

**Design Patterns**

1. **Singleton pattern**
   * Singleton Beans
   * Spring restricts a singleton to one object per IoC container, multiple objects of the same class can exist in a single application if we have multiple containers

**Autowired Singletons**

* + Can create two controller within a single application context and inject a bean of same type into each

When you annotate a class with @Component or its specialized annotations like @Service, @Repository, or @Controller, Spring automatically registers it as a singleton bean in the Spring application context.

1. **Factory Method pattern**

Define an Interface: Create an interface that defines the contract for creating objects.

@Component

**@Scope("prototype")**

public class Car implements Vehicle {

@Override

public void start() {

System.out.println("Car started!");

}

}

1. **Proxy pattern**

the proxy pattern is a technique that allows one object — the proxy — to control access to another object — the subject or service.

1. **Template Method pattern**

For example, when executing a query on a database, the same series of steps must be completed:

1. Establish a connection
2. Execute query
3. Perform cleanup
4. Close the connection

These steps are an ideal scenario for the template method pattern.

**Scopes**

Singleton / Prototype / Request / Session / Application/ Websocket

The request, session, application, and websocket scopes are available only if you use a **web-aware Spring ApplicationContext** implementation (such as XmlWebApplicationContext). If you use these scopes with regular Spring IoC containers, such as the **ClassPathXmlApplicationContext, an IllegalStateException** that complains about an unknown bean scope is thrown.

**Caching**

Hazelcast

- Add dependencies

- Create cache configuration

- Enable and use caching

- @EnableCaching

Evict Cache - LRU, LFU, NONE, RANDOM

**@EnableCaching**

**@CacheEvict**

**JPA**

For artifacts

- H2/HSQL, DERBY all will configures automatically

For like mysql

- Url, Username, Password

By default hibernate as orm in springboot

|  |  |  |
| --- | --- | --- |
| Aspect | CrudRepository | JpaRepository |
| Inheritance | Extends Repository | Extends CrudRepository and  thus indirectly Repository |
| Query methods | Basic CRUD operations | Additional JPA-related methods  for querying and managing entities |
| Batch operations | Basic CRUD operations | Batch operations such as  deleteInBatch and saveAll |
| Use case | Basic CRUD operations without JPA-specific | When you need more  sophisticated querying,  batching, or JPA-specific features |

**Actuator health check**

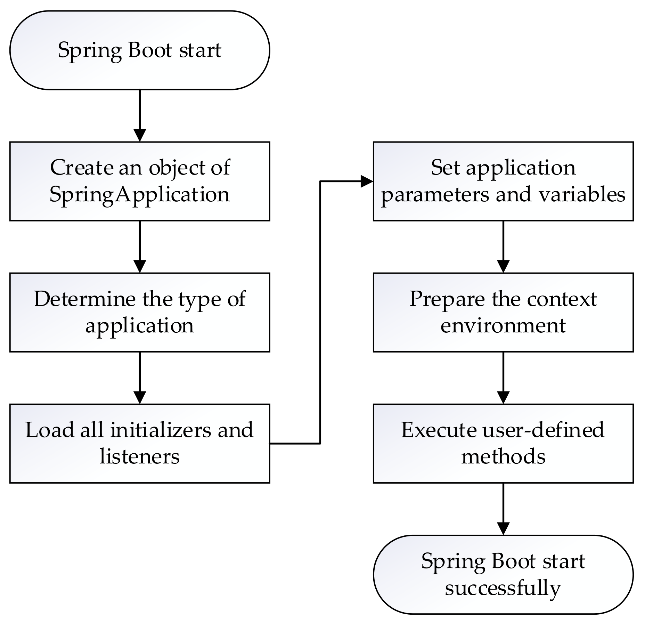
/health info Liquibase logfile shutdown env beans trace

**Spring security**

@EnableWebSecurity / @Secured / @PreAuthorize /

@EnableGlobalMethodSecurity / @AuthenticationPrincipal

@EnableResourceServer / @EnableAuthorizationServer



**Springboot AOP**

**@Aspect** – defince aspect

Join Point – where aspect applied **@Pointcut**

Advice - **@Before,@After, @Around**

**Weaving** – applying aspect. Compiletime, runtime, load time

Target object

**Caching** - spring-boot-starter-cache

@EnableCaching| @Cacheable("addresses") | @CacheEvict

Spring Boot Applications

@SpringBootApplication

- @SpringbootConfig

- @EnableautoConfig

- @ComponentScan

@EnableWebSecurity

@ComponentScan

- @Service

- @Repository

- @Controller

- @RestController

@Configuration

@EnableScheduling

@Async

@Required

@Autowired  
@Transactional - @EnableTransactionManagement

**Types of DI**

* + Constructor based
  + Setter / property
  + Interface injection

**Inversion of control** – receives dependencies from external source (not creating by self)

**Hibernate**

States – Transient / Persistent / Detached / Removed

**Components of Microservices**

**1. Service Discovery**

client side pattern - client check service registry. uses load balancing algm to makes a request

server side pattern - client requests to LB, LB check service registry

**2. Load Balancer**

Scalability — Vertical or Horizontal Scaling when Designing Architectures

Server-side Load Balancing - It is generally a hardware load balancer.

Client-side Load Balancing - The load balancer's logic in the client, it holds the list of servers and decides to which server. also known as software load balancers.

**3. API Gateway**

**4. Service Registry**

**5. Circuit Breaker**

**6. Service Monitoring**

**7. Service Orchestration**

**8. Configuration Server**

**-------------------------------------------------------------------------------------------**

**12 Factors**

**I. Codebase**

One codebase tracked in revision control, many deploys

**II. Dependencies**

Explicitly declare and isolate dependencies

**III. Config**

Store config in the environment

**IV. Backing services**

Treat backing services as attached resources

**V. Build, release, run**

Strictly separate build and run stages

**VI. Processes**

Execute the app as one or more stateless processes

**VII. Port binding**

Export services via port binding

**VIII. Concurrency**

Scale out via the process model

**IX. Disposability**

Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**

Keep development, staging, and production as similar as possible

**XI. Logs**

Treat logs as event streams

**XII. Admin processes**

Run admin/management tasks as one-off processes

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| Spring is an open-source lightweight framework widely used to develop enterprise applications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. |
| The most important feature of the Spring Framework is dependency injection. | The most important feature of the Spring Boot is Autoconfiguration. |
| It helps to create a loosely coupled application. | It helps to create a stand-alone application. |
| To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc. |
| To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |
| Developers need to write boilerplate code for smaller tasks. | In Spring Boot, there is reduction in boilerplate code. |
| Developers have to define dependencies manually in the pom.xml file. | pom.xml file internally handles the required dependencies |

**Java 8**  
**Lambda Expressions** - (instance of the functional interface)

Functional Interfaces - numList.stream().filter(n -> n > 5).sorted().forEach(System.out::println);

(An interface that contains only one abstract method is known as a functional interface)

**Method Reference** - numList.stream().filter(n -> n > 5).sorted().forEach(System.out::println);

**Streams**

**Comparable and Comparator**

class Movie implements Comparable<Movie>

public int compareTo(Movie m)

class RatingCompare implements Comparator<Movie> {

public int compare(Movie m1, Movie m2)

{

**Optional Class**

**Date/Time API**

**Local / Zoned LocalDate/LocalTime** / LocalDateTime API

**Miscellaneous**

**Java 11**

**Oracle to OpneJDK** (java 10, last free)

**New string features**

isBlank, lines, strip, stripLeading, stripTrailing, and repeat.

**New File Methods**

We can use the new **readString** and **writeString** static methods from the **Files class**:

Path filePath = Files.writeString(Files.createTempFile(tempDir, "demo", ".txt"), "Sample text");

String fileContent = Files.readString(filePath);

assertThat(fileContent).isEqualTo("Sample text");

**Collection to an Array**

**var keyword**

**HTTP Client**

HttpClient.newBuilder()

.version(HttpClient.Version.HTTP\_2)

.connectTimeout(Duration.ofSeconds(20))

.build();

HttpRequest httpRequest = HttpRequest.newBuilder()

.GET()

.uri(URI.create("http://localhost:" + port))

.build();

**Running Java Files** - java HelloWorld.java

Java 21

String Template - ‘Hello {your actual variable value} welcome to the GeeksForGeeks!’

Libraries Improvements

- Virtual threads

Performance Improvements

Z Garbage Collection (ZBC) which performs all the expensive works concurrently, without stopping the execution of application threads