**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**](https://en.wikipedia.org/wiki/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**](https://beginnersbook.com/2015/04/json-tutorial/).

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. Lot of Typos

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.

What happens if we have prototype beans in singleton bean?

When you use singleton-scoped beans with dependencies on prototype beans, be aware that dependencies are resolved at instantiation time. Thus, if you dependency-inject a prototype-scoped bean into a singleton-scoped bean, **a new prototype bean is instantiated and then dependency-injected into the singleton bean**.

- What's the problem with eager loading and lazy loading advantage-disadvantage?

Eager Loading

- Advantage

- data can be retrieved at once, with single join query and no extra n queries required

- useful for case, where we are sure we need associated data

- Disadvantage

- Large memory consumption, as all associated data needs to store in memory

- It requires excessive database access, resulting bad performance result

Lazy Loading

- Advantage

- less database access, and only retrieved what's needed

- performant, as data isn't fetched eagerly as we retrieved only what is required

- Disadvantage

- detachment problem, as data can’t be retrieved after transaction commits

- for data retrieval, fires n queries resulting into (n+1) problem

- DB connection needs to be available

- Framework vs Library (advantage-disadvantage)?

Library

- Advantage

- Light-weight

- faster development

- application can control the flow, easy to debug

- Disadvantage

- We need to know about library to invoke its methods

- if library code is updated, we need to recompile our application

- It depends on may slow our application code

Framework

- Advantage

- Faster development, provides default code, easy to get started

- More secured, flexible and testable (can tweak and add behaviors)

- Is scalable and provides lots of functionalities

- Disadvantage

- Application loses the flow of control, hard to debug and track code flow

- Can't extend or modify all internal behaviors of framework

- Takes time to startup, as it needs to load and setup

- What is Dependency Inversion Principle?

Dependency Inversion adheres to the concepts of programming to abstraction. The class or modules should not be dependent on concretion.

But should invert and rely only on abstraction and not on details. So, the dependency can be passed to it from outside, resulting in loosely coupled system.

- Dependency Injection (advantage-disadvantage)?

- Advantage:

- decrease coupling and adheres to separation of concerns and DRY principle

- (a lot of code changes aren't required to change behaviors)

- adheres to Program to Interface (P2I)

- reusable, flexible, and more testable code

- allows concurrent and independent development

- Disadvantage:

- Errors can be unclear or not visible early as injection occurs at runtime

- as it separates behavior from construction

- enforce coupling with framework, as code can be coupled with dependency injection framework

- Injecting and configuring dependency are extra overhead and complex

- as it demands configuration details (config code or xml)

- creates complexity if there's circular dependency

- can cause ripple effect, one change in configuration may affect another

- IOC Advantage:

- Centralized configuration and hold and manages beans and its lifecycle

- Support addition and invocation of behaviors to beans managed by its container.

- Helps us to write modular and loosely coupled code and takes care of long nested dependency chains

- guides the flow of application code, and provides flexible and faster development

- IOC Disadvantage:

- IOC Requires configuration. We need to provide details for dependencies configuration and need to adapt to the framework toolbox.

- Hard to figure out the flow in the application. As in, what is the actual implementation of the dependency you are looking at.

- Startup time increases as it needs to configure and manage beans at startup.

- Advantage-Disadvantage callbacks in JPA?

- Advantage:

- we get control back to the application code to perform specific task before or after

- useful to intercept JPA operations and carry some business logic

- Disadvantage:

- Doesn't guarantee the operations success and its order.

- Doesn't provide a way to handle the exception

- Why callback method cannot be final or static?

Java classes do not inherit annotations automatically. ORM provider will use reflection to check the annotations on the callback methods and override it. It uses the proxy pattern, to dynamically create the class for an entity or entity listener by extending it. Since, java doesn't allow extending class with final and static method it’s not allowed, as ORM provider can't delegate to the entity listener methods.

- Why should we not use EM and not run queries in callback method?

callback methods are called from the ORM provider after persist, remove, or find is called on entity manager.

And if we use EM to perform some operations with persist, remove or find on new entity, which links back to the original entity, then this can trigger circular callbacks. Also, if we run queries then the modification of it may cause same issue or may modify or fail the queries that were committed originally.

- Is it better to place callbacks on entity class or externally?

It's better to have event callback Outside,

The Entity listener may include programmatic logic to process the event

And, to put the logic within the entity class may not be the good approach, doing this, will clutter entity classes and violate single responsibility

Separating Entity Listener class, we can create it as a context which holds a reference to other services and use them to process the events.

This will separate the event handling logic and the entity class and can perform their own responsibility.

- How merge works with cascade and no-cascade?

merge will attach an entity into the persistence context, if context already contains the entity, it will override its attributes with the provided one.

Then make a copy and return it.

Without cascade the association aren't merged and it leaves as it is.

With cascade, it attaches the association and checks if the associated entity in the context, then override its attributes too.

---

- Spring (advantage-disadvantage)?

Umbrella open-source framework.

- Advantage:

- Promotes Loosely coupled architecture with IOC and DI

- Supports cross-cutting concerns through aspect-oriented programming

- Easily testable application

- Disadvantage:

- Takes time to startup, as need to configure and bootstrap IOC

- Requires lots of configuration before starting

- Ways to achieves DI in spring?

- Constructor, Setter, Field, Method

- XML or Java-Annotation or Mixed (all advantage-disadvantage)?

XML Advantage:

- decoupling of configuration values and application logic

- supports legacy system and backward compatibility

- changes in xml doesn't require code recompilation, only restart or refresh. So, it’s faster

- promotes grouped configuration (so all configurations are at one place, and adheres to separation of concerns)

XML Disadvantage:

- can have typos

- xml can grow large and verbose, and it could be difficult to manage

- moving back-and-forth while debugging on xml-based configuration is hard

Java-Annotation Advantage:

- early typo checking

- code completion and refactoring support

Java-Annotation Disadvantage:

- Configuration could go large and spread across multiple classes and modules

- Each time change in configuration code requires re-compilation, so could be slow for frequent compilation and deployment

- XML, we need to define the dependencies in the order.

No, Spring can read the configuration file and figure out the dependency map between the objects.

So, from the tree structure of its dependency map, it can configure beans as required in the order.

- Is constructor injection better than setter? (Constructor vs Setter vs Method vs Property)?

No, it depends. If the dependency is required or should be immutable,

then the constructor injection, makes sure that the object is constructed with it, and values don't mutate and produce side effects.

Setter injection offers more configurable features, as values can be optional or can be mutable.

And we can also inject it as required dynamically at runtime.

So, we use constructor injection to ensures all mandatory properties have been satisfied, and object is in valid state.

Setter injection is useful if properties are optional and can be used to solve circular dependency problem.

========================

- Profiles (advantage-disadvantage)?

- Advantage:

- allows a way to configure application as per environment, separating the env specific configurations from code

- can switch profile without rebuilding the application

- Disadvantage:

- requires extra configurations, for all separate environment and its pain to manage all specific environments and keep in-sync.

- increase test overhead, need to carry combinatorial tests for all environments

- influence beans loaded in context and other properties, harder to evaluate at runtime

- AOP (advantage-disadvantage)?

- Advantage:

- Decouples the code, achieves separation of concerns and DRY (allows developer to concentrate on business code, instead of cross-cutting concerns)

- centralize or groups the concerns in one place, making it easier to maintain and add functionality

- reduce code complexity and makes code clean and readable

- Disadvantage:

- Debugging is difficult, as classes are advised behind the scenes

- harder to unit test, if its runtime-based weaving

- Weaving at runtime, takes performance hit and with compile time takes extra startup overhead

- A bug in an advice code can create a chain effect to all the functionality that uses it

- Join Point, Pointcut, Advice, Aspect, Weaving

- Aspect:

- A cross-cutting concern or a reusable module. You can have one or more aspects in an application.

A Class that's used to declare that this includes additional methods and attributes for a particular type.

- Join Point: A point where you can plug an aspect in.

- Advice: The action that is performed at a particular join point. It's also used to define the action that should be performed preceding or succeeding a method execution.

- Weaving: Provides you solution to your tangled code. It enables you to link different aspects with other objects of the application.

- Target Object: A target object may be defined as one that is advised by one or more aspects in your application.

- Pointcut: Specifies the weaving rules, i.e., it is used to define the join point where a particular advice can be applied in your application.

- Spring AOP vs AspectJ AOP (advantage-disadvantage)

- @Before, @After, @AfterReturning, @AfterThrowing, @Around ?

- @Before – Run before the method execution

- @After – Run after the method returned a result

- @AfterReturning – Run after the method returned a result, intercept the returned result as well.

- @AfterThrowing – Run after the method throws an exception

- @Around – Run around the method execution, combine all three advice above.

- IOC container is DI.

No, IOC container is responsible for creation, configuration and management of beans.

And as a part of this, for bean creation and configuration, it achieves via Dependency injection.

So, IOC uses DI in its phase to perform its task, but itself is not a DI.

- What's the use of aware interfaces?

- Singleton (advantage-disadvantage)?

- Advantage:

- better memory usage, as only one instance of object and stores globally (single bean per container per bean Name)

- one time initialization i.e., can be eagerly initialized or on demand only once, thus no initialization required every-time when requested

- lifecycle managed by spring IOC as it only needs to keep track of one instance per beans

- Disadvantage:

- Unit testing is more difficult (as it introduces a global state into an application)

- It's difficult to track where they are use, as they are globally accessible and tightly coupled with other objects that are using it.

- They stay for the full lifetime of the application.

- Cyclic dependency is not possible in spring.

No, Cyclic dependency is possible when an object from class is dependent to other object and other object to another, this, creates cyclic dependency problem, to solve this we need to make one setter based injection

- Spring context manages the full lifecycle of all beans.

No, Spring context only manages lifecycle of singleton beans.

- concern vs cross-cutting concern?

Concern can be of specific things, whereas cross-cutting concern targets wider scope of grouped things.

- AspectJ @pointcut (advantage-disadvantage).

- Advantage:

- can target concerns using regular expression, providing much broader options

- AspectJ pointcut can narrow down to specific Join Points to package, method, modifier and arguments.

- The bytecode implementation of pointcut and advice is generated at compiled time.

- Disadvantage:

- Introduces extra compilation overhead in our build process. Runtime weaving is not available

- requires additional dependencies, AspectJ compiler.