**Answer all the questions**

Here are the answers to all the questions from your Spring Data JPA hands-on document, formatted concisely and clearly as requested.

**1. Need and Benefit of ORM**

**ORM (Object-Relational Mapping)** allows developers to interact with a database using Java objects, abstracting away SQL.  
**Benefits:**

* Simplifies database interactions.
* Reduces boilerplate code.
* Makes code database-agnostic.
* Manages transactions and relationships automatically.

**2. Need and Benefit of Spring Data JPA**

* Provides an abstraction over JPA and Hibernate.
* Reduces boilerplate code for repository and CRUD operations.
* Supports database-independent queries.
* Integrates easily with Spring Boot and supports automatic transaction management.

**3. Core Objects of Hibernate Framework**

* **SessionFactory:** Creates Session objects; thread-safe and heavy-weight.
* **Session:** Represents a single unit of work with the database.
* **Transaction:** Handles atomic units of work.
* **ConnectionProvider:** Provides JDBC connections to Hibernate.

**4. ORM Implementation with Hibernate XML and Annotation Configuration**

* **XML Configuration:** Maps classes to tables and fields to columns in XML files. Requires manual configuration of mappings and properties.
* **Annotation Configuration:** Uses annotations like @Entity, @Table, @Id, @Column in Java classes for mapping.
* Both require a configuration file (hibernate.cfg.xml) for database connection settings.

**5. Difference Between JPA, Hibernate, and Spring Data JPA**

* **JPA:** Specification for ORM in Java (no implementation).
* **Hibernate:** Popular ORM implementation of JPA.
* **Spring Data JPA:** Abstraction over JPA/Hibernate, reduces boilerplate, provides repository pattern, and integrates with Spring.

**6. DML Using Spring Data JPA**

* **findById()**: Retrieves an entity by its ID.
* **save()**: Persists or updates an entity.
* **deleteById()**: Deletes an entity by ID.
* **Query Methods:** Define custom queries by method naming conventions.

**7. Hibernate Table Creation Configuration (ddl-auto)**

* **create:** Drops and recreates tables each time.
* **validate:** Validates schema, throws error if mismatched.
* **update:** Updates schema without dropping data.
* **create-drop:** Creates tables and drops them after session ends.

**8. Country Table O/R Mapping**

**Entity:**

@Entity  
@Table(name="country")  
public class Country {  
 @Id  
 @Column(name="code")  
 private String code;  
 @Column(name="name")  
 private String name;  
 // getters, setters, toString()  
}

**Repository:**

@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {}

**Service:**

@Service  
public class CountryService {  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

**9. Country Service Features**

* **Find by code:** countryRepository.findById(code)
* **Add:** countryRepository.save(country)
* **Update:** countryRepository.save(country)
* **Delete:** countryRepository.deleteById(code)
* **Find by partial name:** countryRepository.findByNameContaining(String partialName)

**10. Hibernate XML Config Implementation**

* **SessionFactory:** Configured in hibernate.cfg.xml.
* **Session:** Opened from SessionFactory.
* **Transaction:** Begin, commit, or rollback as needed.
* **CRUD:** Use session.save(), session.get(), session.delete(), session.createQuery().list().

**11. Hibernate Annotation Config Implementation**

* **Entity Class:** Annotated with @Entity, @Table, @Id, @GeneratedValue, @Column.
* **Configuration:** Database connection settings in hibernate.cfg.xml.
* **Operations:** Use SessionFactory, Session, and Transaction as in XML config.

**12. Example: Adding Employee Using Hibernate vs Spring Data JPA**

**Hibernate:**

Session session = factory.openSession();  
Transaction tx = session.beginTransaction();  
session.save(employee);  
tx.commit();  
session.close();

**Spring Data JPA:**

@Autowired  
private EmployeeRepository employeeRepository;  
  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
}

**13. Populate Country Table**

Use SQL insert statements to add all countries to the country table.

**14. Testing Country Service**

* Autowire CountryService in main application.
* Call service methods to test CRUD and search operations.

If you need this content as a PDF, please use the "Export as PDF" feature in your editor or let me know if you want a downloadable PDF file generated.

⁂