Instantiation

1. By default, all beans are singletons, so whenever Application context gets created, they are all pre-loaded. If, specifically, any singleton bean has an attribute lazy-init="true" set, it will be lazy-loaded, i.e. it will be instantiated when the getBean method is called for the first time.
2. For other scopes, beans will be instantiated whenever they are requested.

It depends what is the scope of the bean you are calling with getBean() method. If it is 'Singleton', it is pre-instantiated by the ApplicationContext.
If you are using BeanFactory as an IOC Container, then it uses lazy initialization and the beans will be instantiated only when you call the getBean() method. This is an advantage of ApplicationContext over BeanFactory that it solves Circular Dependency problem.

Spring guarantees that the defined beans will be initialized before attempting an initialization of the current bean. Let's say we have a FileProcessor which depends on a FileReader and FileWriter. In this case, FileReader and FileWriter should be initialized before the FileProcessor.

https://www.baeldung.com/spring-depends-on#:~:text=Spring%20guarantees%20that%20the%20defined,be%20initialized%20before%20the%20FileProcessor

For reference, see
- Lazy-initialized beans and
- Bean scopes

Here's a brief description of when beans are created:
- A singleton bean (which is the default scope) that does not have the lazy-init property set to true (default is false) is constructed when the application context is created
- A singleton bean that does have the lazy-init property set to true is constructed when it is first requested
- A bean set in any other scope is created when it is first requested (for that scope).

I understand the confusion. Let's clarify:

1. **When a Bean is Needed: In Spring, beans are objects managed by**

the Spring IoC (Inversion of Control) container. These beans represent various components of your application, such as services, controllers, data access objects, etc. Beans are typically needed when your application requests them to perform a certain task. For example, if your application needs to fetch data from a database, it might request a data access object (DAO) bean to perform the database operation.

2. **When a Bean is Created**: Beans in Spring are created by the Spring IoC container. The container is responsible for instantiating, configuring, and managing the lifecycle of beans. Beans can be created in one of two ways:
   - **Eager Initialization**: By default, Spring initializes beans eagerly, meaning that beans are created as soon as the application context is loaded, even if they are not immediately needed. This approach ensures that beans are available for use when required.
   - **Lazy Initialization**: Alternatively, you can configure beans for lazy initialization. With lazy initialization, beans are created only when they are first requested by the application. This can improve application startup time and resource usage, especially for large applications with many beans.

To summarize, beans in Spring are created by the Spring IoC container, and they are typically created either eagerly at application startup or lazily when they are first needed by the application. The choice between eager and lazy initialization depends on factors such as application requirements, performance considerations, and resource usage.

Yes, that's correct. In Spring, beans are typically created lazily, meaning that they are instantiated and configured by the Spring IoC container only when they are first requested by the application through the `getBean()` method or other mechanisms such as dependency injection.

When you call the `getBean()` method on the ApplicationContext or BeanFactory to retrieve a bean by its name or type, Spring checks if the bean has already been instantiated. If the bean has not been created yet, Spring creates it at that moment, initializes its properties, and then returns it to the caller.

Here's a simplified example:

```java
// Assuming applicationContext is an instance of ApplicationContext
MyBean myBean = applicationContext.getBean(MyBean.class);
```

In this example, if the `MyBean` bean has not been created yet, Spring will

create it when `getBean()` is called. Subsequent calls to `getBean()` for the same bean will return the same instance that was previously created.

This lazy initialization behavior allows Spring to optimize resource usage by deferring the creation of beans until they are actually needed by the application. It's particularly beneficial for large applications with many beans, where eager initialization of all beans at startup could lead to increased memory consumption and longer startup times.