Q1.

Answer: Spring Boot simplifies Spring application development by providing auto-configuration, embedded servers, and production-ready defaults to reduce boilerplate and setup time.

Q2.

Answer: Spring Framework is a comprehensive framework, while Spring Boot builds on it to provide auto-configuration, opinionated defaults, and faster development.

Q3.

Answer: Starters are curated dependency descriptors that simplify dependency management by bundling commonly used libraries.

Q4.

Answer: @SpringBootApplication combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.

Q5.

Answer: Auto-configuration works by conditionally configuring beans based on classpath, properties, and existing beans.

Q6.

Answer: It uses conditional annotations like @ConditionalOnClass, @ConditionalOnMissingBean, and @ConditionalOnProperty.

Q7.

Answer: They list auto-configuration classes that Spring Boot loads automatically.

Q8.

Answer: Exclude it using @SpringBootApplication(exclude=...) or spring.autoconfigure.exclude.

Q9.

Answer: Spring Boot provides embedded servers like Tomcat, Jetty, or Undertow, allowing apps to run standalone.

Q10.

Answer: It manages dependency versions via starters and dependency management.

Q11.

Answer: Both store config; YAML is hierarchical and more readable, properties is flat key-value.

Q12.

Answer: Through properties files, environment variables, command-line args, and config servers.

Q13.

Answer: @ConfigurationProperties binds groups of properties; @Value injects single values.

Q14.

Answer: Profiles allow environment-specific configurations like dev, test, prod.

Q15.

Answer: Using profile-specific files or environment variables.

Q16.

Answer: Command-line args > environment vars > application.yml/properties > defaults.

Q17.

Answer: Using vaults, encrypted values, environment variables, and secret managers.

Q18.

Answer: @Controller returns views; @RestController returns JSON/XML directly.

Q19.

Answer: @GetMapping is a shortcut for @RequestMapping(method=GET).

Q20.

Answer: Using Jackson library automatically.

Q21.

Answer: It allows full control over HTTP response including status and headers.

Q22.

Answer: Using @ControllerAdvice with @ExceptionHandler.

Q23.

Answer: It provides global exception handling and cross-cutting concerns.

Q24.

Answer: Using Bean Validation annotations like @NotNull, @Valid.

Q25.

Answer: Via spring-boot-starter-data-jpa and auto-configuration.

Q26.

Answer: Spring Data JPA simplifies JPA repositories with minimal boilerplate.

Q27.

Answer: CrudRepository basic CRUD, Paging adds pagination, JpaRepository adds JPA features.

Q28.

Answer: Using DataSource auto-configuration and connection pools.

Q29.

Answer: HikariCP is a high-performance JDBC connection pool, default for speed and efficiency.

Q30.

Answer: Using @Transactional annotation.

Q31.

Answer: Via spring-boot-starter-security auto-configuration.

Q32.

Answer: JWT is a stateless authentication mechanism using signed tokens.

Q33.

Answer: Using Spring Security, OAuth2, JWT, HTTPS.

Q34.

Answer: Authentication verifies identity; authorization verifies permissions.

Q35.

Answer: Using roles with @PreAuthorize or access rules.

Q36.

Answer: By enabling independent deployment, REST APIs, and messaging.

Q37.

Answer: Spring Cloud provides tools for distributed systems on top of Spring Boot.

Q38.

Answer: Using REST, gRPC, or messaging systems.

Q39.

Answer: Using Spring for Apache Kafka or Spring AMQP.

Q40.

Answer: Using Resilience4j or Spring Cloud Circuit Breaker.

Q41.

Answer: Using unit, slice, and integration tests.

Q42.

Answer: @SpringBootTest loads full context; @WebMvcTest loads only web layer.

Q43.

Answer: It replaces a bean with a mock in the Spring context.

Q44.

Answer: Using MockMvc or TestRestTemplate.

Q45.

Answer: Provides production-ready features like health, metrics, info.

Q46.

Answer: They indicate application health and readiness.

Q47.

Answer: Using Actuator, Prometheus, Grafana.

Q48.

Answer: Using external config files or logging frameworks.

Q49.

Answer: As a JAR, Docker container, or cloud deployment.

Q50.

Answer: Common issues include memory leaks, slow DB calls; fixed via tuning, caching, and profiling.