**What is JPA?**

**Java Persistence API (JPA)** is a specification for object-relational mapping (ORM) in Java. It allows developers to manage relational data in Java applications using objects rather than direct SQL queries. JPA simplifies database access by abstracting low-level JDBC operations, providing an easier way to persist, retrieve, and manipulate data.

### ****Key Features of JPA****

* Maps Java objects to database tables.
* Provides a query language (**JPQL**) to perform database operations.
* Supports annotations to define mappings and relationships between objects.
* Works with providers like Hibernate, EclipseLink, and OpenJPA.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **JPA** | **JDBC** |
| **Definition** | A high-level ORM specification that abstracts database operations. | A low-level API to execute SQL queries directly. |
| **Ease of Use** | Simplifies development by eliminating boilerplate code. | Requires manual handling of SQL queries and mappings. |
| **Query Language** | Uses **JPQL** (object-oriented query language). | Uses **SQL** directly. |
| **Data Mapping** | Maps Java objects to database tables (ORM). | Requires manual mapping between objects and tables. |
| **Transaction Management** | Simplifies transactions through annotations and configuration. | Requires explicit transaction management. |
| **Vendor Independence** | Independent of the database vendor. | Tied to the SQL dialect of the specific database. |
| **Performance** | Adds overhead due to abstraction but improves developer productivity. | Faster as it directly communicates with the database. |
| **Example** | entityManager.persist(student); | PreparedStatement ps = con.prepareStatement(...); |

|  |  |  |  |
| --- | --- | --- | --- |
| **Method in Service Interface** | **Operation** | **Description** | **Example Implementation in Service** |
| void createStudent(Student student); | Create | Adds a new student to the database. | studentRepository.save(student); |
| Student getStudentById(Long id); | Read (Single Record) | Fetches a student by their ID. | studentRepository.findById(id).orElseThrow(() -> new RuntimeException("Student not found")); |
| List<Student> getAllStudents(); | Read (All Records) | Retrieves all students. | studentRepository.findAll(); |
| Student updateStudent(Long id, Student student); | Update | Updates an existing student's details. | studentRepository.save(student); |
| void deleteStudent(Long id); | Delete | Deletes a student by their ID. | studentRepository.deleteById(id); |
| List<Student> getStudentsByAge(int age); | Filter Using WHERE | Fetches students matching a specific age. | studentRepository.findByAge(age); |
| List<Student> getStudentsByGrade(String grade); | Filter Using = AND | Retrieves students of a specific grade. | studentRepository.findByGrade(grade); |
| List<Student> getStudentsWithMarksGreaterThan(int marks); | Filter Using > | Finds students with marks above a threshold. | studentRepository.findByMarksGreaterThan(marks); |
| List<Student> getStudentsWithinAgeRange(int minAge, int maxAge); | Filter Using BETWEEN | Fetches students within a specific age range. | studentRepository.findByAgeBetween(minAge, maxAge); |
| List<Student> getStudentsByNamePattern(String pattern); | Filter Using LIKE | Finds students whose names match a given pattern. | studentRepository.findByNameLike(pattern); |
| List<Student> getStudentsInGrades(List<String> grades); | Filter Using IN | Retrieves students whose grades are in a specific list. | studentRepository.findByGradeIn(grades); |
| boolean isStudentEmailExist(String email); | Check Using EXISTS | Checks if a student with a given email exists. | studentRepository.existsByEmail(email); |
| List<Student> getStudentsWithNullEmail(); | Filter Using IS NULL | Retrieves students who do not have an email address. | studentRepository.findByEmailIsNull(); |
| List<Student> getTopNStudentsByMarks(int n); | Sorting Using ORDER BY | Retrieves the top N students sorted by marks in descending order. | studentRepository.findTopNByOrderByMarksDesc(n); |
| List<Student> getStudentsByGradeGrouped(); | Aggregation Using GROUP BY | Retrieves the count of students grouped by their grades. | studentRepository.countStudentsGroupedByGrade(); |
| List<Student> getStudentsAboveAverageMarks(); | Filter Using HAVING | Retrieves students whose marks are above the average. | Implemented using JPQL: SELECT s FROM Student s GROUP BY s.grade HAVING AVG(s.marks) > :averageMarks |
| List<Student> getStudentsByCourse(String course); | Join Query | Retrieves students enrolled in a specific course using a join. | @Query("SELECT s FROM Student s JOIN s.courses c WHERE c.name = :courseName") |
| List<Student> getStudentsWithPagination(int page, int size); | Pagination | Fetches students with pagination. | Pageable pageable = PageRequest.of(page, size); studentRepository.findAll(pageable); |
| Student refreshStudentDetails(Long id); | Refresh | Refreshes the student entity state from the database. | entityManager.refresh(student); |
| void lockStudent(Long id); | Locking | Applies a lock to a student record for preventing concurrent modifications. | entityManager.lock(student, LockModeType.PESSIMISTIC\_WRITE); |

|  |
| --- |
| public interface StudentService {  // CRUD Operations  void createStudent(Student student);  Student getStudentById(Long id);  List<Student> getAllStudents();  Student updateStudent(Long id, Student student);  void deleteStudent(Long id);  // Additional Methods  List<Student> getStudentsByAge(int age);  List<Student> getStudentsByGrade(String grade);  List<Student> getStudentsWithMarksGreaterThan(int marks);  List<Student> getStudentsWithinAgeRange(int minAge, int maxAge);  List<Student> getStudentsByNamePattern(String pattern);  List<Student> getStudentsInGrades(List<String> grades);  boolean isStudentEmailExist(String email);  List<Student> getStudentsWithNullEmail();  List<Student> getTopNStudentsByMarks(int n);  List<Student> getStudentsByGradeGrouped();  List<Student> getStudentsAboveAverageMarks();  List<Student> getStudentsByCourse(String course);  List<Student> getStudentsWithPagination(int page, int size);  Student refreshStudentDetails(Long id);  void lockStudent(Long id);  } |

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Operation** | **Example** | **Description** |
| persist(entity) | Create | entityManager.persist(student); | Inserts a new record into the database. |
| find(entityClass, id) | Read | Student student = entityManager.find(Student.class, 1); | Fetches a single entity by primary key. |
| merge(entity) | Update | entityManager.merge(student); | Updates an existing record or inserts if it doesn't exist. |
| remove(entity) | Delete | entityManager.remove(student); | Deletes a specific record from the database. |
| createQuery("...") | Custom Query | entityManager.createQuery("SELECT s FROM Student s WHERE s.age > 18"); | Executes a JPQL query with conditions. |
| setParameter(name, value) | Parameterized Query | query.setParameter("age", 18); | Sets parameters dynamically in a JPQL query. |
| JOIN | Joining Tables | SELECT s FROM Student s JOIN s.courses c WHERE c.name = 'Math'; | Retrieves data using inner joins between related tables. |
| LEFT JOIN | Left Join | SELECT s FROM Student s LEFT JOIN s.courses c ON c.id = s.courseId; | Retrieves data including non-matching rows from the left table. |
| WHERE | Filtering | SELECT s FROM Student s WHERE s.age > 18; | Filters results based on a condition. |
| AND | Multiple Conditions | SELECT s FROM Student s WHERE s.age > 18 AND s.grade = 'A'; | Combines multiple conditions where all must be true. |
| OR | Alternative Conditions | SELECT s FROM Student s WHERE s.age > 18 OR s.grade = 'A'; | Combines conditions where at least one must be true. |
| <, >, = | Comparison Operators | SELECT s FROM Student s WHERE s.age > 18 AND s.grade = 'A'; | Compares values in conditions. |
| BETWEEN | Range Filter | SELECT s FROM Student s WHERE s.age BETWEEN 18 AND 25; | Selects rows where a value lies in a specific range. |
| IN | Set Membership | SELECT s FROM Student s WHERE s.grade IN ('A', 'B', 'C'); | Filters rows where a value matches any in a given set. |
| LIKE | Pattern Matching | SELECT s FROM Student s WHERE s.name LIKE 'J%'; | Filters rows based on a pattern (e.g., starts with J). |
| IS NULL | Null Check | SELECT s FROM Student s WHERE s.email IS NULL; | Filters rows where a column is null. |
| IS NOT NULL | Non-Null Check | SELECT s FROM Student s WHERE s.email IS NOT NULL; | Filters rows where a column is not null. |
| ORDER BY | Sorting | SELECT s FROM Student s ORDER BY s.age DESC; | Sorts rows based on a specified column. |
| GROUP BY | Grouping | SELECT s.grade, COUNT(s) FROM Student s GROUP BY s.grade; | Groups rows by a specific column and performs aggregate functions. |
| HAVING | Filtering Groups | SELECT s.grade, COUNT(s) FROM Student s GROUP BY s.grade HAVING COUNT(s) > 5; | Filters groups created using GROUP BY. |
| DISTINCT | Unique Results | SELECT DISTINCT s.grade FROM Student s; | Selects unique rows by eliminating duplicates. |
| COUNT() | Count Rows | SELECT COUNT(s) FROM Student s; | Counts the number of rows in a result set. |
| AVG() | Average | SELECT AVG(s.age) FROM Student s; | Calculates the average of a column's values. |
| SUM() | Sum | SELECT SUM(s.marks) FROM Student s; | Calculates the sum of a column's values. |
| MAX() | Maximum Value | SELECT MAX(s.marks) FROM Student s; | Finds the maximum value in a column. |
| MIN() | Minimum Value | SELECT MIN(s.age) FROM Student s; | Finds the minimum value in a column. |
| EXISTS | Subquery Check | SELECT s FROM Student s WHERE EXISTS (SELECT c FROM Course c WHERE c.studentId = s.id); | Checks for the existence of rows satisfying a condition in a subquery. |
| NOT | Negation | SELECT s FROM Student s WHERE NOT s.age > 18; | Reverses a condition's truth value. |
| CASE | Conditional Logic | SELECT CASE WHEN s.marks > 50 THEN 'Pass' ELSE 'Fail' END FROM Student s; | Implements conditional logic within a query. |
| INNER JOIN FETCH | Eager Fetch | SELECT s FROM Student s JOIN FETCH s.courses; | Retrieves related entities in a single query to avoid lazy loading. |
| LEFT JOIN FETCH | Left Fetch Join | SELECT s FROM Student s LEFT JOIN FETCH s.courses; | Eagerly fetches relationships, including non-matching entities. |
| NativeQuery | Raw SQL Query | entityManager.createNativeQuery("SELECT \* FROM student WHERE age > 18", Student.class).getResultList(); | Executes raw SQL queries directly in the database. |
| NamedQuery | Predefined Query | @NamedQuery(name = "Student.findByName", query = "SELECT s FROM Student s WHERE s.name = :name") | Defines and reuses queries using annotations. |
| LIMIT | Limit Results | SELECT s FROM Student s WHERE s.age > 18 LIMIT 10; | Restricts the number of rows returned (requires native SQL in JPA). |
| OFFSET | Pagination Start | SELECT s FROM Student s WHERE s.age > 18 OFFSET 20; | Skips a specified number of rows (requires native SQL in JPA). |
| flush() | Synchronize Context | entityManager.flush(); | Synchronizes the persistence context with the database. |
| refresh(entity) | Reload Entity | entityManager.refresh(student); | Reloads an entity's state from the database. |
| detach(entity) | Detach Entity | entityManager.detach(student); | Detaches an entity from the persistence context. |
| clear() | Clear Context | entityManager.clear(); | Clears the persistence context, detaching all managed entities. |
| lock(entity, LockModeType) | Lock Entity | entityManager.lock(student, LockModeType.PESSIMISTIC\_WRITE); | Applies a database lock to an entity. |
| commit() | Commit Transaction | entityManager.getTransaction().commit(); | Commits changes made within a transaction. |
| rollback() | Rollback Transaction | entityManager.getTransaction().rollback(); | Rolls back changes made within a transaction. |

### ****Thymeleaf in Spring Boot****

**Thymeleaf** is a modern, server-side Java template engine for web and standalone applications. It is widely used with Spring Boot to build dynamic, server-rendered HTML pages. Thymeleaf supports natural templating, which allows templates to be run as static mock-ups without a backend.

### ****Key Features of Thymeleaf****

* **Natural Templates**: Can be opened directly in a browser as valid HTML.
* **Spring Integration**: Offers seamless integration with Spring MVC and Spring Boot.
* **Template Engine**: Supports conditional rendering, loops, and expression-based dynamic content.
* **Internationalization**: Supports message bundles for localization.
* **Fragment Inclusion**: Reuses common template parts using fragments.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **JSP (JavaServer Pages)** | **Thymeleaf** |
| **Definition** | JSP is a server-side technology that uses Java code embedded in HTML for dynamically generating web pages. | Thymeleaf is a modern, server-side Java template engine that focuses on serving well-formed XML/HTML and can work both server-side and client-side. |
| **Syntax Style** | Uses a combination of HTML and Java code (scriptlets, expressions, or JSTL tags). | Uses pure HTML-like syntax with attributes prefixed by th: for dynamic content. |
| **Integration with Spring Boot** | Can be integrated with Spring Boot but requires additional configurations. | Seamlessly integrated with Spring Boot and is often the default choice in Spring Boot projects. |
| **Rendering Approach** | The rendered HTML is generated only on the server-side. | Allows natural templating, meaning the templates are valid HTML even without being processed by the backend. |
| **Ease of Use** | Requires knowledge of Java code to write dynamic content. | Easier for designers to use, as templates are valid HTML and don’t need Java knowledge. |
| **Code Separation** | Allows mixing Java code with HTML using scriptlets, leading to less separation of concerns. | Promotes clean separation of logic and presentation by removing Java code from the templates. |
| **Debugging** | Can be harder to debug due to scriptlet-based code and lack of support for IDE-friendly syntax highlighting. | Easier to debug as templates are well-formed and IDEs provide support for Thymeleaf syntax. |
| **Performance** | Faster rendering as it is compiled into servlets, but can lead to slower performance when overused with scriptlets or heavy logic. | May have slightly slower rendering compared to JSP due to parsing the HTML structure but scales better with modern development practices. |
| **Template Validation** | JSP files are not valid HTML files and cannot be previewed directly in a browser without server processing. | Thymeleaf templates are valid HTML and can be viewed directly in browsers for mock-ups. |
| **Dynamic Content Handling** | Uses <%= %> for outputting expressions and JavaScript logic. | Uses ${} for outputting variables and offers a wide range of th:\* attributes for dynamic handling. |
| **Reusability** | Allows inclusion with <jsp:include> and tag libraries. | Encourages reusability through fragments and layouts (e.g., th:fragment, th:replace, th:include). |
| **Community and Popularity** | Older technology with a long history; still in use but gradually being replaced in modern Spring projects. | Gaining popularity, especially with Spring Boot projects, due to its clean syntax and powerful features. |
| **Form Handling** | Handles forms via JSP tags or custom Java code. | Provides built-in support for binding forms to objects using attributes like th:field, and integrates seamlessly with Spring’s form backing beans. |
| **Tooling Support** | IDEs like Eclipse and IntelliJ support JSP but may lack modern features for syntax checking or debugging. | Better tooling support in IDEs, including syntax highlighting and error detection. |
| **Error Handling** | Errors can propagate directly to the user if not handled correctly, and debugging stack traces can be cumbersome. | Offers better error handling with more descriptive messages in templates and server logs. |
| **Use Cases** | Suitable for legacy projects or when backward compatibility is required. | Preferred for modern, MVC-based Spring Boot applications, especially when clean code and maintainability are priorities. |
| **Internationalization** | Requires external tools or libraries for i18n. | Built-in support for internationalization through Spring message bundles. |
| **Learning Curve** | Easier for Java developers but may be harder for non-developers due to the need to understand Java and server-side scripting. | Easier for both developers and designers, as it allows for a clear division of labor and uses natural templating. |
| **Template Execution** | Tightly coupled with the servlet container and cannot be executed in a client-side context. | Can be executed server-side (for dynamic pages) or client-side (for static mock-ups). |

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Example** |
| th:text | Replaces the content of the element with dynamic data. | <p th:text="${student.name}">Name Placeholder</p> |
| th:utext | Inserts unescaped HTML content. | <p th:utext="${htmlContent}"></p> |
| th:if | Conditionally renders the element if the expression is true. | <p th:if="${student.age > 18}">Adult</p> |
| th:unless | Conditionally renders the element if the expression is false. | <p th:unless="${student.active}">Inactive</p> |
| th:each | Iterates over a collection or array. | <li th:each="student : ${students}">${student.name}</li> |
| th:href | Dynamically sets the href attribute of a link. | <a th:href="@{/students}">Students</a> |
| th:src | Dynamically sets the src attribute of an image or script. | <img th:src="@{/images/logo.png}" alt="Logo"/> |
| th:class | Dynamically sets the class attribute. | <div th:class="${isActive ? 'active' : 'inactive'}"></div> |
| th:style | Dynamically sets the style attribute. | <div th:style="'color:' + color"></div> |
| th:attr | Dynamically sets multiple attributes at once. | <input th:attr="type='text', placeholder='Enter name'"/> |
| th:value | Sets the value attribute for form inputs. | <input type="text" th:value="${student.name}"/> |
| th:action | Sets the form's action attribute dynamically. | <form th:action="@{/submit}" method="post"></form> |
| th:method | Sets the HTTP method of a form dynamically. | <form th:method="'post'"></form> |
| th:checked | Dynamically sets the checked attribute for checkboxes or radio buttons. | <input type="checkbox" th:checked="${student.active}"/> |
| th:selected | Dynamically sets the selected attribute for dropdown options. | <option th:selected="${option == selectedOption}">Option</option> |
| th:replace | Replaces the content of the tag with a fragment. | <div th:replace="fragments/header :: header"></div> |
| th:include | Includes a fragment inside the tag. | <div th:include="fragments/footer :: footer"></div> |
| th:remove | Removes the element based on the specified condition (all, tag, body, etc.). | <p th:remove="all">Temporary Content</p> |
| th:object | Binds a model object to a form for data binding. | <form th:object="${student}"></form> |
| th:field | Dynamically binds form fields to model properties. | <input th:field="\*{name}"/> |
| th:errorclass | Adds a CSS class to fields with validation errors. | <input th:field="\*{name}" th:errorclass="error"/> |
| th:fragment | Declares a reusable template fragment. | <div th:fragment="header">Header Content</div> |
| th:with | Defines variables scoped within the element and its children. | <div th:with="total=${students.size()}">Total: <span th:text="${total}"></span></div> |
| th:inline | Enables inlining for JavaScript or CSS within templates. | <script type="text/javascript" th:inline="javascript">var name = [[${student.name}]];</script> |
| th:switch and th:case | Implements switch-case functionality. | <div th:switch="${status}"><p th:case="'active'">Active</p><p th:case="'inactive'">Inactive</p></div> |
| th:block | Groups multiple elements without adding a new HTML tag. | <th:block th:if="${condition}"><p>Condition Met</p></th:block> |
| th:disabled | Dynamically sets the disabled attribute. | <button th:disabled="${!isEnabled}">Click</button> |
| th:readonly | Dynamically sets the readonly attribute for form inputs. | <input type="text" th:readonly="${isReadonly}"/> |
| th:lang | Dynamically sets the language attribute of the element. | <html th:lang="${lang}"></html> |
| th:data-\* | Sets custom data attributes dynamically. | <div th:data-id="${student.id}" th:data-role="${student.role}"></div> |
| th:on\* (e.g., th:onclick) | Dynamically binds event handlers such as onclick, onchange, etc. | <button th:onclick="'handleClick(' + student.id + ')'">Click</button> |
| Attribute | Description | Example |
| th:text | Replaces the content of the element with dynamic data. | <p th:text="${student.name}">Name Placeholder</p> |
| th:utext | Inserts unescaped HTML content. | <p th:utext="${htmlContent}"></p> |
| th:if | Conditionally renders the element if the expression is true. | <p th:if="${student.age > 18}">Adult</p> |
| th:unless | Conditionally renders the element if the expression is false. | <p th:unless="${student.active}">Inactive</p> |
| th:each | Iterates over a collection or array. | <li th:each="student : ${students}">${student.name}</li> |
| th:href | Dynamically sets the href attribute of a link. | <a th:href="@{/students}">Students</a> |
| th:src | Dynamically sets the src attribute of an image or script. | <img th:src="@{/images/logo.png}" alt="Logo"/> |
| th:class | Dynamically sets the class attribute. | <div th:class="${isActive ? 'active' : 'inactive'}"></div> |
| th:style | Dynamically sets the style attribute. | <div th:style="'color:' + color"></div> |
| th:attr | Dynamically sets multiple attributes at once. | <input th:attr="type='text', placeholder='Enter name'"/> |
| th:value | Sets the value attribute for form inputs. | <input type="text" th:value="${student.name}"/> |
| th:action | Sets the form's action attribute dynamically. | <form th:action="@{/submit}" method="post"></form> |
| th:method | Sets the HTTP method of a form dynamically. | <form th:method="'post'"></form> |
| th:checked | Dynamically sets the checked attribute for checkboxes or radio buttons. | <input type="checkbox" th:checked="${student.active}"/> |
| th:selected | Dynamically sets the selected attribute for dropdown options. | <option th:selected="${option == selectedOption}">Option</option> |
| th:replace | Replaces the content of the tag with a fragment. | <div th:replace="fragments/header :: header"></div> |
| th:include | Includes a fragment inside the tag. | <div th:include="fragments/footer :: footer"></div> |
| th:remove | Removes the element based on the specified condition (all, tag, body, etc.). | <p th:remove="all">Temporary Content</p> |
| th:object | Binds a model object to a form for data binding. | <form th:object="${student}"></form> |
| th:field | Dynamically binds form fields to model properties. | <input th:field="\*{name}"/> |
| th:errorclass | Adds a CSS class to fields with validation errors. | <input th:field="\*{name}" th:errorclass="error"/> |
| th:fragment | Declares a reusable template fragment. | <div th:fragment="header">Header Content</div> |
| th:with | Defines variables scoped within the element and its children. | <div th:with="total=${students.size()}">Total: <span th:text="${total}"></span></div> |
| th:inline | Enables inlining for JavaScript or CSS within templates. | <script type="text/javascript" th:inline="javascript">var name = [[${student.name}]];</script> |
| th:switch and th:case | Implements switch-case functionality. | <div th:switch="${status}"><p th:case="'active'">Active</p><p th:case="'inactive'">Inactive</p></div> |
| th:block | Groups multiple elements without adding a new HTML tag. | <th:block th:if="${condition}"><p>Condition Met</p></th:block> |
| th:disabled | Dynamically sets the disabled attribute. | <button th:disabled="${!isEnabled}">Click</button> |
| th:readonly | Dynamically sets the readonly attribute for form inputs. | <input type="text" th:readonly="${isReadonly}"/> |
| th:lang | Dynamically sets the language attribute of the element. | <html th:lang="${lang}"></html> |
| th:data-\* | Sets custom data attributes dynamically. | <div th:data-id="${student.id}" th:data-role="${student.role}"></div> |
| th:on\* (e.g., th:onclick) | Dynamically binds event handlers such as onclick, onchange, etc. | <button th:onclick="'handleClick(' + student.id + ')'">Click</button> |

|  |
| --- |
| <!DOCTYPE html>  <html xmlns:th="http://www.thymeleaf.org">  <head>  <title>Student Page</title>  </head>  <body>  <h1 th:text="${title}">Student Information</h1>  <ul>  <li th:each="student : ${students}" th:text="${student.name}"></li>  </ul>  <form th:action="@{/saveStudent}" th:object="${student}" method="post">  <input type="text" th:field="\*{name}" placeholder="Name"/>  <button type="submit">Save</button>  </form>  </body>  </html> |





