: Prototype Design Pattern :-

=> Problem Statement :-

- Given an Object of a class
- Create a deep copy of that Object

↓

new obj with same attributes & values as orig obj

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

=> 2 ways :

1st way => use getter/setter or attributes to copy the value

```
Student st = new Student();
Student stCopy = new Student();
st.copy.setId(st.getId());
st.copy.setName(st.getName());
```

CONS

1) Client will need to know all implementation details of Student ⇒ prone to errors, tightly coupled

2) Some attributes inside Student might be hidden

↓

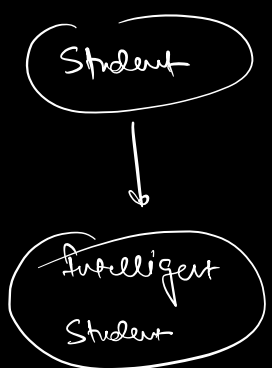
(private + no setter/getter)

2nd way

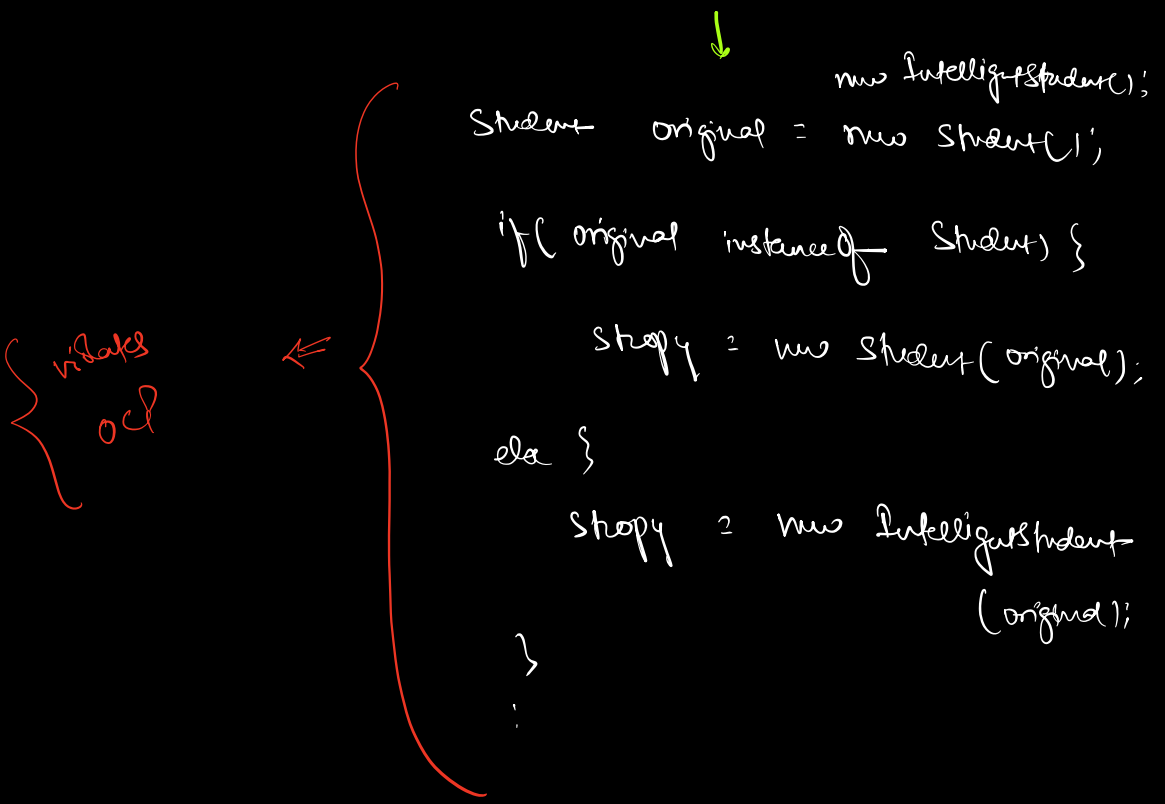
copy constructor

```
Student {

    Student() {

    }

    Student( Student stCopy ) {

    }

}
```

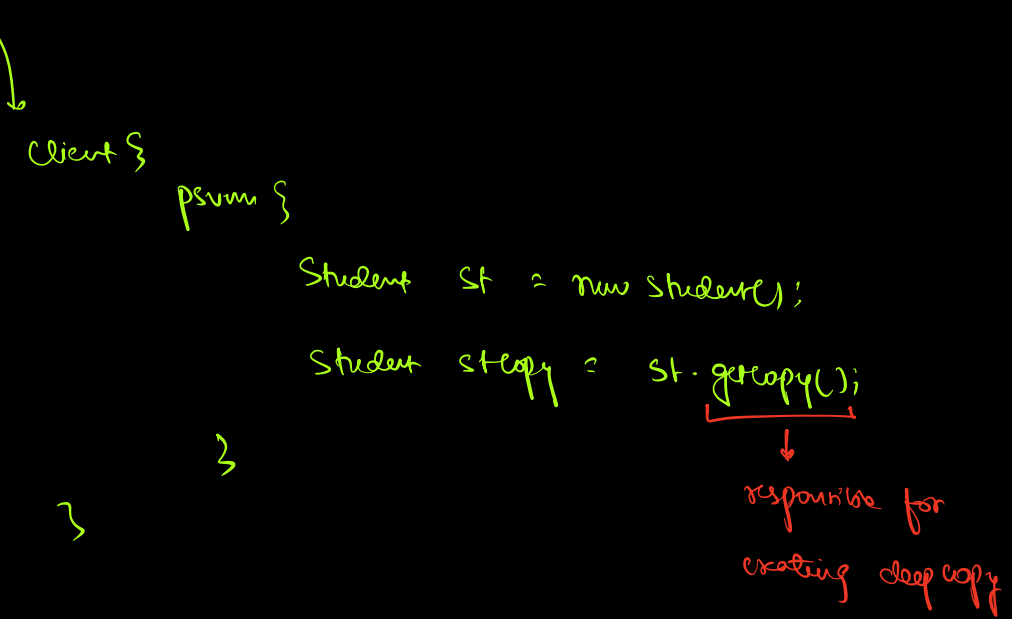
```
Client {

    psvm {

        Student st = new Student();

        Student stCopy = new Student(stCopy);

    }

}
```

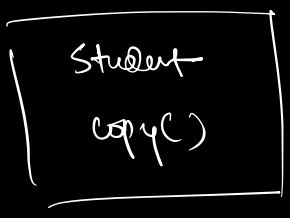



```
Client {

    psvm {

        Student st = new IntelligentStudent();

        Student stCopy = new Student(stCopy);

    }

}
```

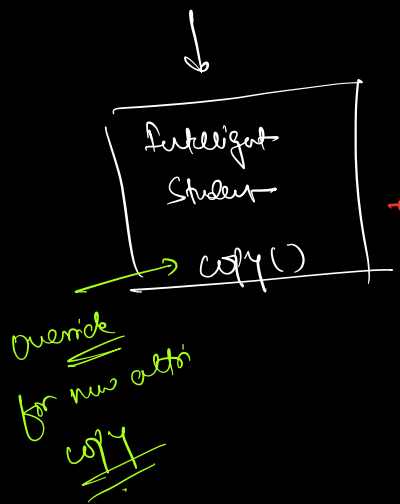


1. Using either copy constructor or copying the values in the client class using getter/setter, are prone to errors, and also violate SOLID principles.
2. Ideal Solution can be that the client outsources the work to create the copy of the object to the object itself.



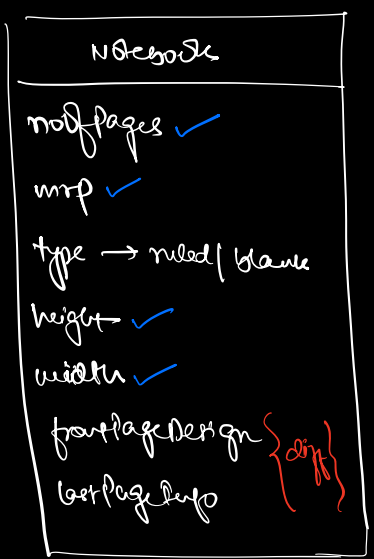


1. Not tight coupling between client and Student
2. No need to worry about the type of object, either Student or Intelligent student, or any other child of student. It follows OCP.



: Prototype Design Pattern :-

⇒ Classmate Notebook manufacturing unit

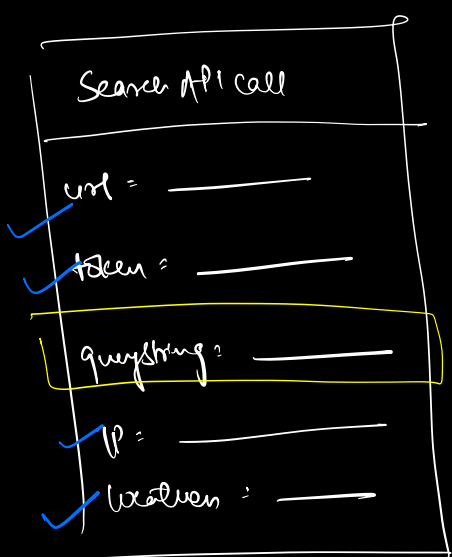


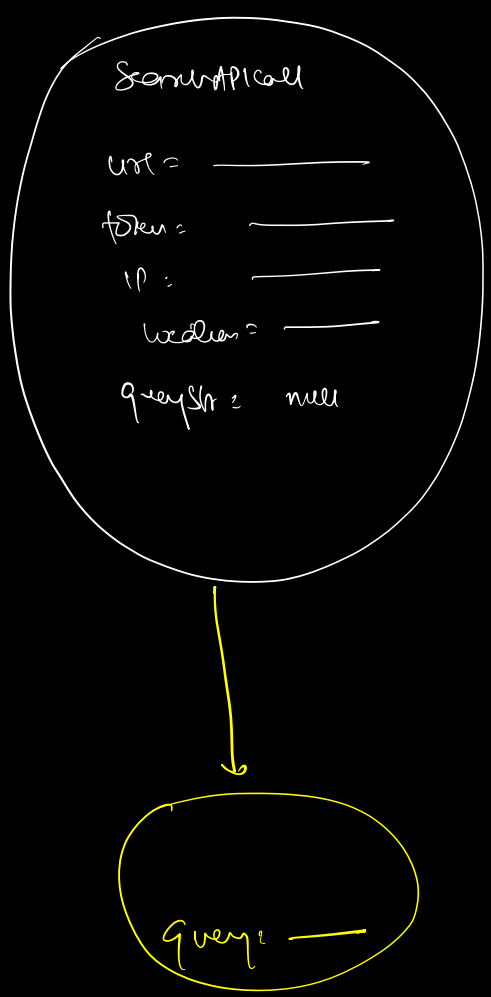
⇒ Create 10000 Notebooks

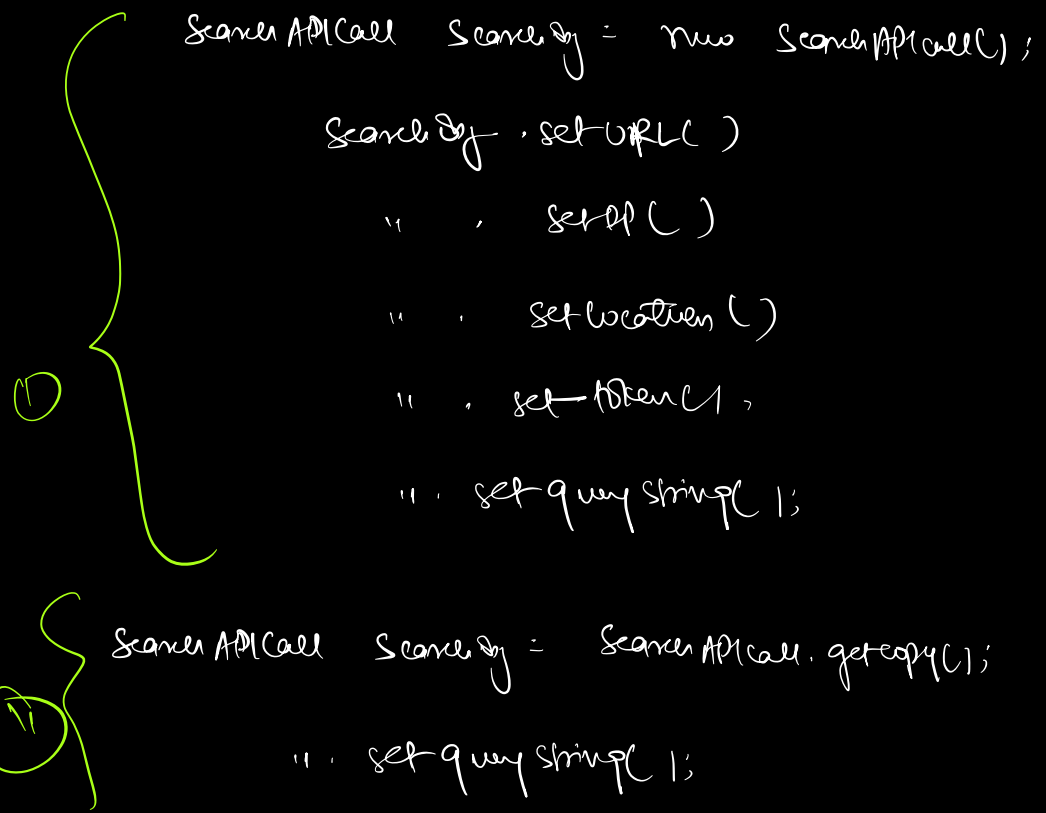
- A4
- 120 pages
- ruled

When the manufacturing starts, create the notebooks with the fixed info(size, no pages, ruling) . and then update the front page and last page.

Often times, there are scenarios, where we create an instance of a prototype, create a copy of prototype, update values and use it.



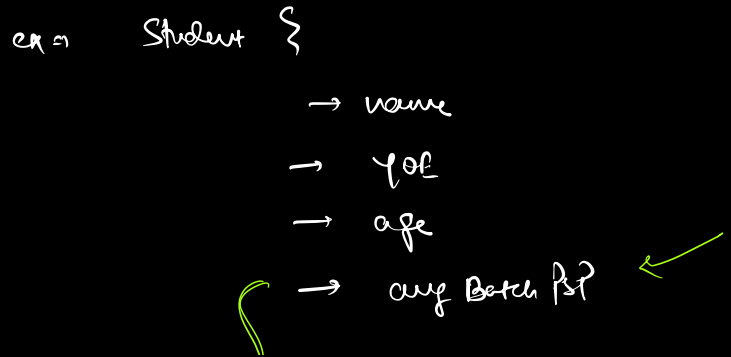




(2) is much more convenient than (1).

Often times, we don't want to create an object from scratch, we would want to create a copy of the template [prototype] and update some values on the copy of the template, and use it further.

This will save us from the creation of object and updating it's value every time it is used, leading to lesser code.



{ → batchName ←
→ YearofEnrollment ←

}

i) Create a few templates:

```
i) Student apsBatchStu = new Student();
          apsBatchStu. setAvgPsf();
             "    . setBatchName();
             "    . setYearofEnrollment();
```

ii) Insert them inside registry;

```
Register {
        Map< String, Student> map;

                 String      Student)
                   ↑          ↑
           add ( key,     value) {

              map.add( k, v);
           }

           get( key) {
                 map.get(k)
           }
}
```

III) use template:

Register.get(k).copy();

Prototype

1) when we have single or minm no. of templates, with very less use-cases we use normal prototype (without registry)

Registry

1) when we have a lot of types of templates which are being used again & again, we use registry

↓

Spring Boot