Singleton

1. For some components it only makes sense to have one in the system
   1. Database repository
   2. Object factory
2. E.g. the constructor call is expensive
   1. We only do it once
   2. We provide everyone with the same instance
3. Want to prevent anyone creating additional copies
4. Need to take care of lazy instantiation and thread safety
5. Singleton: A component which is instantiated only once.

Basic Singleton

1. Have a private constructor
2. Create a private instance of the class
3. Have a static method that returns the private instance

Serialization Problems

1. Using reflection you can access the private constructor that we have defined previously.
2. Serialization and deserialization. JVM doesn't care if your constructor is private.
3. Object implements Serializable and has a readResolve method that returns the instance letting JVM know to use the same instance when deserializing.

Static Block Singleton

1. You may want to do additional operations in private constructor that may throw exceptions.
2. To solve you create a static block that has a try catch block when creating the instance.

Laziness and thread safety

1. Create the instance when the user wants it.
2. Check if the instance is null. If so, create it and return the instance. → This has thread safety issues.
   1. One way to solve this is to add a synchronized keyword to the method of getInstance but thi has its performance issues.
   2. Double-checked locking → It is mostly outdated but we should be aware of it.

Inner Static Singleton

1. Create a private inner class that actually creates the instance and return this instance in the outer class.
2. This avoids synchronization problems.

Enum Based Singleton

1. Enums came with 1.5
2. No reflection problem with this approach.
3. constructor is always private for enum.
4. Enums are serializable by default. Because fields ont get serialized for enums.
5. And you cannot inherit from enum.

Monostate

1. Making the storage elements (fields) static.
2. So these fields are shared between objects.

Multiton

1. Allowed to make multiple instances.
2. Look at the example code.

Testability Issues

1. In the given example in the course, we need to directly access the actual data and know the data inside.

Singleton Dependency Injection

1. With constructor ğinjection inject a database interface and put the actual code in that class. With this way when you are writing your tests you can create a dummy database and inject that.
2. This way you don't directly access the data. And you don't need to know the actual data.

Summary

1. Making a safe singleton is easy: constructor a static Lazy<T> and return its value.
2. Singletons are difficult to test
3. Instead of directly using a singleton, consider depending on an abstraction (eg. an interface)
4. Consider defining singleton lifetime in the DI container.