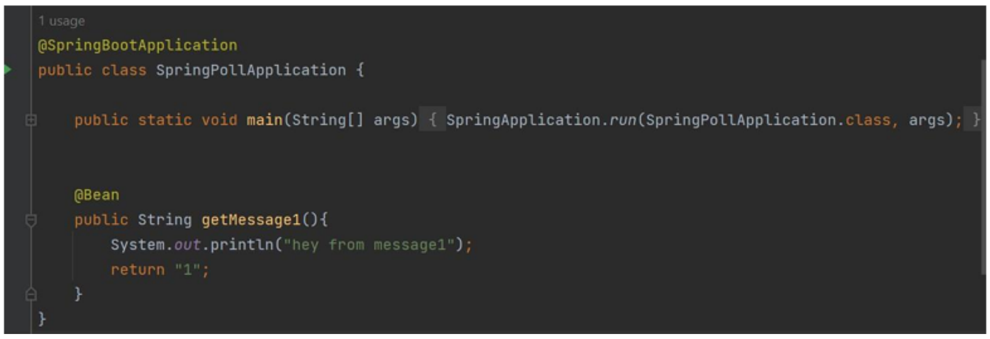
Exercise (Spring Container)

Preliminary Explanation:
Spring Boot applications run over a Spring Container or IoC container which in simple words is a place that creates, manages and configures the lifecycles of objects I.e beans. Bean are objects that are managed by Spring container writing it over any function that includes it into the container.

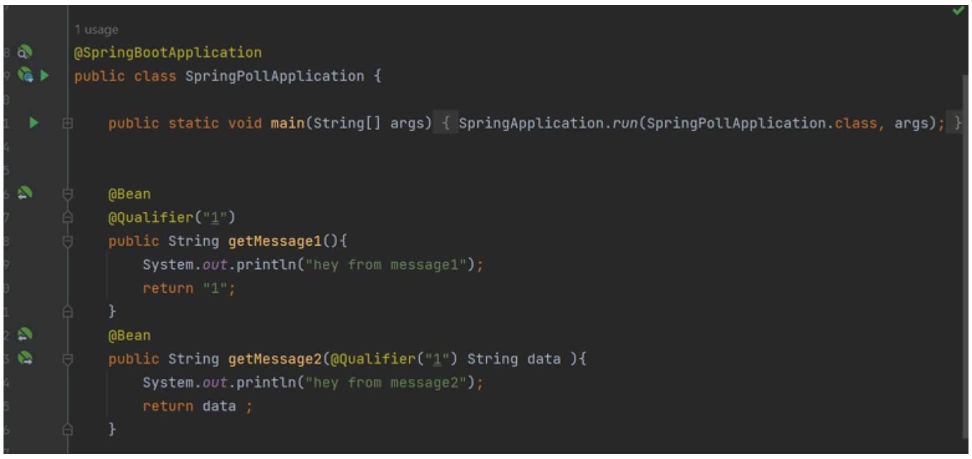
Q1.



```java
1 usage
@SpringBootApplication
public class SpringPollApplication {

    public static void main(String[] args) { SpringApplication.run(SpringPollApplication.class, args); }


    @Bean
    public String getMessage1(){
        System.out.println("hey from message1");
        return "1";
    }
}
```

A1.
Creating a single bean would result in “**hey from message1”** being printed due to getMessage being in the container, but asking for multiple beans to be included might result in unpredictable behavior or sometimes outright issues as seen in class.

Q2.

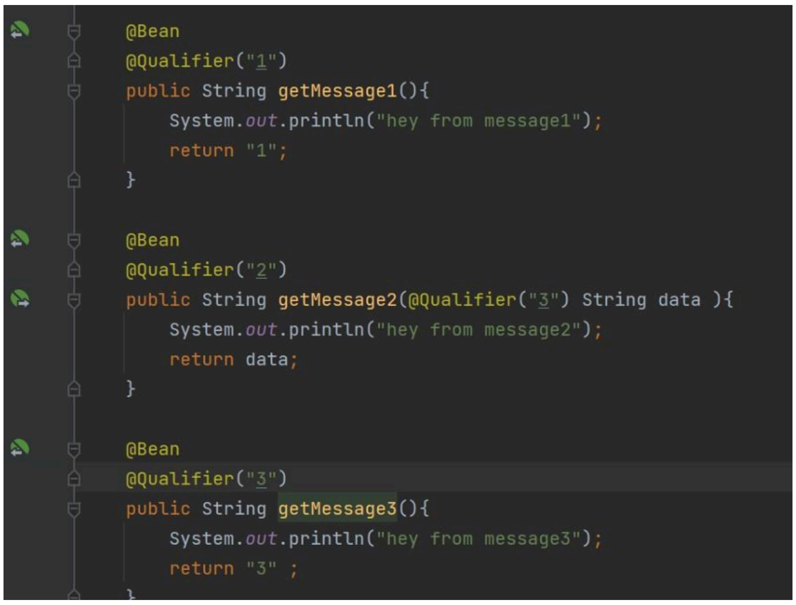


```java
1 usage
@SpringBootApplication
public class SpringPollApplication {

    public static void main(String[] args) { SpringApplication.run(SpringPollApplication.class, args); }


    @Bean
    @Qualifier("1")
    public String getMessage1(){
        System.out.println("hey from message1");
        return "1";
    }
    @Bean
    public String getMessage2(@Qualifier("1") String data ){
        System.out.println("hey from message2");
        return data ;
    }
```

A2.
when multiple beans exist in the container of the same type using qualifier would specify which bean is used.

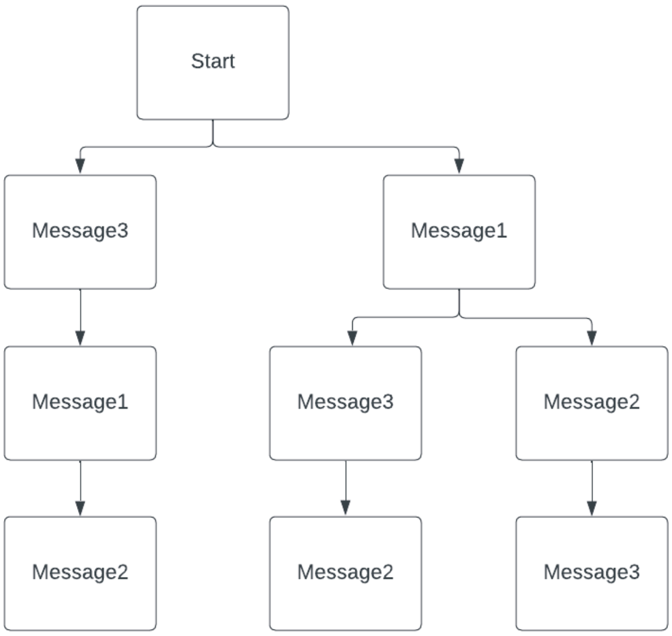
output: “**hey from message1”** followed by “**hey from message2”** due to the Qualifier specifying that bean: getMessage 1 to be injected first followed by getMessage2 that is dependent on it.

Q3.



```
@Bean
@Qualifier("1")
public String getMessage1(){
    System.out.println("hey from message1");
    return "1";
}

@Bean
@Qualifier("2")
public String getMessage2(@Qualifier("3") String data ){
    System.out.println("hey from message2");
    return data;
}

@Bean
@Qualifier("3")
public String getMessage3(){
    System.out.println("hey from message3");
    return "3" ;
}
```

A3.
Three possible outcomes being:



1. “**hey from message1” - “hey from message3” - “hey from message2”**
2. “**hey from message1” - “hey from message2” - “hey from message3”**

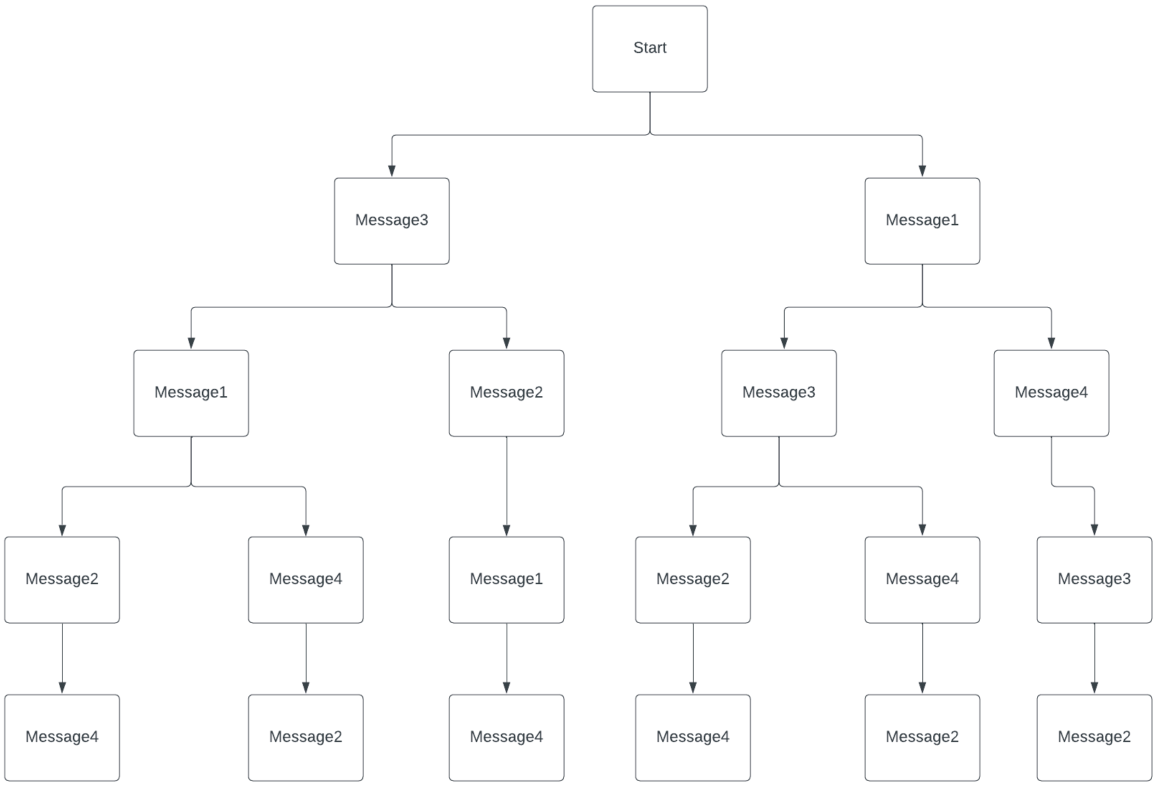
3. “**hey from message3”** - “**hey from message1”** - “**hey from message2”**

Since getMessage2 can't be injected before getMessage1, that leaves getMessage3 and getMessage1 both can be injected first, if 1 is injected that leaves two choices of injection being 3 and 2….

A4.

Adding a Component which does the same as beans but for classes a class with the annotation Component is included into the container.

the question has 6 possible outcomes which are:



1. **“hey from message3” - “hey from message1” - “hey from message2” - “hey from message4”**
2. **“hey from message3” - “hey from message1” - “hey from message4” - “hey from message2”**
3. **“hey from message3” - “hey from message2” - “hey from message1” - “hey from message4”**
4. **“hey from message1” - “hey from message4” - “hey from message3” - “hey from message2”**
5. **“hey from message1” - “hey from message3” - “hey from message2” - “hey from message4”**
6. **“hey from message1” - “hey from message3” - “hey from message4” - “hey from message2”**

A5.

uses the component as a bean that is needed for getMessage1, so the order goes: “**hey from message3”** - “**hey from message2”** - “**hey from mainController”** - “**hey from message1”** which is the only possible output.