Mapper

If the Doctor ID is present in the Doctor DTO but not in the Doctor Entity, you can still use MapStruct to map between them. You just need to define the mapping to explicitly ignore the Doctor ID field when mapping from DTO to Entity.

Here's how you can do it:

```java
// DoctorDto.java
public class DoctorDto {
    private Long id;
    private String name;
    private String specialization;
    private Long doctorId;

    // Getters and setters
}

// DoctorEntity.java
public class DoctorEntity {
    private Long id;
    private String name;
    private String specialization;

    // Getters and setters
}
```

Your Doctor DTO has an additional field `doctorId` compared to the Doctor Entity.

Now, let's define the mapper interface using MapStruct:

```java
// DoctorMapper.java
@Mapper
public interface DoctorMapper {

    DoctorMapper INSTANCE = Mappers.getMapper(DoctorMapper.class);

    // Method to map from DoctorDto to DoctorEntity
    @Mapping(target = "id", ignore = true) // Ignore Doctor ID when mapping to
Entity
    DoctorEntity dtoToEntity(DoctorDto doctorDto);
    // Method to map from DoctorEntity to DoctorDto
    DoctorDto entityToDto(DoctorEntity doctorEntity);
}
```

In the `dtoToEntity` method, we use `@Mapping` annotation to ignore the `id` field of the DoctorEntity. This way, when mapping from DoctorDto to DoctorEntity, the Doctor ID field will be ignored.

Now, you can use these methods to map between DoctorDto and DoctorEntity objects:

```java
// Mapping from DoctorDto to DoctorEntity
DoctorDto doctorDto = new DoctorDto();
doctorDto.setId(1L);
doctorDto.setName("Dr. John");
doctorDto.setSpecialization("Cardiologist");
doctorDto.setDoctorId(101L);

DoctorEntity doctorEntity = DoctorMapper.INSTANCE.dtoToEntity(doctorDto);

// Mapping from DoctorEntity to DoctorDto
DoctorEntity doctorEntity = new DoctorEntity();
doctorEntity.setId(1L);
doctorEntity.setName("Dr. John");
doctorEntity.setSpecialization("Cardiologist");

DoctorDto doctorDto = DoctorMapper.INSTANCE.entityToDto(doctorEntity);
```

MapStruct will generate the mapping implementation code during compile-time, and it will ignore the Doctor ID field when mapping from DTO to Entity as specified in the mapper interface.

The `@Mapping(target = "id")` annotation in MapStruct is used to specify the target property for a mapping. It tells MapStruct to map the source property to the target property with the same name.

For example, let's say you have a source class `Source` and a target class `Target`, and you want to map the `sourceId` property of the source class to the `id` property of the target class. You can use `@Mapping(target = "id")` to specify this mapping.

Here's an example:

```java
// Source.java
public class Source {
    private Long sourceId;

    // Getter and setter
}

// Target.java
public class Target {
    private Long id;

    // Getter and setter
}
```

And your mapper interface would look like this:

```java
// MyMapper.java
@Mapper
public interface MyMapper {

    MyMapper INSTANCE = Mappers.getMapper(MyMapper.class);

    @Mapping(target = "id", source = "sourceId")
    Target sourceToTarget(Source source);
}
```

In this example, the `@Mapping(target = "id", source = "sourceId")` annotation specifies that the `sourceId` property of the `Source` class should be mapped to the `id` property of the `Target` class. The `source` attribute is optional and can be omitted if the source property name matches the target property name.

So, whenever you invoke `sourceToTarget` method of `MyMapper`, MapStruct will automatically map the `sourceId` property of the source object to the `id` property of the target object based on the mapping defined in the annotation.

To ignore mapping for null fields that are not present in the DTO, you can use the `nullValuePropertyMappingStrategy` attribute in MapStruct. This attribute allows you to define a strategy for handling null values during mapping.

Here's how you can configure MapStruct to ignore mapping for null fields:

```java
// MyMapper.java
@Mapper(nullValuePropertyMappingStrategy =
NullValuePropertyMappingStrategy.IGNORE)
public interface MyMapper {

    MyMapper INSTANCE = Mappers.getMapper(MyMapper.class);

    @Mapping(target = "id", source = "sourceId")
    Target sourceToTarget(Source source);
}
```

In this example, by setting `nullValuePropertyMappingStrategy` to `IGNORE`, MapStruct will not map the properties from the source object to the target object if the source properties are null.

So, if the `sourceId` property in the `Source` object is null, MapStruct will not set the `id` property in the `Target` object, effectively ignoring the mapping for null fields that are not present in the DTO.

If you have extra properties in the DTO that are not present in the entity and you want to ignore mapping for those properties, you can use the `ignoreUnknownProperties` attribute in MapStruct. This attribute allows you to specify whether extra properties in the source object (DTO) should be ignored during mapping.

Here's how you can configure MapStruct to ignore mapping for extra properties in the DTO:

```java
// MyMapper.java
@Mapper(ignoreUnknownProperties = true)
public interface MyMapper {

    MyMapper INSTANCE = Mappers.getMapper(MyMapper.class);

    @Mapping(target = "id", source = "sourceId")
    Target sourceToTarget(Source source);
}
```

In this example, by setting `ignoreUnknownProperties` to `true`, MapStruct will ignore any extra properties in the DTO (source object) that do not have corresponding properties in the entity (target object).

So, if the DTO contains extra properties apart from `sourceId`, those properties will be ignored during mapping, and only the mapped properties will be considered. This allows you to safely map objects even if they have additional fields.