DTO + Validation

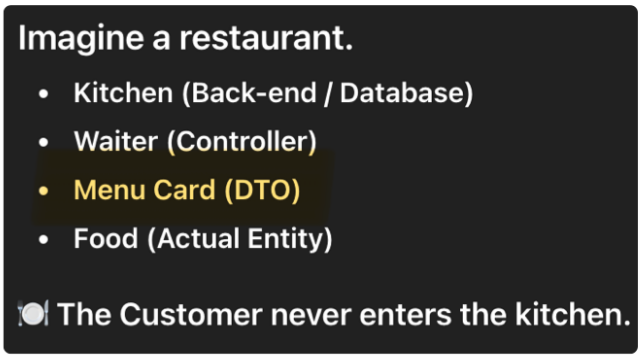
Data Transfer objects (DTO)

→ It is a simple object used to transfer data b/w diff. layers of an Applicatⁿ

→ Data transfer from client → Server → client

→ It is a saperate object used only for input/output of API's.

→ DTO is a custom shaped object created only for communicatⁿ b/w your API & outside world


Imagine a restaurant.

- Kitchen (Back-end / Database)
- Waiter (Controller)
- Menu Card (DTO)
- Food (Actual Entity)

🍽️ The Customer never enters the kitchen.

@Valid

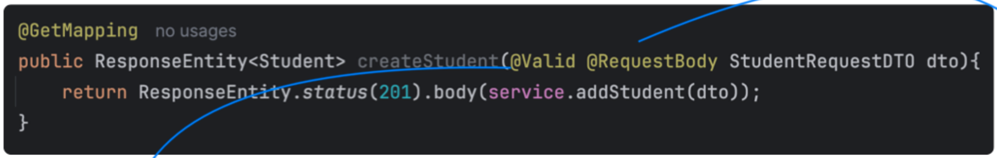
→ Triggers validation checks written inside your DTO using annotatⁿ like –

@ not null　　@ min　　@ size
@ Email　　@ Max　　etc.

But !

Validation only runs when @valid is used on the controller method Parameter.

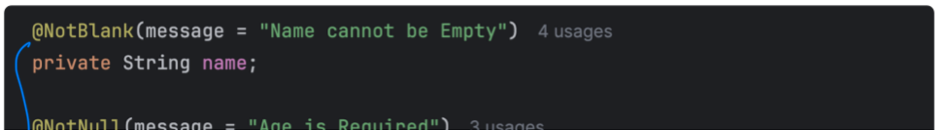
→ Used to execute rules set in DTO.



```
@GetMapping  no usages
public ResponseEntity<Student> createStudent(@Valid @RequestBody StudentRequestDTO dto){
    return ResponseEntity.status(201).body(service.addStudent(dto));
}
```

controller method needed for @valid.

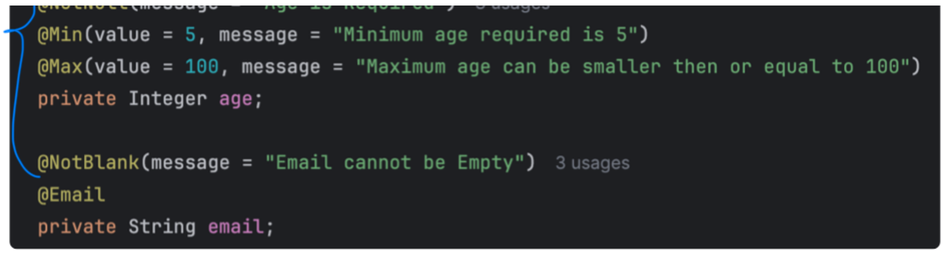
checks for all annotatⁿ [illegible]



```
@NotBlank(message = "Name cannot be Empty")  4 usages
private String name;

@NotNull(message = "Age is Required")  3 usages
```

if they are being followed or not.



```
@NotNull(message = "Age is Required")  3 usages
@Min(value = 5, message = "Minimum age required is 5")
@Max(value = 100, message = "Maximum age can be smaller then or equal to 100")
private Integer age;

@NotBlank(message = "Email cannot be Empty")  3 usages
@Email
private String email;
```

When to use –

① @ Pathvariable – when body contains JSON.
POST, PUT → almost alway request body.

② @ Pathvariable – when value comes from URL path
get /students/{id}

③ @ RequestParam – when value comemes from URL query parameter.
get /students? age = 20.

Response Entity

Gives control of –
- status code
- headers
- body

→ .ok() returns – 200 OK.
→ .status – use when you want 201, 400, 404, 500 etc.
→ .notfound().build() – shortcut for 404 with Empty body
→ .build() – No body : useful for delete.
→ . created – for post API – 201 created.

- when creating something → 201
  ↓
return ResponsEntity.status(HttpStatus.created).body(dto)

- when fetching something → 200

return ResponseEntity.ok(student);

• when deleting something → 204 / 200

return ResponsEntity. nocontent. build()

DTO contains –

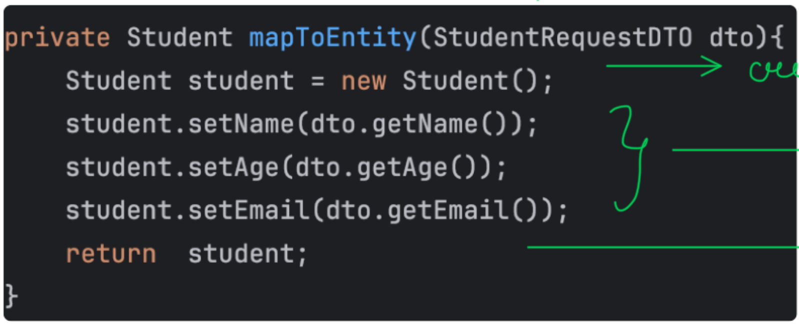
a) Request DTO – what client can request from DB Eg → name, age, email

b) Response DTO – what client will recieve as per his request
Eg → Id, name, age, email.

used in service + controllers.

→ Under DTO we create methods such as MapToEntity & MapToResponseDTO

Return type ↘ method ↓ calls ↓ what student can request – name age email.



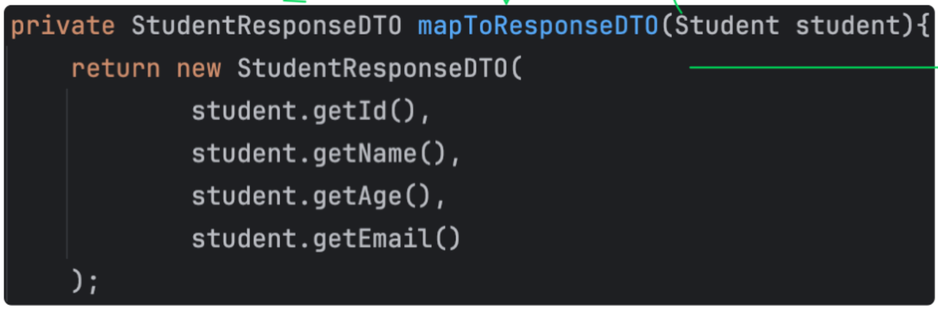
```java
private Student mapToEntity(StudentRequestDTO dto){
    Student student = new Student();
    student.setName(dto.getName());
    student.setAge(dto.getAge());
    student.setEmail(dto.getEmail());
    return  student;
}
```

→ creates a new student

→ assigns all those values to new student

→ returns the new student created

Return type ↘ method ↓ → calls student



```java
private StudentResponseDTO mapToResponseDTO(Student student){
    return new StudentResponseDTO(
            student.getId(),
            student.getName(),
            student.getAge(),
            student.getEmail()
    );
```

→ Returns the new student to response DTO.