LAZY AND EAGER INITIALIZATION OF SPRING BEANS

EAGER INITIALIZATION:

Eager initialization is the default behaviour of the Spring Framework, where beans are created and initialized during the application startup. This means that all the beans defined in the application context are created and initialized, regardless of whether they are used or not.

LAZY INITIALIZATION:

Lazy initialization is a technique in Spring Framework where an object or bean is created only when it is needed, instead of being created during the application startup. This is useful in scenarios where the creation of an object is expensive, and you want to defer the initialization until the object is required.

EXAMPLE: EAGER INITIALIZATION

```
package com.naveen.learnspringframework.LazyVSEagerInitialization;

import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.stereotype.Component;

@Component
class Manager{

    Employee employee;

    public Manager(Employee employee) {
        super();
        this.employee = employee;
        System.out.println("Manager bean created and got employee.");
    }

    public void doWork() {
        System.out.println("Manager doing work");
    }
}

@Component
class Employee{

}

@Configuration
@ComponentScan
```

```
public class LazyInitializationApp {

    public static void main(String[] args) {

        try(var context =
                new AnnotationConfigApplicationContext
                    (LazyInitializationApp.class)){

        }
    }
}
```

Once the spring context is prepared in the above example, the beans are instantiated within the context. This is example of Eager Initialization of spring beans.

OUTPUT:

```
13:14:21.600 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.605 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.608 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.611 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.628 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.637 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.638 [main] DEBUG org.springframework.beans.factory.suppc
13:14:21.668 [main] DEBUG org.springframework.beans.factory.suppc
Manager bean created and got employee.
13:14:21.759 [main] DEBUG org.springframework.context.annotation.
```

EXAMPLE: LAZY INITIALIZATION

```
package com.naveen.learnspringframework.LazyVSEagerInitialization;

import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

@Component
@Lazy
class Manager{

    Employee employee;

    public Manager(Employee employee) {
        super();
        this.employee = employee;
        System.out.println("Manager bean created and got employee.");
    }
```

```
    public void doWork() {
        System.out.println("Manager doing work");
    }
}

@Component
@Lazy
class Employee{

}

@Configuration
@ComponentScan
public class LazyInitializationApp {

    public static void main(String[] args) {

        try(var context =
                new AnnotationConfigApplicationContext
                    (LazyInitializationApp.class)){

        }
    }
}
```

OUTPUT:

```
13:13:37.495 [main] DEBUG org.springframework.context.annotation.
13:13:37.533 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.634 [main] DEBUG org.springframework.context.annotation.
13:13:37.641 [main] DEBUG org.springframework.context.annotation.
13:13:37.831 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.836 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.838 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.841 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.858 [main] DEBUG org.springframework.beans.factory.suppc
13:13:37.949 [main] DEBUG org.springframework.context.annotation.
```

In the above example, we have added the **@Lazy** annotation. So, it will create the bean only when it is needed. That's why we couldn't see any message which is in Manager constructor.

```
package com.naveen.learnspringframework.LazyVSEagerInitialization;

import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Component;

@Component
```

```java
class Manager{

    Employee employee;

    public Manager(Employee employee) {
        super();
        this.employee = employee;
        System.out.println("Manager bean created and got employee.");
    }

    public void doWork() {
        System.out.println("Manager doing work");
    }
}

@Component
class Employee{

}

@Configuration
@ComponentScan
public class LazyInitializationApp {

    public static void main(String[] args) {

        try(var context =
                new AnnotationConfigApplicationContext
                    (LazyInitializationApp.class)){

            System.out.println("Initialization of context is completed");

            context.getBean(Manager.class).doWork();
        }
    }
}
```

OUTPUT:

```
13:15:18.039 [main] DEBUG org.springframework.beans.factory.suppc
13:15:18.043 [main] DEBUG org.springframework.beans.factory.suppc
13:15:18.062 [main] DEBUG org.springframework.beans.factory.suppc
13:15:18.071 [main] DEBUG org.springframework.beans.factory.suppc
13:15:18.072 [main] DEBUG org.springframework.beans.factory.suppc
13:15:18.101 [main] DEBUG org.springframework.beans.factory.suppc
Manager bean created and got employee.
Initialization of context is completed
Manager doing work
13:15:18.189 [main] DEBUG org.springframework.context.annotation.
```

DISADVANTAGES OF EAGER INITIALIZATION:

Eager initialization can also have drawbacks, such as slowing down the application startup time and increasing the memory footprint of the application. It can also result in unnecessary resource consumption if certain beans are not actually used during the application runtime. In such cases, lazy initialization may be a better approach.

DISADVANTAGES OF LAZY INITIALIZATION:

1. **Increased latency**: If a bean is lazily initialized, there will be a slight delay the first time it is used as the initialization process takes place. This can add some latency to the application, which may be undesirable in certain performance-sensitive scenarios.
2. **Increased complexity**: Lazy initialization can make the application context more complex, as you need to carefully consider which beans should be lazy-loaded and which should not. This can make the application harder to manage and maintain.
3. **Possible errors at runtime**: If a bean is not initialized during the application startup, it can potentially result in errors at runtime if the bean is actually used. This can be difficult to debug, especially if the initialization process involves complex logic or external dependencies.
4. **Inconsistent resource usage**: Lazy initialization can result in inconsistent resource usage, as beans may be created and initialized at different times during the application runtime. This can make it harder to predict the memory usage and performance of the application.
5. **Difficulty in managing dependencies**: Lazy initialization can make it harder to manage dependencies between beans, as you need to ensure that all the required beans are initialized before they are used. This can result in additional complexity and potential errors in the application context.