Advantages and Disadvantages

Lazy Loading

Advantages:

- Much smaller initial load time than in the other approach
- Less memory consumption than in the other approach

Disadvantages:

- Delayed initialization might impact performance during unwanted moments.
- In some cases we need to handle lazily initialized objects with special care, or we might end up with an exception.

Eager Loading

Advantages:

- No delayed initialization-related performance impacts
- Fetch all data

Disadvantages:

- Long initial loading time
- Loading too much unnecessary data might impact performance

Separation of Concerns (SoC)

Concerns are the different aspects of functionality that the software system provides. **Separation of Concerns** (**SoC**) is a design principle that manages complexity by partitioning the software system so that each partition is responsible for a separate concern, minimizing the overlap of concerns as much as possible.

Following the principle involves decomposing a larger problem into smaller, more manageable concerns. SoC reduces complexity in a software system, which reduces the effort needed to make changes and improves the overall quality of the software.

When the DRY principle is followed, and logic is not repeated, a SoC is usually a natural result if the logic is organized properly.

SoC is a principle ...

Benefits:

· Manage complexity and increases maintainability of the module

· Enhance cohesion and reduce coupling between modules

· Enable scalable program that is open to extension

· Allow easy reuse of modules and functionalities

· Reduces fragility of modules

Advantages of Java Config over XML Config

1. Compile-Time Feedback due to Type-checking

2. Ability to integrate Spring with external libraries.

Disadvantage

1. Configuration class cannot be final

2. Configuration class methods cannot be final

3. All Beans must be listed, for big applications, it might be a challenge compared to Component Scanning

Advantages of XML

- XML uses human, not computer, language. XML is readable and understandable, even by novices, and no more difficult to code than HTML.
- XML is completely compatible with Java™ and 100% portable. Any application that can process XML can use your information, regardless of platform.
- XML is extendable.

1. XML is platform independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.

2. XML supports Unicode. **Unicode** is an international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs. This feature allows XML to transmit any information written in any human language.

3. The data stored and transported using XML can be changed at any point of time without affecting the data presentation. Generally other markup language such as HTML is used for data presentation, HTML gets the data from XML and display it on the GUI (graphical user interface), once data is updated in XML, it does reflect in HTML without making any change in HTML GUI.

4. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.

5. XML simplifies data sharing between various systems because of its platform independent nature. XML data doesn't require any conversion when transferred between different systems.

Disadvantages of XML

1. XML syntax is verbose and redundant compared to other text-based data transmission formats such as **JSON**.

2. The redundancy in syntax of XML causes higher storage and transportation cost when the volume of data is large.

3. XML document is less readable compared to other text-based data transmission formats such as JSON.

4. XML doesn't support array.

5. XML file sizes are usually very large due to its verbose nature, it is totally dependent on who is writing it.

What are the advantages of annotations in Java?

Annotations have a lot of advantages over XML, to name a few :

- Static type checking - the compiler will check for you where the annotation (once defined properly) is applicable and how.
- Clean code - it's much easier to see (visually) the meta data defined in annotations.
- It can help them understand the desired output and recognize any recurring patterns, questions, etc. It offers two main benefits: **More accurate output**. Annotated data helps any machine learning solution or application to be more accurate and relevant.

Auto wiring

Good

▪ Save typing

▪ Adding additional beans in your Java code does not require changing configuration xml

Bad

▪ Cannot use auto wiring for String and primitive types for parameters.

▪ Not a good idea to mix auto wiring and explicit configuration (mixing will result in confusion in the future)

▪ Explicit configuration will override auto wiring. ▪ Tools use explicit configuration (not many uses auto wired, why?

In most cases, the answer to the question of whether you should use auto wiring is definitely "no!" Auto wiring can save you time in small applications, but in many cases, it leads to bad practices and is inflexible in large applications. Using by Name seems like a good idea, but it may lead you to give your classes artificial property names so that you can take advantage of the auto wiring functionality. The whole idea behind Spring is that you can create your classes how you like and have Spring work for you, not the other way around.

Advantages of Profiling

1. **Spring Profiles helps to easily set right configurations on right environments**.

Dependency Injection

Dependency injection (DI) is a process whereby objects define their dependencies (that is, the other objects with which they work) only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then injects those dependencies when it creates the bean. This process is fundamentally the inverse (hence the name, Inversion of Control) of the bean itself controlling the instantiation or location of its dependencies on its own by using direct construction of classes or the Service Locator pattern.

Code is cleaner with the DI principle, and decoupling is more effective when objects are provided with their dependencies. The object does not look up its dependencies and does not know the location or class of the dependencies. As a result, your classes become easier to test, particularly when the dependencies are on interfaces or abstract base classes, which allow for stub or mock implementations to be used in unit tests.

The concern is the behavior we want to have in a particular module of an application. It can be defined as a functionality we want to implement.

The cross-cutting concern is a concern which is applicable throughout the application. This affects the entire application. For example, logging, security and data transfer are the concerns needed in almost every module of an application, thus they are the cross-cutting concerns.

Advantages and Disadvantages of AOP

Advantages:

- Complements object orientation.
- Modularizes cross-cutting concerns improving code maintainability and understandability.
  - Reuse of classes and aspects, thanks to modularity
  - Reduced cost of coding thanks to modularity and reuse
  - Shorter code thanks to the ability to have an aspect with the code that would otherwise be implemented (scattered) into several classes
  - Ability to add behavior to a class without introducing in it code unrelated with its main responsibility
  - Ability to redefine the semantic of methods/classes without letting client classes know about it

Disadvantage:

- Not the easiest of concepts to grasp - not as well documented as O-O
- O-O goes far enough in the separation of concerns.
- Since it uses proxy based AOP, only method level advising is supported, doesn’t support field level interception
- Only methods with public visibility will be advised

- Aspects cannot advise other Aspects - Not possible to have aspects to be the target of advice from other aspects.

AOP Comparison

Spring AOP

▪ Weaving occurs during run time. May result in performance cost (runtime overhead).

▪ Proxy based, so can only be executed on methods (cannot be applied to constructors).

▪ Can only be applied to beans.

▪ Not applied to internal method calls

AspectJ

▪ Weaving occurs more during compile time and less during runtime. Less impact on performance.
▪ More control over Join Points

▪ Way more powerful than Spring AOP

▪ Uses annotation

▪ Check that only what you want to be weaved is weaved

. ▪ Extra build time overhead

Advantages of DI

- Makes testing easier by enabling the use of mock objects or stubs.
- Reduces coupling between client and dependency classes.
- Reduces boilerplate code since the initialization of all dependencies is done once by the injector.
- The code is easier to maintain and reuse.

Disadvantages of DI

- DI increases complexity, usually by increasing the number of classes since responsibilities are separated more, which is not always beneficial.
- Your code will be (somewhat) coupled to the dependency injection framework you use (or more generally how you decide to implement the DI pattern)

Aware Interface

Spring Aware interfaces allow you to look into the inner workings of the Spring Framework. Through Spring Aware interfaces, you can access the Spring context, or Spring bean lifecycle events.

Your Spring beans might require access to framework objects, such as Application Context, Bean Factory, and Resource Loader. To gain access, a bean can implement one of the many Aware interfaces of the Spring Framework.

Singleton Bean Advantages

Singleton beans as in the name are singleton. This means that **once a singleton bean is initialized, the same instance will be reused throughout the application context**. This also means that every-time the bean is requested from the application context, the same instance of the bean will be provided.

Singleton Bean disadvantages

Unit testing is more difficult (because it introduces a global state into an application). This pattern reduces the potential for parallelism within a program, because to access the singleton in a multi-threaded system, an object must be serialized (by locking)

We have compiled below a list of reasons or usages for both types of beans. Let us start with singleton beans:

- Useful as "stateless" beans.
- For beans that require expensive resources on start up such as database connections.
- For beans that are used for caching and sharing data throughout your application.
- When an orderly and controlled shutdown is required.

The following scenarios could be more appropriate for prototype beans:

- Stateful beans that fulfill a purpose of a single call or a single use.
- For entities where Spring support is required. For example, when having a list of "Accounts", where each account holds different data. Each account can be a prototype bean. You might need such design if you need to use functionalities from Spring inside each Account instance such as performing calls to other Spring beans. Sometimes you might need such unusual workarounds.