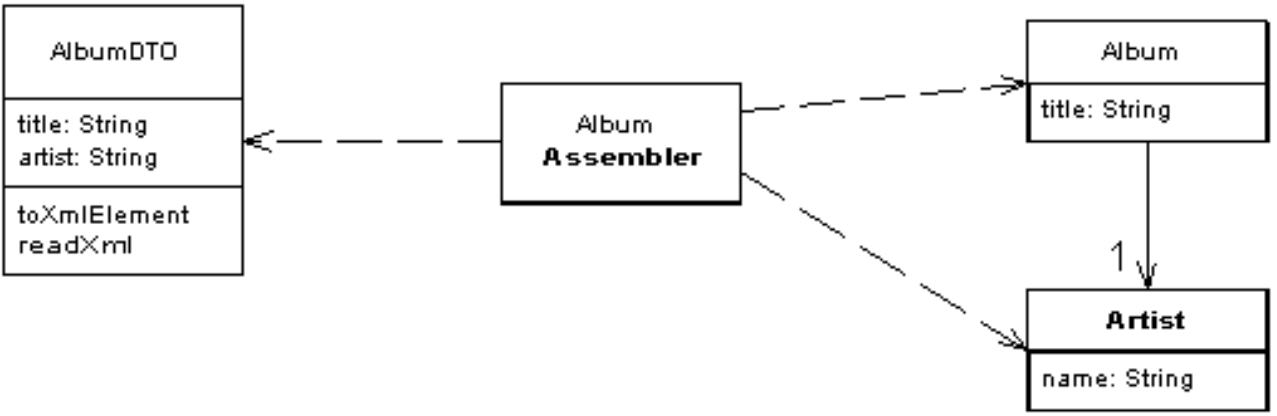
TÓPICO 14 - DTO E MAPPING

Disciplina de Backend - Professor Ramon Venson - SATC 2026.1

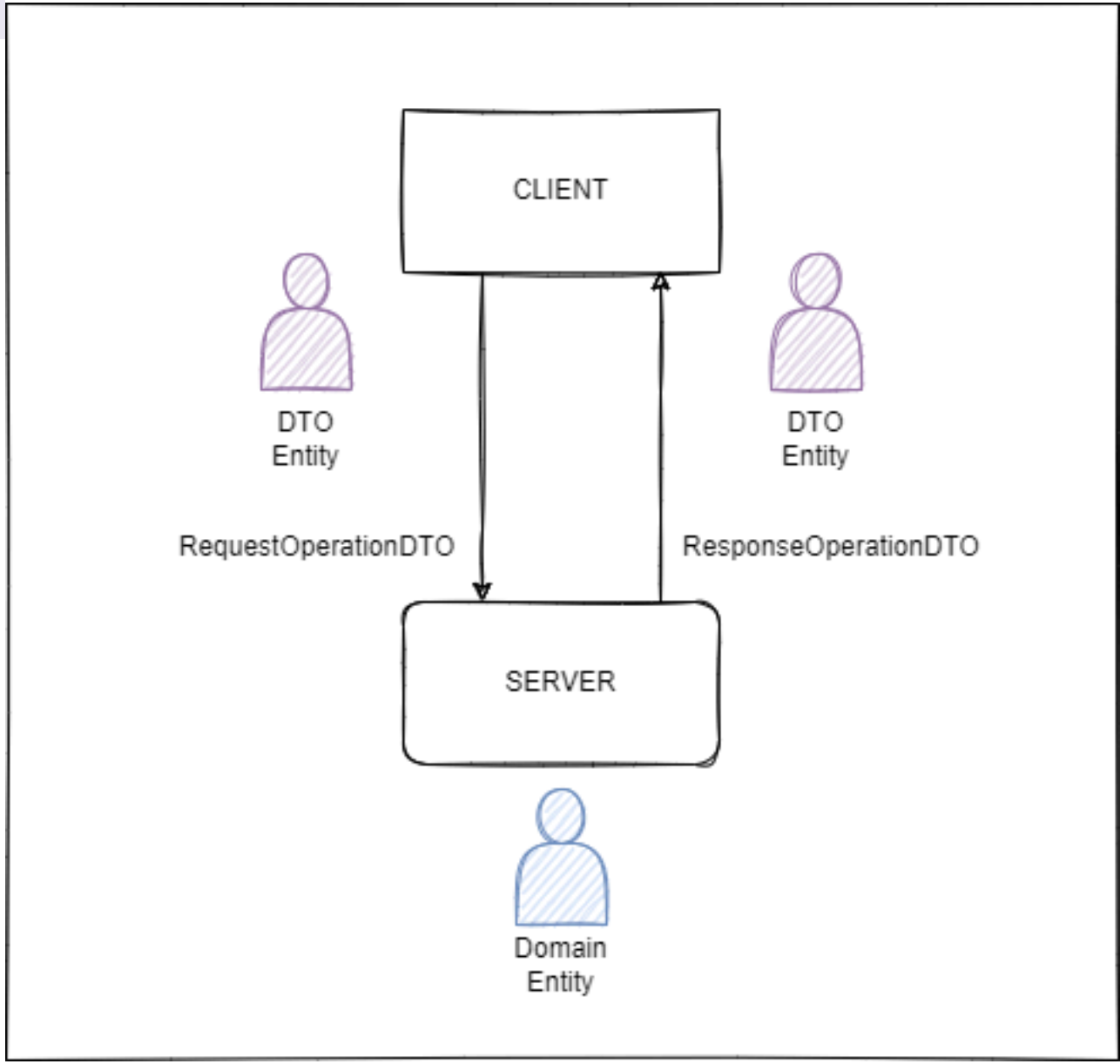


Data Transfer Objects

Data Transfer Objects, ou **DTO**s, são objetos desenhados exclusivamente para a transferência de dados entre camadas ou aplicações.

Vantagens do DTO

- Evitar expor as entidades diretamente
- Controlar o que é recebido ou enviado
- Simplificar as estruturas de dados
- Tornar a comunicação mais eficiente



DTO / Records

Um DTO é um padrão de design e por isso pode ser criado usando diferentes estruturas de programação. Por exemplo, podemos usar uma classe simples do Java. Vamos dar o nome de `PessoaDto.java`:

```java
public class Pessoa {
    private String nome;
    private int idade;

    // Construtor
    // Getters e Setters
}
```

No entanto, na linguagem java, podemos criar a mesma estrutura usando outros recursos, como **Records**.

Java Records

Introduzido no Java 14, os Java Records são uma forma de declarar classes no Java cujo o propósito seja a de prover um objeto imutável (onde não se pode alterar o valor após a criação) e com o mínimo de código possível para acessar os atributos desse objeto, reduzindo o chamado *boilerplate*.

Na prática, o Java Record serve como substituto para uma classe POJO que carrega apenas dados para serialização.

Declarando um Record

Para declarar um novo record, podemos criar um novo arquivo e definir a seguinte sintaxe:

```
public record Pessoa(String nome, int idade) {}
```

Instanciando um Record

```
Pessoa pessoa = new Pessoa("Tarsila", 86);
```

Por ser imutável, os parâmetros associados ao record no momento de sua instância **não poderão ser mais modificados** dentro desse objeto.

Acessando um Record

Para acessar os atributos de um record, usaremos métodos criados automaticamente que correspondem ao mesmo nome do atributo. Por exemplo:

```
String nome = pessoa.nome();
System.out.println(pessoa.idade())
```

Repare que os atributos de um record são privados e não podem ser acessados diretamente, por isso acessamos sempre usando `()`, que representa que estamos usando um método.

Record ou Classe?

Repare que ambas as estruturas de dados podem ser utilizadas para atingir os mesmos objetivos: a criação de um DTO.

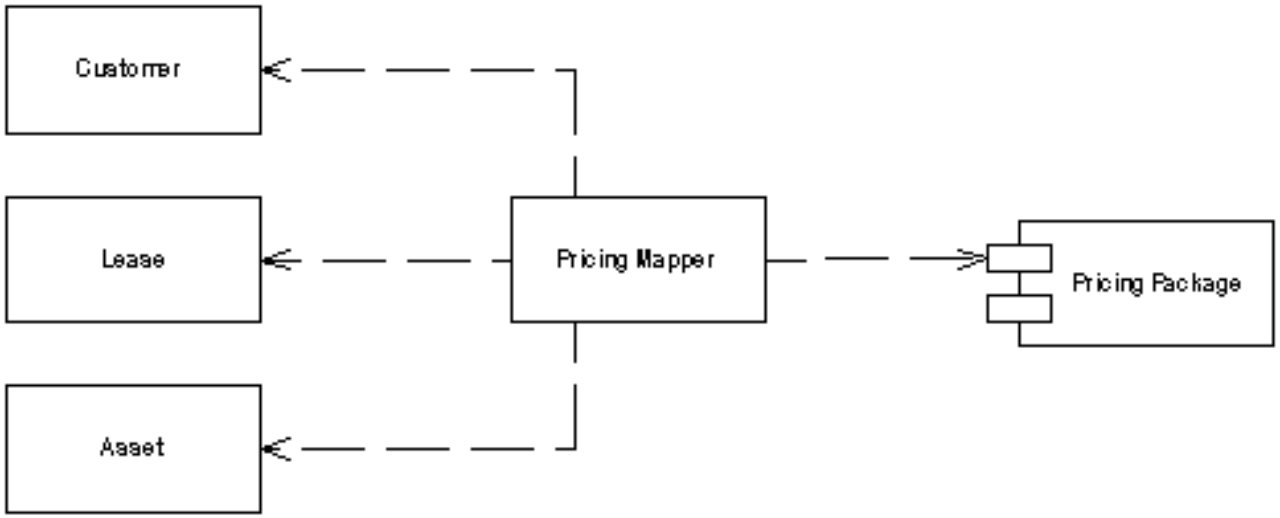
É importante frisar que um DTO é um padrão de design.

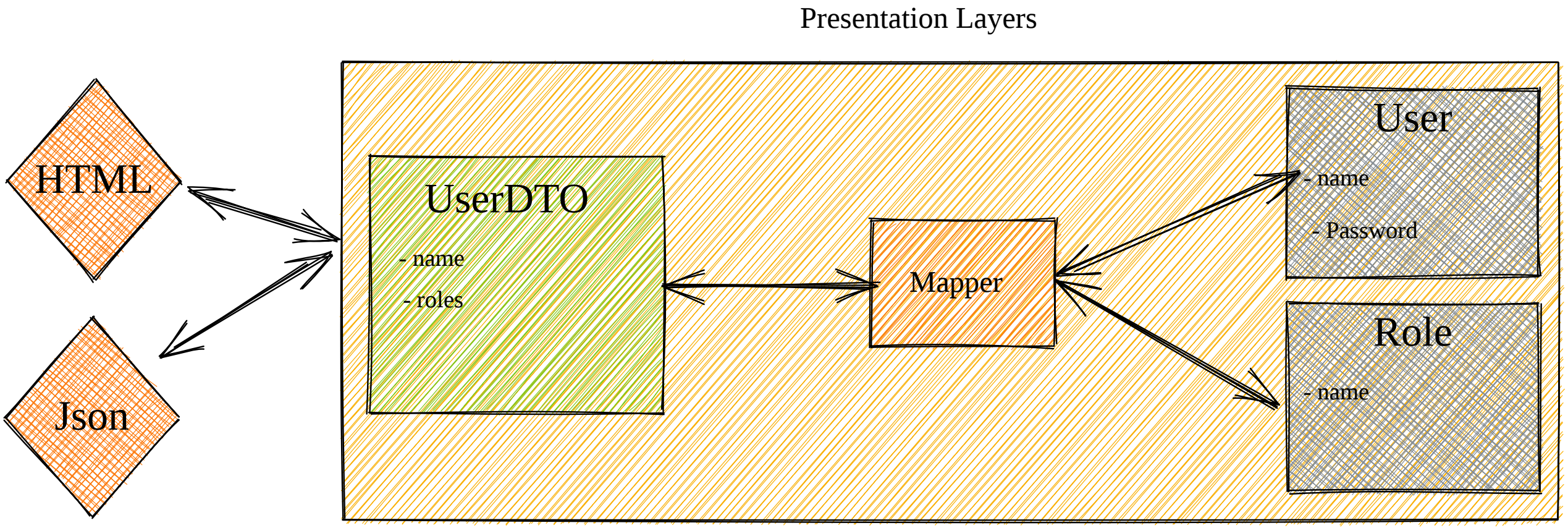
O ponto forte do Record é redução de boilerplate e a imutabilidade forçada, enquanto estruturas de Classe são mais flexíveis.

Mapeamento de Dados

Ao criar um DTO, é comum que seja necessário implementar o mapeamento de um DTO para uma entidade do modelo de dados (e vice versa).

Para isso precisaremos de um padrão de design (**pattern**) chamada de **Mapper** (Mapeador).




Presentation Layers
HTML
Json
UserDTO
- name
- roles
Mapper
User
- name
- Password
Role
- name

Mapeando Entidade

Model

Vamos usar como exemplo a integração de um modelo de entidade chamado **Usuario**:

`Usuario.java`

```
package com.example.models;
public class Usuario {
    private String nome;
    private String email;
    private String senha;
}
```

DTO

Considere que teremos também um DTO, que servirá como estrutura da aplicação para fornecer dados de um usuário (o qual criamos anteriormente), porém **sem o atributo** `senha`;

`UsuarioResponseDto.java`

```
package com.example.models;
public class Usuario(
    String nome,
    String email
) {}
```

Mapper

Criaremos também um mapeador (**mapper**) para fazer a conversão de uma entidade modelo para o DTO:

`UsuarioResponseDto.java`

```
package com.example.models;
public class UsuarioMapper() {
    public UsuarioDto toDto(Usuario usuario) {
        UsuarioDto dto = new UsuarioDto(
            usuario.nome,
            usuario.email
        );
        return dto;
    }
}
```

Service

Nesse exemplo, vamos implementar também um serviço:

```
@Service
public class UsuarioService {
    @Autowired
    UsuarioMapper usuarioMapper;
    public UsuarioDto retornaAdministrador() {
        Usuario usuario = new Usuario(
            "Admin",
            "admin@localhost",
            "123456"
        );
        return usuarioMapper.toDto(usuario);
    }
}
```

Mapeamento Automático

Você também pode optar por usar bibliotecas que geram o mapeamento automático.

No java, uma biblioteca popular é a `MapStruct`:

```
@Mapper(componentModel = "spring")
public interface UserMapper {
    UserDTO toDto(UserEntity entity);
    UserEntity toEntity(UserDTO dto);
}
```

`MapStruct` gera a implementação automaticamente em tempo de compilação.

Considerações gerais

- O **DTO** é um objeto imutável usado para encapsular dados de entrada e saída
- O **modelo de dado** é o objeto que representa a descrição lógica de conceito
- O **mapper** é o responsável pela conversão entre os dois modelos anteriores

O que aprendemos hoje?

- O que é um Data Transfer Model;
- Como criar um record;
- O que é e como criar uma classe de mapeamento (mapper);