Creational

- → Singleton
- → Builder
- → Prototype.

PROTOTYPE.

Problem Statement

Given an object of a class, we need to create a copy of that object

(Creating a new object in the memory with exact same attributes)

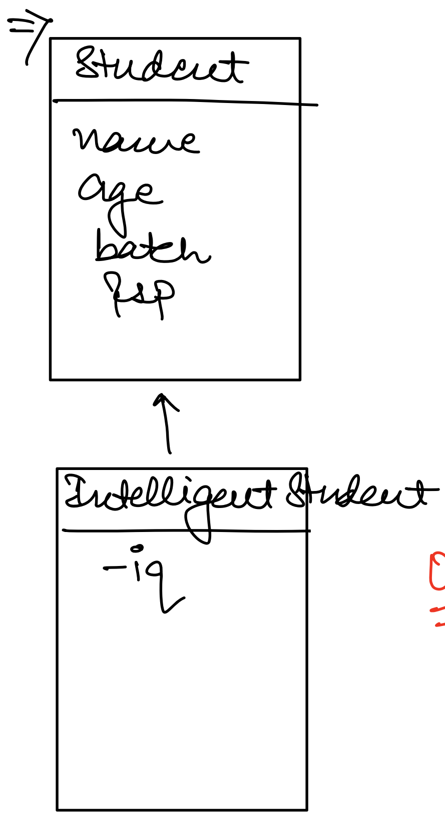
Approach 1:-

Student | Child Class.

```
Client {
    PSVM() {
        Student original = ________;
        Student Copy = ___;
        copy.name = original.name;
        copy.age = original.age;
        ___
        ___
        ___
    }
}
```

Cons.

1. Too many lines of code.
2. Client needs to know all the internal details of Student class.
   ↳ Tightly coupled
3. Student might have some private attrs then client won't be able to access those private attrs.

⇒ 

Student | IntelligentStudent

```
Student original = ______;
Student copy;

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

OCP

II Copy Constructor

```
Student {

    Student(Student st) {
        this.name = st.name;
        this.age = st.age;
        this.psp = st.psp;
        ===
    }
}
```

```
                                  Student | IntelligentStudent
                                        ↓
Student original = ______;
Student copy = new Student(original);
```

⇒ IS should also contain a copy constructor.

```
      if (original instanceof Student) {
          copy = Student(original);
OCP.  }
      else if (original instanceof IS) {
          copy = IntelligentStudent(original);
      }
```

If client wants to create a 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 to work to create copy to the object itself.

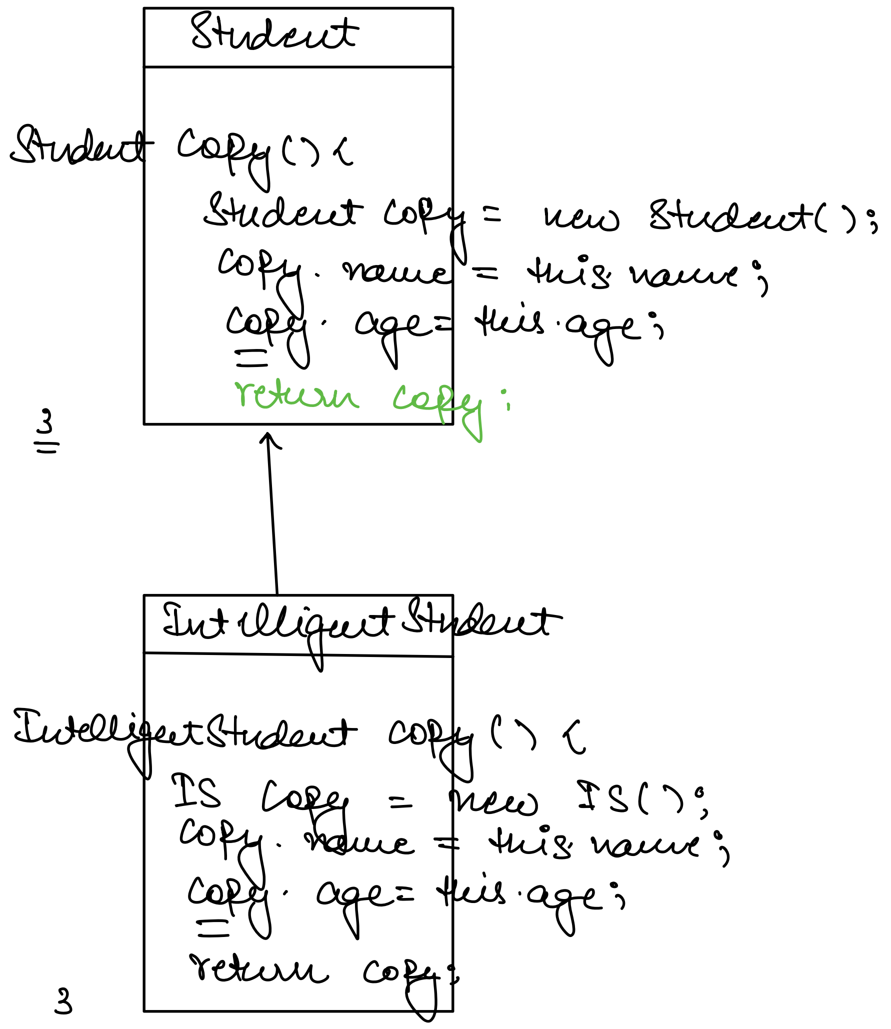
Client

```
Student original = ______;
Student copy = original.copy();
```

Pros:

1. No tight coupling b/w Student & Client class.
   ⇒ Client need not to know all the internal details about Student class.

2. No OCP Violation.



```
Student copy() {
    Student copy = new Student();
    copy. name = this. name;
    copy. age = this.age;
    return copy;
}
```

```
IntelligentStudent copy() {
    IS copy = new IS();
    copy. name = this. name;
    copy. age = this.age;
    return copy;
}
```

⇒ Student original = ________;
Student copy = original. copy()

} Runtime Polymorphism.

Note : All the child classes must override the copy method, else it can lead to inconsistent results.

Prototype Design Pattern

$\downarrow$

Demo / Template

⇒ Classmate Notebooks.

Notebook

- noOfPage ✓
- size of page ✓
- mrp ✓
- type ✓ → Blank, Lines, Ruled
- height ✓
- width ✓
- thickness ✓
- frontPage ✗
- funfact ✗

⇒ Classmate wants to create 100000 notebooks of A4 size with 120 pages of ruled type.

A4_Ruled_120	
- noOfPage	= 120
- size of page	= A4
- mrp	= 100
- type	= Ruled
- height	= —
- width	= —
- thickness	= —
- frontPage	= null
- funfact	= null

→ Prototype (template) ⇒ Copy

⇒ Create a copy of prototype object & only set the attrs which are different for NB's.

⇒ Often there are scenarios where we create an instance of a prototype, change few attributes and we are DONE!

acting like a template.

```
Class Student {
        name
        age
        batchName
        psp
        avgBatchPsp
  }
  3
```

```
→  Student madhu = new Student();
   madhu.name = "Madhu";
   madhu.age = 28;
   madhu.batchName = "Aug22";
   madhu.psp = 85.0
   madhu.avgBatchPsp = 75.0

(I)  Student rahul = new Student();
     rahul.name = "Rahul";
     rahul.age = 26;
     rahul.batchName = "Aug22";
     rahul.psp = 84.0;
     rahul.avgBatchPsp = 75.0;
```

II

Registry

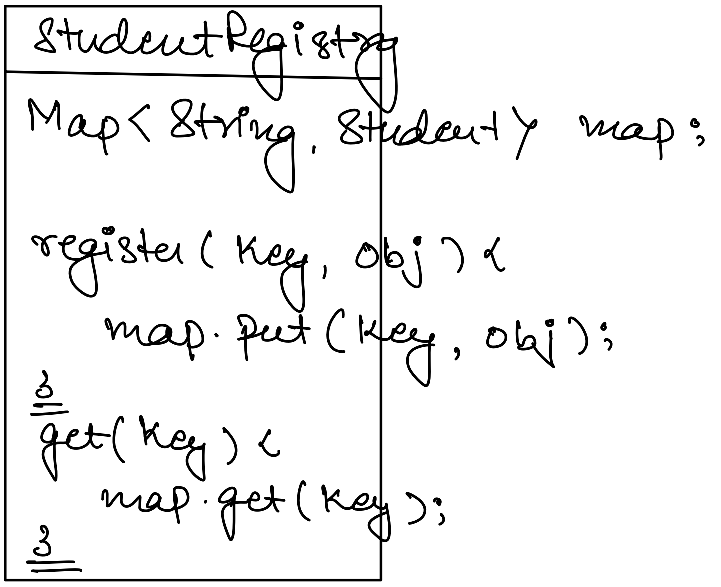
```
Student aug22 = new St();
aug22.batchName = "Aug22";
aug22.avgPsp = 75.0;
```

```
Student vijaya = Registry.getPrototype("aug22").copy();
    vijaya.name = "Vijaya";
    vijaya.age = "24";
    vijaya.psp = 90.0
```

Step 1

In the class that we want to create prototype of, declare a method called clone(), This method creates a copy of the current object.
All the child classes must also implement the clone() method.

Step 2: Store the prototype objects in Registry



```
StudentRegistry
-----------------
Map<String, Student> map;

register(key, obj) {
    map.put(key, obj);
}
get(key) {
    map.get(key);
}
```

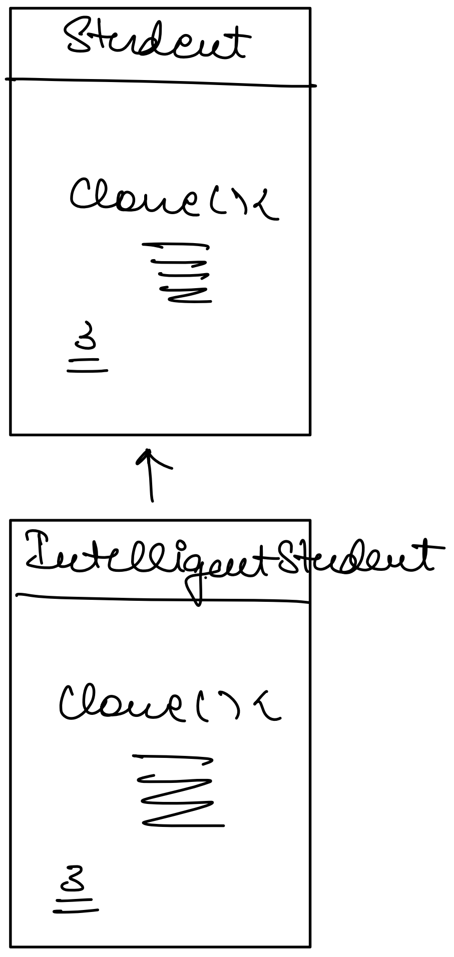
③ Client calls the registry to get the prototype, It then creates a copy of it and updates/ changes the value in the copy object.

PROTOTYPE.

⇒ Often there are scenarios where don't want to create an object from scratch, rather we want to create an copy of an already existing object and change few attributes on that.

Registry :

If we need some object again and again then we can store those objects in the Registry.



——— * ———