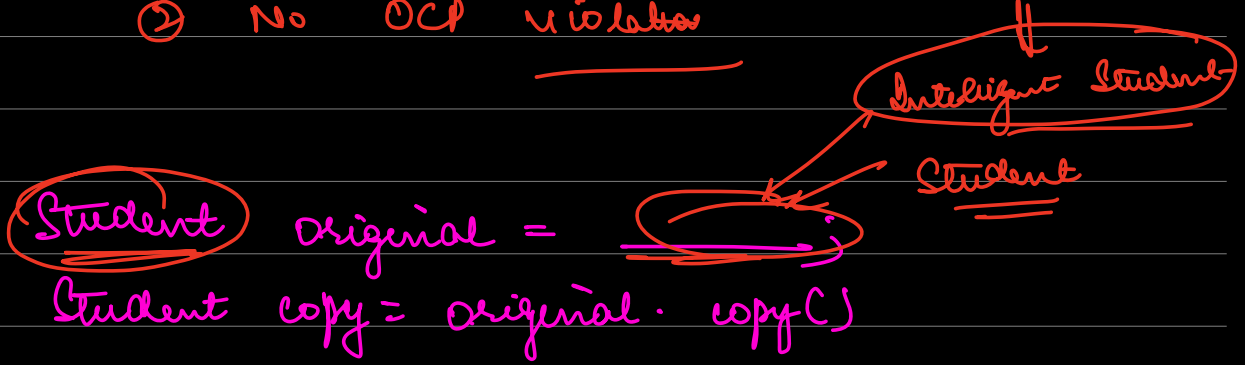
→ Ideal Sol^{n} can be that client outsources the work to create copy to the object itself.

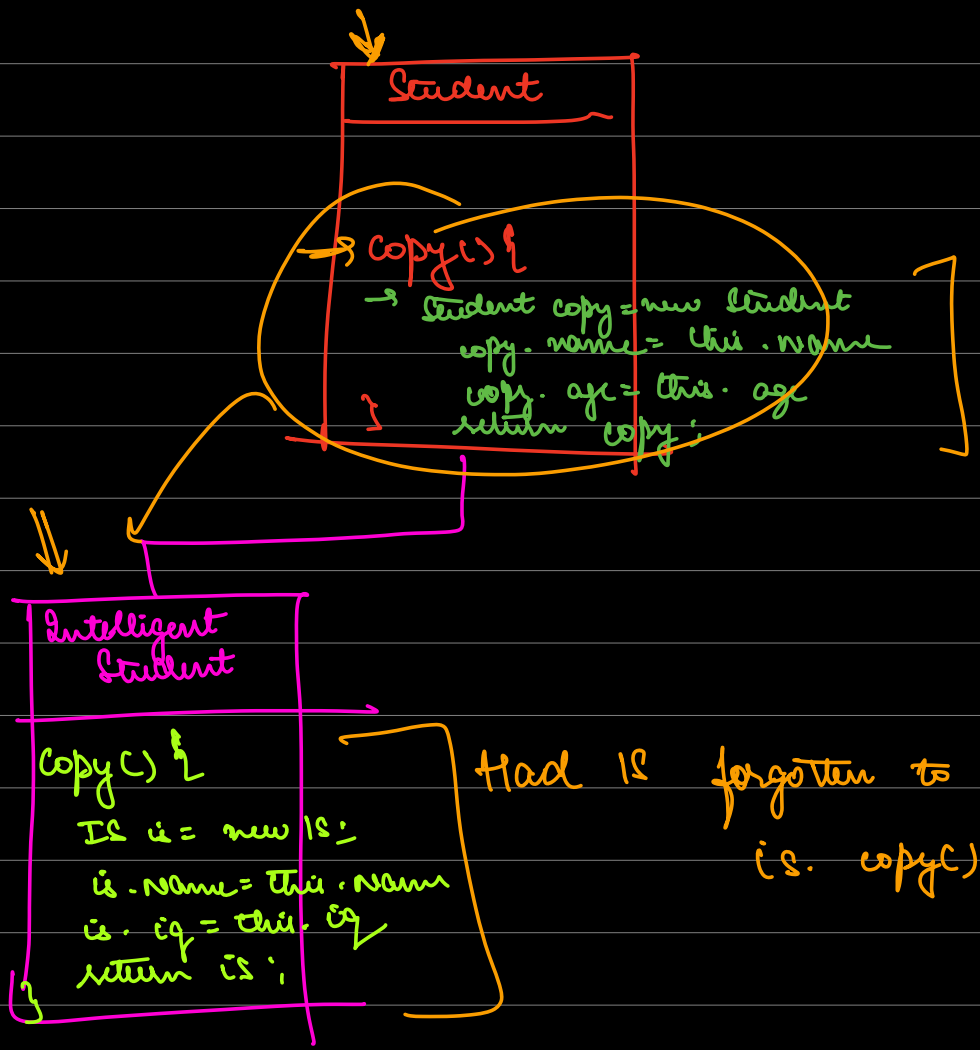


Benefits ① No tight coupling b/w client and obj class.

→ Client doesn't need to know internals of the class to be copied

② No OCP violation

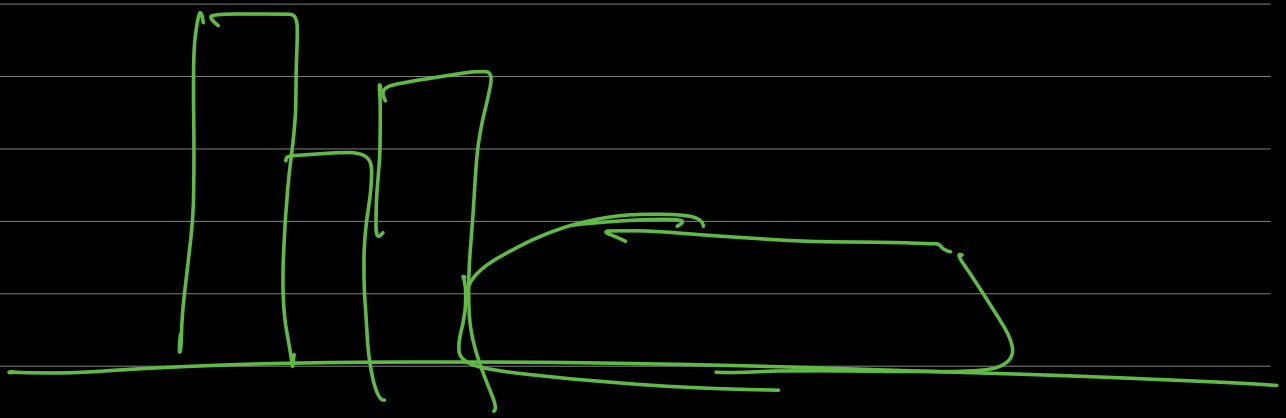




Had IS forgotten to override copy (i.e. copy())

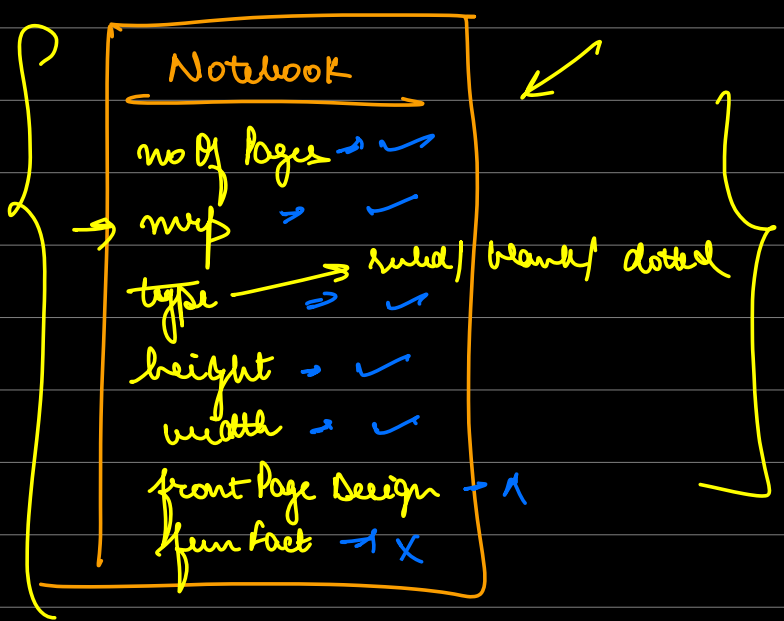
⇒ All the child classes must also override the copy method. Otherwise it can lead to unexpected results.

Prototype Design Pattern

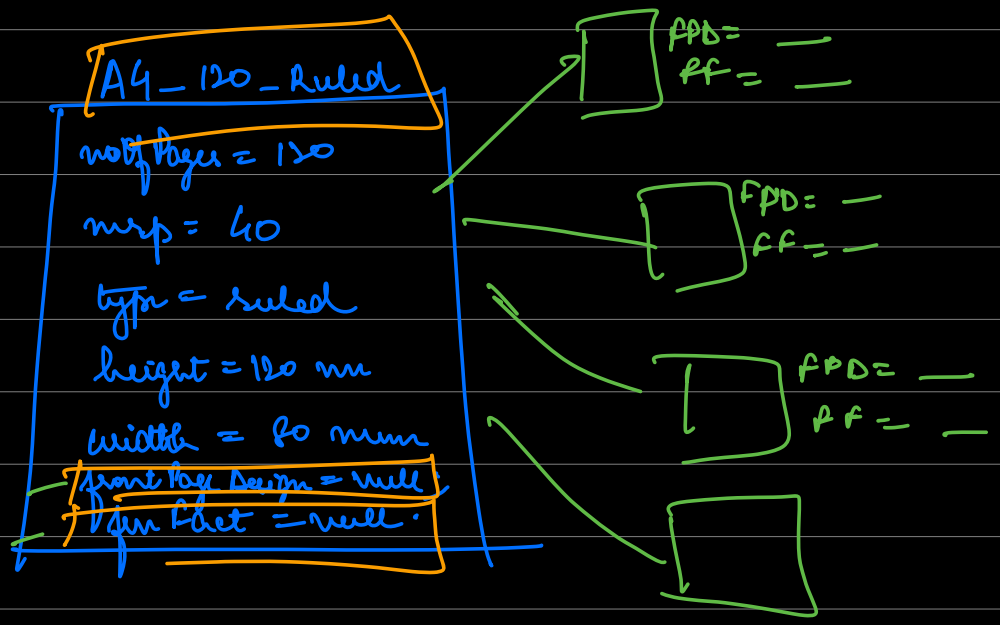


Classmate Notebook Factory

⇒ Factory Management System



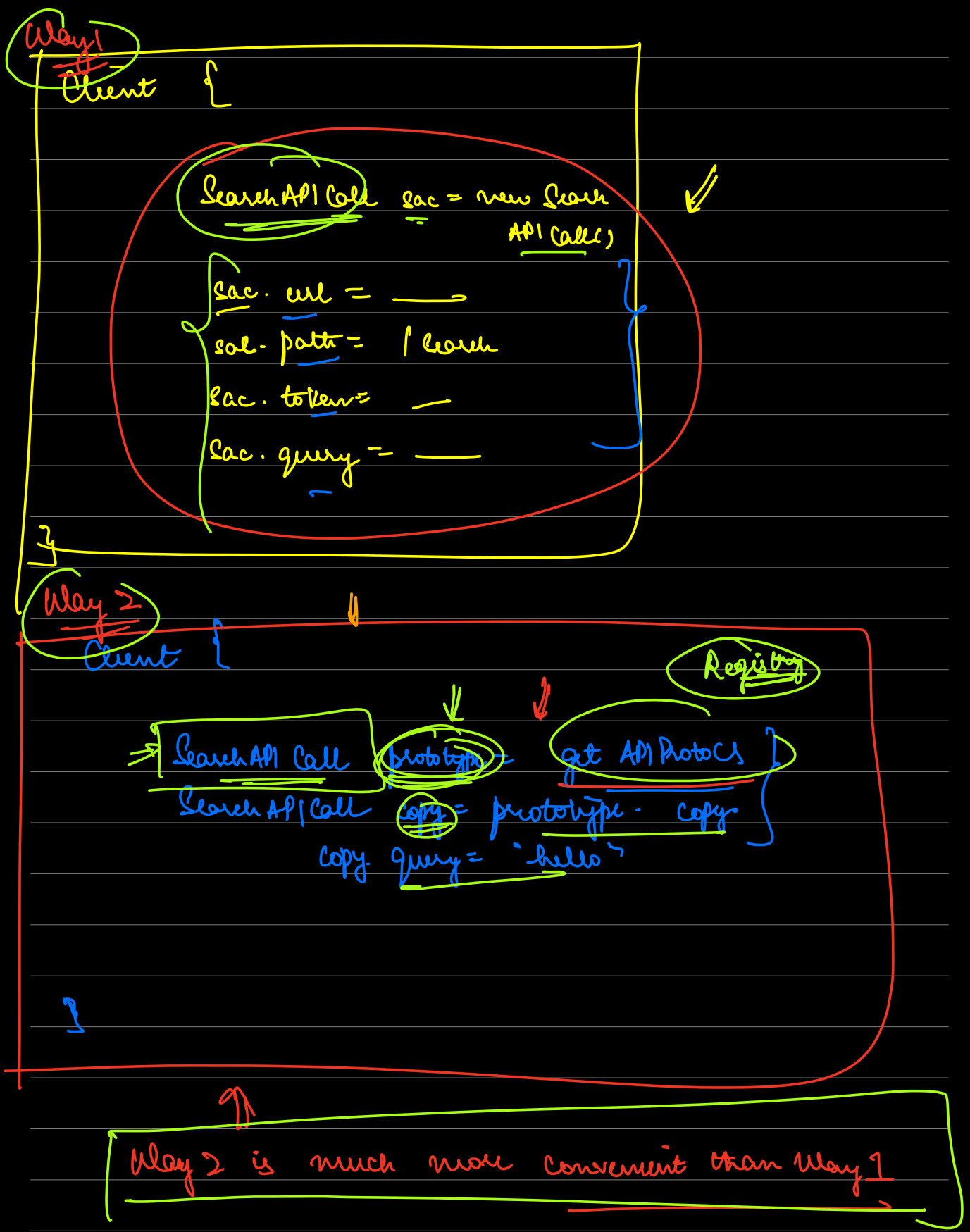
⇒ Classmate wants to create 10000 notebooks of A4 size with 120 pages that are ruled.

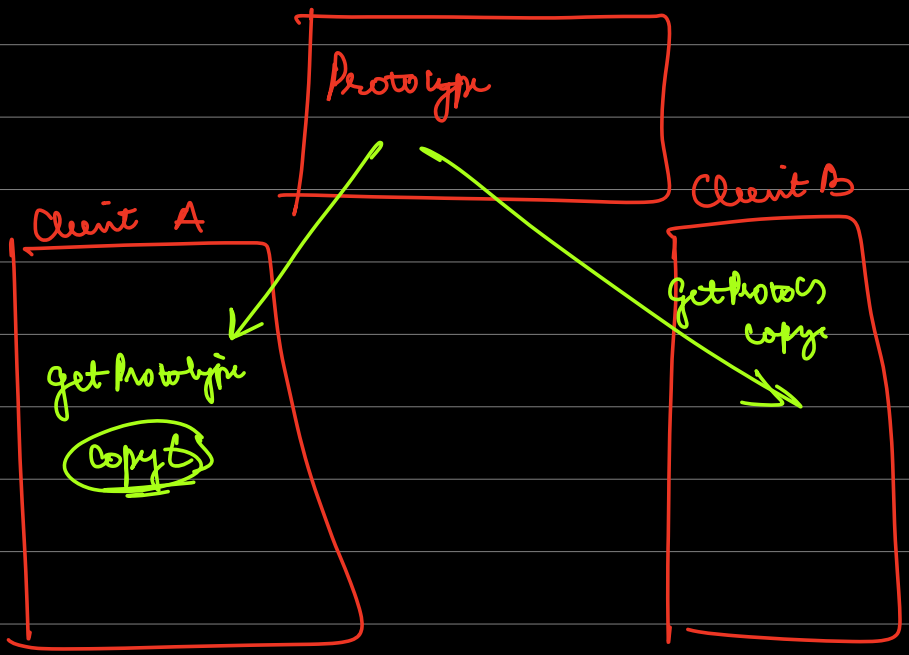


When the factory works, it creates a copy of the prototype, adds the value of frontPage and funFact and DONE!

Often there are scenarios where we create an instance of a prototype, change a few attributes and done! → Prototype acts like a template.

SearchAPICall
url = _____
path = _____
token = _____
queryString =


Way 1
Client {
SearchAPICold sac = new SearchAPICall();
sac.url =
sac.path = /search
sac.token =
sac.query =
}
Way 2
Client {
Registry
SearchAPICall
prototype =
getAPIProtocol();
SearchAPICall copy = prototype.copy
copy.query = "hello"
}
Way 2 is much more convenient than Way 1



PROTOTYPE DESIGN PATTERN

- Often there are scenarios where we don't want to create an object from scratch.
- Rather, we prefer creating a copy from the template and changing the values in that copy

Eg

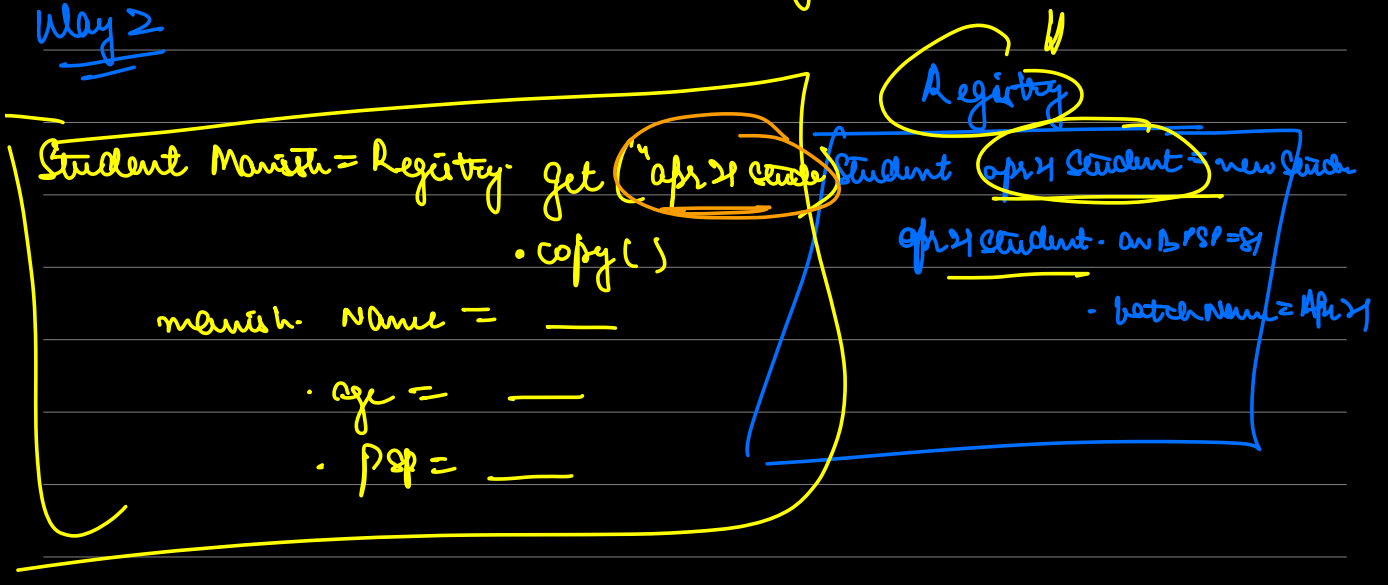
```
Class Student {
    → name
    → age
    → psp
    → averageBatchPsp
    → batchName
}
```

```
Student trishik = new Student()
trishik.name = "Trishik"
. age = 24
. psp = 75
. averageBatchPsp = 81
. batchName = "Dsa 24"
```

Way 1

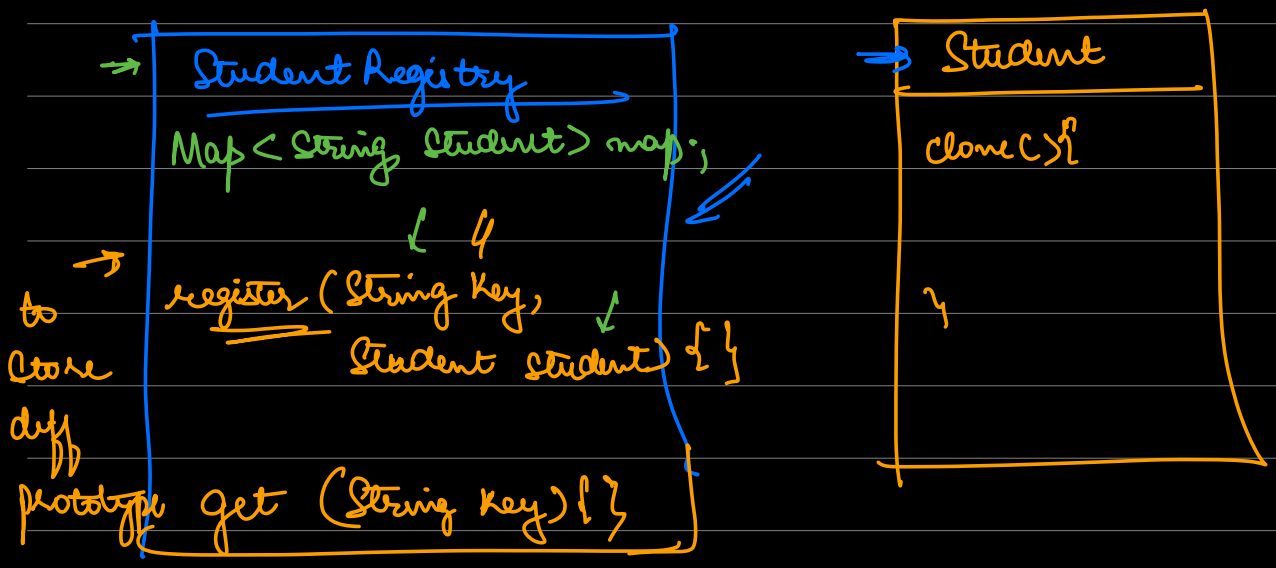
```
Student manish = new Student()
manish.name = "Manish"
manish.age = 28
      .psp = 91
      .averageBatchPsp = 84
      .batchName = "Apr 21"
```

Way 2



Step 1 : In the class that you want to create prototypes of, declare a method called as clone(). This method creates a copy of the current object. NOTE : If you have child classes, they must also override clone()

Step 2 Store the prototypes in a registry.



Step 3 : Client calls the registry to get prototype. It then creates a copy of them and does its work on the copy

⇒ Usually prototypes are stored at app^n start time

⇒ Registry can be singleton.

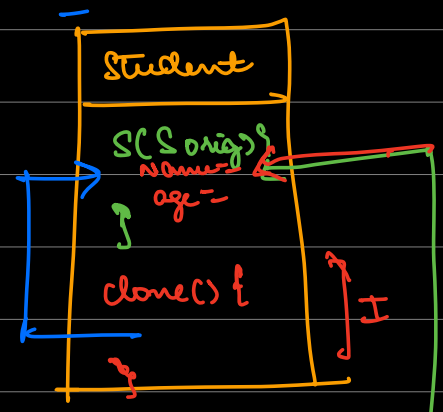
Prototype Design Pattern

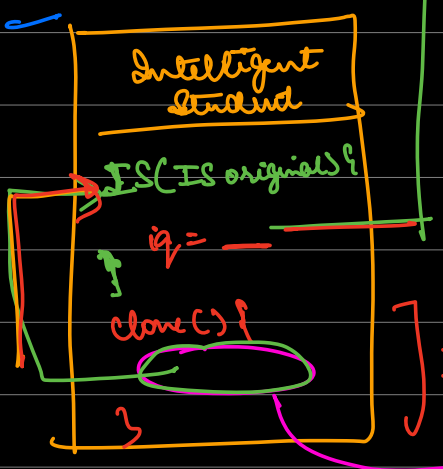
If you want to create copy of object rather than copying yourself, the object should have the responsibility to create copy

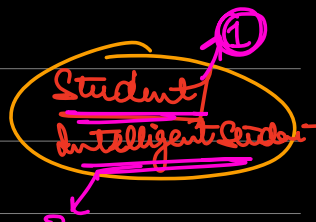
Registry Design Pattern

If you need something again and again, store those things in the registry

Spring Boot Container





Student st= 

→ We needed clone so that method of correct class gets called automatically Otherwise we will need to depend on concrete object violating DCP

→ will not have access to private properties of Student

Assgn

① Implement
Student, IS, StudentRegistry, Client

② Read about Prototype DP
from Refactoring, Source Making