Controlling Beans Loading Order by using @DependsOn

The order in which Spring container loads beans cannot be predicted. There's no specific ordering logic specification given by Spring framework. But Spring guarantees if a bean A has dependency of B (e.g. bean A has an instance variable @Autowired B b;) then B will be initialized first. But what if bean A doesn't have direct dependency of B and we still want B to initialize first?

When we want to control beans initializing order

There might be scenarios where A is depending on B indirectly. For example assume A is some kind of events publisher and B is listening to those events. This is a typical scenario of observer pattern. We don't want B to miss any events and would like to have B being initialize before A.

@DependsOn annotation

Example

Following example shows a very basic observer pattern.

EventManager, a facility to register listeners and publishing events

```
public class EventManager {
  private final List<Consumer<String>> listeners = new ArrayList<>();

  @PostConstruct
  public void initialize() {
      System.out.println("initializing: "+this.getClass().getSimpleName());
```

```
  }

  public void publish(final String message) {
      listeners.forEach(l -> l.accept(message));
  }

  public void addListener(Consumer<String> eventConsumer){
      listeners.add(eventConsumer);
  }
}
```

EventPublisher

```
public class EventPublisher {
  @Autowired
  private EventManager eventManager;

  @PostConstruct
  public void initialize() {
      System.out.println("initializing: "+this.getClass().getSimpleName());
      eventManager.publish("event published from EventPublisherBean");
  }
}
```

EventListener

```
public class EventListener {
  @Autowired
  private EventManager eventManager;

  @PostConstruct
  private void initialize() {
      System.out.println("initializing: "+this.getClass().getSimpleName());
      eventManager.addListener(s ->
              System.out.println("event received in EventListenerBean : " + s));

  }
}
```

Defining beans and running main class

```
@Configuration
@ComponentScan("com.piseth.java.school")
public class AppConfig {

  @Bean
  @DependsOn("eventListenerBean")
  public EventPublisher eventPublisherBean() {
      return new EventPublisher();
```

```
  }

  @Bean
  public EventListener eventListenerBean() {
      return new EventListener();
  }

  @Bean
  public EventManager eventManagerBean() {
      return new EventManager();
  }

  public static void main(String... strings) {
      AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
      context.close();

  }
}
```

Output

```
initializing: EventManager
initializing: EventListener
initializing: EventPublisher
event received in EventListenerBean : event published from EventPublisherBean
```

If we don't use @DependsOn, there's no guarantee that EventListener will initialize first:

```
@Configuration
@ComponentScan("piseth.java.school")
public class AppConfig {

    @Bean
   // @DependsOn("eventListenerBean")
    public EventPublisher eventPublisherBean() {
        return new EventPublisher();
    }

    @Bean
    public EventListener eventListenerBean() {
        return new EventListener();
    }

    @Bean
    public EventManager eventManagerBean() {
        return new EventManager();
    }

    public static void main(String... strings) {
```

```
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
        context.close();


    }
}
```

Output

```
initializing: EventManager
initializing: EventPublisher
initializing: EventListener
```