Enterprise module (Java branch)

Java Enterprise and Spring

What are the possible uses of reflection?
Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime. The required classes for reflection are provided under java.lang.reflect package which is essential in order to understand reflection.
Reflection gives us information about the class to which an object belongs and also the methods of that class that can be executed by using the object.
Through reflection, we can invoke methods at runtime irrespective of the access specifier used with them.
Reflection can be used to get information about class, constructors, and methods.

What is Spring?
Spring is the most popular application development framework for enterprise Java. The term "Spring" can be used for the Spring Framework itself, but most people use it for the entire family of projects as well.

What is Spring Boot?
Spring Boot has been built on top of the Spring Framework, and it comes with many dependencies that can be plugged into the Spring application. The main goal of Spring Boot is to reduce development time and difficulty of configurations.

Spring Boot
is a lightweight framework
is easy to get started quickly with
is a perfect choice if you want to avoid XML configurations
has default setup for unit and integration tests

What is the major difference between the Standard edition (JSE) and Enterprise edition (JEE)? You can choose Spring (Spring Boot) instead of JavaEE. Focus on comparing them.
Java technology is both a programming language and a platform. The Java programming language is a high-level object-oriented language that has a particular syntax and style. A Java platform is a particular environment in which Java programming language applications run.
There are several Java platforms. Many developers, even long-time Java programming language developers, do not understand how the different platforms relate to each other. Four platforms.
-Java Platform, Standard Edition (Java SE)
-Java Platform, Enterprise Edition (Java EE)
-Java Platform, Micro Edition (Java ME)
-JavaFX
All Java platforms consist of a Java Virtual Machine (VM) and an application programming interface (API). The Java Virtual Machine is a program, for a particular hardware and software platform, that runs Java technology applications. An API is a collection of

software components that you can use to create other software components or applications. Each Java platform provides a virtual machine and an API, and this allows applications written for that platform to run on any compatible system with all the advantages of the Java programming language: platform-independence, power, stability, ease-of-development, and security.

Java SE
When most people think of the Java programming language, they think of the Java SE API. Java SE's API provides the core functionality of the Java programming language. It defines everything from the basic types and objects of the Java programming language to high-level classes that are used for networking, security, database access, graphical user interface (GUI) development, and XML parsing.
In addition to the core API, the Java SE platform consists of a virtual machine, development tools, deployment technologies, and other class libraries and toolkits commonly used in Java technology applications.

Java EE
The Java EE platform is built on top of the Java SE platform. The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.

What are the advantages of the Spring Framework? Focus on the Core part.
There are the following advantages of the Spring framework:

Light Weight: Spring is a lightweight framework because of its POJO implementation. It does not force the programmer to inherit any class and implement any interface. With the help of Spring, we can enable powerful, scalable applications using POJOs (Plain Old Java Object).
Flexible: It provides flexible libraries trusted by developers all over the world. The developer can choose either XML or Java-based annotations for configuration options. The IoC and DI features provide the foundation for a wide-ranging set of features and functionality. It makes the job simpler.
Loose Coupling: Spring applications are loosely coupled because of dependency injection. It handles injecting dependent components without a component knowing where they came from.
Powerful Abstraction: It provides a powerful abstraction to JEE specifications such as JMS, JDBC, JPA, and JTA.
Declarative Support: It provides declarative support for caching, validation, transaction, and formatting.
Portable: We can use server-side in web/EJB app, client-side in swing app business logic is completely portable.
Cross-cutting behavior: Resource management is a cross-cutting concern, easy to copy and paste everywhere.
Configuration: It provides a consistent way of configuring everything, separate configuration from application logic, varying configuration.

Lifecycle: Responsible for managing all your application components, particularly those in middle-tier container sees components through well-defined lifecycle: init(), destroy().
Dependency Injection: The use of dependency injection makes the easy development of JavaEE.
Easier Testing: The use of dependency injection makes the testing easy. The spring framework does not require a server while the EJB and Struts application requires a server.
Fast: The team of Spring engineers deeply cares about the performance. Its fast startup, fast shutdown, and optimized execution maintain performance make it fast. Even, we can start a new Spring project in seconds by using Spring Initializr.
Secure: It monitors third-party dependencies closely. The regular update is issued that make our data and application secure. We can make our application secure by using the Spring Security framework. It provides industry-standard security schemes and delivers a trustworthy solution that is secure by default.
Supportive: The Spring community provides support and resources to get you to the next level QuickStart guides, tutorials, videos, and meetup helps a lot.
Productive: It is more productive because the spring application can integrate with other Spring-based applications. For example, we can combine the Spring Boot application with Spring Cloud.

What is a servlet? What is the purpose of DispatcherServlet in Spring?
Simply put, a Servlet is a class that handles requests, processes them and reply back with a response.
For example, we can use a Servlet to collect input from a user through an HTML form, query records from a database, and create web pages dynamically.
It's important to understand that the Servlet technology is not limited to the HTTP protocol.
In practice it almost always is, but Servlet is a generic interface and the HttpServlet is an extension of that interface – adding HTTP specific support – such as doGet and doPost, etc.

Spring MVC framework is built around a central servlet called DispatcherServlet that handles all the HTTP requests and responses.
The DispatcherServlet does a lot more than that:
It seamlessly integrates with the IoC container and allows you to use each feature of Spring in an easier manner.
The DispatcherServlet contacts HandlerMapping to call the appropriate Controller for processing the request on receiving it.
Then, the controller calls appropriate service methods to set or process the Model data. The service processes the data and returns the view name to DispatcherServlet. DispatcherServlet then takes the help of ViewResolver and picks up the defined view for the request.
Once the view is decided, the DispatcherServlet passes the Model data to View where it is finally rendered on the browser.

When do you use RestControllers, when just simple Controllers?
We can use @Controller annotation for traditional Spring controllers, and it has been part of the framework for a very long

time.
Spring 4.0 introduced the @RestController annotation in order to simplify the creation of RESTful web services. It's a convenient annotation that combines @Controller and @ResponseBody, which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.
Every request handling method of the controller class automatically serializes return objects into HttpResponse.

What is Spring Application Context?
One of the main features of the Spring framework is the IoC (Inversion of Control) container. The Spring IoC container is responsible for managing the objects of an application. It uses dependency injection to achieve inversion of control.
The interfaces BeanFactory and ApplicationContext represent the Spring IoC container. Here, BeanFactory is the root interface for accessing the Spring container. It provides basic functionalities for managing beans.
On the other hand, the ApplicationContext is a sub-interface of the BeanFactory. Therefore, it offers all the functionalities of BeanFactory.
Furthermore, it provides more enterprise-specific functionalities. The important features of ApplicationContext are resolving messages, supporting internationalization, publishing events, and application-layer specific contexts. This is why we use it as the default Spring container.
Spring provides different types of ApplicationContext containers suitable for different requirements. These are implementations of the ApplicationContext interface.

What are the main ways to define a bean in the Application Context?
1. Java-Based Configuration: Java configuration typically uses @Bean-annotated methods within a @Configuration class. The @Bean annotation on a method indicates that the method creates a Spring bean. Moreover, a class annotated with @Configuration indicates that it contains Spring bean configurations.
2. Annotation-Based Configuration: In this approach, we first enable annotation-based configuration via XML configuration. Then we use a set of annotations on our Java classes, methods, constructors, or fields to configure beans. Some examples of these annotations are @Component, @Controller, @Service, @Repository, @Autowired, and @Qualifier.
Notably, we use these annotations with Java-based configuration as well.
3. XML-Based Configuration: This is the traditional way of configuring beans in Spring.
Obviously, in this approach, we do all bean mappings in an XML configuration file.

Difference between .jar and .war files.
The main difference between JAR and WAR Files is that the JAR files are the files that have Java class files, associated metadata and resources aggregated into a single file to execute a Java

application while the WAR files are the files that contain Servlet, JSP, HTML, JavaScript and other files necessary for developing web applications.

What are the major differences between Maven, Ant and Gradle?
All 3 are build tools for building java applications.

ANT

Ant was initially used by many software developers to build applications.
Ant build script should be written in xml and it is called build.xml
Ant doesn't follow any project structure. As a result developers has to write complete build script which sometimes makes a build.xml huge and difficult to maintain
Initially Ant doesn't have dependency management. This leads to the development of Maven. Later on, support had been enabled to integrate with dependency management tools like IVY.

MAVEN

Maven uses dependency management to manage the dependencies using repository based approach unlike ANT where developers need to place the dependencies in the class path and refer it in build.xml
Maven also uses xml style build script called pom.xml
Maven uses pre defined build commands to compile and build the projects unlike ANT where developers has to write a script from scratch.
Maven follows standard project structure.

GRADLE

Gradle is developed on the concept of both Ant and Maven.
Gradle is DSL based script(Domain Specific Language) avoiding xml syntax based language which makes code easier to understand even for the common user.
Gradle also uses dependency management and repository based approach to compile and build projects.

As of now Maven occupied the majority of the market share.

What is Maven used for?
Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.
In short terms we can tell **maven is a tool that can be used for building and managing any Java-based project.** maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

Maven does a lot of helpful task like

-We can easily build a project using maven.

-We can add jars and other dependencies of the project easily using the help of maven.
-Maven provides project information (log document, dependency list, unit test reports etc.)
-Maven is very helpful for a project while updating central repository of JARs and other dependencies.
-With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
-Using maven we can easily integrate our project with source control system (such as Subversion or Git).

What does a pom.xml file contains in Maven?
A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.
Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

Elements in the POM that are merged are the following:

dependencies
developers and contributors
plugin lists (including reports)
plugin executions with matching ids
plugin configuration
resources

Object Relational Mapping, JPA

What is an ORM? What are the benefits, when to use?
Object-relational mapping (ORM) is the phenomenon of mapping application domain model objects to the relational database tables and vice versa.
They write correct and optimized SQL queries, thereby eliminating the hassle for developers
They make the code easier to update, maintain, and reuse as the developer can think of, and manipulate data as objects
ORMs will shield your application from SQL injection attacks since the framework will filter the data for you!
ORMs provide the concept of Database Abstraction which makes switching databases easier and creates a consistent code base for your application.

If you know your data access pattern is going to be complex or you

plan to use a lot of database-specific features, you may not want to use an ORM. While many ORMs (like Hibernate) let you access the underlying data source connection pretty easily , if you know you're going to have to throw around a lot of custom SQL, you may not get a lot of value out of ORM to begin with because you're constantly going to have to break out of it.

What is the difference between JDBC and JPA? Which are the advantages and disadvantages of each? Give a general overview.
JDBC: JDBC stands for Java Database Connectivity. It is a java application programming interface to provide a connection between the Java programming language and a wide range of databases (i.e), it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.
JDBC is an Application Programming Interface(API) for Java, which is helpful for interaction with the database and for executing the SQL query. JDBC is an abbreviation used for Java Database Connectivity. It uses JDBC drivers for connecting with the database. JDBC API is used to access tabular data stored in relational databases like Oracle, MySQL, MS Access, etc.

The Java Persistence API (JPA) is a standard API for accessing databases from within Java applications. The main advantage of JPA over JDBC (the older Java API for interacting with databases) is that in JPA data is represented by classes and objects rather than by tables and records as in JDBC. Using plain old Java objects (POJO) to represent persistent data can significantly simplify database programming.
A JPA implementation (sometimes referred to as a JPA provider) is needed in order to interact with a relational database such as Oracle, DB2, SQL Server or MySQL. The popular JPA implementations are Hibernate, TopLink, EclipseLink, Open JPA and DataNucleus. These implementations are Object Relational Mapping (ORM) tools. The mapping bridges between the data representation in the relational database (as tables and records) and the representation in the Java application (as classes and objects).

The most obvious benefit of JDBC over JPA is that it's simpler to understand. On the other side, if a developer doesn't grasp the internal workings of the JPA framework or database design, they will be unable to write good code.
Also, JPA is thought to be better suited for more sophisticated applications by many developers. But, JDBC is considered the preferable alternative if an application will use a simple database and we don't plan to migrate it to a different database vendor.
The main advantage of JPA over JDBC for developers is that they can code their Java applications using object-oriented principles and best practices without having to worry about database semantics. As a result, development can be completed more quickly, especially when software developers lack a solid understanding of SQL and relational databases.
Also, because a well-tested and robust framework is handling the interaction between the database and the Java application, we should

see a decrease in errors from the database mapping layer when using JP

What is Hibernate? What are the advantages, limitations?
Hibernate is the most commonly used java based ORM framework.
Hibernate: Hibernate is an open-source, non-invasive, light-weight java ORM(Object-relational mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE. It simplifies the interaction of java applications with databases. Hibernate is an implementation of JPA(Java Persistence API).

Name 3 different annotations used in JPA, what can they do for you?
-@Entity: Specifies that the class is an entity. This annotation can be applied on Class, Interface of Enums.
-@Table: It specifies the table in the database with which this entity is mapped. In the example below the data will be stores in the “employee” table. Name attribute of @Table annotation is used to specify the table name.
-@Id: This annotation specifies the primary key of the entity.
-@GeneratedValue: This annotation specifies the generation strategies for the values of primary keys.

What is object-relational impedance mismatch?
Impedance mismatch is the term used to refer to the problems that occurs due to differences between the database model and the programming language model. The practical relational model has 3 components these are:
Attributes and their data types
Tuples
Tables
1. The first problem that may occur is that is data type mismatch means the programming language attribute data type may differ from the attribute data type in the data model.
2. The second problem that may occur is because the results of most queries are sets or multisets of tuples and each tuple is formed of a sequence of attribute values. In the program, it is necessary to access the individual data values within individual tuples for printing or processing.

What is a JpaRepository? What are the 2 main methods to define queries in them?
JpaRepository is a JPA (Java Persistence API) specific extension of Repository. It contains the full API of CrudRepository and PagingAndSortingRepository. So it contains API for basic CRUD operations and also API for pagination and sorting.
In order to define SQL to execute for a Spring Data repository method, we can annotate the method with the @Query annotation – its value attribute contains the JPQL or SQL to execute.
JPQL
By default, the query definition uses JPQL.

Let's look at a simple repository method that returns active User

entities from the database:

`
@Query("SELECT u FROM User u WHERE u.status = 1")
Collection<User> findAllActiveUsers();`

Native
We can use also native SQL to define our query. All we have to do is set the value of the nativeQuery attribute to true and define the native SQL query in the value attribute of the annotation:

`@Query(
value = "SELECT * FROM USERS u WHERE u.status = 1",
nativeQuery = true)
Collection<User> findAllActiveUsersNative();`

Why is the Set preferred over List when we want to store OneToMany relations?
A list, if there is no index column specified, will just be handled as a bag by Hibernate (no specific ordering).
One notable difference in the handling of Hibernate is that you can't fetch two different lists in a single query. For example, if you have a Person entity having a list of contacts and a list of addresses, you won't be able to use a single query to load persons with all their contacts and all their addresses. The solution in this case is to make two queries (which avoids the cartesian product), or to use a Set instead of a List for at least one of the collections.
It's often hard to use Sets with Hibernate when you have to define equals and hashCode on the entities and don't have an immutable functional key in the entity.

What kind of inheritance strategies are available? Which annotations are used to solve this?
Relational databases don't have a straightforward way to map class hierarchies onto database tables.

To address this, the JPA specification provides several strategies:

MappedSuperclass – the parent classes, can't be entities (@MappedSuperclass)
Single Table – The entities from different classes with a common ancestor are placed in a single table. (@Inheritance(strategy = InheritanceType.SINGLE_TABLE))
Joined Table – Each class has its table, and querying a subclass entity requires joining the tables. (@Inheritance(strategy = InheritanceType.JOINED))
Table per Class – All the properties of a class are in its table, so no join is required. (@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS))
Each strategy results in a different database structure.

Entity inheritance means that we can use polymorphic queries for retrieving all the subclass entities when querying for a superclass.