Agenda

1. Prototype Design Pattern
2. Registry Design Pattern

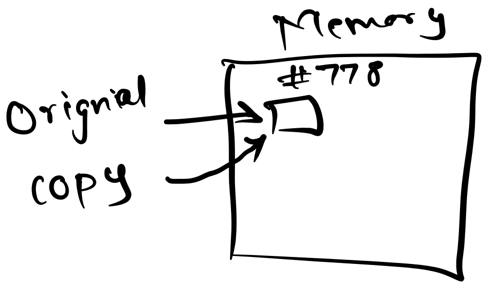
Prototype Design Pattern

Given an object of a class, you want to clone it.

Approaches to achieve this

#1]

```
Student original = ....
Student copy = original;
```



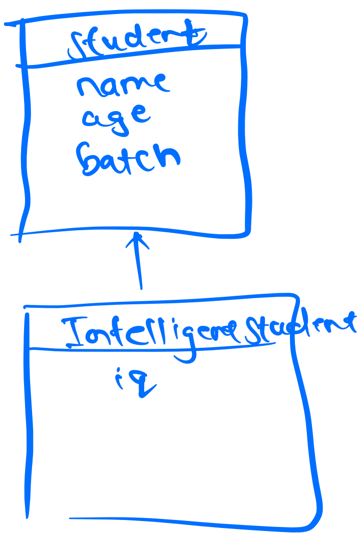
#2) Manually copy attributes

```
Client {
                                          Student()
                                          Intelligent
    Student original = . . . . . .        Student()
    Student copy = new Student();
        copy.name = original.name;
        copy.batch = original.batch;
            :
        copy.email = original.email;

}
```

Problems:

1. Clients cannot access private attributes
2. Tight coupling b/w client & student



SRP ✗

OCP ✗

```
Client

Student original = . . . . .
Student copy = null;
if ( original instanceOf
          Student ) {
      copy = new Student();
         :
} else if (                    ) {
         :
}
```

3. OCP getting violated when we have inheritance.

#3] Copy constructor

```
Student {

public Student () { }

    public student ( Student original) {
      Student  copy = new Student();
        copy.name = original.name;
            :
          return copy;
}
```

Student
IntelligentStudent

```
Client {

    Student  original = . . . . . .
      ~~Student~~ copy = (new Student (~~original~~);
    if ( original instanceOf  Student ){
          copy = new Student (original);
                                  )
    } else if (

}
```

Pros : ① Private attributes can be accessed ② Tight coupling wont happen.

Cons : ① OCP problem still exists

Analysis:

① Giving the responsibility of copying to client is prone to errors & leads to OCP violation.

② Lets give this responsibility to the Student itself.

```
Student  {

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

}
```

```
Client                                  Student/
                                          IntelligentStudent

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

① Private attribute ✓

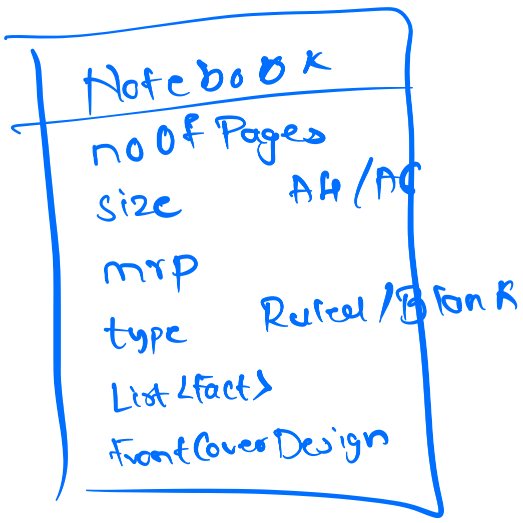
② Tight coupling ✓

③ OCP violation ✓

```
class IntelligentStudent{
    int iq;
    public IntelligentStudent copy(){
        :
        :
    }
}
```

Purpose behind Prototype

Factory management system for Classmate



Order: Give me 10000 notebooks of type Ruled, size A4 & noofPages 200 & all the notebooks should have Facts & beautiful cover design.

Prototype

(Template / 1st version / sample)

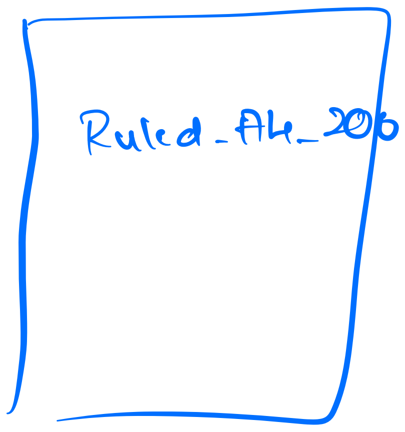
```
Notebook nb = new Notebook()
nb. size = A4
nb. type = "Ruled"
nb. mrp = 100
nb. noof Pages = 200
List <Notebook> order = new Arraylist () ;
for ( int i = 0; i < 10000 ; i++ ) {
    Notebook copy = nb.copy();
    copy. fact = generate RandomFacts();
    copy . front - page . design = . . . . . . ;
    order. add(copy);
}
```

When an object creation requires a lot time due to involvement of a database call or an api call, it makes sense to use prototype design pattern to create multiple objects of it.

Registry Design Pattern

Registry is used to store the prototype objects

Registry



Notebook
: {
type: ruled
size: A4
noOfPages : 200
}

Registry has 2 methods

① Register (String key , Notebook nb) :

Used to store an object in the registry.

② Get (String key) : Notebook :

Used to get objects from registry

Client

```
List <Notebook> order = new ArrayList();
for ( int i = 0; i < 10000; i++ ) {
    Notebook  copy = NotebookRegistry.get("Ruled_A4_200").copy();
    copy.fact = generateRandomFacts();
    copy.front_page.design = . . . . . . ;
    Order.add(copy);
}
```

Main

```
NoteBook nb = new Notebook();
nb.size = A4
nb.type = "ruled"
  :
NotebookRegistry reg = new NotebookRegistry();

reg.register("Ruled_A4_200", notebook);
```

```
class NbRegistry {
  private Map<String, Notebook> map =
                    new HashMap();

  public void register (String key,
                        Notebook nb) {
     map.put (key, nb);
  }
  public Notebook get (String key) {
     return map.get (key);
} }
```

Prototype Design Pattern	Registry Design Pattern
Whenever an obj creation requires api call or db call & you need multiple copies of this, you use Prototype DP	Used to store & get prototype objects.

Registry class should be made a singleton.

Real life example of Registry

```
ChatGpt Query {

        host  }
        url   }  Remain same
        token }
        query → Change per object

}
```

Option #1

```
ChatGPT Query q = new
                ChatGPTQuery();

    q.host = ____ . _ ;
    q.url = ____ . _ _ ;
    q.token = fetch_api_token();
    q.query = . . . . . . ;
```

✓

Option #2

```
ChatGPT
Query     q = ChatGPTQueryRegistry.get("Base Query").
                                       copy();

        q.query = "            ";
```

Assignment

→ Implement Classmate example
(Prototype + Registry Design Pattern)

→ Read up about Prototype on refactoring.guru