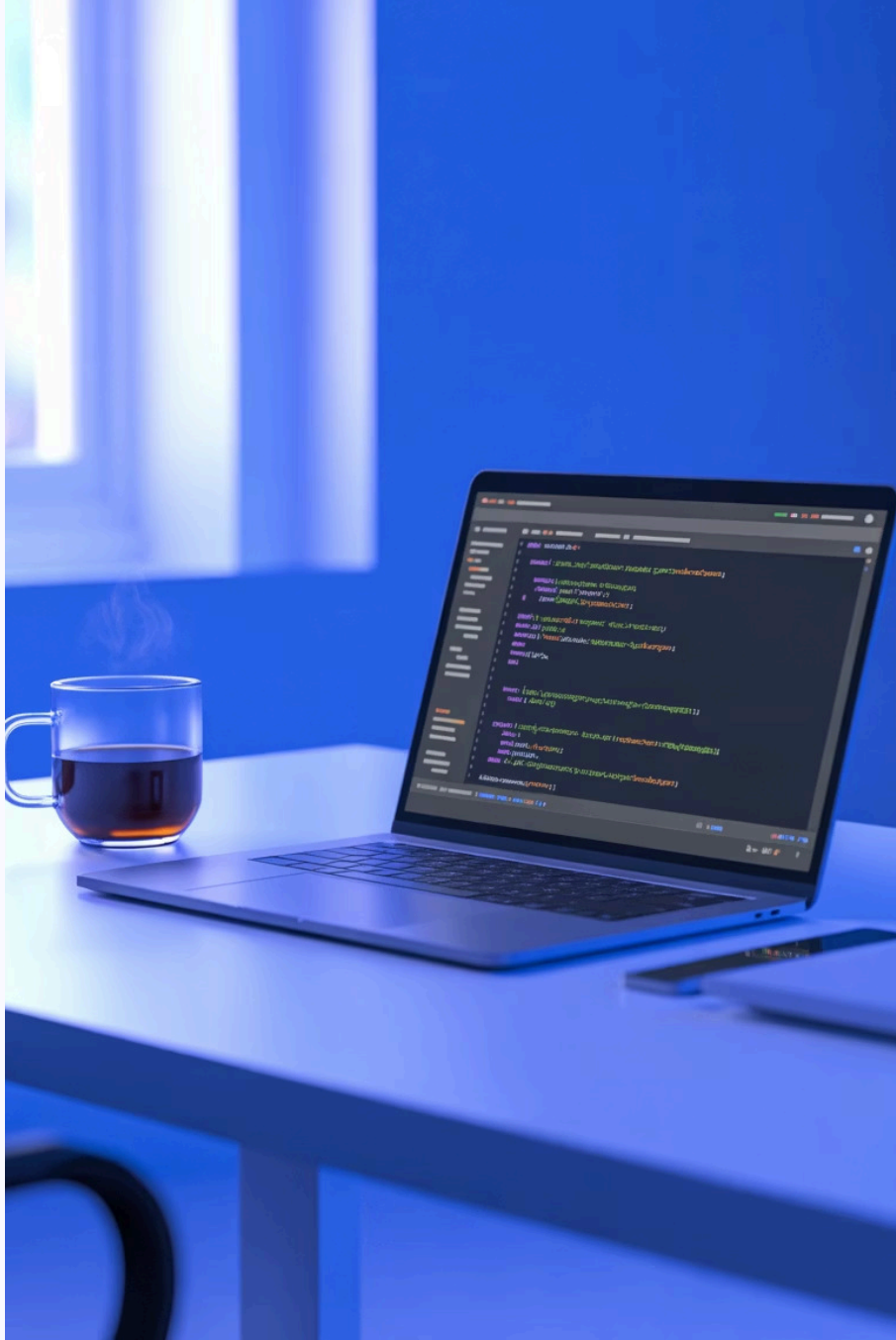
Day-21: What Is @Primary in Spring and Why Is It Used?

DURGASOFT

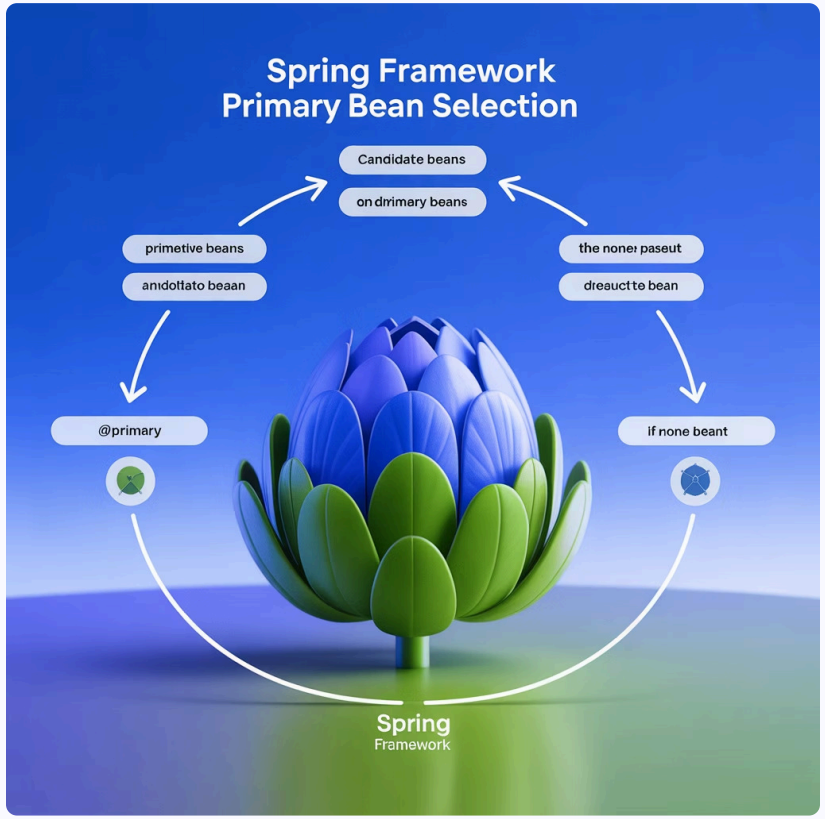


Understanding @Primary Annotation

The @Primary annotation is a powerful tool in Spring Framework that helps you manage situations where multiple Beans of the same type exist in your application context. When Spring encounters multiple Bean candidates during autowiring, it can become confused about which one to inject. This is where @Primary comes to the rescue.

By marking one Bean as @Primary, you're telling Spring: "When in doubt, use this one by default." This significantly reduces the need for using @Qualifier annotations throughout your codebase, making your code cleaner and easier to maintain.

This annotation is particularly valuable in large enterprise applications where you might have multiple implementations of the same interface serving different purposes or environments.



@Primary in Action: Real-World Example

Primary Service

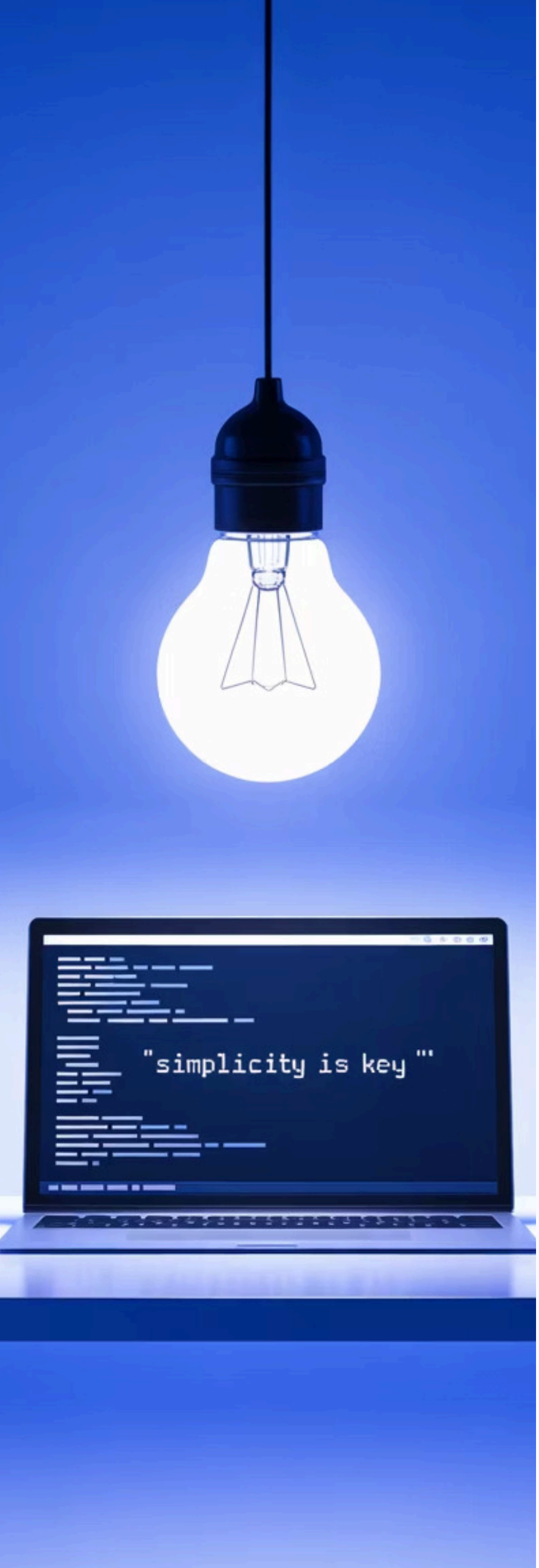
EmailNotificationService marked with @Primary becomes the default choice

Alternative Service

SmsNotificationService available but requires explicit @Qualifier

```
@Service
@Primary
public class EmailNotificationService implements NotificationService {
    // Email notification implementation
}

@Service
public class SmsNotificationService implements NotificationService {
    // SMS notification implementation
}

@Autowired
private NotificationService service;
// Spring automatically injects EmailNotificationService
```

In this example, Spring intelligently injects **EmailNotificationService** automatically because it's marked as @Primary. You only need to use @Qualifier when you specifically want the SMS service instead.



"Default choices reduce complexit y — both in code and in life."

Start Your Spring Framework Journey Today

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

📞 Contact Numbers

9246212143, 8885252627

🌐 Website

www.durgasoftonline.com

📩 Email

durgasoftonlinetraining@gmail.com

Day-22: What Is @PostConstruct in Spring?

DURGASOFT

The Power of @PostConstruct

The @PostConstruct annotation is a lifecycle callback annotation that executes a method **automatically after a Bean is created** and all dependency injection is complete. This makes it perfect for initialization tasks that need to run before the Bean is actually used in your application.

This annotation is extremely useful for several scenarios: loading configuration data, initializing cache systems, establishing database connections, preparing resources, or running any setup code that must execute before the Bean handles its first request.

The beauty of @PostConstruct is its timing - it runs after all dependencies are injected, so you can safely use all autowired fields inside the annotated method.

@PostConstruct: Practical Implementation

01

Bean Creation

Spring creates the CacheLoader Bean

02

Dependency Injection

All @Autowired dependencies are injected

03

@PostConstruct Execution

The init() method runs automatically

04

Bean Ready

Bean is now ready for use in the application

```
@Component
public class CacheLoader {

    @Autowired
    private DataService dataService;

    @PostConstruct
    public void init() {
        System.out.println("Cache loaded successfully!");
        // Load initial data into cache
        // Establish connections
        // Prepare resources
    }
}
```

As soon as Spring creates this Bean, the init() method will run automatically. This is perfect for tasks like loading application settings, preparing database connections, or initializing in-memory services.

"A small setup at the beginning creates smooth performance later."

Join DURGASOFT for Expert Spring Training

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

Call Us Today

9246212143

8885252627

Visit Our Website

www.durgasoftonline.com

Email Us

durgasoftonlinetraining@gmail.com

Day-23: What Is @PreDestroy in Spring?

DURGASOFT

Understanding @PreDestroy Lifecycle Hook

The @PreDestroy annotation is the counterpart to @PostConstruct, and it's equally important for building robust Spring applications. This annotation marks a method that should run **just before a Bean is removed** from the Spring container during application shutdown.

This cleanup phase is critical for preventing resource leaks and ensuring your application shuts down gracefully. Common use cases include closing database connections, stopping background threads, releasing file handles, flushing buffers, and properly terminating external service connections.

Without proper cleanup using @PreDestroy, your application might leave dangling connections, unclosed files, or running threads even after shutdown, which can cause problems in production environments.

@PreDestroy: Ensuring Clean Shutdowns


```
@Component
public class NotificationService {

    private Connection connection;
    private ExecutorService executor;

    @PreDestroy
    public void cleanup() {
        System.out.println("Releasing NotificationService resources...");
        // Close database connections
        if (connection != null) connection.close();
        // Shutdown thread pools
        if (executor != null) executor.shutdown();
    }
}
```

As the application stops, Spring automatically calls the cleanup() method. This ensures your app shuts down cleanly without leaving open database connections, running threads, or other resources hanging.

"A clean shutdown is as important as a perfect startup."

Master Spring Framework with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

📞 Contact Us

9246212143

8885252627

🌐 Online Portal

www.durgasoftonline.com

📩 Email Support

durgasoftonlinetraining@gmail.com

Day-24: What Is @Resource in Spring?

DURGASOFT

@Resource: Name-Based Dependency Injection

The @Resource annotation is part of the JSR-250 specification and provides an alternative approach to dependency injection in Spring. Unlike @Autowired which primarily injects by type, @Resource follows a different strategy: it looks for Beans **by name first**, and only falls back to type matching if no name match is found.

This annotation gives you more precise control over which specific Bean Spring should inject, which is particularly useful in applications where multiple Beans of the same type exist. Since @Resource is a standard Java annotation (not Spring-specific), it helps keep your code more portable across different Java frameworks.

Using @Resource is ideal when you want explicit, name-based injection without mixing too many Spring-specific annotations into your codebase.

@Resource vs @Autowired: A Comparison

@Resource Approach

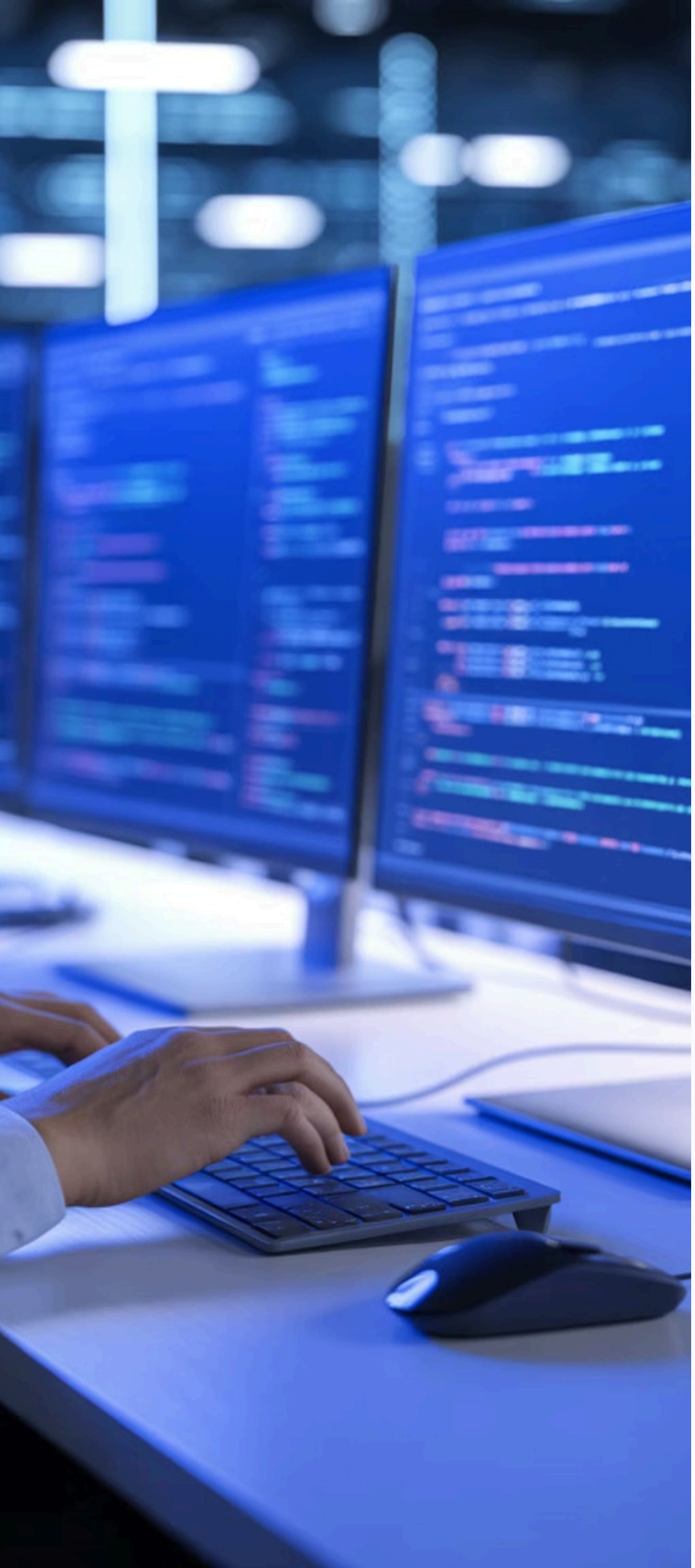
- Searches by name first
- Falls back to type matching
- Standard JSR-250 annotation
- Framework-independent

@Autowired Approach

- Matches by type first
- Uses @Qualifier for names
- Spring-specific annotation
- More commonly used

```
@Service("emailService")
public class EmailService {
    // Email service implementation
}

@Component
public class NotificationClient {

    @Resource(name = "emailService")
    private EmailService service;

    // Spring injects the Bean with exact name "emailService"
}
```

Here, Spring injects the Bean with the **exact name** "emailService". This avoids any confusion or ambiguity when multiple Beans of the same type exist in your application context. The name-based lookup provides clarity and precision.



"Precision in selection leads to perfection in results."

Learn Spring the Right Way with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

Call for Course Details

9246212143, 8885252627

Explore Online

www.durgasoftonline.com

Get in Touch

durgasoftonlinetraining@gmail.com

Day-25: What Is @Inject in Spring?

DURGASOFT

@Inject: Standard-Based Dependency Injection

The @Inject annotation comes from the **Java CDI (Contexts and Dependency Injection)** specification, making it a standard Java annotation rather than a Spring-specific one. This standardization is important because it allows your code to remain portable across different Java frameworks and dependency injection containers.

Spring Framework provides full support for @Inject to give developers flexibility and to encourage the use of industry-standard annotations. It works very similarly to @Autowired, performing type-based injection to find and inject matching Beans from the application context.

Using @Inject is especially beneficial when you're building applications that might need to work with multiple Java frameworks or when you want to follow pure Java standards rather than framework-specific conventions.

@Inject in Practice: Clean Standard Code

Standard Annotation

Part of JSR-330 specification

Spring Support

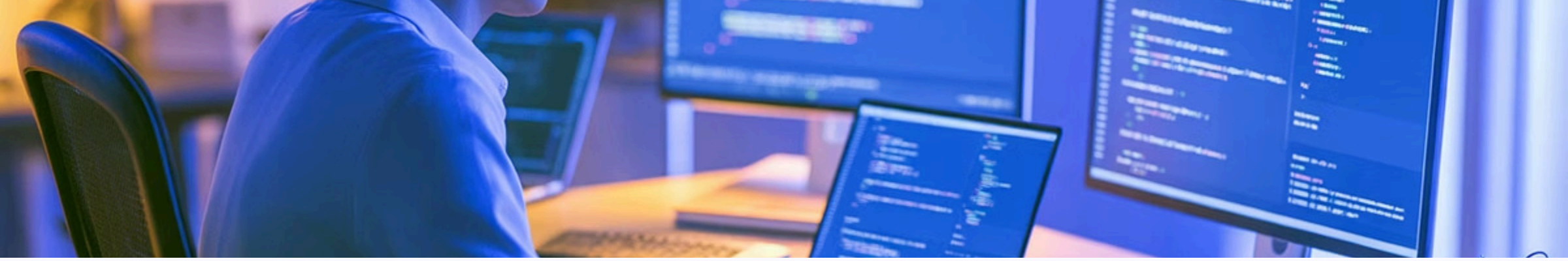
Fully supported by Spring

Framework Independent

Works across Java frameworks

```
@Component
public class OrderController {

    @Inject
    private PaymentService paymentService;

    @Inject
    private InventoryService inventoryService;

    public void processOrder(Order order) {
        // Use injected services
        paymentService.processPayment(order);
        inventoryService.updateStock(order);
    }
}
```

Spring automatically injects the matching Bean based on **type**. This is particularly useful when you want your code to follow Java-standard annotations rather than relying solely on Spring-specific annotations. It keeps your code clean and portable.



"Learn the standards, and you can work with any framework confidently."

Build Your Spring Expertise at DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

1

Contact Us

9246212143

8885252627

2

Visit Website

www.durgasoftonline.com

3

Email Inquiry

durgasoftonlinetraining@gmail.com

Day-26: What Is @Named in Spring?

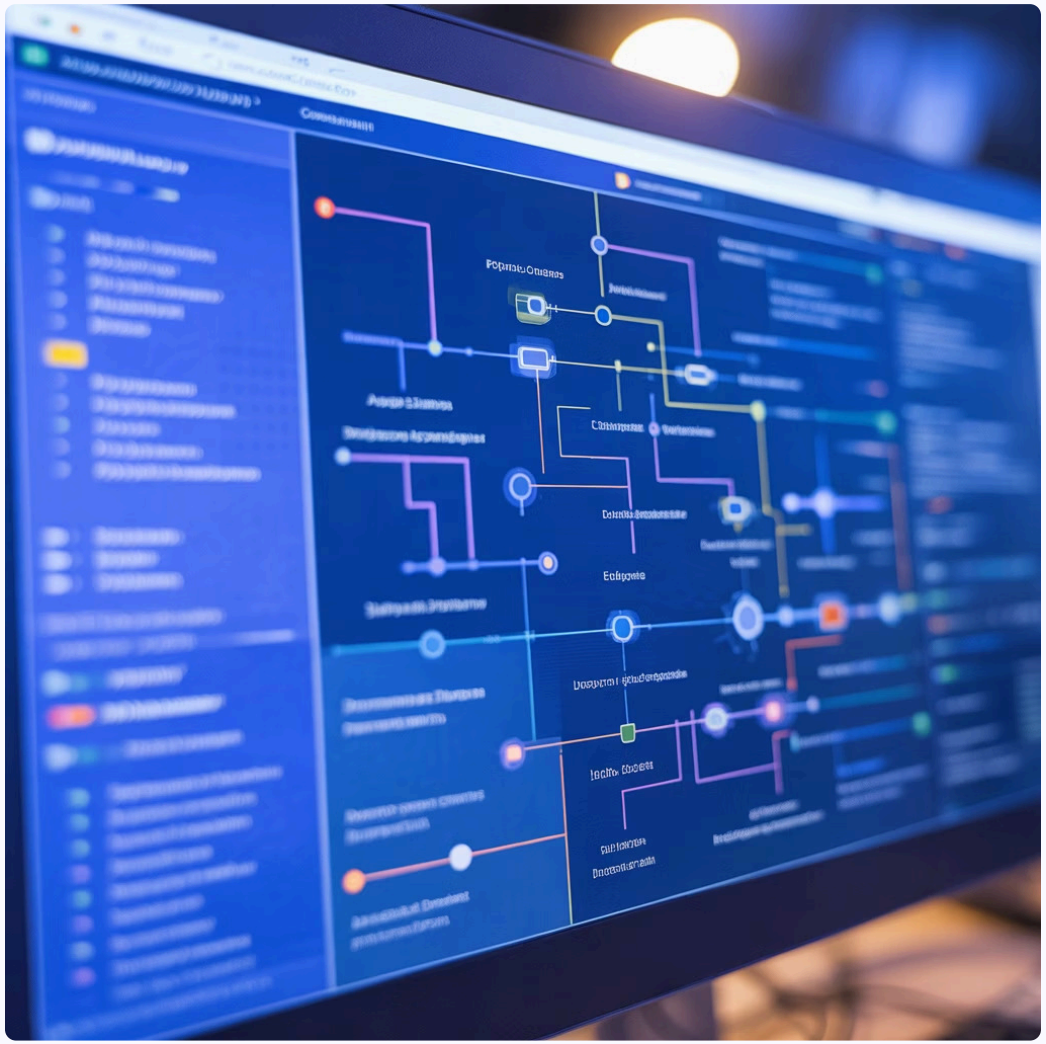
DURGASOFT

@Named: CDI's Alternative to @Component

The @Named annotation is part of the CDI (Contexts and Dependency Injection) standard and serves as an alternative to Spring's @Component annotation. When you mark a class with @Named, Spring detects it during component scanning and creates a managed Bean in the application context.

The key advantage of @Named is that it allows you to specify a **custom Bean name** directly in the annotation, making it easy to identify and reference specific Bean implementations when multiple options exist. This is particularly useful when working with interfaces that have multiple implementations.

Using @Named keeps your code aligned with standard Java annotations rather than Spring-specific ones, which can be beneficial in enterprise environments where code portability and framework independence are valued.



@Named with @Inject: Clean Standard Injection

01

Define Named Bean

Create Bean with custom name using @Named

02

Spring Detection

Component scanning finds and registers the Bean

03

Named Injection

Use @Inject with @Named to inject specific Bean

04

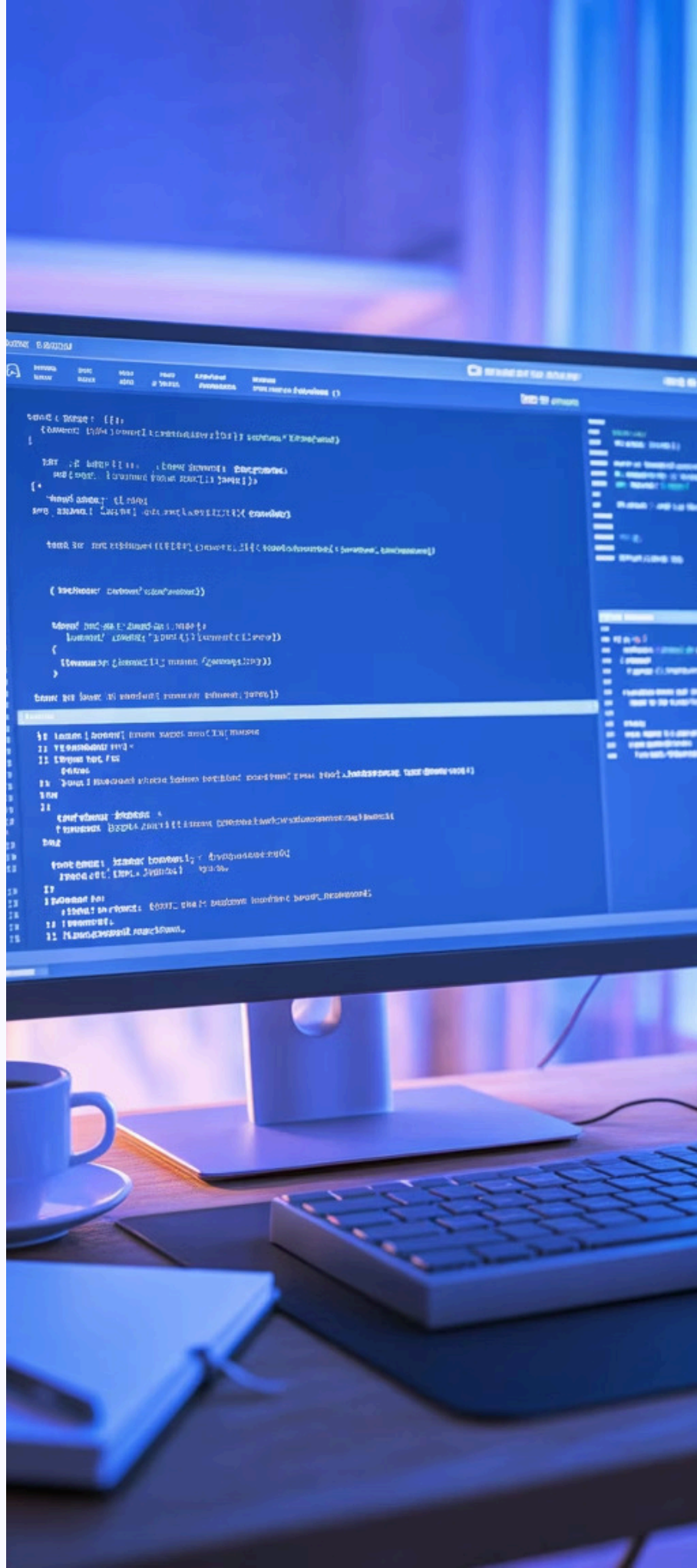
Clean Resolution

Spring injects the exact Bean by name

```
@Named("emailService")
public class EmailService implements MessageService {
    public void sendMessage(String message) {
        System.out.println("Sending email: " + message);
    }
}

@Component
public class NotificationManager {

    @Inject
    @Named("emailService")
    private MessageService service;

    public void notify(String msg) {
        service.sendMessage(msg);
    }
}
```

Spring creates a Bean named **"emailService"** and injects it when explicitly requested. This approach is useful when you want clean, standard Java annotations instead of Spring-specific ones, keeping your code portable and framework-independent.

"The right name brings clarity — in code and in life."



Transform Your Career with DURGASOFT Training

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

Call Today

9246212143

8885252627

Visit Us Online

www.durgasoftonline.com

Email Us

durgasoftonlinetraining@gmail.com

Day-27: What Is @Lookup in Spring?

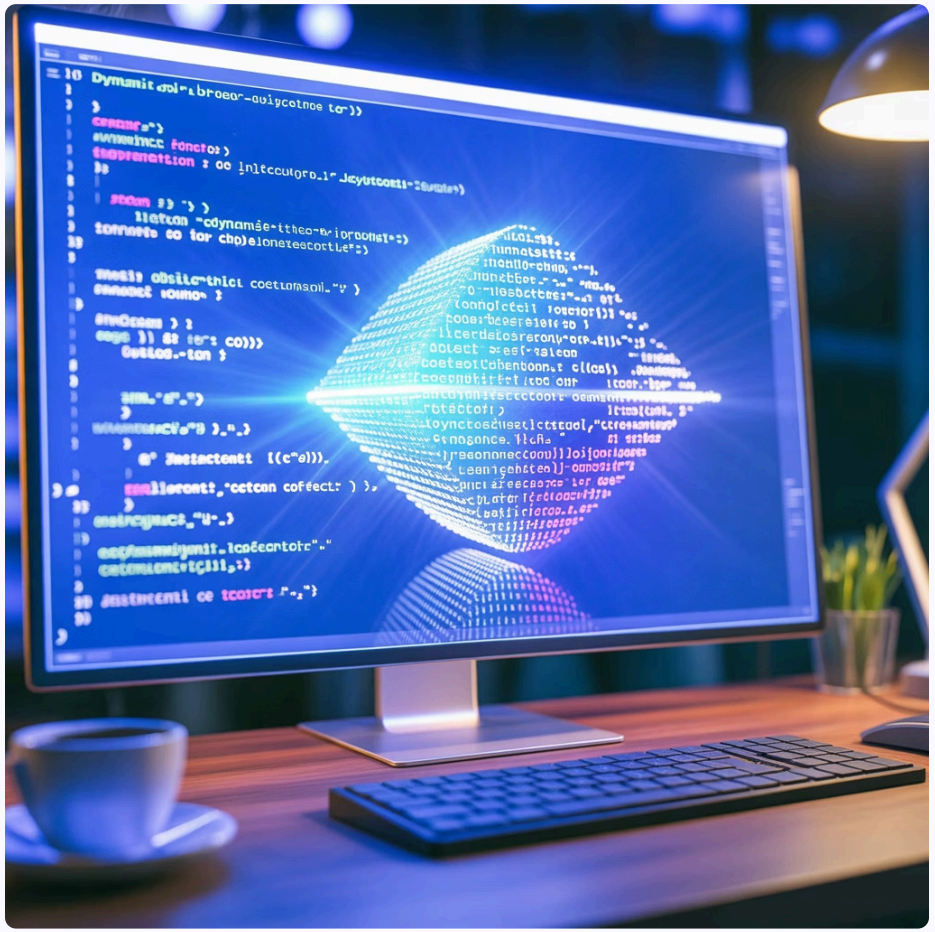
DURGASOFT

@Lookup: Dynamic Bean Instance Creation

The @Lookup annotation solves a unique challenge in Spring: how to inject **prototype-scoped Beans** into **singleton-scoped Beans**. Normally, when a singleton Bean autowires another Bean, it gets the same instance every time. But what if you need a fresh instance on every method call?

That's where @Lookup comes in. It tells Spring to return a **new Bean instance every single time** the annotated method is called, even though the containing Bean is a singleton. Spring achieves this through method injection using CGLIB proxying.

This annotation is particularly valuable for scenarios like generating reports, processing batch jobs, creating temporary services, or any situation where you need stateless, fresh objects for each operation rather than reusing the same instance.



@Lookup: Solving Singleton-Prototype Challenge

The Problem

Singleton Beans can't directly autowire prototype Beans because they get only one instance during creation

The Solution

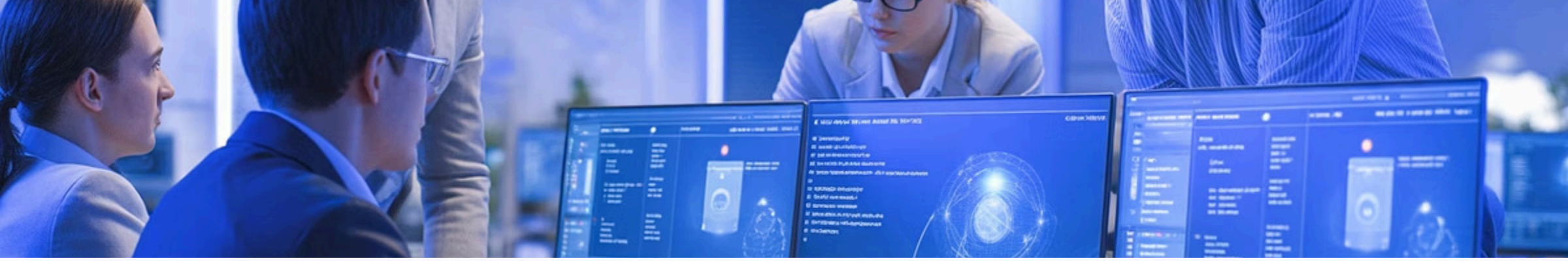
@Lookup tells Spring to override the method and return a new prototype instance every time it's called

The Result

Your singleton Bean can now get fresh prototype instances on demand without breaking singleton behavior

```
@Component
@Scope("prototype")
public class ReportGenerator {
    public void createReport() {
        System.out.println("Generating new report: " + this.hashCode());
    }
}

@Component
public abstract class ReportManager {

    @Lookup
    protected abstract ReportGenerator getReportGenerator();

    public void generateDailyReport() {
        ReportGenerator generator = getReportGenerator(); // New instance
        generator.createReport();
    }
}
```

Every time getReportGenerator() is called, Spring returns a **brand new ReportGenerator object**. This is extremely useful for tasks that need fresh, stateless objects – like generating reports, processing jobs, or creating temporary services with clean state.



"Smart object creation leads to smarter application performance."

Excel in Spring with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

📞 Phone Support

9246212143

8885252627

🌐 Online Learning

www.durgasoftonline.com

📩 Email Contact

durgasoftonlinetraining@gmail.com

Day-28: What Is @EventListener in Spring?

DURGASOFT

@EventListener: Building Event-Driven Systems

The @EventListener annotation enables you to create event-driven architectures in Spring applications. It allows you to write methods that automatically execute when specific Spring events occur, whether they're built-in system events or your own custom application events.

This powerful feature helps you build loosely-coupled systems where different modules can react automatically to important actions without direct dependencies. For example, when a user registers, you could automatically trigger email notifications, logging, analytics tracking, and cache updates – all through events.

Event-driven programming with @EventListener makes your code more modular, testable, and maintainable. Components communicate through events rather than direct method calls, reducing coupling and increasing flexibility.

@EventListener: Responding to Application Events

Event Published

Application or component publishes an event

Spring Dispatches

Event is broadcast to all registered listeners

Actions Performed

Each listener handles the event independently

Listeners Respond

@EventListener methods automatically execute

```
@Component
public class StartupListener {

    @EventListener
    public void handle(ApplicationReadyEvent event) {
        System.out.println("Application is ready to serve!");
        // Perform initialization tasks
        // Load cache, send notifications, log startup
    }

    @EventListener
    public void handleUserRegistration(UserRegisteredEvent event) {
        System.out.println("New user registered: " + event.getUsername());
        // Send welcome email, track analytics, etc.
    }
}
```

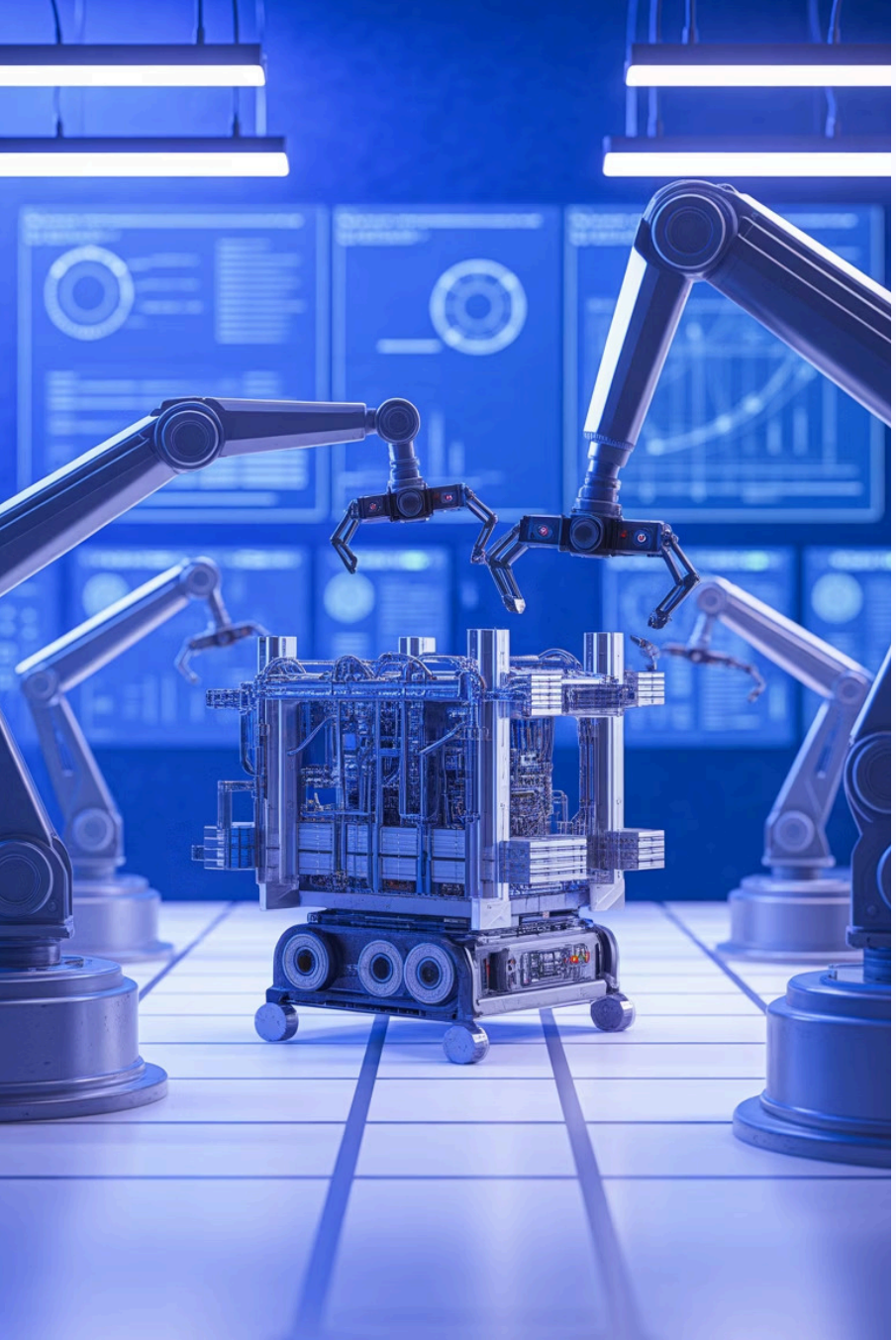
The first method runs automatically when the Spring Boot application becomes fully ready. The second responds to custom user registration events. This pattern is perfect for logging, cache loading, sending notifications, or triggering background tasks whenever important events occur.

"When your system responds automatic ally, your architectu re becomes truly intelligent ."

Advance Your Skills with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

1	2	3
Call Us **9246212143** **8885252627**	Visit Website **www.durgasoftonline.com**	Email Us **durgasoftonlinetraining@gmail.com**



Day-29: What Is Dependency Injection (DI) in Spring?

Constructor vs Setter Injection

DURGASOFT

Constructor vs Setter Injection: Choosing Wisely

Spring Framework supports two primary methods for dependency injection: **Constructor Injection** and **Setter Injection**. Understanding when to use each approach is critical for building robust, maintainable applications.

Constructor Injection is the recommended approach for required dependencies. When you inject dependencies through the constructor, the object is fully initialized at creation time. This ensures the class is always in a valid state and makes it immutable, which is ideal for thread-safe, cloud-ready applications.

Setter Injection is better suited for optional dependencies that might be set later or changed during the object's lifecycle. However, it allows objects to exist in a partially initialized state, which can lead to NullPointerExceptions if not handled carefully.

Constructor vs Setter: Side-by-Side Comparison

Constructor Injection ✓

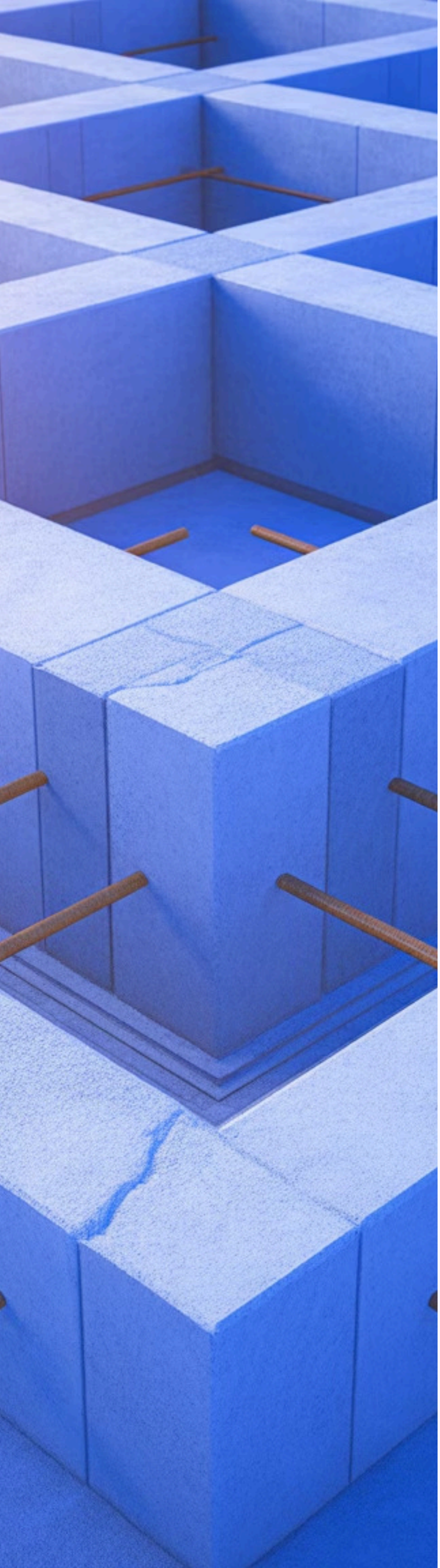
- Required dependencies
- Immutable objects
- Always fully initialized
- Easier to test
- Better for final fields
- **Recommended by Spring**

Setter Injection

- Optional dependencies
- Mutable objects
- Can be partially initialized
- Runtime modification possible
- Legacy code compatibility
- Use when truly optional

```
// Constructor Injection (Recommended)
@Service
public class OrderService {
    private final PaymentService payment;
    private final InventoryService inventory;

    public OrderService(PaymentService payment, InventoryService inventory) {
        this.payment = payment;
        this.inventory = inventory;
    }
}

// Setter Injection (For optional dependencies)
@Service
public class ReportService {
    private EmailService emailService; // Optional

    @Autowired
    public void setEmailService(EmailService emailService) {
        this.emailService = emailService;
    }
}
```

Constructor injection is strongly preferred because it makes the class immutable, ensures all required dependencies are present at creation time, and makes the code easier to test and maintain in production environments.



"Strong foundations come from strong wiring — choose your injection wisely."

Master DI Patterns with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

Call Now

9246212143

8885252627

Visit Portal

www.durgasoftonline.com

Email Inquiry

durgasoftonlinetraining@gmail.com

Day-30: What Is Spring AOP (Aspect-Oriented Programming)?

DURGASOFT

Spring AOP: Separating Cross-Cutting Concerns

Spring AOP (Aspect-Oriented Programming) is a powerful paradigm that helps you separate **cross-cutting concerns** from your core business logic. Cross-cutting concerns are functionalities that span multiple parts of your application – like logging, security checks, transaction management, performance monitoring, and error handling.

Without AOP, you'd need to duplicate the same code across hundreds of methods. For example, adding a log statement before and after every service method would mean copying that logging code everywhere. AOP lets you write this common logic **once** and apply it automatically wherever needed.

This separation makes your code cleaner, more modular, easier to maintain, and less prone to errors. Your business logic stays focused on business problems, while AOP handles the infrastructure concerns.

Spring AOP: Core Concepts and Example

Aspect

A modularized cross-cutting concern (like logging or security)

Join Point

A point in your application where an aspect can be applied (method execution)

Advice

Action taken by an aspect (@Before, @After, @Around)

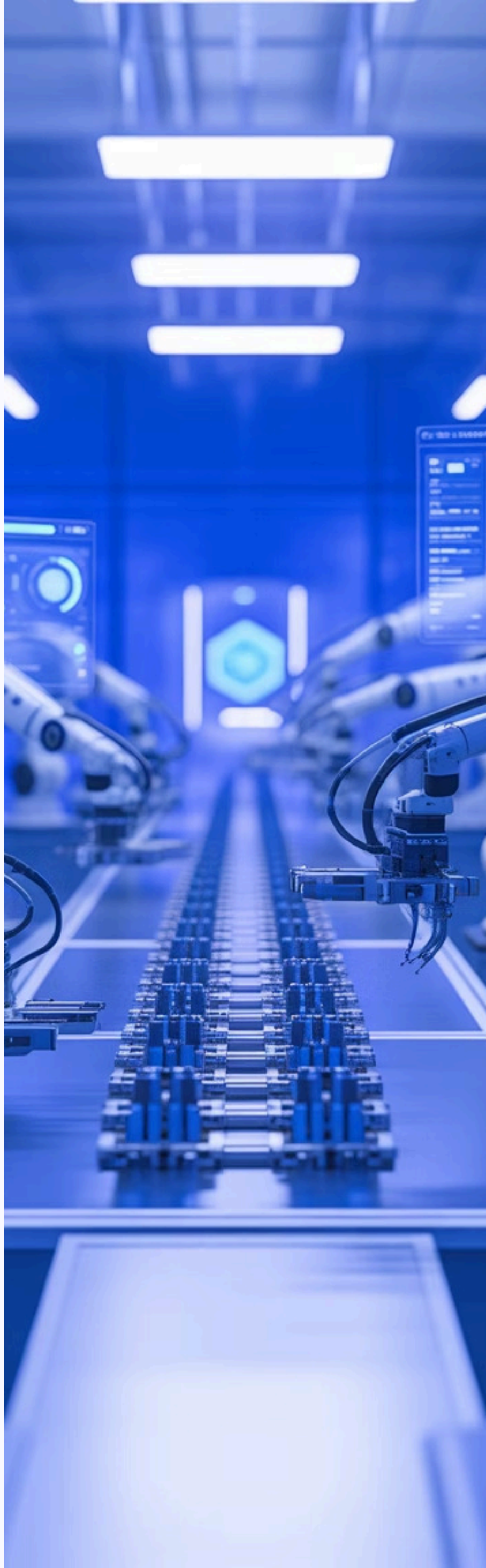
Pointcut

Expression that matches join points where advice should apply

```
@Aspect
@Component
public class LogAspect {

    @Before("execution(* com.durgasoft.service.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Executing: " + joinPoint.getSignature().getName());
    }

    @After("execution(* com.durgasoft.service.*.*(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("Completed: " + joinPoint.getSignature().getName());
    }
}
```

This @Before advice runs **automatically before every method** inside the service package. There's no need to copy logging code into every class – AOP handles it automatically. You can easily add security checks, performance monitoring, or transaction management the same way.

"Work smart: write common logic once and let AOP handle the rest."



Complete Your Spring Journey with DURGASOFT

✨ *If you want online training in SPRING Framework with SPRING BOOT Cloud, join DURGASOFT - India's trusted training institute.*

📞 Connect Today	🌐 Explore Courses
9246212143, 8885252627	**www.durgasoftonline.com**

📩 Get Started

durgasoftonlinetraining@gmail.com

> **"Master Spring Framework and Spring Boot Cloud with India's most trusted online training institute. Join thousands of successful developers who learned with DURGASOFT!"**