# SAYYED SAMEER BASIR (PRODUCT ENGINEERING INTERN – FULL STACK)

# TASK ASSIGNED

# Dependency Injection in Spring Boot

## Introduction to Dependency Injection

Dependency Injection (DI) is a design pattern used in Spring Boot to manage dependencies between different components of an application. Instead of an object creating its dependencies, the Spring framework injects them at runtime. This promotes loose coupling, enhances testability, and improves maintainability.

## Types of Dependency Injection

Spring Boot supports multiple types of dependency injection, but the most commonly used ones are:

1. Setter Dependency Injection

2. Constructor Dependency Injection

## 1. Setter Dependency Injection

Setter Injection involves injecting dependencies through setter methods after the object is created.

### How it Works?

• Spring first creates an object using a default constructor.  
• Then, it calls the appropriate setter method to inject the dependency.

### Advantages of Setter Injection

More flexible since dependencies can be changed at runtime.  
 Useful for optional dependencies where injection may not always be required.  
 Allows modification of dependencies after object creation.

### Disadvantages of Setter Injection

Dependencies are not mandatory, which can lead to uninitialized objects if not properly managed.  
Increases boilerplate code due to multiple setter methods.

### When to Use Setter Injection?

• When dependencies are optional.  
• When we need to change dependencies dynamically at runtime.  
• When working with legacy code where direct modifications to constructors are not feasible.

## 2. Constructor Dependency Injection

Constructor Injection involves passing dependencies through a constructor at the time of object creation.

### How it Works?

• Spring invokes the constructor with required dependencies while creating the bean.  
• This ensures all required dependencies are available immediately.

### Advantages of Constructor Injection

Ensures required dependencies are always present.  
 Promotes immutability by making dependencies `final`.  
 Reduces boilerplate code by eliminating setter methods.  
 Works well for dependency injection with multiple components.

### Disadvantages of Constructor Injection

Less flexible, as dependencies cannot be changed after object creation.  
 If a class has too many dependencies, the constructor can become complex.

### When to Use Constructor Injection?

• When dependencies are mandatory.  
• When working with immutable objects.  
• When following best practices in Spring Boot (recommended approach).

## Comparison Table: Setter Injection vs Constructor Injection

|  |  |  |
| --- | --- | --- |
| Feature | Setter Injection | Constructor Injection |
| Object Creation | Uses default constructor | Uses parameterized constructor |
| Flexibility | More flexible (dependencies can be changed) | Less flexible (dependencies cannot be changed) |
| Optional Dependencies | Good for optional dependencies | Requires all dependencies |
| Code Complexity | More boilerplate (setter methods required) | Cleaner code (no setters needed) |
| Unit Testing | Easier to modify dependencies for testing | Requires constructor mocks in tests |
| Best Practice | Not recommended for required dependencies | Recommended by Spring |

## Which One is Better?

• Constructor Injection is generally preferred in Spring Boot as it ensures dependencies are required and promotes immutability.  
• Setter Injection should only be used when dependencies are truly optional.  
  
Spring’s latest recommendations strongly favor Constructor Injection as the default approach, especially when working with modern Spring Boot applications.

## Conclusion

Dependency Injection is a crucial concept in Spring Boot for managing component dependencies. While Setter Injection provides flexibility, Constructor Injection is the preferred approach due to its benefits in ensuring required dependencies and improving maintainability. Choosing the right method depends on the specific use case, but Constructor Injection is generally the best practice in most scenarios.