Exercise

(Spring Container)

Faisal Alkhrayef

Table of Contents

Q1: 2

1. Expected Output: 2

Q2: 2

1. Expected Output: 2

Q3: 3

1. Expected Output: 3
2. Expected Output: 3
3. Expected Output: 4

Q4: 5

1. Expected Output: 5
2. Expected Output: 6
3. Expected Output: 6
4. Expected Output: 6
5. Expected Output: 6
6. Expected Output: 7

Q5: 8

1. Expected Output: 8

Q1:

```
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";
    }
}
```

1. Expected Output:

   hey from message1

   Explanation: it's added to the container and has no barrier to its execution.

Q2:

```
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 ;
    }
```

1. Expected Output:

   hey from message1

   hey from message2

   Explanation: getMessage2() is dependent on getMessage1()

Q3:

```java
@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" ;
}
```

1. Expected Output:

   `hey from message1`
   `hey from message3`
   `hey from message2`
   Explanation: `getMessage1()` has no barriers to execute, `getMessage3()` must execute before `getMessage2()`

2. Expected Output:

   `hey from message3`
   `hey from message1`
   `hey from message2`
   Explanation: `getMessage3()` has no barriers to execute and it must execute before `getMessage2()`, `getMessage1()` has no barriers to execute as well, lastly `getMessage2()` can execute because its dependent has executed.

3. Expected Output:

```
hey from message3
hey from message2
hey from message1
```

Explanation: `getMessage3()` has no barriers to execute and it must execute before `getMessage2()`, now `getMessage2()` can execute because its dependent has executed, and lastly `getMessage1()` has no barriers to execute.

Q4:

```java
@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" ;
}
```

```java
@Component
public class MainController {

    1 usage
    String data;

    public MainController(@Qualifier("1") String data){
        this.data=data;
        System.out.println("hey from Main controller");
    }

}
```

1. Expected Output:

```
hey from message1
hey from message3
hey from message2
```

hey from Main controller

Explanation: getMessage1() has no dependencies so it can execute, as well as getMessage3() it can execute after it, now getMessage2() can execute that getMessage3() executed before it, and lastly the MainController() can execute as well because getMessage1() executed before.

2. Expected Output:

hey from message1
hey from message3
hey from Main controller
hey from message2

Explanation: Same as the previous output but the MainController() and getMessage2() can be flipped because both of their dependencies have executed.

3. Expected Output:

hey from message3
hey from message1
hey from message2
hey from Main controller

Explanation: As we flipped the last two (dependents) we can flip the dependencies getMessage3() and getMessage1() .

4. Expected Output:

hey from message3
hey from message1
hey from Main Controller
hey from message2

Explanation: Flip the dependents with flipped dependencies

5. Expected Output:

hey from message1
hey from Main controller
hey from message3
hey from message2

Explanation: Let's start this way: dependency 1 > dependent 1 > dependency 2 > dependent 2.

6. Expected Output:

```
hey from message3
hey from message2
hey from message1
hey from Main controller
```

Explanation: now let's flip it: dependency 2 > dependent 2 > dependency 1 > dependent 1.

Q5:

```java

    @Bean
    @Qualifier("1")
    public String getMessage1(MainController mainController){
        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" ;
    }
```

```java
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

1 usage
@Component
public class MainController {

    1 usage
    String data;

    public MainController(@Qualifier("2") String data){
        this.data=data;
        System.out.println("hey from Main controller");
    }

}
```

1. Expected Output:

   hey from message3
   hey from message2
   hey from Main controller
   hey from message1

Explanation: Let’s reverse it, `getMessage1()` needs `MainController()` to execute before it, but the `MainController()` needs `getMessage2()` to execute before it, but `getMessage2()` needs `getMessage3()` to execute before it, and lastly `getMessage3()` has no barriers to its execution.