Different Types of Dependency Injection

- Constructor injection
- Field injection
- Property/Setter injection

Spring Context Loader Sequence of Events

1. Start reading and processing the **configuration**
2. Instantiate all the **singleton** beans and do all the **constructor** injections
3. **Field** and **property** injection happen here
4. All init methods (or methods annotated with @PostConstruct) will be called
5. Application will run for a while
6. Once context.close() is called and before the Spring application shuts down, all the @PreDestroy methods are called

Issues with Session/Operation Anti-Pattern

1. We lose the maximum benefit of caching
2. You can only have one transaction per operation (no unit of work)
3. Code problem. Session open/close and transaction start/commit needs to be repeated many times unnecessarily
4. Exception problem. We are catching the exception at the lowest layer of the application (data access layer)

How do we solve the session/operation anti-pattern?

By applying the Hibernate ThreadLocal/Current Session Pattern

What happens when you call sessionFactory.getCurrentSession() (with ThreadLocal pattern?

Hibernate will load the current session (will create one if it does not exist) from the ThreadLocal context. As a result, each thread gets its own session. In this case (and only in this case), hibernate closes the session when you tx.commit()

What is the one issue that still remains?

After you close the session (in the Controller or Service layer), if you try to access lazy-loaded data in JPA entities, you get the LazyInitialization exception. This is normally in the context of a web-application.

How do you solve that?

- Open-session-in-view filter
- Hibernate.initialize()
- Eager loading of everything