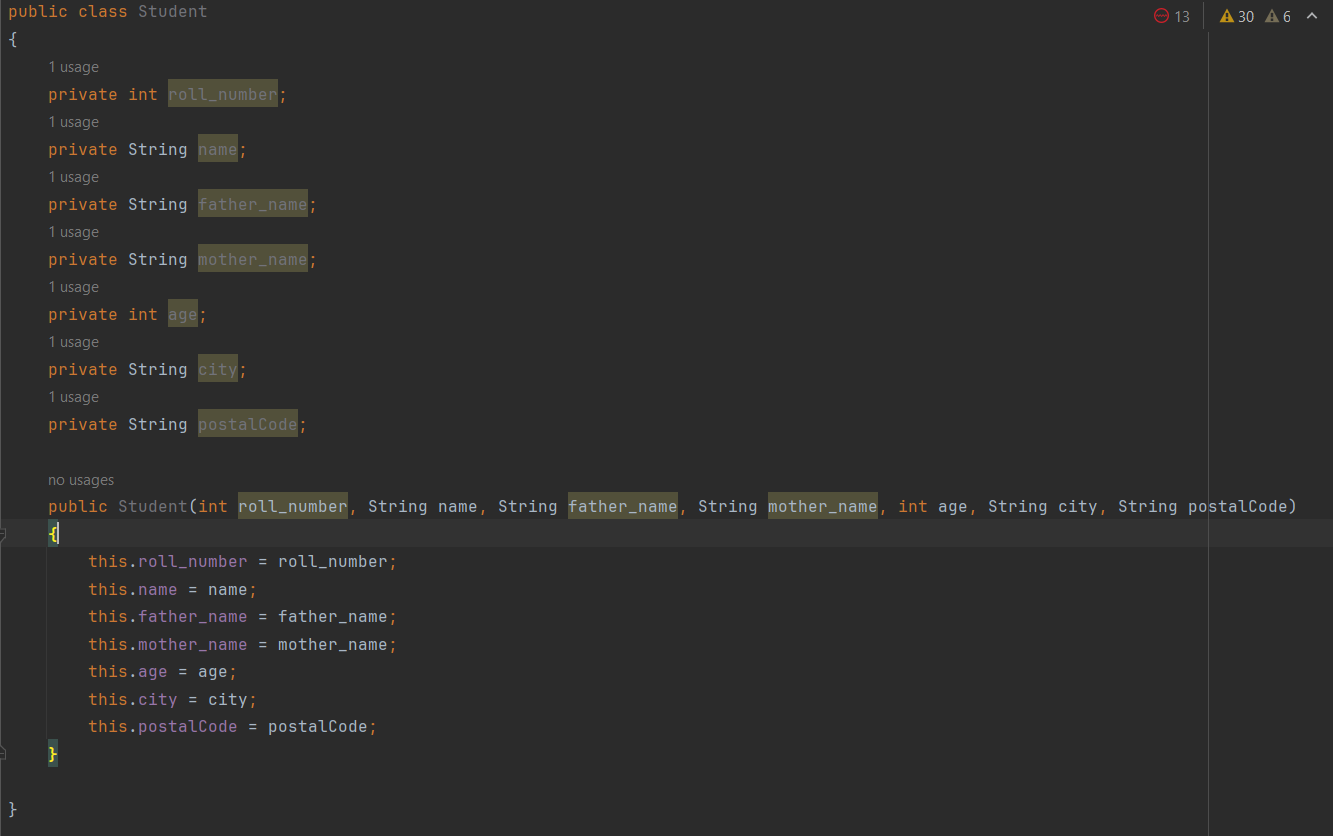
**Builder Design Pattern:**

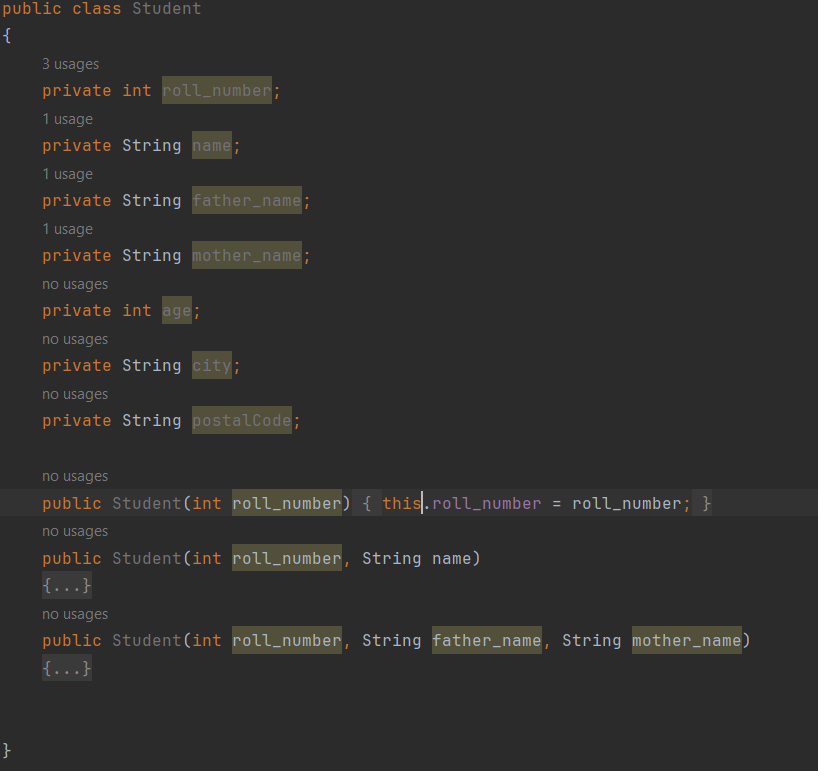
Builder pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes. There are three major issues with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

1. Too Many arguments to pass from client program to the Factory class that can be error prone because most of the time, the type of arguments are same and from client side its hard to maintain the order of the argument.
2. Some of the parameters might be optional but in Factory pattern, we are forced to send all the parameters and optional parameters need to send as NULL.
3. If the object is heavy and its creation is complex, then all that complexity will be part of Factory classes that is confusing.

We can solve the issues with large number of parameters by providing a constructor with required parameters and then different setter methods to set the optional parameters. The problem with this approach is that the Object state will be **inconsistent** until unless all the attributes are set explicitly. Builder pattern solves the issue with large number of optional parameters and inconsistent state by providing a way to build the object step-by-step and provide a method that will actually return the final Object.



* Increased constructor arguments it increases constructor size.
* If the user have only roll\_number has mandatory field then it forces the developer to insert default values for other fields.
* To resolve this issue we have to make multiple arguments.



* In above case if we write our method like this it will increase the number of constructors.
* It will also throw error if the user try to create multiple constructors with same argument.
* To resolve this issue we have to use Builder design pattern, it will going to create the object step by step.

Example: for example, if we looking to construct a house, we need multiple things like basement, roof, walls, windows, doors etc.

Step - 1: Housebuilder

add(basement) return Housebuilder

Step - 2: Housebuilder

add(walls) return Housebuilder

Step - 3: Housebuilder

add(windows) return Housebuilder

Step - 4: Housebuilder

add(roof) return Housebuilder

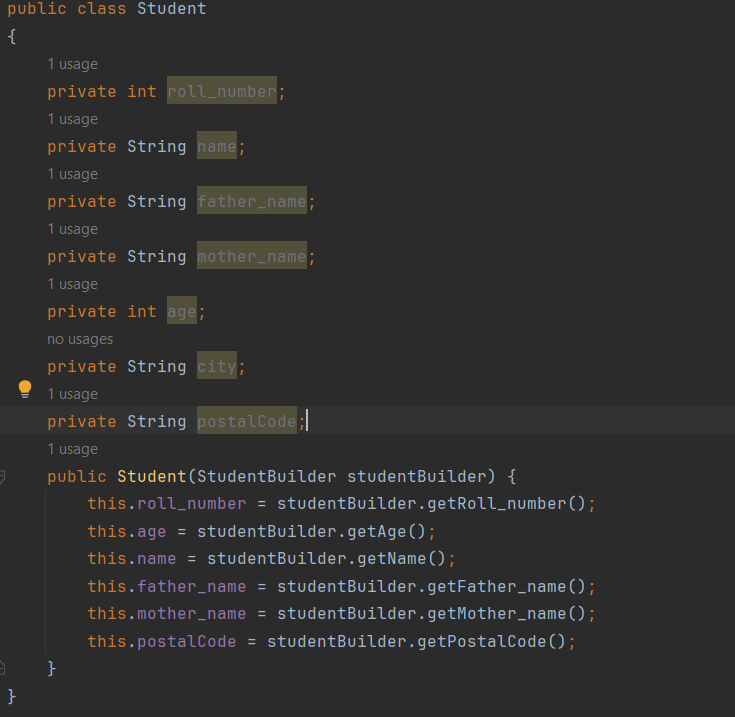
Step - 5: Housebuilder

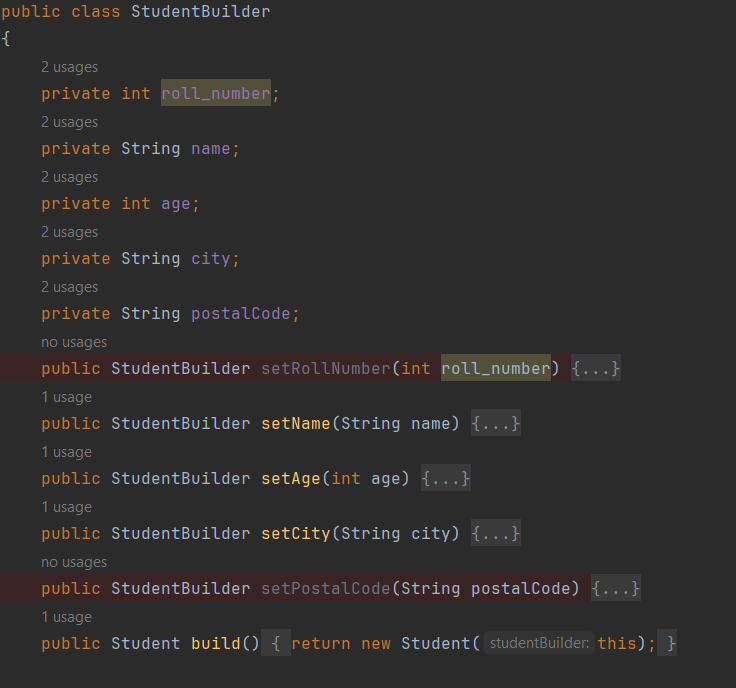
add(doors) return Housebuilder

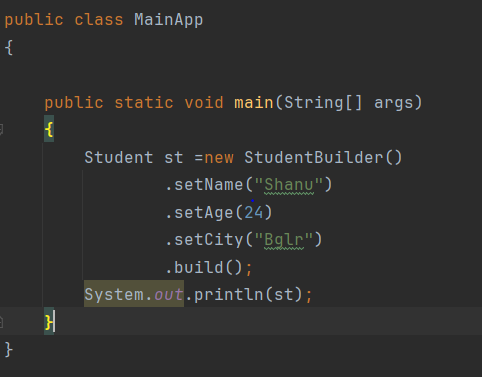
Step – 6: Housebuilder

build() return House

* disadvantage of builder design pattern is code duplicacy
* to implement builder we can use either interface or abstract so that we can achieve abstraction.







* client code shown above.

