Hibernate Notes

1. **Hibernate Configuration**

Hibernate configuration is essential to set up the environment for interacting with the database. This includes the Hibernate configuration file (`hibernate.cfg.xml`), which contains database connection settings and Hibernate-specific properties.

Key Components of ******`hibernate.cfg.xml`********:**

- **Database Connection Settings:**

  ```xml

  <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>

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

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

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

  ```

- **Dialect:** Specifies the SQL dialect for the database.

  ```xml

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

  ```

- **Hibernate Properties:**

  - `hbm2ddl.auto`: Manages schema updates (`update`, `create`, `validate`, `none`).

  - `show_sql`: Displays SQL queries in the console.

  ```xml

  <property name="hibernate.hbm2ddl.auto">update</property>

  <property name="hibernate.show_sql">true</property>

  ```

- **Mapping Resource:** Connects entities to Hibernate.

  ```xml
  ```

<mapping class="com.example.EntityClass" />
```

### 2. **SessionFactory and Session**

#### **SessionFactory:**

- A heavyweight, thread-safe object responsible for creating `Session` objects.
- Initialized once during application startup.

#### **Session:**

- A lightweight, non-thread-safe object for interacting with the database.
- Handles CRUD (Create, Read, Update, Delete) operations.

**Usage Example:**

```java
SessionFactory factory = new Configuration().configure().buildSessionFactory();
Session session = factory.openSession();
session.beginTransaction();
// Perform database operations
session.getTransaction().commit();
session.close();
```

3. **Entity and Annotations**

An **Entity** represents a table in the database, and each instance of the entity represents a row.

Key Annotations:
```

- `@Entity`: Marks a class as an entity.

- `@Table(name = "table_name")`: Specifies the table name (optional if class name matches table name).

- `@Id`: Denotes the primary key.

- `@GeneratedValue`: Defines how the primary key is generated.

- `@Column(name = "column_name")`: Maps a field to a specific column.

Example:

```java
@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "email")
    private String email;

    // Getters and Setters
}
```

4. **Common Hibernate Annotations:**

- `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`: For defining relationships.

- `@JoinColumn`: Specifies the foreign key column.
- `@Transient`: Excludes a field from being persisted.
- `@Lob`: For large objects like text and binary data.
- `@Temporal`: Specifies date/time type (`DATE`, `TIME`, `TIMESTAMP`).

5. **Workflow of Hibernate Application:**

1. **Create Configuration:** Define database connection and Hibernate properties.
2. **Build SessionFactory:** Initialize a single instance.
3. **Open Session:** For database interactions.
4. **Begin Transaction:** To ensure atomicity.
5. **Perform Operations:** CRUD operations using `session.save()`, `session.get()`, etc.
6. **Commit Transaction:** Save changes to the database.
7. **Close Session:** Release resources.

6. **CRUD Operations Example:**

```java
// Create
Session session = factory.openSession();
session.beginTransaction();
Student student = new Student("John Doe", "john@example.com");
session.save(student);
session.getTransaction().commit();
session.close();

// Read
session = factory.openSession();
Student student = session.get(Student.class, 1);
session.close();
```

// Update

session = factory.openSession();

session.beginTransaction();

student.setName("Jane Doe");

session.update(student);

session.getTransaction().commit();

session.close();

// Delete

session = factory.openSession();

session.beginTransaction();

session.delete(student);

session.getTransaction().commit();

session.close();

```

**### 7. **Conclusion:****

**- Hibernate simplifies database interactions using ORM principles.**

**- Configuration, SessionFactory, and Entities are the core components.**

**- Annotations make the mapping process easier and cleaner.**

**- Efficient for managing large-scale applications with complex data models.**

**Running a Restaurant: Hibernate Analogy**

1. **Configuration File (config.xml) → The Restaurant Blueprint**
    - **Role:** Sets up the rules and instructions for how the restaurant should operate.
    - **Real-life Example:**
        - Where the restaurant is located (database URL)
        - Who's the manager with access (username/password)
        - The type of food served (SQL dialect)
        - Whether to update the menu automatically (hbm2ddl.auto = update)
```

        - Without this blueprint, the restaurant wouldn't know how to connect to suppliers (the database) or how to operate.
    - **Hibernate Configuration:** The restaurant's blueprint is represented by the hibernate.cfg.xml file. This file contains settings for the restaurant's database, such as the database connection, SQL dialect, and Hibernate-specific settings like automatic schema updates (hbm2ddl.auto).

Key Components of hibernate.cfg.xml:

xml

CopyEdit

<property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>

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

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

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

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

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="hibernate.show_sql">true</property>

<mapping class="com.example.EntityClass" />

2. **Entity Class (Users.java) → The Menu Card**
    - **Role:** Defines what items (data) the restaurant offers and how they are organized.
    - **Real-life Example:**
        - Each dish on the menu has a name (username) and ingredients (password).
        - The menu card follows a specific layout (table structure in the database).
        - Some dishes are special and must be unique (primary key with @Id).
        - This class helps the restaurant keep track of the dishes (user data) it can serve to customers.
    - **Entity in Hibernate:** The menu card corresponds to an Entity in Hibernate, which represents a table in the database. The entity class uses annotations to define how the table is structured, just like the menu defines how the dishes are organized.

Entity Class Example:

java

CopyEdit

@Entity

@Table(name = "users")

```
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;

   @Column(name = "username")
   private String username;

   @Column(name = "password")
   private String password;

   // Getters and Setters
}
```

3. **Main Application (mainApp.java) → The Waiter**
    - **Role:** Takes orders from customers and interacts with the kitchen (database) to process them.
    - **Real-life Example:**
        - The waiter receives an order (creates a new Users object).
        - Checks the restaurant rules (loads the config.xml).
        - Takes the order to the kitchen (opens a Session).
        - Confirms the order with the chef (starts a Transaction).
        - Finalizes the order and delivers it to the customer (commits the transaction).
    - **Session in Hibernate:** The waiter represents the Session in Hibernate, which interacts with the database. The waiter ensures that each order is handled following the restaurant's rules (configuration file) and makes sure everything runs smoothly by committing the transaction (saving changes to the database).

Session Example:

```
java
CopyEdit
SessionFactory factory = new Configuration().configure().buildSessionFactory();
Session session = factory.openSession();
session.beginTransaction();
```

```
User user = new User("johnDoe", "password123");

session.save(user);


session.getTransaction().commit();

session.close();
```

Additional Hibernate Notes:

1. **SessionFactory and Session:**
    - **SessionFactory:** A heavyweight, thread-safe object that creates Session objects, much like the restaurant's kitchen setup, which can take multiple orders at once.
    - **Session:** A lightweight, non-thread-safe object for interacting with the database, like an individual waiter serving one order.
2. **Hibernate Annotations:**
    - **@Entity:** Marks a class as an entity (like the menu item being served).
    - **@Id:** Denotes the primary key for the table (just like a unique dish ID).
    - **@GeneratedValue:** Specifies how the primary key is generated (like dish identifiers).
    - **@Column:** Maps class fields to table columns (like the dish details being presented).

DTD (Document Type Definition):

The DTD is used in Hibernate configuration to define the structure of the hibernate.cfg.xml file. It ensures that all required properties are present and correctly formatted.

xml

CopyEdit

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <!-- Configuration details -->

</hibernate-configuration>
```