Definitions

- *** Independent:** An Independent function or controller etc. does not rely on any other component or variable in the Spring Boot container. This means that these un-depend components can execute in any order without waiting for others.

- *** Dependent :** A dependent function or controller relies on another component or variable in the Spring Boot container. Dependent components must wait until the components they depend on have been executed before they can run (become Independent).

- *** Primary:** A primary bean act as the default value in the Spring Boot container. For example, if we set a primary of type "String", any function that requires a parameter of type String will automatically receive this primary value from the Spring Boot container.

-

- *** Qualifier:** A qualifier allows us to specify a particular value to use when there are multiple values of the same type in the Spring Boot container. For example, if we define an integer bean with a qualifier "*" (e.g., @Qualifier("*")), then any function requiring an integer with this specific qualifier will receive it from the Spring Boot 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";
    }
}
```

We added getMessage1 to the Spring Context using the @Bean annotation. the getMessage1 is independent*. The expected output is "hey from message1." Additionally, a string value is stored in the Spring Boot container.

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 ;
    }
```

We modified getMessage1 to have a Qualifier and it still independent. Also, we added getMessage2 to the Spring Context using the @Bean annotation. getMessage2 is dependent* on getMessage1. So, we expect the output to start with "hey from message1." Since getMessage2 depends on getMessage1, it will specifically request the "1" qualifier from the Spring Boot container. The expected output for getMessage2 is "hey from message2." Additionally, a string value related with the qualifier "data" is stored in the Spring Boot container

Potential output sequences:
1. getMessage1 → getMessage2

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" ;
    }
```

Both getMessage1 and getMessage3 are **independent*** and have a **Qualifier***, meaning they can execute in unorderly and return values stored in the Spring Boot container when requested.

However, getMessage2 is **dependent*** on getMessage3, so getMessage2 will witting until getMessage3 has executed.

    Potential output sequences:
2. getMessage1 → getMessage3 → getMessage2
3. getMessage3 → getMessage1 → getMessage2
4. getMessage3 → getMessage2 → getMessage1

Q4

```
@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" ;
}
```

```
@Component
public class MainController {

    1 usage
    String data;

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

Similar to before, but now we have a new component: a class named MainController. The MainController constructor is **dependent*** on getMessage1, so it will store the value linked with the **Qualifier*** "1" in its data variable.

Potential output sequences:
1. getMessage1 → getMessage3 → getMessage2-> MainController
2. getMessage1 → MainController-> getMessage3 → getMessage2
3. getMessage3 → getMessage1 → getMessage2-> MainController
4. getMessage3 → getMessage2 → getMessage1-> MainController

Q5

```
15
16  @Bean
17  @Qualifier("1")
18  public String getMessage1(MainController mainController){
19      System.out.println("hey from message1");
20      return "1";
21  }
22
23  @Bean
24  @Qualifier("2")
25  public String getMessage2(@Qualifier("3") String data ){
26      System.out.println("hey from message2");
27      return data;
28  }
29
30  @Bean
31  @Qualifier("3")
32  public String getMessage3(){
33      System.out.println("hey from message3");
34      return "3" ;
35  }
```

```
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");
    }

}
```

- Similar to before, but now getMessage1 is **dependent*** on
  the MainController constructor, and the MainController constructor
  is **dependent*** on getMessage2. Note that now
  only getMessage3 remains **independent*** in the initial state.

  Potential output sequence:

1. getMessage3 → getMessage2 → MainController → getMessage1

  Since each component is dependent on the next in a strict sequence, we
  only have one possible output order.