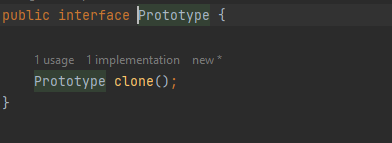
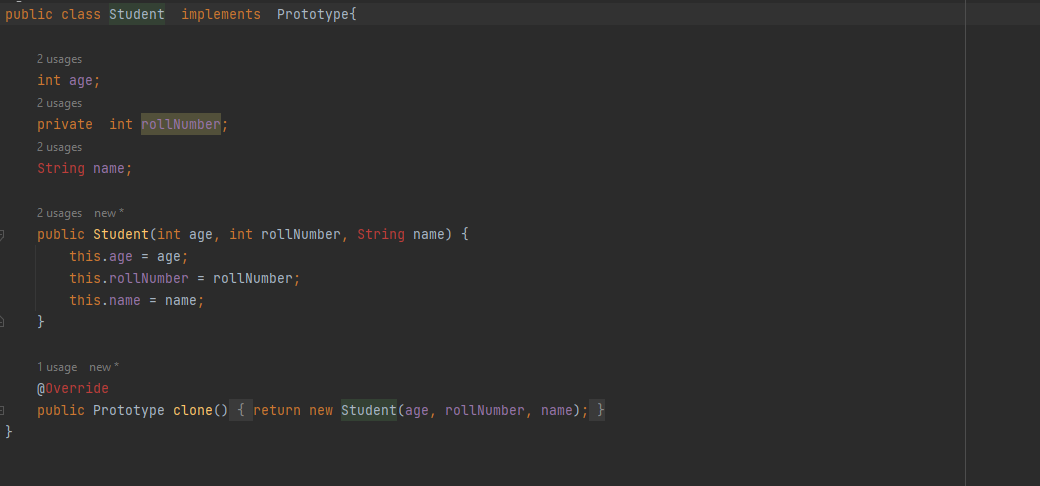
**CREATIONAL DESIGN PATTERN**

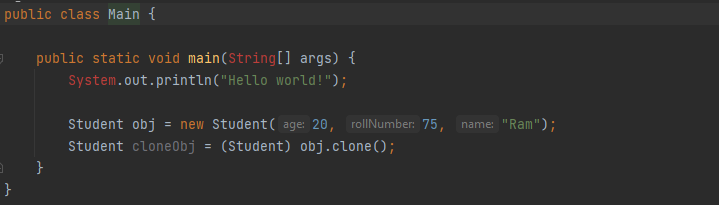
Controls creation of an object.

**PROTOTYPE**

Used when we have to copy/clone from existing obj

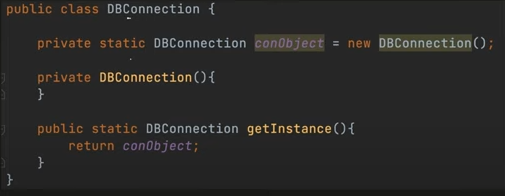
****

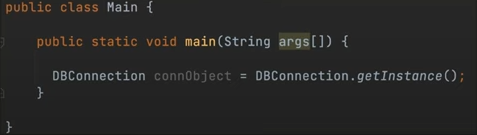
****

****

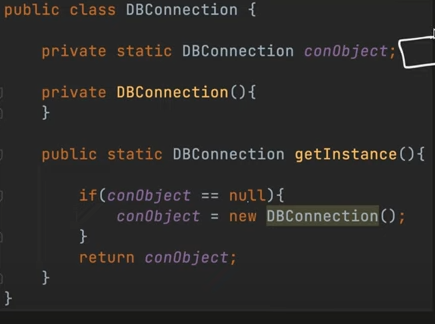
**SINGLETON PATTERN  
  
4 ways**

1. **Eager Initialization**

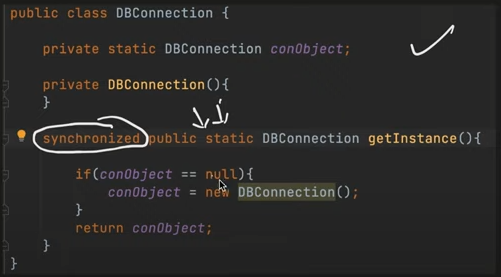
****

****

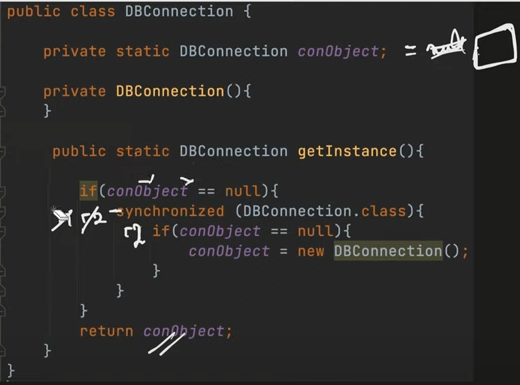
1. **Lazy**

**  
  
The problem with this approach is in case we have multiple threads entering the critical section where we check conObject == null. In this scenario both the threads will try to create the conObject and 2 objects will be created.**

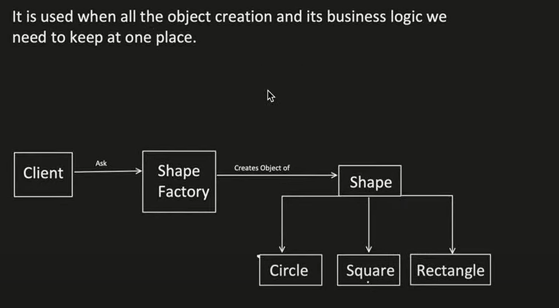
1. **Synchronized**

**  
  
The problem we saw in lazy is solved with this. But since locking each and everytime getInstance() is called is very expensive we do not use synchronized method because of this.**

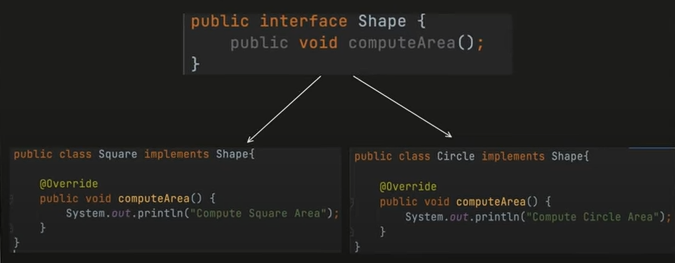
1. **Double Locking**

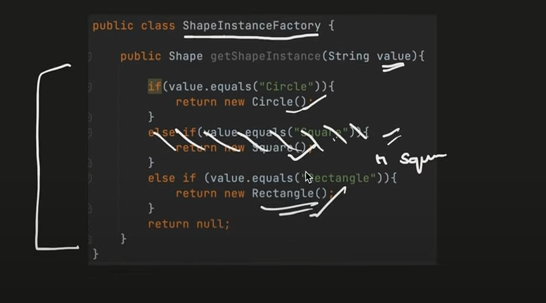
****

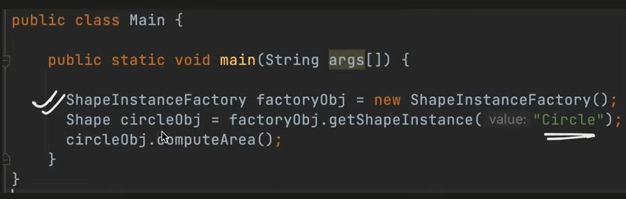
**Factory Pattern  
  
  
Simple Factory**

****

**This says that if there is any change in the logic of creation of an object we need to change it only at one place and not in 100s of other classes**

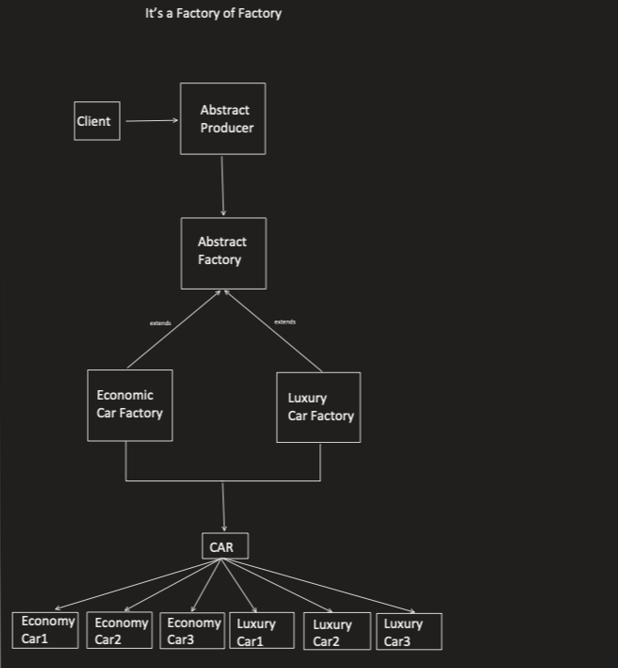
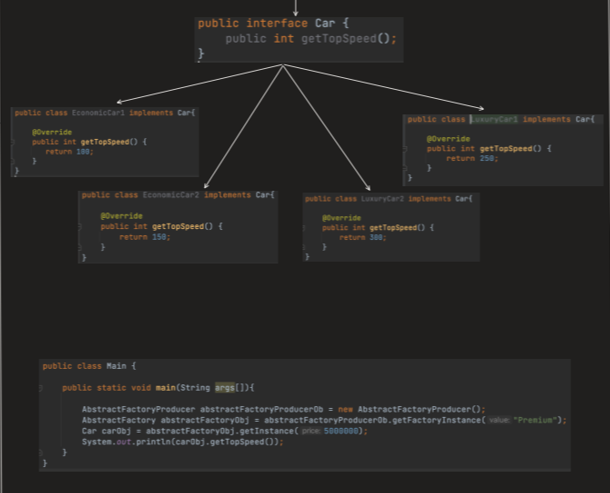
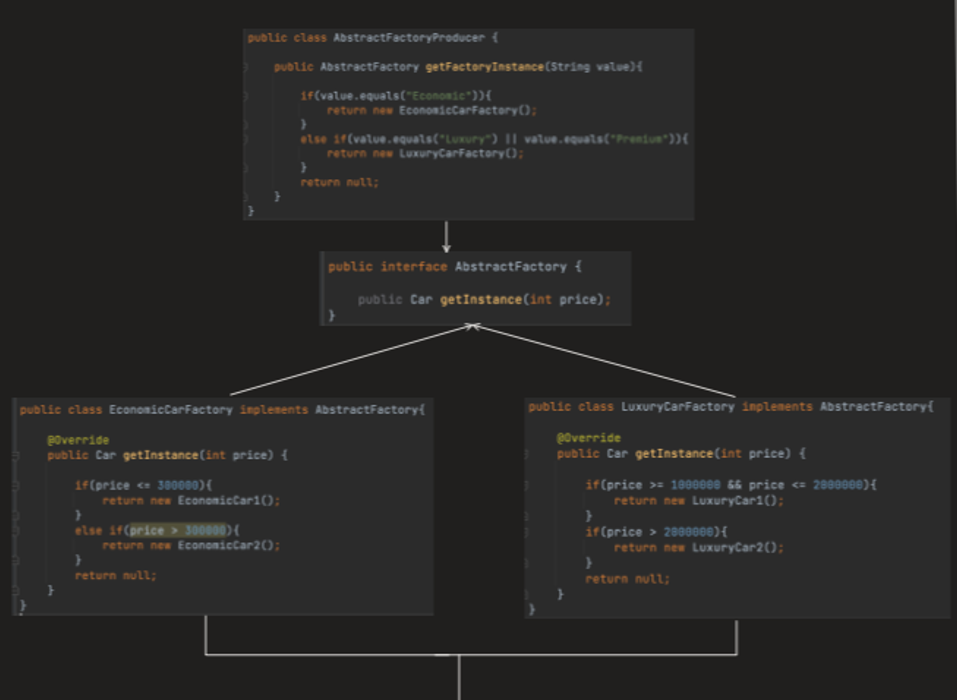
****

**  
Now as we can see if there is a change in logic say suppose if value ==”Rectangle” or “Square” then we have to return Rectangle only. In this case we just need to change this logic at one place and not in every place that is using a square or rectangle.**

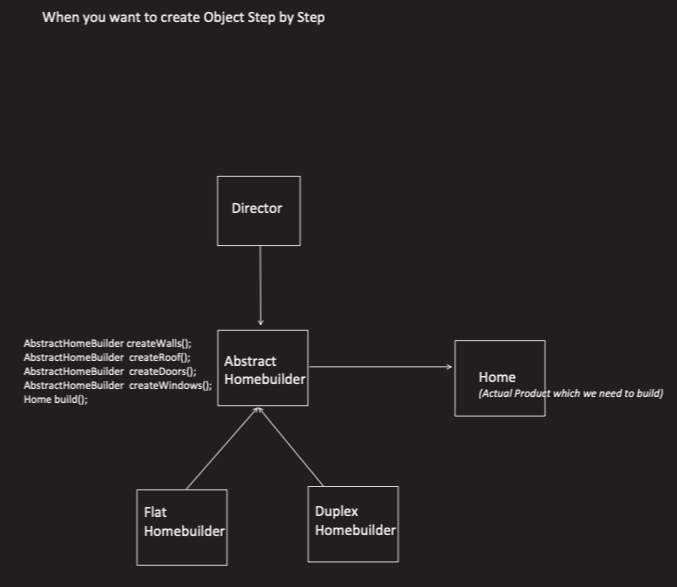
****

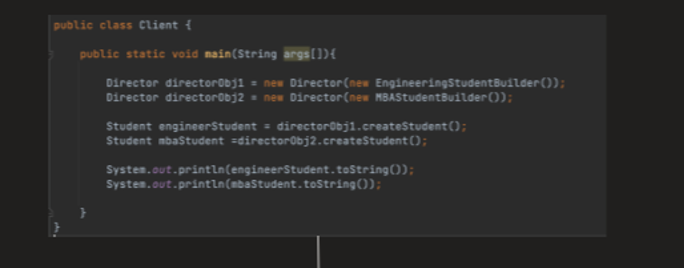
**Abstract Factory**

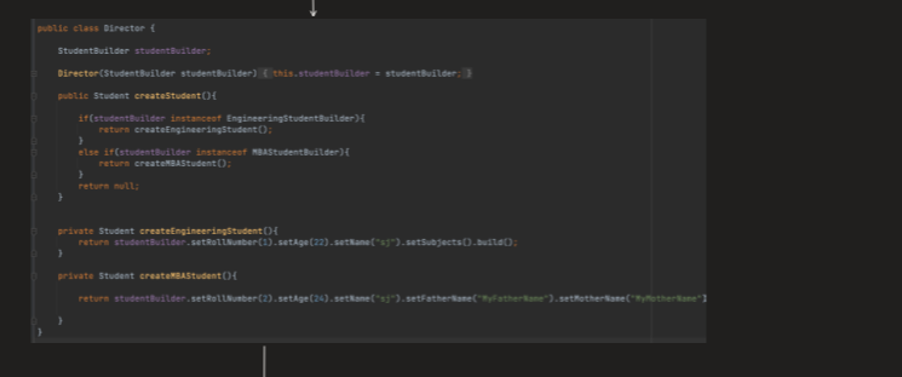
**It’s a Factory of Factory**

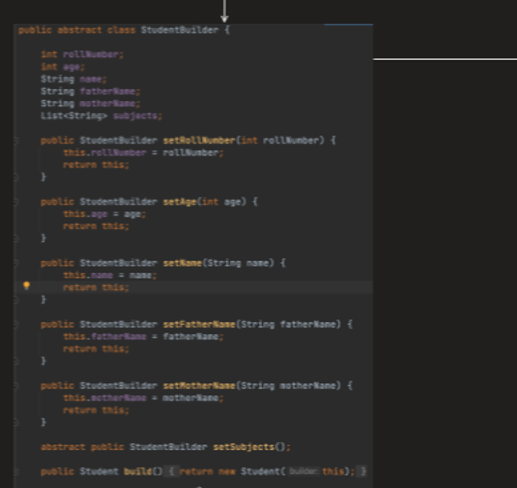
**  
  
**

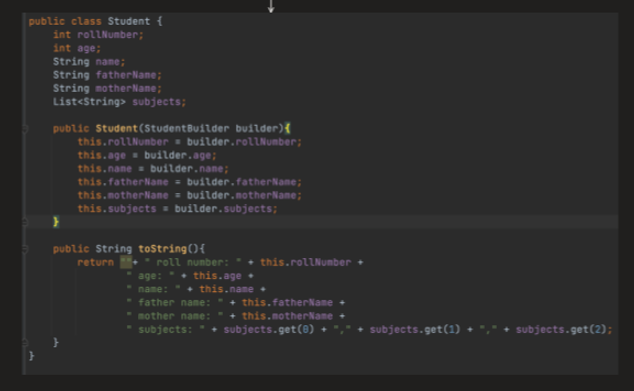
**Builder Pattern  
  
When we want to create an obj step by step**

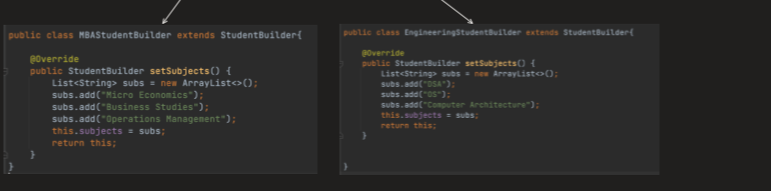
****

****

****

****

****

****