Spring Professional Exam Tutorial v5.0

Question 16

Question 16 - What is the behavior of the annotation @Autowired with regards to field injection, constructor injection and method injection?

`@Autowired` is an annotation that is processed by `AutowiredAnnotationBeanPostProcessor`, which can be put onto class constructor, field, setter method or config method. Using this annotation enables automatic Spring Dependency Resolution that is primary based on types.

`@Autowired` has a property `required` which can be used to tell Spring if dependency is required or optional. By default dependency is required. If `@Autowired` with required dependency is used on top of constructor or method that contains multiple arguments, then all arguments are considered required dependency unless argument is of type `Optional`, is marked as `@Nullable`, or is marked as `@Autowired(required = false)`.

If `@Autowired` is used on top of `Collection` or `Map` then Spring will inject all beans matching the type into `Collection` and key-value pairs as BeanName-Bean into `Map`. Order of elements depends on usage of `@Order`, `@Priority` annotations and implementation of `Ordered` interface.

`@Autowired` uses following steps when resolving dependency:

1. Match exactly by type, if only one found, finish.
2. If multiple beans of same type found, check if any contains `@Primary` annotation, if yes, inject `@Primary` bean and finish.
3. If no exactly one match exists, check if `@Qualifier` exists for field, if yes use `@Qualifier` to find matching bean.
4. If still no exactly one bean found, narrow the search by using bean name.
5. If still no exactly one bean found, throw exception (`NoSuchBeanDefinitionException`, `NoUniqueBeanDefinitionException`, …).

Question 16 - What is the behavior of the annotation @Autowired with regards to field injection, constructor injection and method injection?

`@Autowired` with field injection is used like this:

```
@Autowired
public DbRecordsReader recordsReader;
@Autowired
protected DbRecordsBackup recordsBackup;
@Autowired
private DbRecordsProcessor recordsProcessor;
@Autowired
DbRecordsWriter recordsWriter;
```

- Autowired fields can have any visibility level
- Injection is happening after Bean is created but before any init method (`@PostConstruct`, `InitializingBean`, `@Bean(initMethod)`) is called
- By default field is required, however you can use `Optional`, `@Nullable` or `@Autowired(required = false)` to indicate that field is not required.

Question 16 - What is the behavior of the annotation @Autowired with regards to field injection, constructor injection and method injection?

`@Autowired` can be used with constructor like this:

```
@Autowired
public RecordsService(DbRecordsReader recordsReader, DbRecordsProcessor recordsProcessor) {
    this.recordsReader = recordsReader;
    this.recordsProcessor = recordsProcessor;
}
```

Constructor can have any access modifier (public, protected, private, package-private).

If there is only one constructor in class, there is no need to use `@Autowired` on top of it, Spring will use this default constructor anyway and will inject dependencies into it.

If class defines multiple constructor, then you are obligated to use `@Autowired` to tell Spring which constructor should be used to create Spring Bean. If you will have a class with multiple constructor without any of constructor marked as `@Autowired` then Spring will throw `NoSuchMethodException`.

By default all arguments in constructor are required, however you can use `Optional`, `@Nullable` or `@Autowired(required = false)` to indicate that parameter is not required.

Question 16 - What is the behavior of the annotation @Autowired with regards to field injection, constructor injection and method injection?

`@Autowired` can be used with method injection like this:

```java
@Autowired
public void setRecordsReader(DbRecordsReader recordsReader) {
    this.recordsReader = recordsReader;
}
```

`@Autowired` method can have any visibility level and also can contain multiple parameters. If method contains multiple parameters, then by default it is assumed that in `@Autowired` method all parameters are required. If Spring will be unable to resolve all dependencies for this method, `NoSuchBeanDefinitionException` or `NoUniqueBeanDefinitionException` will be thrown.

When using `@Autowired(required = false)` with method, it will be invoked only if Spring can resolve all parameters.

If you want Spring to invoke method only with arguments partially resolved, you need to use `@Autowired` method with parameter marked as `Optional`, `@Nullable` or `@Autowired(required = false)` to indicate that this parameter is not required.