Below is a **clear, short, and exact explanation** of **what `EmployeeMapper` does**, why it exists, and where it fits in your Spring Boot architecture.

---

1. What Is `EmployeeMapper`? (One Line)

> `EmployeeMapper` **converts data between different object types**, typically **Entity** ↔ **DTO**, without containing business logic.

---

2. Why `EmployeeMapper` Exists

In a clean Spring application, **each layer uses a different model**:

Layer	Object Type
Controller	DTO
Service	DTO
Repository	Entity

Mapper **bridges this gap**.

---

3. What `EmployeeMapper` Actually Does

Example Conversions

- `Employee` → `EmployeeDTO`
- `EmployeeDTO` → `Employee`

That's it. No validation. No database access. No business rules.

---

4. Typical `EmployeeMapper` Example

```
@Component
public class EmployeeMapper {

    public EmployeeDTO toDTO(Employee employee) {
        return new EmployeeDTO(
            employee.getId(),
            employee.getFirstName(),
            employee.getLastName(),
            employee.getEmail()
        );
    }
```

```
    public Employee toEntity(EmployeeDTO dto) {
        Employee employee = new Employee();
        employee.setFirstName(dto.getFirstName());
        employee.setLastName(dto.getLastName());
        employee.setEmail(dto.getEmail());
        return employee;
    }
}
```

---

5. Who Uses `EmployeeMapper`

```
Controller → Service → Mapper → Entity
```

More precisely:

- **Service calls Mapper**
- **Mapper returns converted object**
- **Service continues its logic**

Controller never calls mapper directly (best practice).

---

6. Why Not Put Mapping in Service?

You *can*, but it causes:

- Bloated service classes
- Repeated mapping code
- Poor testability

Mapper gives:

- Single responsibility
- Cleaner services
- Easier refactoring

---

7. Where `EmployeeMapper` Fits

```
Controller
   ↓
Service
   ↓
EmployeeMapper
```

```
    ↓
Repository
```

---

8. What `EmployeeMapper` Must NOT Do

❌ Access database ❌ Contain business logic ❌ Handle HTTP ❌ Throw domain exceptions

It is a **pure transformation utility**.

---

9. Interview-Quality Answer (Use This)

> "An `EmployeeMapper` is responsible for transforming data between DTOs and entities, ensuring separation between API models and persistence models without leaking concerns across layers."

---

10. Manual Mapper vs MapStruct

Manual Mapper	MapStruct
Easy to understand	Compile-time generated
More boilerplate	Less boilerplate
Beginner-friendly	Industry-preferred

---

11. One Sentence to Remember

> **Mapper = Translator, not Decision Maker**

If you want next:

- Where mapper should live in package structure
- MapStruct implementation
- When mapper is overkill

Just say the topic.