**Hibernate Guide: Basics to Advanced**

**1. Introduction to Hibernate**

Hibernate is an open-source ORM (Object-Relational Mapping) framework for Java applications, enabling developers to interact with databases using Java objects. It simplifies database operations by handling object persistence and retrieval.

**Features of Hibernate:**

* Eliminates boilerplate JDBC code
* Provides caching mechanisms
* Supports automatic table creation
* Offers HQL (Hibernate Query Language) for querying
* Supports transaction management

**2. Hibernate Architecture**

Hibernate operates on a layered architecture:

1. **Session Factory** - Manages sessions and database connections.
2. **Session** - Acts as an interface between Java objects and the database.
3. **Transaction** - Manages database transactions.
4. **Query** - Provides APIs for database queries.
5. **Configuration** - Reads the hibernate.cfg.xml file to configure the database.
6. **JDBC Connection** - Provides low-level JDBC interaction.

**3. Configuring Hibernate**

To configure Hibernate, follow these steps:

1. Add Hibernate dependencies in pom.xml (for Maven projects):

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.6.9.Final</version>

</dependency>

1. Create a Hibernate configuration file (hibernate.cfg.xml):

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">password</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

</session-factory>

</hibernate-configuration>

**4. Hibernate Annotations**

Hibernate supports annotations to define entity relationships and table mapping.

* @Entity - Defines a class as an entity.
* @Table(name="table\_name") - Maps entity to a database table.
* @Id - Defines the primary key.
* @GeneratedValue(strategy = GenerationType.IDENTITY) - Enables auto-increment.
* @Column(name="column\_name") - Maps a field to a database column.
* @OneToOne, @OneToMany, @ManyToOne, @ManyToMany - Define relationships.

Example:

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name")

private String name;

// Getters and Setters

}

**5. Hibernate CRUD Operations**

**Create:**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Student student = new Student("John Doe");

session.save(student);

tx.commit();

session.close();

**Read:**

Session session = sessionFactory.openSession();

Student student = session.get(Student.class, 1);

System.out.println(student.getName());

session.close();

**Update:**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Student student = session.get(Student.class, 1);

student.setName("Jane Doe");

session.update(student);

tx.commit();

session.close();

**Delete:**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Student student = session.get(Student.class, 1);

session.delete(student);

tx.commit();

session.close();

**6. Hibernate Caching**

Hibernate provides two levels of caching:

1. **First-Level Cache**: Enabled by default, stores objects in session.
2. **Second-Level Cache**: Stores objects across multiple sessions.
   * Common providers: EhCache, Hazelcast, Redis.

Enable second-level caching:

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

**7. Hibernate Query Language (HQL)**

HQL is an object-oriented query language similar to SQL but works with entity objects.

Example:

String hql = "FROM Student WHERE name = :name";

Query query = session.createQuery(hql);

query.setParameter("name", "John Doe");

List<Student> students = query.list();

**8. Integration with Spring Boot**

Hibernate is commonly integrated with Spring Boot for database handling.

**Steps:**

1. Add dependencies:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

1. Configure application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/mydb

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

1. Create a repository:

@Repository

public interface StudentRepository extends JpaRepository<Student, Integer> {}

**9. Spring Boot Annotations**

**Key Annotations:**

* @SpringBootApplication - Main entry point of the application.
* @Autowired - Dependency injection.
* @RestController - Defines a REST API controller.
* @RequestMapping, @GetMapping, @PostMapping - Define HTTP routes.
* @ComponentScan(basePackages = {"com.example"}) - Scans for Spring components.

Example:

@RestController

@RequestMapping("/students")

public class StudentController {

@Autowired

private StudentRepository repository;

@GetMapping

public List<Student> getAllStudents() {

return repository.findAll();

}

}

**10. Spring Boot Validation**

Spring Boot provides built-in validation using javax.validation.

* @NotNull - Ensures field is not null.
* @Size(min=, max=) - Defines length constraints.
* @Email - Ensures valid email format.
* @Valid - Validates an object.

Example:

public class User {

@NotNull

@Size(min = 2, max = 30)

private String name;

@Email

private String email;

}

This guide provides a complete overview of Hibernate, from basic setup to advanced integrations.