**Singleton Design Pattern**

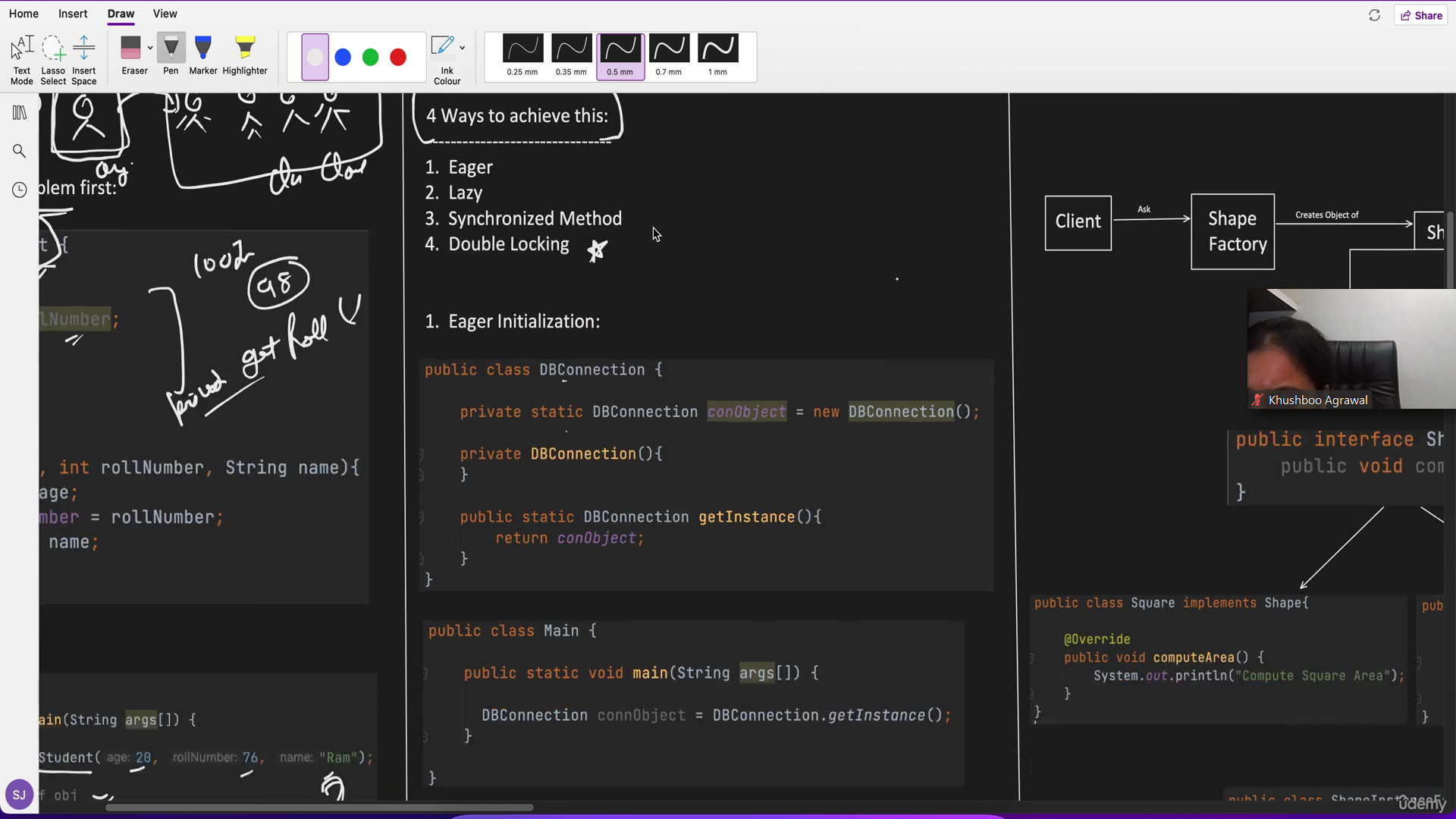
The Singleton design pattern is a creational pattern that ensures a class has only one instance and provides a global point of access to that instance. This pattern is useful when exactly one object is needed to coordinate actions across the system.

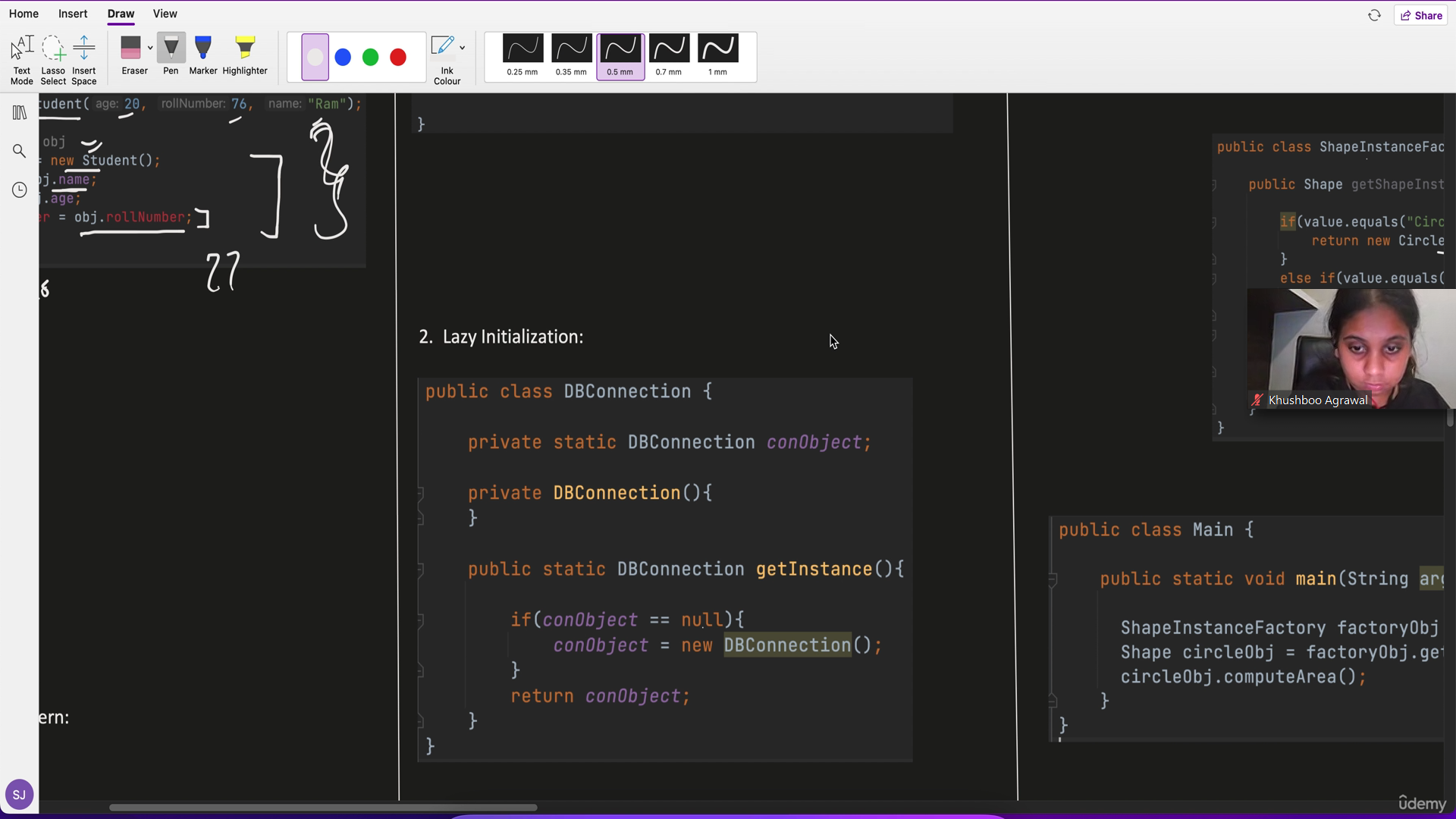
**Definition**

The Singleton design pattern ensures that a class has only one instance and provides a global point of access to that instance. It is particularly useful when a single object is needed to coordinate actions across the system.

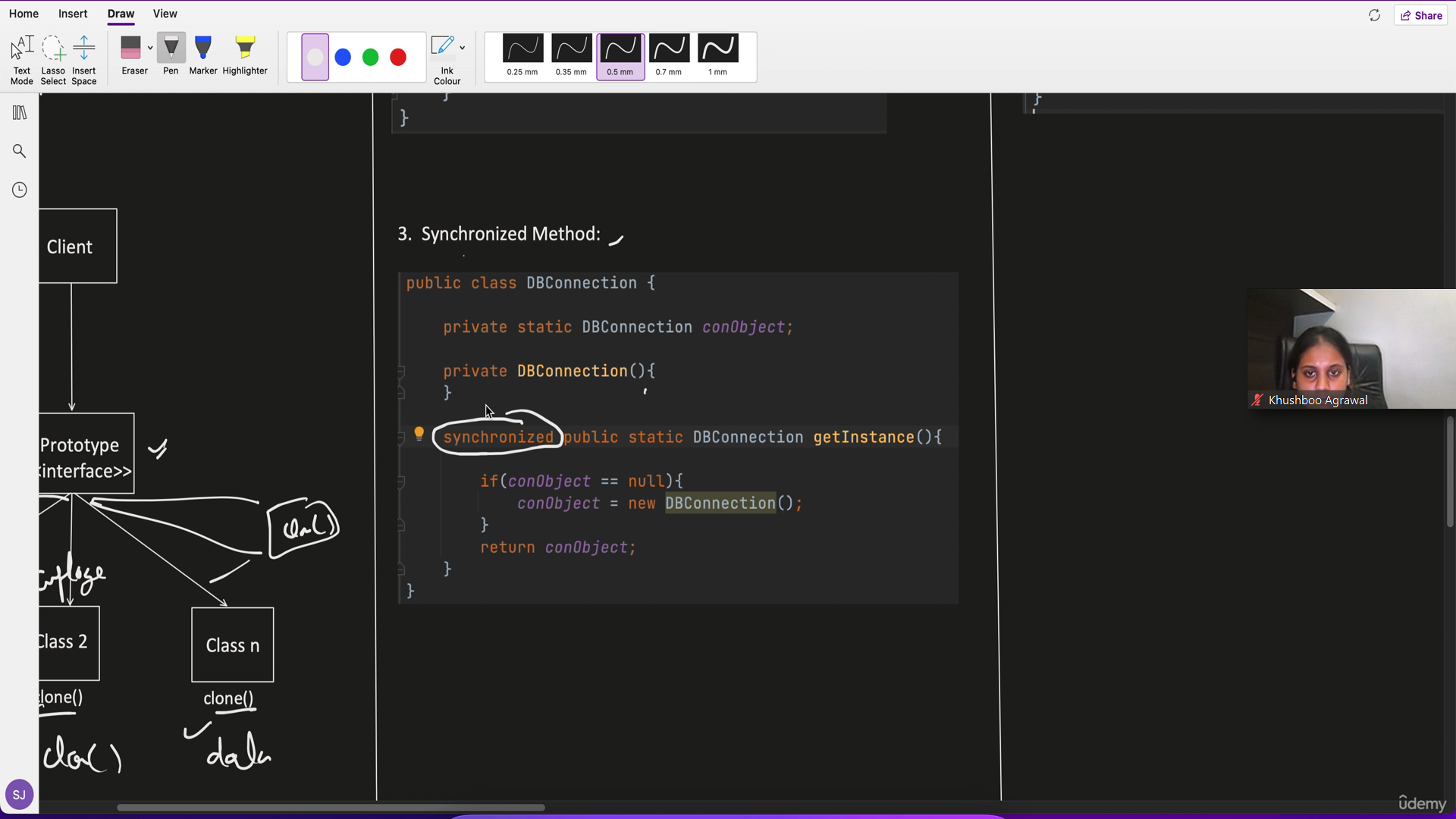
Four ways to achieve this –

1. EAGER
2. LAZY (No thread safety)
3. SYNCHRONIZED METHOD
4. DOUBLE LOCKING (Most used in the industry)

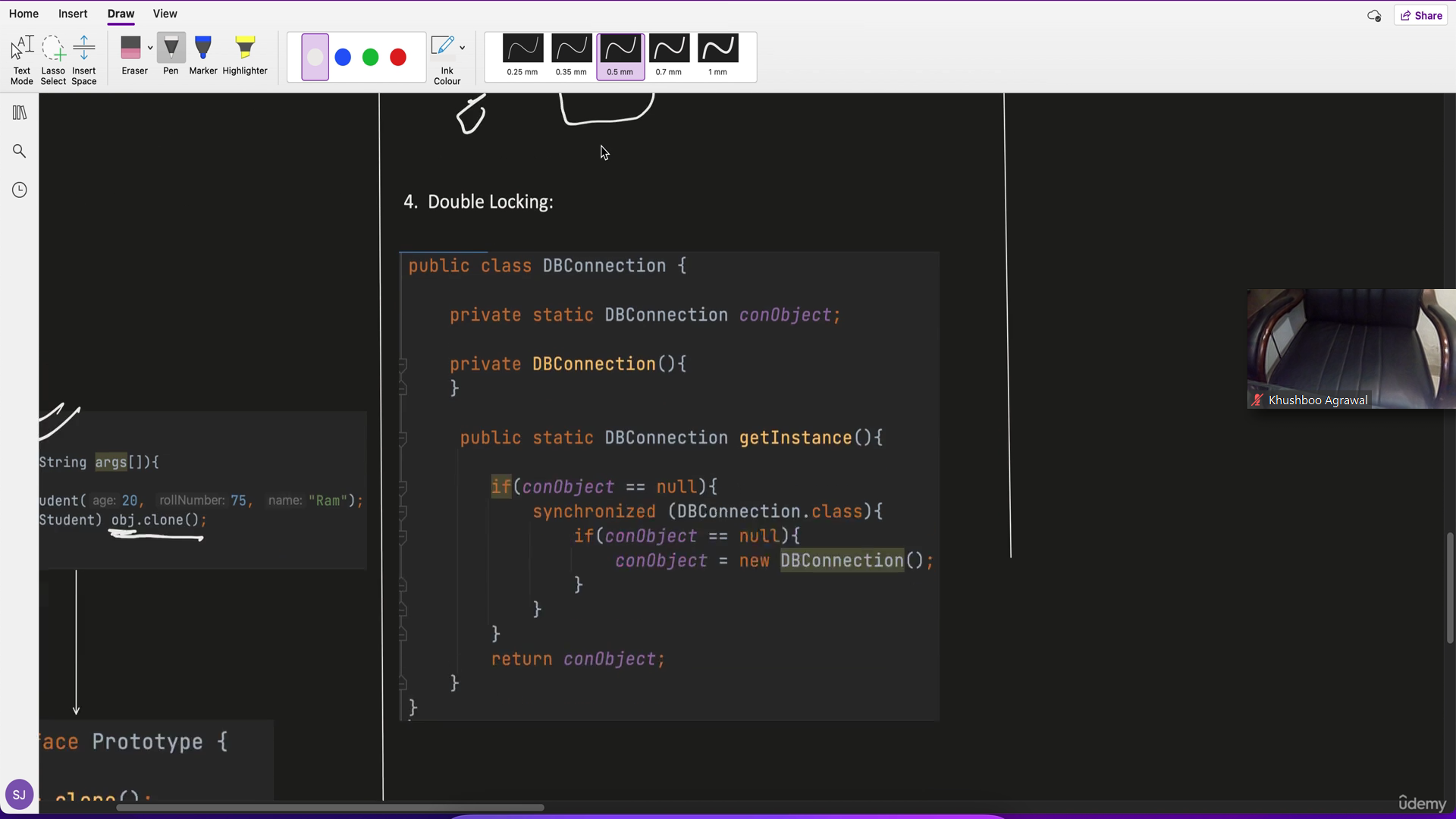




Two objects can be created in getInstance method if accessed by two threads simultaneously. Therefore, it is not thread safe.

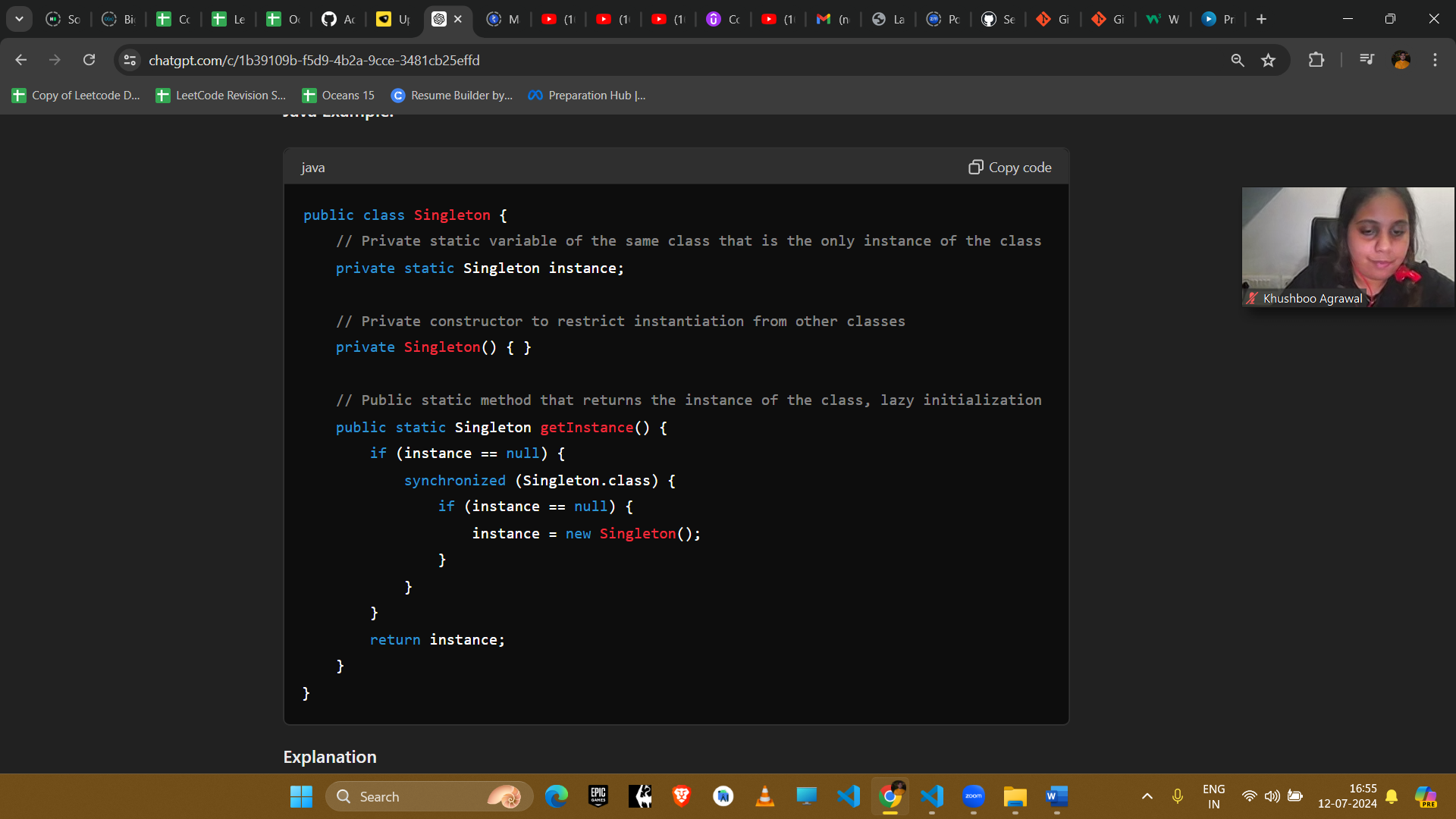


Locking (using synchronized) is very expensive if multiple threads show up (eg 1000)



**Example**

Consider a scenario where we need a single instance of a configuration manager.



**Explanation**

1. **Private Static Variable**: Holds the single instance of the Singleton class.
2. **Private Constructor**: Prevents instantiation from outside the class.
3. **Public Static Method**: Provides the global access point to get the instance of the class. The double-checked locking mechanism ensures that the instance is created only when it is needed and is thread-safe.

**Example Uses in Amazon Interviews**

1. **Configuration Management**
   * **Scenario**: Managing application configurations.
   * **Implementation**: A singleton class holds the configuration settings, ensuring that all parts of the application use the same configuration data.
2. **Logging**
   * **Scenario**: Logging application activities.
   * **Implementation**: A singleton logger ensures that all classes log messages to the same logging instance, providing a centralized logging mechanism.
3. **Database Connection Pool**
   * **Scenario**: Managing a pool of database connections.
   * **Implementation**: A singleton class manages the database connections, ensuring efficient use of resources by reusing existing connections.
4. **Caching**
   * **Scenario**: Storing frequently accessed data.
   * **Implementation**: A singleton cache stores frequently accessed data, providing quick access and improving performance.

**Conclusion**

The Singleton pattern is useful when a single instance of a class is needed to control access to resources or manage shared state. It is commonly used in scenarios requiring centralized control, such as logging, configuration management, and database connection pooling.