Spring:

Java Database Connectivity (JDBC):

- RDBMS understand SQL language only.
- JDBC driver converts Java requests in database understandable form and database response in Java understandable form.
- JDBC drivers are of 4 types
    - Type I - Jdbc Odbc Bridge driver:
        - ODBC is standard of connecting to RDBMS (by Microsoft).
        - Needs to create a DSN (data source name) from the control panel.
        - From Java application JDBC Type I driver can communicate with that ODBC driver (DSN).
        - The driver class: sun.jdbc.odbc.JdbcOdbcDriver -- built-in in Java.
        - database url: jdbc:odbc:dsn
        - Advantages: Can be easily connected to any database.
        - Disadvantages: Slower execution (Multiple layers). The ODBC driver needs to be installed on the client machine.
    - Type II - Partial Java/Native driver:
        - Partially implemented in Java and partially in C/C++. Java code calls C/C++ methods via JNI.
        - Different driver for different RDBMS. Example: Oracle OCI driver.
        - Advantages: Faster execution
        - Disadvantages: Partially in Java (not truly portable) Different driver for Different RDBMS
    - Type III - Middleware/Network driver:
        - Driver communicates with a middleware that in turn talks to RDBMS.
        - Example: WebLogic RMI Driver
        - Advantages: Client coding is easier (most task done by middleware)
        - Disadvantages: Maintaining middleware is costlier. Middleware specific to database
    - Type IV:
        - Database specific driver written completely in Java.
        - Fully portable.
        - Most commonly used.
        - Example: Oracle thin driver, MySQL Connector/J, ...

MySQL Programming Steps:

- step 0: Add JDBC driver into project/classpath.
    - Project Properties -> Java Build Path -> Libraries - Classpath -> Add External Jas -> select mysql driver jar -> Ok
- step 1: Load and register JDBC driver class. These drivers are auto-registered when loaded first time in JVM. This step is optional in Java SE applications from JDBC 4 spec.
  //Class.forName("com.mysql.cj.jdbc.Driver");
- step 2: Create JDBC connection using helper class DriverManager.
  //Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/dbname", "root", "manager");
- step 3: Create the statement.
  //Statement stmt = con.createStatement();
- step 4: Execute the SQL query using the statement and process the result.
  1.// String sql = "select query";
  ResultSet rs = stmt.executeQuery(sql);
  2.// String sql = "non-select query";
  int count = stmt.executeUpdate(sql);
- step 5: Close statement and connection.
  //stmt.close();
  con.close();

DAO class:

- In enterprise applications, there are multiple tables and frequent data transfer from database is needed.
- Instead of writing a JDBC code in multiple Java files of the application (as and when needed), it is good practice to keep all the JDBC code in a centralized place -- in a single application layer.
- DAO (Data Access Object) class is standard way to implement all CRUD operations specific to a table. It is advised to create different DAO for different table.
- DAO classes make application more readable/maintainable.

Transaction Management: Transaction is set of DML operations to be executed as a single unit. Either all queries in tx should be successful or all should be discarded. The transactions must be atomic. They should never be partial.

Result Set:

Result Set types: -

- TYPE_FORWARD_ONLY -- default type
  - next () -- fetch the next row from the db. and return true. If no row is available, return false.
- TYPE_SCROLL_INSENSITIVE
  - next() -- fetch the next row from the db and return true. If no row is available, return false.
  - previous() -- fetch the previous row from the db and return true. If no row is available, return false.
  - absolute(rownum) -- fetch the row with given row number and return true. If no row is available (of that number), return false.
  - relative(rownum) -- fetch the row of next rownum from current position and return true. If no row is available (of that number), return false.
  - first(), last() -- fetch the first/last row from db.
  - beforeFirst(), afterLast() -- set ResultSet to respective positions.
  - INSENSITIVE -- After taking ResultSet if any changes are done in database, those will NOT be available/accessible using ResultSet object. Such ResultSet is INSENSITIVE to the changes (done externally).
- TYPE_SCROLL_SENSITIVE
  - SCROLL -- same as above.
  - SENSITIVE -- After taking ResultSet if any changes are done in database, those will be available/accessible using ResultSet object. Such ResultSet is SENSITIVE to the changes (done externally).

Result Set concurrency: -

- CONCUR_READ_ONLY -- Using this ResultSet one can only read from db (not DML operations). This is default concurrency.
- CONCUR_UPDATABLE -- Using this ResultSet one can read from db as well as perform INSERT, UPDATE and DELETE operations on database.

Maven:

- Maven is Java Build Tool.
- pom.xml:
  - pom.xml is heart of Maven.
  - POM - Project Object Model.
  - It is located into root of Maven project.
  - pom.xml holds build details of the project ➔ profiles dependencies build plugins

dependencies:

Third-party jars to be added into the project.

Dependency is uniquely identified by the groupId, artifactId and version.

All jars auto-downloaded from Maven repository and added into project CLASSPATH.

Profiles:

Maven enable building projects in different configurations like dev, test, production, etc.

It enables doing changes in build steps/config for certain profile.

build plugins:

Allows to add user-defined actions in the build process.

Implemented by frameworks for customization in build process.

Build phases:

A build life cycle is divided into sequence of multiple build phases.

Important build phases in default build life cycle.

- validate: Check project pom.xml syntax. Downloads all dependencies (if not present in local repository).
- compile: Compile source code of the project.
- test: Execute given unit tests against the compiled source code using a suitable unit testing framework.
- package: Pack the generated files into given package (jar or war).
- install: Copy the package into the local repository. It can be used in other projects on local machine.
- deploy: Copy the final package to the remote repository for sharing with other developers and projects.

==========

Spring:

- Initially developed by "Rod Johnson" in 2003.

- Early versions of spring were relying on XML config (till Java 1.4).

-Later versions (3.0+) added support of annotation config (from Java 5.0+).

-Spring 3.0 became popular spring version.

Spring is lightweight comprehensive framework that simplifies Java development. "light-weight" - basic version of Spring framework is around 2 MB. "comprehensive" - dependency injection "Simplifies Java development" - Ready-made support/wrappers for different Java technologies, Unit testing, ... Inversion of Control - Dependency injection.

JDBC:

Load & register class (managed by spring)

Create connection (managed by spring)

Create statement (managed by spring)

Execute the query (managed by spring)

Process the result (ResultSet) -- supply SQL statement, parameters and process result.

Close all (managed by spring)

Hibernate

Hibernate configuration (.cfg.xml or coding) (managed by spring)

Create SessionFactory (managed by spring)

Create session (managed by spring)

transaction management (managed by spring)

CRUD operations or Query execution -- user-defined

Cleanup (managed by spring)

- Unified transaction management (local & distributed/global) -- @Transactional Local transactions: Within same database Global transactions - JTA: Across the databases
- Easier Java EE development through Spring Web MVC (Pull) and Web sockets (Push).
- Consistent and "readable" unchecked exceptions - wraps technology-specific exceptions.

Traditional Approach

In a traditional application, the **dependent object** (e.g., `OrderServiceImpl`) is responsible for **creating** and **managing** its dependencies (e.g., `PaymentService`). This creates a tight coupling between the dependent and dependency objects, making the code less flexible, harder to test, and more difficult to maintain.

Inversion of Control

With **IoC**, the **control** of creating and injecting dependencies is inverted. Instead of the dependent class creating its dependencies, an external framework (e.g., Spring) is responsible for:

1. **Creating the dependencies** (e.g., `CreditCardServiceImpl`).
2. **Injecting them** into the dependent object (e.g., `OrderServiceImpl`).

This allows the developer to focus on **business logic** instead of managing the dependencies.

IoC ensures that the framework manages the initialization and wiring of dependencies, so you can focus on implementing business methods and logic.

Core Concept of IoC

The key idea of IoC is to **delegate the control** of creating and managing dependencies to an external framework or container (e.g., Spring IoC container). This enables:

1. **Loose coupling**: Dependencies are no longer tightly coupled to the dependent class.
2. **Better testability**: Dependencies can easily be mocked or replaced for testing.
3. **Flexibility**: Dependencies can be swapped or reconfigured without modifying the dependent class.

Dependency Injection and IoC

- **Dependency Injection (DI)** is a design pattern that implements IoC.

Spring Beans:

Spring beans are Java class objects instantiated by Spring container.

Beans are created and initialized as per user-defined configuration (bean definition).

Spring bean classes are simple Java POJO classes with one or more business logic method.

Application Context:

- Spring container is created while ApplicationContext is created.
- ApplicationContext enable accessing Spring container features in the application.
- ApplicationContext is an interface and have various implementations for different scenarios/applications.
- ApplicationContext
    - ClassPathXmlApplicationContext
        - ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml"); // beans.xml in resources or src dir.
    - FileSystemXmlApplicationContext
        - FileSystemXmlApplicationContext ctx = new FileSystemXmlApplicationContext("/home/nilesh/beans.xml"); // beans.xml is in some folder.
    - AnnotationConfigApplicationContext
        - AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class); // AppConfig is @Configuration class
    - WebApplicationContext (interface)
        - XmlWebApplicationContext
        - AnnotationConfigWebApplicationContext
- ApplicationContext (Spring Container) create all (singleton) beans when application context is loaded.
- ApplicationContext interface itself is inherited from BeanFactory interface.
    - It provides basic facility of Beans loading and initialization.
- Typical spring applications have single ApplicationContext.

Dependency Resolution Process:

- The ApplicationContext is created and initialized with configuration metadata (XML or annotations) that describes all the beans.

- For each bean, its dependencies are expressed in the form of properties or constructor arguments. These dependencies are injected when the bean is actually created.
- Each property or constructor argument can be a value (primitive type) or reference of another spring bean.
- The value is converted from its specified format to the actual type of that property or constructor argument.

Spring Boot:

Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.

- Framework (Not a completely new framework -- rather it is abstraction on existing spring frameworks)
- Ease bootstrapping - Quick Start (Rapid Application Development)
- New applications - Not good choice for legacy applications

Abstraction/integration/simplification over existing spring framework/modules.

- Spring core
- Spring Web MVC
- Spring Security
- Spring ORM

NOT replacement or performance improvement of Spring framework Not replacement, but it is abstraction.

Underlying same Spring framework is working -- with same speed.

Spring Boot = Spring framework + Embedded web-server + Auto-configuration - XML configuration - Jar conflicts.

Why Spring Boot:

Provide a radically faster and widely accessible "Quick Start". Very easy to develop production grade applications. Reduce development time drastically. Increase productivity. Spring Boot CLI Spring Initializer (https://start.spring.io/) IDE support - Eclipse based Spring Tools IDE (STS) Build tools - Maven/Gradle support.

Managing versions of dependencies with starter projects:

Set of convenient dependency descriptors

Hierarchically arranged so that minimal dependencies to be added in application

e.g. spring-boot-starter-web

- spring-web
- spring-webmvc
- spring-boot-starter
    - spring-core
    - spring-context
    - spring-boot-autoconfigure
    - spring-boot-starter-logging

log4j-to-slf4j

jul-to-slf4j

spring-boot-starter-json

jackson-databind

spring-boot-starter-tomcat

tomcat-embed-core

tomcat-embed-websocket

jakarta.el

hibernate-validator

No extra code generation (boiler-plate is pre-implemented) and No XML config.

Minimal configuration required.

Ready to use in-memory/embedded databases, ORM/JPA, MVC/REST, ...

Opinionated configuration, yet quickly modifiable for different requirements.

Easy/quick integration with standard technologies/frameworks.

Examples:

Web applications – Tomcat

ORM -- Hibernate

JSON -- Jackson

Test – Junit

You can change the configuration/technologies by adding alternates on pom.xml (classpath). Provide lot of non-functional common features (e.g. security, servers, health checks, ...). Easy deployment and Containerisation. Embedded Web Server for web applications.

- @SpringBootApplication = @ComponentScan + @Configuration + @EnableAutoConfiguration
- @ComponentScan
    - Auto detection of spring stereo-type annotated beans e.g. @Component, @Service, @Repository, @Configuration, @Controller, @RestController, ...
    - By default, basePackage is to search into current package (and its sub-packages).
    - @ComponentScan ("other. package") can be added explicitly to search of beans/config into given package.
- @Configuration
    - Spring annotation configuration to create beans.
    - Contains @Bean methods which create and return beans.
    - @Configuration classes are also auto-detected by @ComponentScan
- @EnableAutoConfiguration annotation
    - Intelligent and automatic configuration.
    - Auto-configuration class are internally Spring @Configuration classes.
    - @Conditional beans:
        - @ConditionalOnClass
        - @ConditionalOnMissingBean

Spring Bean Life Cycle

- Life cycle of a java object is governed by the container that creates the object.
- Container do call certain methods the object to initialize it or give some information.
- Java applets are executed by JRE plugin in the browser (referred as applet container). It used to call methods of applet.
    - init(), start (), paint (), stop (), destroy ().
- Java servlets are executed by web container in the Java web server. It calls the certain methods of the servlet object.
    - init(), service (), destroy ()
- The life cycle methods are callback methods i.e. they are implemented by the programmer and are invoked by the container.
- init() method is called by the container and all container services are accessible from there. This is not possible while construction of the object.

Explanation:

- If @Configuration classes are in same package they are auto-detected due to @SpringBootApplication (@ComponentScan).

- All @Bean are by default singleton beans and hence created as soon as ApplicationContext is created (by SpringApplication.run()).
- ApplicationContext can be accessed in any Spring bean using ApplicationContextAware interface.
- ApplicationContext is used to access the beans using ctx.getBean() method.
- Spring bean provides various callback interfaces.

Reference: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-lifecycle

- Possible bean initialization callbacks

  InitializingBean interface is legacy way of Spring bean initialization callback. Not recommended in modern applications.

  XML configuration relies on <bean ... init-method="initMethodName"/>.

  Annotation config usually follows JSR-250 annotation @PostConstruct.

BeanPostProcessor:

- To perform certain actions on mutliple beans initialization, custom BeanPostProcessor is used.
- There are multiple pre-defined BeanPostProcessor.
  AutowiredAnnotationBeanPostProcessor
  InitDestroyAnnotationBeanPostProcessor
  BeanValidationPostProcessor

Bean creation:
1. Constructor
2. Fields/setters initialized (DI)
3. BeanNameAware.setBeanName()
4. ApplicationContextAware.setApplicationContext()
5. CustomBeanPostProcessor.postProcessBeforeInitialization()
6. @PostConstruct method invoked
7. InitializingBean.afterPropertiesSet()
8. CustomBeanPostProcessor.postProcessAfterInitialization()
9. Bean is ready to use

Bean destruction:
1. @PreDestroy method invoked
2. DisposableBean.destroy()
3. Object.finalize()
4. Bean will be garbage collected

Stereo-type annotations:

- Spring container can auto-detect certain beans (without defining as @Bean method or XML config). It also manages their bean life cycle.
- Following annotations are auto-detected by Spring container using @ComponentScan.
  - @Component -- general-purpose (no special significance) spring bean.
  - @Service-- Spring beans containing business logic
  - @Repository-- Spring beans handling data/database connectivity

    - @Controller-- Spring beans handling navigation, user-interaction in Spring web-mvc applications.
    - @RestController -- It is @Controller used for REST services.
    - @Configuration -- To be used for spring annotation config (not to be used as spring bean).
- Stereo-type annotations are always written at class level.
- Using @ComponentScan, all classes with above annotations in given package and its sub-packages will be instatiated as Spring beans.
- If no package is given, all classes in current package will be scanned for stereo-type annotations.
- This behaviour can be customized using includeFilters and/or excludeFilters.
- XML configuration equivalent of @ComponentScan <context:component-scan>.

Auto-wiring:

- Auto-detection of dependencies and injecting them into dependent objects is also referred as "Autowiring".
- Auto-wiring can be done in XML config using <bean id="..." class="…"autowire="default|no|byType|byName|constructor" .../>.
    - default/no = By default auto-wiring is disabled.
    - byType = auto wire(connect) the bean with matching type (when single bean of matching type is available).
    - byName = auto wire(connect) the bean with matching name.
    - constructor = auto wire(connect) the bean into constructor of dependent bean.

Auto-wiring is done into Java config using spring annotation @Autowired.

@Autowired can be used at

Setter level: setter-based DI

Constructor level: constructor-based DI

Field level: field-based DI

Constructor based @Autowired DI is preferred over Setter based DI & Field based DI.

From Spring 5, single argument constructors have @Autowired implicitly.

@Autowired resolution process:

- @Autowired finds bean of corresponding field "type" and assign it.
- If no bean is found of given type, it throws exception.
- @Autowired(required=false): no exception is thrown, auto-wiring skipped.
- If multiple beans are found of given type, it try to attach bean of same name. If no such bean is found,
- then throw exception.
- If multiple beans are found of given type, programmer can use @Qualifier to choose expected bean.
- @Qualifier can only be used to resolve conflict in case of @Autowired.
- If multiple beans are found of given type, one of the bean can be declared as @Primary. If no
- @Qualifier is mentioned, then @Primary bean will be attached (and no exception is produced).
- @Autowired --> byType, byQualifier (@Qualifier), byName.
- @Resource resolution: DI byName, byType, byQualifier.

Spring Bean Scopes:

- Using a bean definition (@Bean or Stereo-type annotations + Other config), one or more bean objects can be created.
- The bean scope can be set in XML or annotation.

  <bean id="___" class="___" scope="singleton|prototype|request|session" />

  @Scope("singleton|prototype|request|session") on @Bean / @Component ...

- Spring 5 supports six different bean scopes. Four scopes are valid only if you use a web-aware ApplicationContext.

Scope	Description
singleton	(Default) A single object instance for each Spring IoC container.
prototype	Any number of object instances (new instance for each access).
request	A single object for current HTTP request.
session	A single object for current HTTP session.
application	A single object for current application i.e. ServletContext.
websocket	A single object for current websocket.

Singleton:

- Single bean object is created and accessed throughout the application.
- BeanFactory creates object when getBean() is called for first time for that bean.
- All singleton bean objects are created when ApplicationContext is created.
- For each sub-sequent call to getBean() returns same object reference.
- Reference of all singleton beans is managed by spring container.
- During shutdown, all singleton beans are destroyed (@PreDestroy will be called).

Prototype:

- No bean is created during startup.
- Reference of bean is not maintained by ApplicationContext.
- Beans are not destroyed automatically during shutdown.
- Bean object is created each time ctx.getBean() is called.

HTTP protocol:

- HTTP -- Hyper Text Transfer Protocol.
- Connection-less protocol.
- State-less protocol.
- Request-response model.
- Web server is program that enable loading multiple web applications in it.
- Web application is set of web pages (static or dynamic), which are served over HTTP protocol.
- Client makes request by entering URL, click submit, or click hyper link.
- URL: http://server:port/appln/resource:
  - http: protocol/scheme
  - server: machine name or IP address
  - port: default 80
  - URI: /appln/resource
- Request Headers:
  - Server/Host: server name/ip + port
  - User-Agent: Browser type/version
  - URI
  - HTTP version: 1.0 or 1.1

- Content-Type: Type of data in Request body -- application/json, text/...
- Length: Number of bytes in Request body

- Method:
  - GET: Get the resource from the server.
    - Request sent when URL entered in address bar, hyper-link is clicked, html form with method=get is submitted.
    - The data (in html form) is sent via URL.
    - Not secured (because data visible in URL).
    - Faster.
  - POST: Post data to the server.
    - Request sent when html form with method=post is submitted.
    - The data (in html form) is sent via request body.
    - More secure
  - HEAD: Send response headers only.
    - No response data is sent to the client.
  - PUT: Put/upload a resource on server.
  - DELETE: Delete a resource from the server.
  - TRACE: Tracing/Information logging
  - OPTIONS: To know which request methods are supported for the resource.
- Cookies, ...

- Request Body: JSON, Form-Data, or Other.
- Response Headers
  - Status: Code/Text
    - 1xx: Information
    - 2xx: Success e.g. 200 (Ok), 201 (Created), ...
    - 3xx: Redirection e.g. 302
    - 4xx: Client errors e.g. 404 (Not found), 403 (Forbidden), ...
    - 5xx: Server errors e.g. 500 (Internal server error), ...
  - Content-Type: Type of data in Response body
    - text/... : plain, html, xml
    - image/...: png, jpeg, gif, svg
    - audio/...: mp3, wav

Spring Web MVC:

- MVC is a design-pattern / architeture-style.
- Divide application code into multiple relevant components to make application maintainable and extendable.
  - M: Model: Data of the application.
  - V: View: Appearance of data.
  - C: Controller: Interaction between business logic & views.
- Spring MVC components
  - Model (DTO): POJO classes holding data between view & controller.
  - View: JSP** or Thymeleaf or Freemarker pages
  - Controller: Spring Front Controller i.e. DispatcherServlet
  - User defined controller: Interact with front controller to collect/send data to appropriate view, process with service layer.

Spring Web MVC Flow:

- DispatcherServlet receives the request.
- DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to
- the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
- DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.
- HandlerAdapter calls the request handler method of @Controller.
- Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
- DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View Path mapped to
- View name.

- DispatcherServlet dispatches the rendering process to View class / View Engine.
- It renders view with Model data and returns the response.

Request handler Methods

- @RequestMapping attributes
  - value/path = "url-pattern"
  - method = GET | POST | PUT | DELETE
  - params/header = ... (map request only if given param or header is present).
  - consumes = ... (map request only if given request body type is available).
  - produces = ... (produce given response type from handler method)
- One request handler method can be mapped to multiple HTTP request methods (get, post, ...).
- One request handler method can be restrict to set of HTTP request methods.
- To restrict handler method to single request method, shorthand annotations available.
  - @GetMapping --> @RequestMapping(method="GET")
  - @PostMapping --> @RequestMapping(method="POST")
  - @PutMapping --> @RequestMapping(method="PUT")
  - @DeleteMapping --> @RequestMapping(method="DELETE")

REST web services: REST stands for REpresentation State Transfer.

- Object characteristics: State, Behaviour & Identity.
  - State: Values of data members/fields.
- Object state can be transferred (from server to client & vice-versa) in any format like JSON or XML.
- It is a Protocol to invoke web services from any client (with internet connection).
- Client can use any platform/language.
- REST is lightweight (than SOAP).
  - No stub and proxy classes.
  - No XML grammer (xsd) checks.
- REST works on top of HTTP protocol.
- It uses HTTP protocol request methods.
  - GET: to get records.
  - POST: to create new record.
  - PUT: to update existing record.
  - DELETE: to delete record.
- REST services are stateless:
  - Each REST request is independent of another.
  - Request should include all required inputs and produce expected output.

Spring Boot REST services

- Spring REST is subset of Spring Web MVC.
- Spring boot auto-configures WebMvc and messageConverters automatically.
- REST request handlers are implemented similar to web mvc request handlers.
  - Implemented using @Controller
    - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping or @RequestMapping
    - @RequestBody, @ResponseBody / ResponseEntity<>
    - @PathVariable, @RequestParam
  - Can be implemented using @RestController
    - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping or @RequestMapping
    - @RequestBody, ResponseEntity<>
    - @PathVariable, @RequestParam

Response Entity: Response Entity object is used to control response status as well as response body.

If ResponseEntity<> is return type, no need to use @ResponseBody explicitly.

Important methods:

ResponseEntity.ok(body) --> ResponseEntity object

status = 200

body = given object json representation

ResponseEntity.notFound() --> HeaderBuilders object (not to be returned from method)

status = 404

ResponseEntity.notFound().build() --> ResponseEntity object

status = 404

body = empty response body

ResponseEntity.notFound().body(obj) --> ResponseEntity object

status = 404

body = given object json representation

ResponseEntity.noContent() --> HeaderBuilders object (not to be returned from method)

status = 204

ResponseEntity.noContent().build() --> ResponseEntity object

status = 204

body = empty response body

ResponseEntity.noContent().body(obj) --> ResponseEntity object

status = 204

body = given object json representation

ResponseEntity.created(uri) --> BodyBuilder object (not to be returned from method)

status = 201

ResponseEntity.created(uri).build() --> ResponseEntity object

status = 201

body = empty response body

ResponseEntity.created(uri).body(obj) --> ResponseEntity object

status = 201

body = given object json representation

ResponseEntity.internalServerError() --> HeaderBuilders object (not to be returned from method)

status = 500

ResponseEntity.internalServerError().build() --> ResponseEntity object

status = 500

body = empty response body

ResponseEntity.internalServerError(). body(obj) --> ResponseEntity object

status = 500

body = given object json representation

ResponseEntity.status(code) --> BodyBuilder object (not to be returned from method)

status = given code

ResponseEntity.status(code). body(obj) --> ResponseEntity object

status = given code

body = given object json representation

Spring REST Internals:

- Spring REST is subset of Spring MVC.
- Like Spring MVC
    - Each request first encountered by Front controller.
    - The request url is mapped to Request handler method (in controller) by HandlerMapping bean.
    - This request handler method is executed by HandlerAdapter bean.
- To deal with @RequestBody and @ResponseBody, handler-adapter bean creates RequestResponseBodyMethodProcessor bean.
- This bean internally uses Jackson converter (MappingJackson2HttpMessageConverter bean) to convert request body json to required Java object (as given in
- method argument).
- Then request handler method is executed that may call service and dao layer. It produces response Java object.
- Again, RequestResponseBodyMethodProcessor bean internally use Jackson converter to convert Java object to Json format.
- Finally, this Json response is sent back to the client by DispatcherServlet.

Content Negotiation:

- Server is capable of producing/consuming different data formats e.g. JSON, XML, ...
- Client can get any format in which it is interested.
- Spring REST application can support multiple formats by multiple message converters. Example:
    - MappingJackson2HttpMessageConverter -- JSON format
    - MappingJackson2XmlHttpMessageConverter -- XML format
- Spring Boot will auto-create these message converters if respective dependencies are added in project. Spring will also auto add these converters into the list of converters used RequestResponseBodyMethodProcessor.
    - Maven jackson-databind -- auto create -- MappingJackson2HttpMessageConverter
    - Maven jackson-dataformat-xml -- auto create MappingJackson2XmlHttpMessageConverter
- Client application/Postman can now request REST API with request header "Accept"="application/json" to get the Json response and "Accept"="text/xml" to get XML response. The default is "Json".
- It is possible that server may produce only XML or only Json result.

Manipulating JSON output/input

- Using some Jackson annotations we can modify the JSON response data and control JSON request data.
- This is also called as "Static filtering".
- @JsonIgnore -- do not add this field in json output and do not process this field from json input.
- @JsonProperty("newName") -- Json field name is changed to "newName".
- @JsonInclude -- Control which fields to be added in Json.
    - Include.NON_NULL -- do not generate output for null fields in Java object.
    - Include.NON_EMPTY -- do not generate output for null fields or empty collection in Java object.
    - Include.ALWAYS -- always generate output for all fields though they are null or empty.
- @JsonManagedReference -- output is added into json (forward ref)
- @JsonBackReference -- output is not added into json (backward ref).

Response Util Object

- Response/Result object
    - status = "success" or "error"
    - data = any object `{ ... }` or array `[ ... ]` -- in case of success
    - message = string error -- in case of error
    - getters/setters

CORS

- Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.
- CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.
- The server should send response header showing which websites are allowed to access the resource. The "*" indiate that all web-sites can access rge resource.

```
Access-Control-Allow-Origin: *
```

- Spring Boot enable this using @CrossOrigin on @Controller/@RestController.

Jackson Library

- Jackson is a high-performance JSON processor used for Java. It is the most popular library used for serializing Java objects or Map to JSON and vice-versa. It is completely written in Java.
- Convert Java object into JSON

```
ObjectMapper mapper = new ObjectMapper();
Category c = new Category(...);
String json = mapper.writeValueAsString(c);
System.out.println(json);
```

- Convert JSON to Java Object

Spring REST client

- RestTemplate is used to create applications that consume RESTful Web Services. The application can be console application, web application or another REST service.
- RestTemplate can be created directly OR using RestTemplateBuilder.

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

```
@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
    return builder.build();
}
```

- Creating RestTemplate using builder is a good practice. It can be used to customize restTemplate with custom interceptors. It also enables application metrics and work with distributed tracing
- RestTemplate sends GET, POST, PUT or DELETE request to the REST resource/api.

UriComponentsBuilder can be used to build Uri as per following steps:

- Create a UriComponentsBuilder with one of the static factory methods (such as fromPath(String) or fromUri(URI))
- Set the various URI components through the respective methods (scheme (String), userInfo(String), host (String), port(int), path (String),
- pathSegment(String...), queryParam(String, Object...), and fragment (String).
- Build the UriComponents instance with the build () method.

Object relational mapping (ORM)

- Converting Java objects into RDBMS rows and vice-versa is done manually in JDBC code.
- This can be automated using Object Relational Mapping.
    - Java Class is mapped to RDBMS Table, while it's field are mapped to Column.
    - It also map table relations into entities associations/inheritance and auto-generates SQL queries.
- Most used ORM are Hibernate, EclipseLink, iBatis, Torque, ...

ORM (JPA)

- All ORM annotations are in javax.persistence package.
- ORM using annotations
    - @Entity: Mark POJO as hibernate entity.
    - @Table: Map entity class with the RDBMS table.
    - @Column: Map entity class field with the RDBMS table column.
    - @Id: Mark primary key field of the entity class.
    - @Temporal
        - @Temporal(TemporalType.DATE) -- java.util.Date, the java.util.Calendar, or LocalDate
        - @Temporal(TemporalType.TIME) -- java.util.Date, or LocalTime
        - @Temporal(TemporalType.TIMESTAMP) -- java.util.Date, the java.util.Calendar, or LocalDateTime
    - @Lob
        - Mapped with byte[]
    - @Transient: Do not map entity class field to the RDBMS table column.
- @Column can be used on field level or on getter methods.

Query Methods

- Methods available in JpaRepository/CrudRepository provide basic CRUD operations.
- For application specific database queries build queries using keywords.
    - Query expressions are usually property traversals combined with concatenation operators And / Or as well as comparison operators Between, LessThan, Like, etc.
    - IgnoreCase for individual properties or all properties.
    - OrderBy for static ordering.

N+1 Problem

- When ManyToOne and OneToOne is used and multiple entities are searched at once e.g. "blogDao.findAll()", then associated entities are fetched with different query for each entity.
    - If "blogDao.findAll()" --> 3 blogs, Then 3 times queries are fired to get associated Category.
- This slow down execution (due to multiple queries on database).
- It can be done more efficiently by writing JPQL/HQL query.
    - "select b from Blog b join fetch b.category" --> Force to fetch category along with blog in the same join query.

FetchType

- Defines SELECT behaviour of associated entity.
- Association annotations can be have fetch type LAZY or EAGER.
    - @OneToOne default fetch type = EAGER
    - @ManyToOne default fetch type = EAGER
    - @OneToMany default fetch type = LAZY
    - @ManyToMany default fetch type = LAZY

CascadeType

- Defines entity life cycle of associated entity.
- Possible values can be PERSIST, MERGE, DETACH, REMOVE, REFRESH, and ALL.
- If Dept class has @OneToMany(cascade = CascadeType.XXX), then
    - PERSIST: insert Emp in list while inserting Dept (persist())
    - REMOVE: delete Emp in dept while deleting Dept (remove())
    - DETACH: remove Emp in dept from session while removing Dept from session (detach())
    - REFRESH: re-select Emp in dept while re-selecting Dept (refresh())
    - MERGE: add Emp in dept into session while adding Dept into session (merge())

JPA Entity Life Cycle

- JPA entity can have one of four states.
- New
    - New Java object of entity class.
    - This object is not yet associated with hibernate.
- Managed
    - Object in session cache.
    - For all objects created by hibernate or associated with hibernate.
    - State is tracked by hibernate and updated in database during commit.
    - Never garbage collected.
- Detached
    - Object removed from session cache.
- Removed
    - Object whose corresponding row is deleted from database.

JPA (Hibernate) Caching

- Hibernate caches are used to speed up execution of the program by storing data (objects) in memory and hence save time to fetch it from database repeatedly.
- There are two caches
    - Session cache (L1 cache)
    - SessionFactory cache (L2 cache)

JPA L1 (EntityManager) Cache

- Collection (`Map<Serializable,Object>`: Key=Primary Key, Value=Entity Object + Flags) of entities per session – Persistent objects.
    - Flags = New or Deleted or Persistent or Modified (Dirty).
- Hibernate keep track of state of entity objects and update into database.
- Session cache cannot be disabled.
- If object is present in session cache, it is not searched into session factory cache or database.
- Use refresh() to re-select data from the database forcibly.

JPA L2 (EntityManagerFactory) Cache

- Collection of entities per session factory. Entities are stored in serialized form (not as java objects).
- By default disabled, but can be enabled and configured into hibernate.cfg.xml
- Use @Cache on entity class to cache its objects.

JPA

- JPA is specification for ORM.
- JPA specifications are given in form of interfaces and annotations -- javax.persistence package.
  - Annotations: @Entity, @Table, @Column, @Id, @Transient, @Temporal, @OneToMany, @ManyToOne, ...
  - Interfaces:
    - EntityManagerFactory -- create the entity manager -- Hibernate SessionFactory is an impl of EntityManagerFactory.
    - EntityManager -- encapsulate JDBC connection -- Hibernate Session is an impl of EntityManager.
      - find(), persist(), merge(), remove(), detach(), ...
    - Transaction -- tx management.
    - Query -- represent jpql queries.
  - JPQL query language.
- All ORM implementations follow JPA specification e.g. Hibernate, Torque, iBatis, EclipseLink, ...
- Hibernate implements JPA specs.
  - SessionFactory extends EntityManagerFactory
  - Session extends EntityManager
    - find(), persist(), merge(), refresh(), remove(), detach(), ...
  - HQL is similar to JPQL.
- Traditionally JPA is configured with persistence.xml (similar to hibernate.cfg.xml).
- JPA versions
  - Java Persistence API: 1.0, 1.1, 2.0, 2.1, 2.2
  - Jakarta Persistence API: 2.2, 3.0, 3.1

Auto-generated Primary Key

- @GeneratedValue annotation is used to auto-generate primary key.
- This annotation is used with @Id column.
- There are different strategies for generating ids.
  - IDENTITY: RDBMS AUTO_INCREMENT / IDENTITY
    - @GeneratedValue(strategy = GenerationType.IDENTITY)
    - MySQL 8: id will be AUTO_INCREMENT by database.
  - SEQUENCE: RDBMS sequence using @SequenceGenerator
    - @SequenceGenerator(name = "gen", sequenceName = "bookIdSeq", initialValue = 10, allocationSize = 2)
    - @GeneratedValue(generator = "gen", strategy = GenerationType.SEQUENCE)
    - MySQL 8: Emulated with table.
  - AUTO: Depends on database dialect.
    - @GeneratedValue(generator = "gen", strategy = GenerationType.AUTO)
    - MySQL 8: id will be taken from "next_val" column of "gen" table (like emulated sequence).
  - TABLE: Dedicated table for PK generation

- @TableGenerator(name = "gen", table = "shopIdsTable", initialValue = 10, allocationSize = 2, pkColumnValue = "book")
- @GeneratedValue(generator = "gen", strategy = GenerationType.TABLE)

- The behaviour of "nested transaction" is defined by @Transactional(propogation=XXXX).
    - REQUIRED: Support a current transaction, create a new one if none exists.
    - SUPPORTS: Support a current transaction, execute non-transactionally if none exists.
    - REQUIRES_NEW: Create a new transaction, and suspend the current transaction if one exists.
    - MANDATORY: Support a current transaction, throw an exception if none exists.
    - NOT_SUPPORTED: Execute non-transactionally, suspend the current transaction if one exists.
    - NEVER: Execute non-transactionally, throw an exception if a transaction exists.
    - NESTED: Execute within a nested transaction if a current transaction exists (i.e. savepoint), behave like REQUIRED otherwise.

Testing

Unit Testing

- Testing each unit individually.
- JUnit is Java's framework for Unit testing.

MCQ:

📘 Section 1: Spring Framework (1–20)

Q1. Which annotation is used to mark a class as a Spring component?

A) @Bean
B) @Component
C) @Configuration
D) @Service
Answer: B
Explanation: @Component is a generic stereotype for any Spring-managed component.

Q2. What is the default scope of a Spring bean?

A) Prototype
B) Singleton
C) Request
D) Session
Answer: B
Explanation: By default, Spring beans are singletons.

Q3. Which interface must a class implement to be aware of its bean name?

A) BeanFactoryAware
B) ApplicationContextAware
C) BeanNameAware
D) BeanDefinitionAware
Answer: C
Explanation: BeanNameAware allows a bean to be aware of its bean name.

Q4. Which annotation is used to autowire a bean?

A) @Resource
B) @Inject
C) @Autowired
D) @Wire
Answer: C
Explanation: @Autowired is the Spring annotation for automatic dependency injection.

Q5. Which interface allows you to access the Spring ApplicationContext?

A) ContextAware
B) ApplicationContextAware
C) BeanContextAware
D) EnvironmentAware
Answer: B
Explanation: ApplicationContextAware provides access to the Spring ApplicationContext.

Q6. What is the function of the @Qualifier annotation in Spring?

A) Disables bean injection
B) Narrows down bean selection when multiple beans match
C) Declares a new bean
D) Overrides bean scope
Answer: B
Explanation: @Qualifier is used with @Autowired to specify which bean to inject.

Q7. Which lifecycle method is called after a Spring bean is fully initialized?

A) destroy()
B) init()
C) afterPropertiesSet()
D) load()
Answer: C
Explanation: afterPropertiesSet() is called after the properties are set when implementing InitializingBean.

Q8. What is the correct way to define a bean in a configuration class?

A) @Service
B) @Autowired
C) @Bean
D) @Inject
Answer: C
Explanation: @Bean is used inside @Configuration classes to define beans.

Q9. What does the @Scope("prototype") annotation mean?

A) Singleton bean
B) A new bean instance is created every time it's requested
C) Request scope for web apps
D) Session-based bean
Answer: B
Explanation: Prototype scope creates a new bean every time it is requested.

Q10. Which annotation is used to make a class a Spring service component?

A) @Component
B) @Service
C) @Repository
D) @Bean
Answer: B
Explanation: @Service is a specialized version of @Component for service classes.

Q11. What does @PostConstruct do in a Spring bean?

A) Destroys the bean
B) Called after bean creation and dependency injection
C) Initializes the application
D) Creates a singleton instance
Answer: B
Explanation: @PostConstruct is called after the constructor and dependency injection are completed.

Q12. Which annotation is a specialization of @Component for DAO layers?

A) @Service
B) @Bean
C) @Repository
D) @Qualifier
Answer: C
Explanation: @Repository is a marker for persistence/DAO layers and integrates with exception translation.

Q13. What does the @Primary annotation do in Spring?

A) Marks the default bean to inject when multiple candidates are present
B) Registers bean first
C) Disables dependency injection
D) Indicates prototype scope
Answer: A
Explanation: @Primary gives a bean precedence when multiple beans of the same type exist.

Q14. What is ApplicationContext in Spring?

A) Core configuration XML
B) Interface that provides Spring configuration information
C) Wrapper over JDBC
D) Logging framework
Answer: B
Explanation: ApplicationContext provides configuration and manages beans in the Spring container.

Q15. Which Spring container reads bean definitions from XML files?

A) AnnotationContext
B) BeanFactory
C) ClassPathXmlApplicationContext
D) ContextLoader
Answer: C
Explanation: ClassPathXmlApplicationContext loads context from an XML file on the classpath.

Q16. What is the purpose of @Configuration in Spring?

A) Enables component scanning
B) Declares a class with bean definitions
C) Loads external libraries
D) Runs the main application
Answer: B
Explanation: @Configuration defines a class with @Bean methods for Spring context.

Q17. Which of these is NOT a valid bean scope in Spring?

A) Singleton
B) Prototype
C) Session
D) Daemon
Answer: D
Explanation: Daemon is not a Spring bean scope.

Q18. How are circular dependencies resolved in Spring?

A) Automatically using constructor injection
B) Automatically using setter injection
C) Cannot be resolved
D) By using proxy interfaces
Answer: B
Explanation: Spring can resolve circular dependencies using setter injection.

Q19. Which interface provides lifecycle callback after bean creation?

A) DisposableBean
B) BeanFactoryAware
C) InitializingBean
D) FactoryBean
Answer: C
Explanation: InitializingBean defines afterPropertiesSet() method called after bean initialization.

Q20. What is a stereotype annotation in Spring?

A) Used to describe bean scope
B) Meta-annotation for component scanning
C) Used for XML parsing
D) Declares a REST controller
Answer: B
Explanation: Stereotype annotations (@Component, @Service, etc.) mark a class as a Spring-managed component.

Section 2: Spring Boot (21–40)

Q21. What is the main purpose of the @SpringBootApplication annotation?

A) Enable component scanning
B) Enable auto-configuration
C) Mark the main class
D) All of the above

Answer: D
Explanation: @SpringBootApplication is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.

Q22. What file does Spring Boot use for configuration by default?

A) config.yaml
B) application.properties
C) boot.properties
D) settings.xml

Answer: B
Explanation: application.properties is the default configuration file in Spring Boot.

Q23. Which annotation is used to bind properties from application.properties to a Java class?

A) @PropertySource
B) @EnableConfigurationProperties
C) @ConfigurationProperties
D) @Value

Answer: C
Explanation: @ConfigurationProperties binds external config properties to Java POJOs.

Q24. Which embedded server is used by default in Spring Boot?

A) Tomcat
B) Jetty
C) Undertow
D) GlassFish

Answer: A
Explanation: Spring Boot uses embedded Tomcat server by default.

Q25. How do you exclude a specific auto-configuration in Spring Boot?

A) Using excludeAutoConfiguration in @ComponentScan
B) Using @EnableAutoConfiguration
C) Using exclude in @SpringBootApplication
D) Delete the dependency from pom.xml

Answer: C
Explanation: You can disable specific auto-configuration classes using exclude attribute in @SpringBootApplication.

Q26. What is the default port Spring Boot runs on?

A) 80
B) 3000
C) 443
D) 8080

Answer: D
Explanation: Spring Boot applications start on port 8080 by default.

Q27. Where do you define a custom banner in Spring Boot?

A) banner.html
B) application.yml
C) banner.txt
D) startup.txt **Answer: C**
Explanation: Place banner.txt in src/main/resources to customize the startup banner.

Q28. What annotation is used to create a REST controller in Spring Boot?

A) @RestService
B) @RestController
C) @WebController
D) @Api

Answer: B
Explanation: @RestController combines @Controller and @ResponseBody.

Q29. How can you define a custom port in Spring Boot?

A) server.host in application.properties
B) port.host in YAML
C) server.port in application.properties
D) custom.port in application.yaml

Answer: C
Explanation: server.port=XXXX in application.properties changes the default port.

Q30. What does spring.main.web-application-type=none do?

A) Disables all Spring Boot features
B) Enables web application only
C) Disables web environment
D) Enables only servlet context

Answer: C
Explanation: This disables the web environment and is used for CLI/non-web apps.

Q31. Which starter is used for building web applications?

A) spring-boot-starter
B) spring-boot-starter-data
C) spring-boot-starter-web
D) spring-boot-starter-core

Answer: C
Explanation: spring-boot-starter-web includes Tomcat, Spring MVC, and Jackson.

Q32. How do you enable caching in Spring Boot?

A) @EnableCache
B) @EnableCaching
C) @SpringBootCaching
D) @CacheEnable

Answer: B
Explanation: Use @EnableCaching to enable Spring's annotation-driven cache management.

Q33. Which property sets logging level in application.properties?

A) logging.status
B) debug.level
C) log.level.root
D) logging.level.root

Answer: D
Explanation: Use logging.level.root=INFO to configure logging.

Q34. Which annotation is used for scheduled tasks in Spring Boot?

A) @Timer
B) @EnableScheduling
C) @ScheduledTask
D) @RunScheduled

Answer: B
Explanation: @EnableScheduling allows scheduling support.

Q35. How do you define a scheduled method in Spring Boot?

A) @EnableScheduler
B) @Run
C) @Scheduled
D) @Execute

Answer: C
Explanation: @Scheduled is used to define scheduled tasks.

Q36. What is the use of CommandLineRunner interface?

A) Custom REST configuration
B) To load data at startup
C) To enable security
D) To schedule tasks

Answer: B
Explanation: CommandLineRunner.run() executes after application context is loaded.

Q37. What's the file name to configure YAML in Spring Boot?

A) boot.yaml
B) config.yaml
C) application.yaml
D) settings.yml

Answer: C
Explanation: application.yaml is the YAML-based config file.

Q38. What's the use of @SpringBootTest?

A) Starts a full application context for testing
B) Disables testing
C) Only enables JPA tests
D) Tests only beans

Answer: A
Explanation: @SpringBootTest loads full Spring context for integration testing.

Q39. What is the purpose of spring-boot-devtools?

A) Database access
B) Hot reload, auto restart
C) Production deployment
D) Logging

Answer: B
Explanation: spring-boot-devtools enables hot reloading during development.

Q40. Where should application.properties be placed?

A) src/test/java
B) root directory
C) src/main/resources
D) META-INF

Answer: C
Explanation: application.properties must be placed in src/main/resources.

Section 3: JDBC (41–60)

Q41. Which class is used to execute a SQL query in plain JDBC?

A) Connection
B) Statement
C) ResultSet
D) DriverManager

Answer: B
Explanation: The Statement object is used to execute SQL queries.

Q42. What is the method used to load a JDBC driver?

A) Driver.load()
B) Class.forName()
C) DriverManager.getDriver()
D) Connection.loadDriver()

Answer: B
Explanation: Class.forName() loads the JDBC driver class.

Q43. Which interface provides methods for committing or rolling back a transaction?

A) Statement
B) PreparedStatement
C) Connection
D) DriverManager

Answer: C
Explanation: The Connection interface allows committing or rolling back transactions.

Q44. What is the correct way to prevent SQL injection in JDBC?

A) Use raw queries
B) Use Statement
C) Use PreparedStatement
D) Escape user input manually

Answer: C
Explanation: PreparedStatement avoids SQL injection by safely setting parameters.

Q45. Which method is used to retrieve data from a ResultSet?

A) execute()
B) getData()
C) next()
D) fetch()

Answer: C
Explanation: next() moves the cursor to the next row and returns true if there is a row.

Q46. What is returned by executeQuery() method of Statement?

A) int
B) ResultSet
C) Boolean
D) void

Answer: B
Explanation: executeQuery() returns a ResultSet object containing the query results.

Q47. What does setAutoCommit(false) do?

A) Begins a new transaction
B) Commits automatically after every query
C) Disables auto-commit, enabling manual control over commits
D) Rolls back automatically

Answer: C
Explanation: Disabling auto-commit lets you manually manage transactions.

Q48. Which method is used to roll back a transaction in JDBC?

A) rollback() on Statement
B) rollback() on ResultSet
C) rollback() on Connection
D) rollback() on DriverManager

Answer: C
Explanation: The Connection object provides the rollback() method.

Q49. What does close() do on Connection, Statement, or ResultSet?

A) Commits the transaction
B) Cleans resources
C) Kills the database
D) Deletes the schema

Answer: B
Explanation: close() releases resources held by the JDBC object.

Q50. Which method is used to create a PreparedStatement?

A) createStatement()
B) prepareStatement()
C) executeStatement()
D) initStatement()

Answer: B
Explanation: prepareStatement() creates a PreparedStatement object.

Q51. How can you prevent resource leaks in JDBC?

A) Ignore close() calls
B) Use finally block to close connections
C) Let JVM handle it
D) Only use one connection always

Answer: B
Explanation: Resources should be closed in a finally block or using try-with-resources.

Q52. How do you set a String parameter in a PreparedStatement?

A) setParameter(1, value)
B) setString(1, value)
C) put(1, value)
D) assign(1, value)

Answer: B
Explanation: Use setString(index, value) for string parameters.

Q53. What is the JDBC URL format for MySQL?

A) jdbc:mysql://host:port/db
B) jdbc:sql:mysql/host
C) jdbc:mysql:db@host
D) sql:mysql::db//host

Answer: A
Explanation: MySQL JDBC URL follows the format: jdbc:mysql://host:port/dbname.

Q54. What class is used to load JDBC drivers explicitly?

A) DriverLoader
B) DriverManager
C) ClassLoader
D) Class.forName()

Answer: D
Explanation: Class.forName("com.mysql.jdbc.Driver") loads the driver.

Q55. Which of the following is not a valid method on the ResultSet object?

A) getString()
B) getInt()
C) next()
D) execute()

Answer: D
Explanation: execute() is a method on Statement, not ResultSet.

Q56. How do you update a record using JDBC?

A) executeUpdate("UPDATE ...")
B) update()
C) execute()
D) getUpdate()

Answer: A
Explanation: executeUpdate() is used for DML operations like UPDATE.

Q57. What does executeUpdate() return?

A) Boolean
B) String
C) Number of affected rows
D) ResultSet

Answer: C
Explanation: It returns the number of rows affected by the operation.

Q58. Which class is responsible for registering the JDBC driver?

A) DriverManager
B) Connection
C) ResultSet
D) SQLException

Answer: A
Explanation: DriverManager handles JDBC driver registration.

Q59. What is the role of SQLException in JDBC?

A) Indicates a successful operation
B) Used for logging
C) Handles SQL errors
D) Loads SQL schema

Answer: C
Explanation: SQLException handles errors that occur while interacting with the database.

Q60. Which method is used to open a database connection in JDBC?

A) getConnection() on DriverManager
B) connect() on Connection
C) open() on Driver
D) access() on Statement

Answer: A
Explanation: DriverManager.getConnection() is used to open a connection.

Section 4: JPA (61–80)

Q61. Which annotation is used to map a class as a JPA entity?

A) @Model
B) @POJO
C) @Entity
D) @JPAObject

Answer: C
Explanation: @Entity marks a class as a JPA entity.

Q62. Which annotation is used to specify the primary key in a JPA entity?

A) @PK
B) @Id
C) @Key
D) @Primary

Answer: B
Explanation: @Id defines the primary key of a JPA entity.

Q63. Which annotation defines a column in a JPA entity?

A) @Field
B) @Column
C) @Attribute
D) @DataField

Answer: B
Explanation: @Column is used to map an entity field to a table column.

Q64. Which annotation specifies a table name for an entity?

A) @Entity
B) @Table
C) @JoinTable
D) @Database

Answer: B
Explanation: @Table(name="...") specifies the table name.

Q65. Which strategy is NOT valid for ID generation in JPA?

A) GenerationType.AUTO
B) GenerationType.IDENTITY
C) GenerationType.SEQUENCE
D) GenerationType.UUID

Answer: D
Explanation: GenerationType.UUID is not a standard strategy.

Q66. What does the @GeneratedValue annotation do?

A) Encrypts the ID
B) Marks a primary key as auto-generated
C) Declares a relationship
D) Overrides column type

Answer: B
Explanation: @GeneratedValue lets JPA automatically generate primary key values.

Q67. Which annotation is used to define a many-to-one relationship?

A) @JoinColumn
B) @ManyToOne
C) @OneToMany
D) @ManyToMany

Answer: B
Explanation: @ManyToOne establishes a many-to-one relationship.

Q68. What is the default fetch type for @ManyToOne?

A) LAZY
B) EAGER
C) DEFAULT
D) NONE

Answer: B
Explanation: @ManyToOne is EAGER by default.

Q69. Which annotation is used to define a one-to-many relationship?

A) @OneToMany
B) @JoinTable
C) @ManyToOne
D) @Collection

Answer: A
Explanation: @OneToMany maps a one-to-many association.

Q70. Which attribute is used to map the owning side in @OneToMany?

A) name
B) targetEntity
C) mappedBy
D) joinBy

Answer: C
Explanation: mappedBy defines the inverse side of the relationship.

Q71. Which annotation defines the foreign key column for a relationship?

A) @ForeignKey
B) @Relation
C) @JoinColumn
D) @Link

Answer: C
Explanation: @JoinColumn maps a foreign key column.

Q72. What is the use of @Transient in JPA?

A) Declares a derived column
B) Ignores the field from persistence
C) Declares a primary key
D) Used for caching

Answer: B
Explanation: Fields annotated with @Transient are not persisted.

Q73. What is the correct method to persist an entity using EntityManager?

A) save()
B) insert()
C) persist()
D) merge()

Answer: C
Explanation: persist() stores a new entity instance in the database.

Q74. Which method updates a detached entity?

A) persist()
B) update()
C) merge()
D) attach()

Answer: C
Explanation: merge() updates the database with a detached entity.

Q75. Which method removes an entity from the persistence context?

A) remove()
B) delete()
C) destroy()
D) terminate()

Answer: A
Explanation: remove() deletes the entity from the database.

Q76. Which annotation is used for defining a named query?

A) @NamedQuery
B) @QueryName
C) @StaticQuery
D) @JPQL

Answer: A
Explanation: @NamedQuery defines static queries.

Q77. Which annotation allows use of JPQL in Spring Data JPA?

A) @Query
B) @JPQL
C) @Script
D) @SQL

Answer: A
Explanation: @Query lets you define custom JPQL or native SQL queries.

Q78. What does CascadeType.ALL mean in JPA?

A) Cascade only persist
B) Cascade only delete
C) Apply all cascade operations (persist, merge, remove, etc.)
D) No cascading

Answer: C
Explanation: CascadeType.ALL applies all lifecycle operations to child entities.

Q79. What is the role of the EntityManager interface?

A) Configure Spring Boot
B) Persist and manage entity lifecycle
C) Manage JDBC connections
D) Generate DDL

Answer: B
Explanation: EntityManager handles all CRUD operations and entity state transitions.

Q80. What does fetch = FetchType.LAZY do?

A) Loads all entities eagerly
B) Skips loading
C) Delays loading until property is accessed
D) Caches query

Answer: C
Explanation: LAZY loading delays the fetching of associated data until needed.

Section 5: Hibernate (81–100)

Q81. What is the file name used for Hibernate configuration?

A) hibernate-config.xml
B) hibernate.properties
C) hibernate.cfg.xml
D) config.xml

Answer: C
Explanation: hibernate.cfg.xml is the default configuration file in Hibernate.

Q82. Which Hibernate interface is used to interact with the database?

A) SessionFactory
B) Transaction
C) Session
D) EntityManager

Answer: C
Explanation: Session interface is used to perform CRUD operations in Hibernate.

Q83. Which method saves an entity in Hibernate?

A) store()
B) save()
C) create()
D) insert()

Answer: B
Explanation: save() method persists a new entity into the database.

Q84. What is the use of SessionFactory in Hibernate?

A) Manage sessions and cache metadata
B) Execute SQL queries
C) Store configuration
D) Map tables to objects

Answer: A
Explanation: SessionFactory is a factory for Session and caches metadata.

Q85. Which object is responsible for transaction management in Hibernate?

A) Session
B) EntityManager
C) Transaction
D) HibernateContext

Answer: C
Explanation: Transaction handles commit and rollback operations.

Q86. What is the default cache level used in Hibernate?

A) First-level
B) Second-level
C) Query cache
D) No cache

Answer: A
Explanation: Hibernate uses first-level cache by default for each session.

Q87. Which method is used to retrieve an entity by its ID?

A) load()
B) getById()
C) find()
D) get()

Answer: D
Explanation: get() retrieves an entity by ID and returns null if not found.

Q88. What does load() method do differently than get()?

A) Executes immediately
B) Returns null if not found
C) Returns proxy and loads lazily
D) Bypasses session cache

Answer: C
Explanation: load() returns a proxy and throws an exception if not found.

Q89. How do you map one-to-one relationships in Hibernate?

A) @OneToOne
B) @OneToMany
C) @JoinColumn
D) @EntityLink

Answer: A
Explanation: @OneToOne defines a one-to-one mapping.

Q90. Which annotation enables second-level caching in Hibernate?

A) @EnableCaching
B) @Cache
C) @HibernateCache
D) @SecondLevelCache

Answer: B
Explanation: @Cache enables second-level cache configuration.

Q91. What is the purpose of hibernate.hbm2ddl.auto property?

A) Define the database name
B) Set dialect
C) Auto generate schema
D) Enable caching

Answer: C
Explanation: hibernate.hbm2ddl.auto defines how schema is managed (create, update, validate, none).

Q92. What is the purpose of Hibernate Dialect?

A) Translate Java to SQL
B) Specify database-specific SQL syntax
C) Configure JPA mappings
D) Setup connection pool

Answer: B
Explanation: Dialect tells Hibernate how to generate SQL for the specific DB.

Q93. Which class manages the Hibernate lifecycle?

A) HibernateUtil
B) SessionFactory
C) Session
D) HibernateContext

Answer: B
Explanation: SessionFactory is responsible for lifecycle and configuration.

Q94. What does the merge() method do in Hibernate?

A) Deletes the record
B) Merges changes of a detached object into the persistence context
C) Validates a transaction
D) Flushes the cache

Answer: B
Explanation: merge() reattaches and synchronizes changes of a detached entity.

Q95. What happens when you call flush() in Hibernate?

A) Commits the transaction
B) Refreshes the UI
C) Synchronizes session state with DB
D) Deletes cached entities

Answer: C
Explanation: flush() pushes changes from session to the database.

Q96. How can you map an inheritance hierarchy in Hibernate?

A) Using @Inheritance annotation
B) @Polymorphic annotation
C) @Extend annotation
D) @Supertype annotation

Answer: A
Explanation: @Inheritance is used for entity inheritance mapping.

Q97. Which strategy is NOT valid for Hibernate inheritance?

A) SINGLE_TABLE
B) TABLE_PER_CLASS
C) JOINED
D) SPLIT

Answer: D
Explanation: SPLIT is not a valid inheritance strategy in Hibernate.

Q98. What is the default inheritance strategy in Hibernate?

A) JOINED
B) TABLE_PER_CLASS
C) SINGLE_TABLE
D) NONE

Answer: C
Explanation: SINGLE_TABLE is the default strategy.

Q99. What is @DiscriminatorColumn used for?

A) Defines cache strategy
B) Specifies table for joined inheritance
C) Defines column used to distinguish subclasses
D) Configures fetch strategy

Answer: C
Explanation: @DiscriminatorColumn helps identify which subclass the row belongs to.

Q100. What is the difference between save() and persist() in Hibernate?

A) persist() returns ID, save() doesn't
B) persist() works outside transactions
C) save() returns ID; persist() returns void
D) No difference

Answer: C
Explanation: save() returns generated ID; persist() returns void and is JPA-compliant.

Section 6: REST API (101–130)

Q101. What does REST stand for?

A) Representational Stateless Transfer
B) Remote Execution Service Transfer
C) Representational State Transfer
D) Request Encoding Syntax Transfer

Answer: C
Explanation: REST stands for Representational State Transfer, an architectural style for APIs.

Q102. Which HTTP method is idempotent?

A) POST
B) GET
C) PATCH
D) PUT

Answer: D
Explanation: PUT is idempotent because making the same call multiple times will not change the result beyond the first request.

Q103. Which status code represents a successful DELETE request?

A) 200
B) 201
C) 204
D) 404

Answer: C
Explanation: 204 No Content is usually returned for successful DELETE without a response body.

Q104. What is the correct annotation in Spring to handle a GET request?

A) @RequestHandler
B) @Get
C) @GetMapping
D) @Fetch

Answer: C
Explanation: @GetMapping is the specific shortcut for handling GET requests.

Q105. Which annotation maps path variables in Spring Boot REST?

A) @RequestParam
B) @PathParam
C) @PathVariable
D) @Route

Answer: C
Explanation: @PathVariable maps URI template variables to method parameters.

Q106. Which HTTP status code indicates a resource was created?

A) 200
B) 204
C) 302
D) 201

Answer: D
Explanation: 201 Created indicates a resource has been successfully created.

Q107. What does @RequestBody do in Spring MVC?

A) Parses path params
B) Binds the request body to a method parameter
C) Retrieves headers
D) Injects beans

Answer: B
Explanation: @RequestBody maps the entire request body to a method argument.

Q108. Which library is commonly used for JSON conversion in Spring Boot?

A) Gson
B) Jackson
C) FastJson
D) JSONSimple

Answer: B
Explanation: Spring Boot uses Jackson by default for JSON serialization/deserialization.

Q109. What annotation is used for global exception handling in Spring REST?

A) @ControllerAdvice
B) @RestController
C) @ExceptionMapper
D) @ErrorHandler

Answer: A
Explanation: @ControllerAdvice handles exceptions across the whole application.

Q110. What annotation marks a class as a REST controller in Spring Boot?

A) @RestApi
B) @RestController
C) @Controller
D) @ApiController

Answer: B
Explanation: @RestController combines @Controller and @ResponseBody.

Q111. Which HTTP method is not safe?

A) GET
B) PUT
C) HEAD
D) OPTIONS

Answer: B
Explanation: PUT modifies state and is therefore not a safe method.

Q112. What does @ResponseStatus do?

A) Defines a default view
B) Binds status code to an exception or method
C) Logs the request
D) Maps HTTP method

Answer: B
Explanation: @ResponseStatus binds an HTTP status to a method or exception class.

Q113. Which tool is commonly used to document REST APIs?

A) Maven
B) Swagger
C) Jenkins
D) Prometheus

Answer: B
Explanation: Swagger (now OpenAPI) provides API documentation and testing.

Q114. What does HATEOAS add to REST APIs?

A) Authentication
B) Versioning
C) Hypermedia links
D) Rate limiting

Answer: C
Explanation: HATEOAS adds hypermedia links to API responses.

Q115. What is the default response content type in Spring Boot?

A) text/plain
B) application/json
C) application/xml
D) application/x-www-form-urlencoded

Answer: B
Explanation: JSON is the default content type in Spring Boot.

Q116. What is the correct status code for unauthorized access?

A) 403
B) 500
C) 401
D) 404

Answer: C
Explanation: 401 Unauthorized indicates authentication is required.

Q117. Which annotation binds a request parameter to a method argument?

A) @PathVariable
B) @RequestParam
C) @Param
D) @QueryParam

Answer: B
Explanation: @RequestParam maps query parameters to method arguments.

Q118. What is the purpose of @CrossOrigin?

A) Enables URL shortening
B) Allows cross-origin requests
C) Converts XML to JSON
D) Adds cache headers

Answer: B
Explanation: @CrossOrigin enables CORS (Cross-Origin Resource Sharing).

Q119. What is the standard port for REST APIs over HTTPS?

A) 80
B) 443
C) 22
D) 8443

Answer: B
Explanation: 443 is the default port for HTTPS.

Q120. What does OPTIONS method do?

A) Sends request data
B) Gets resource metadata
C) Deletes a resource
D) Updates a resource

Answer: B
Explanation: OPTIONS returns allowed methods and headers for a resource.

Q121. Which HTTP method is used to partially update a resource?

A) PUT
B) PATCH
C) POST
D) UPDATE
✅ **Answer: B**
Explanation: PATCH is used to apply partial updates to a resource.

Q122. What is the use of @RequestHeader in Spring?

A) Access path variables
B) Access query parameters
C) Access request headers
D) Add CORS policy
✅ **Answer: C**
Explanation: @RequestHeader binds method parameters to request header values.

Q123. What is the role of HttpEntity in Spring?

A) Perform async call
B) Carry request and response metadata and body
C) Auto log requests
D) Configure SSL
✅ **Answer: B**
Explanation: HttpEntity encapsulates headers and body of an HTTP request or response.

Q124. What does produces attribute do in @RequestMapping?

A) Sets return type of method
B) Specifies MIME type of response
C) Sets input data format
D) Defines path parameters
✅ **Answer: B**
Explanation: produces tells Spring the content type to be produced in response.

Q125. What does consumes attribute in @PostMapping define?

A) What HTTP verbs are accepted
B) The expected request content type
C) What fields to map
D) Which controller handles it
✅ **Answer: B**
Explanation: consumes defines the MIME type the method can handle in the request body.

Q126. What does ResponseEntity return?

A) Only body
B) Only headers
C) Body and HTTP status code
D) Path parameters
✅ **Answer: C**
Explanation: ResponseEntity represents a complete HTTP response, including headers, body, and status.

Q127. What's the HTTP status code for "Forbidden"?

A) 401
B) 403
C) 404
D) 500
✅ **Answer: B**
Explanation: 403 Forbidden indicates the server understood the request but refuses to authorize it.

Q128. What is the default path for actuator health endpoint in Spring Boot?

A) /check
B) /status
C) /health
D) /monitor
✅ **Answer: C**
Explanation: /actuator/health is the default health-check endpoint.

Q129. What does the @DeleteMapping annotation do?

A) Deletes a Spring bean
B) Maps HTTP DELETE requests to a method
C) Deletes a file
D) Denotes security mapping
✅ **Answer: B**
Explanation: @DeleteMapping is used to map DELETE requests in Spring MVC.

Q130. How do you specify multiple request mappings to a method?

A) Use multiple @RequestMapping annotations
B) Use array in @RequestMapping(value={...})
C) Declare in application.properties
D) Not possible
✅ **Answer: B**
Explanation: You can map multiple URLs using an array in @RequestMapping(value={"/a", "/b"}).