EAPLI : Engenharia de Aplicações

DTO
Data Transfer Object

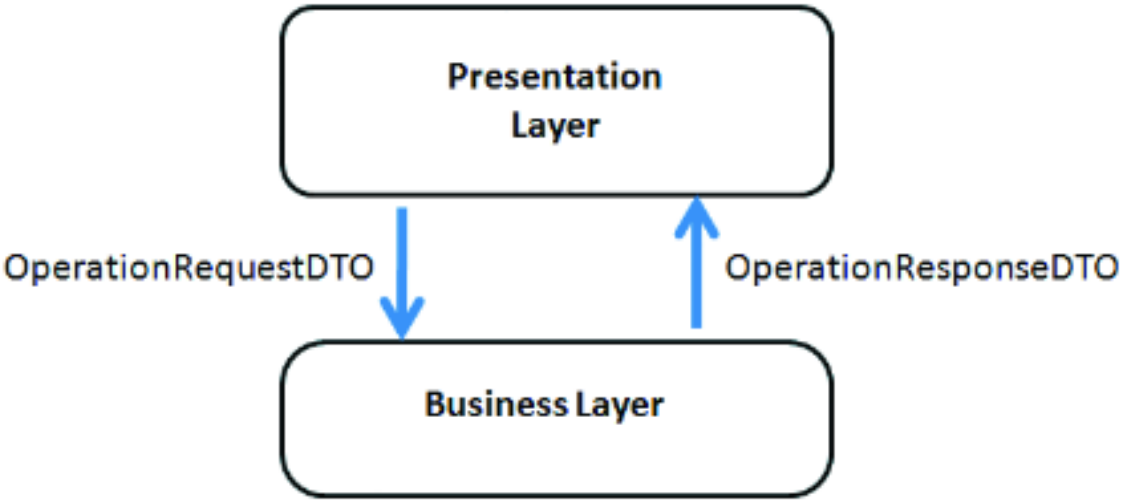
Introdução

Introdução

- Quando temos uma aplicação baseada no domínio, existe a necessidade de analisar seriamente a questão da passagem de informação:
  - Como vamos mover os dados para a camada de apresentação?
  - A camada de apresentação terá referências para objetos do domínio?
  - Como serão enviados os dados entre quaisquer camadas?

Introdução

- Idealmente teremos um cenário como este:



Introdução

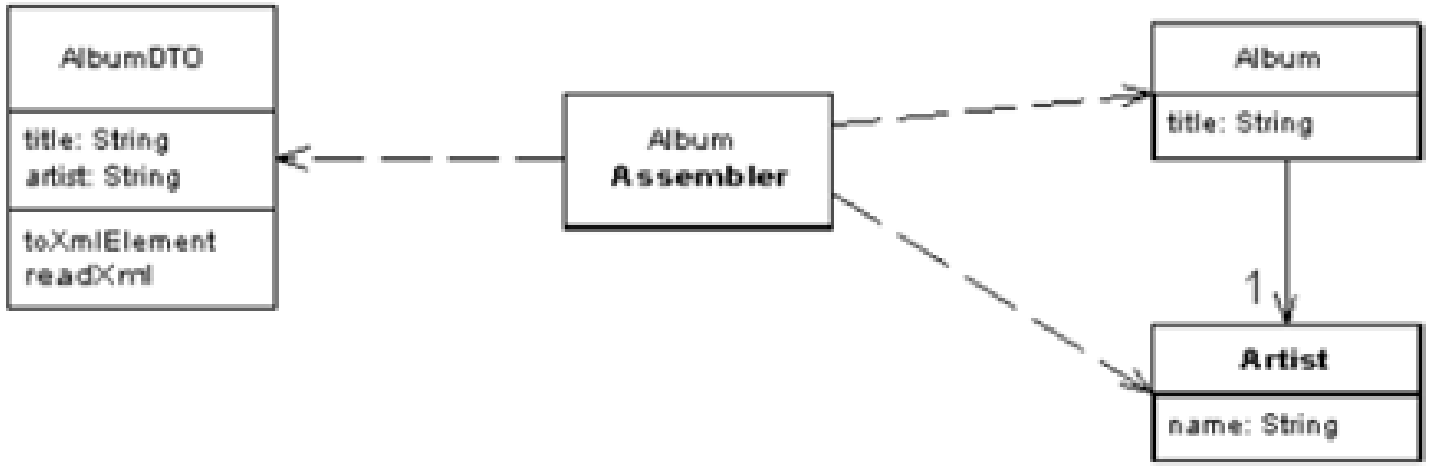
- Um DTO é normalmente uma classe sem métodos e que expõe propriedades
- Um DTO não tem lógica de negócio
- Um DTO é útil sempre que é necessário passar dados agrupados em estruturas.

Data Transfer Object

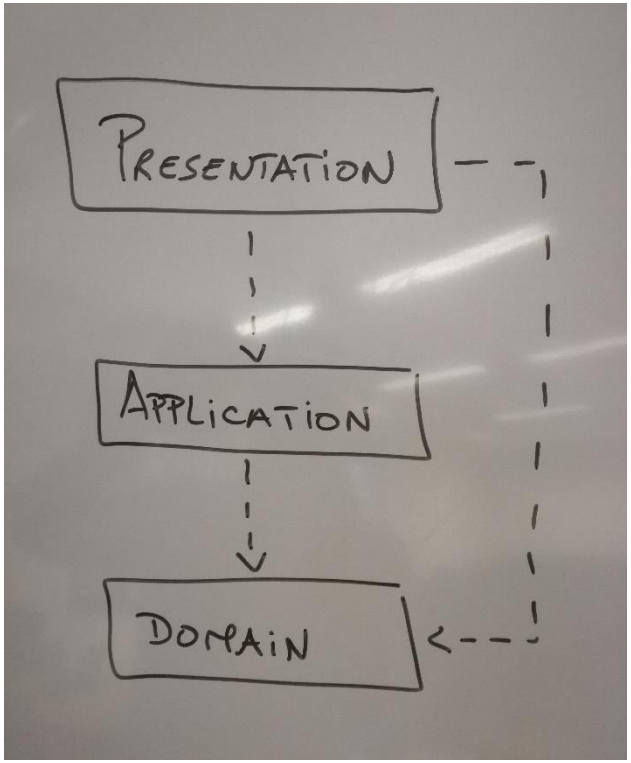
PoEAA

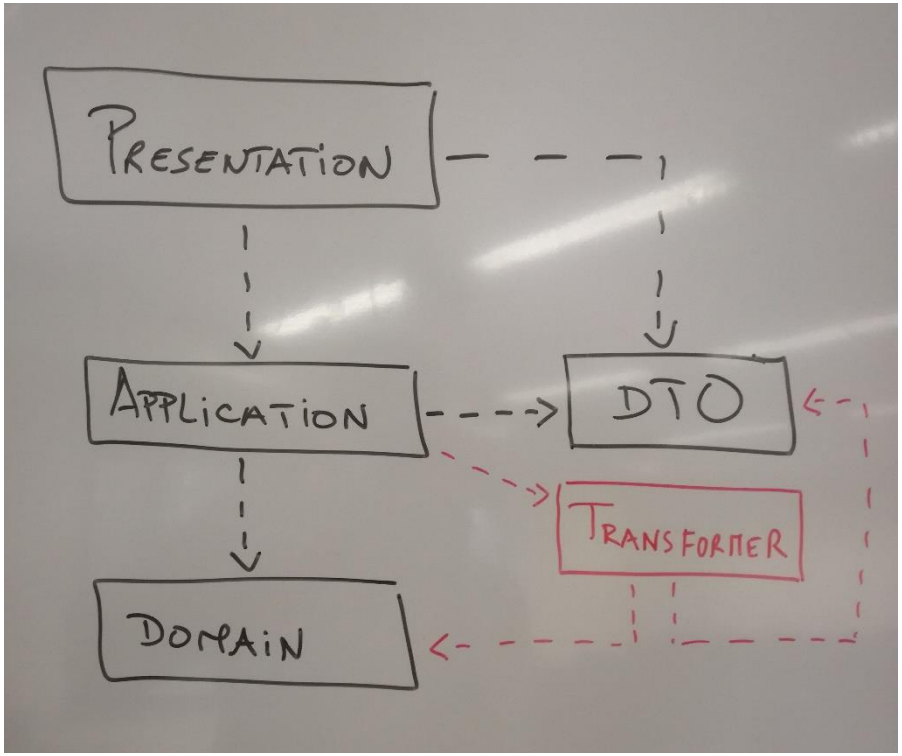
"An object that carries data between processes in order to reduce the number of method calls"

Martin Fowler



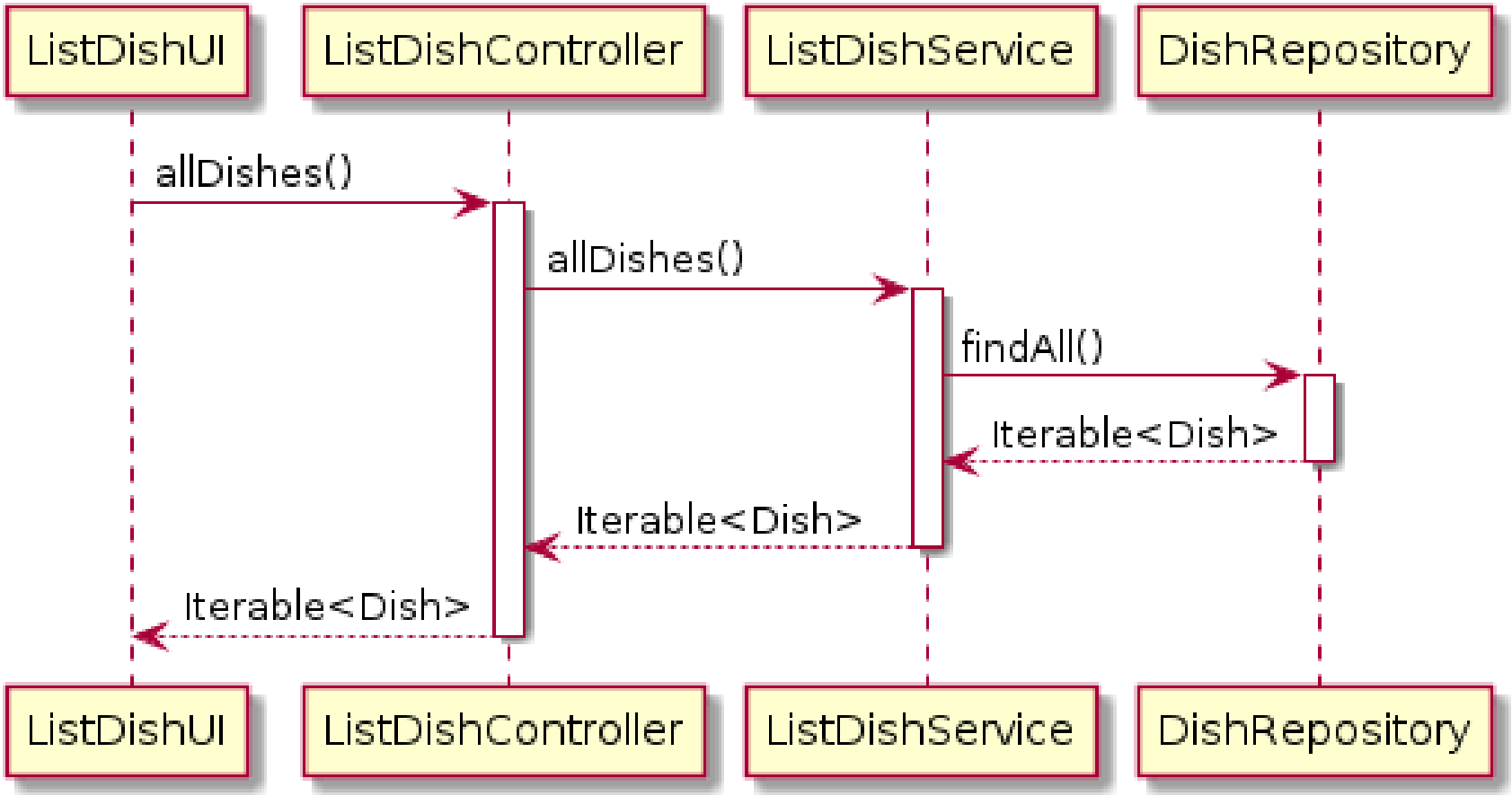
With or Without DTOs





With or Without

ListDish (s/ DTO)

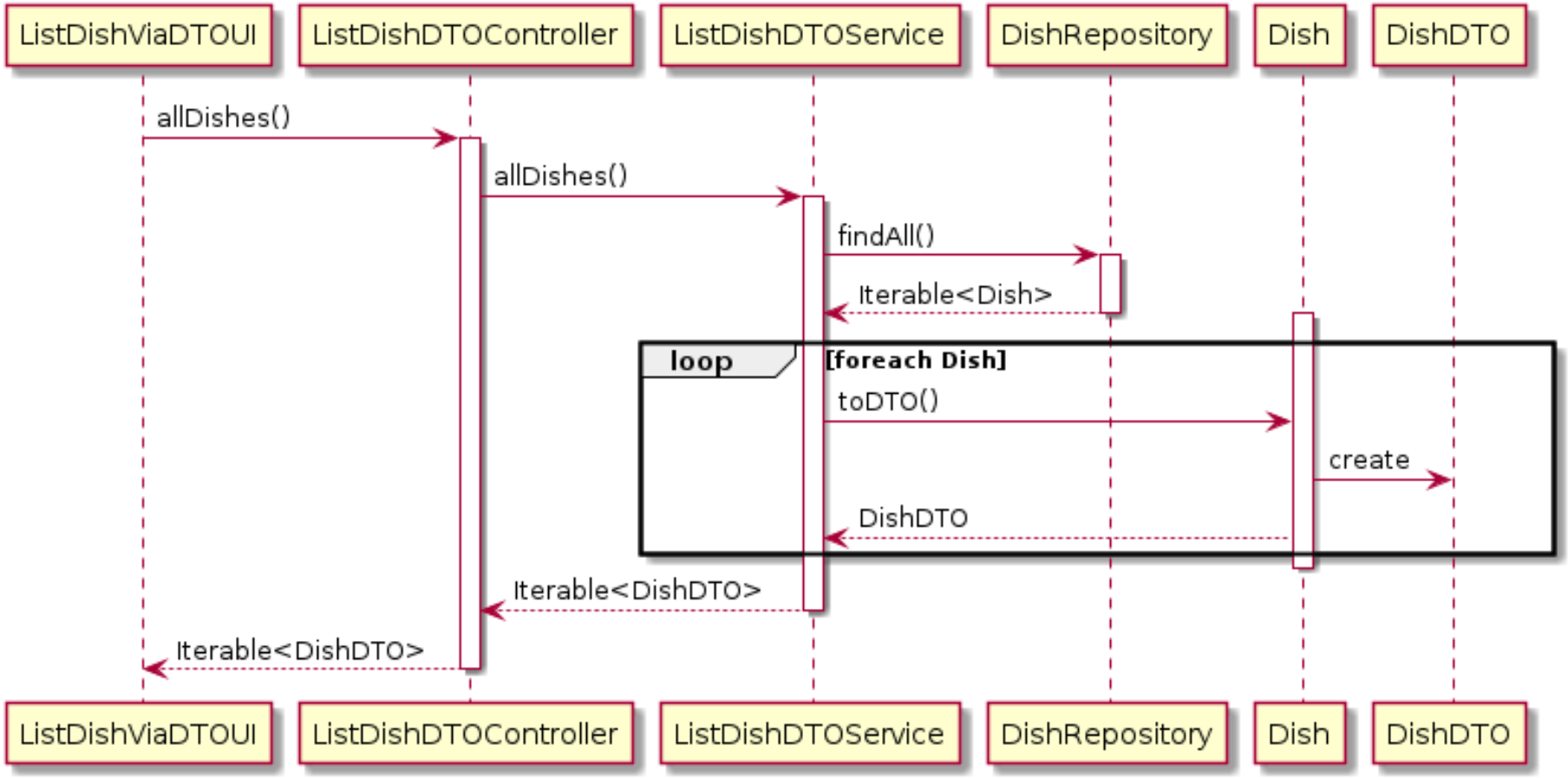


ListDish (c/ DTO)
DishDTO

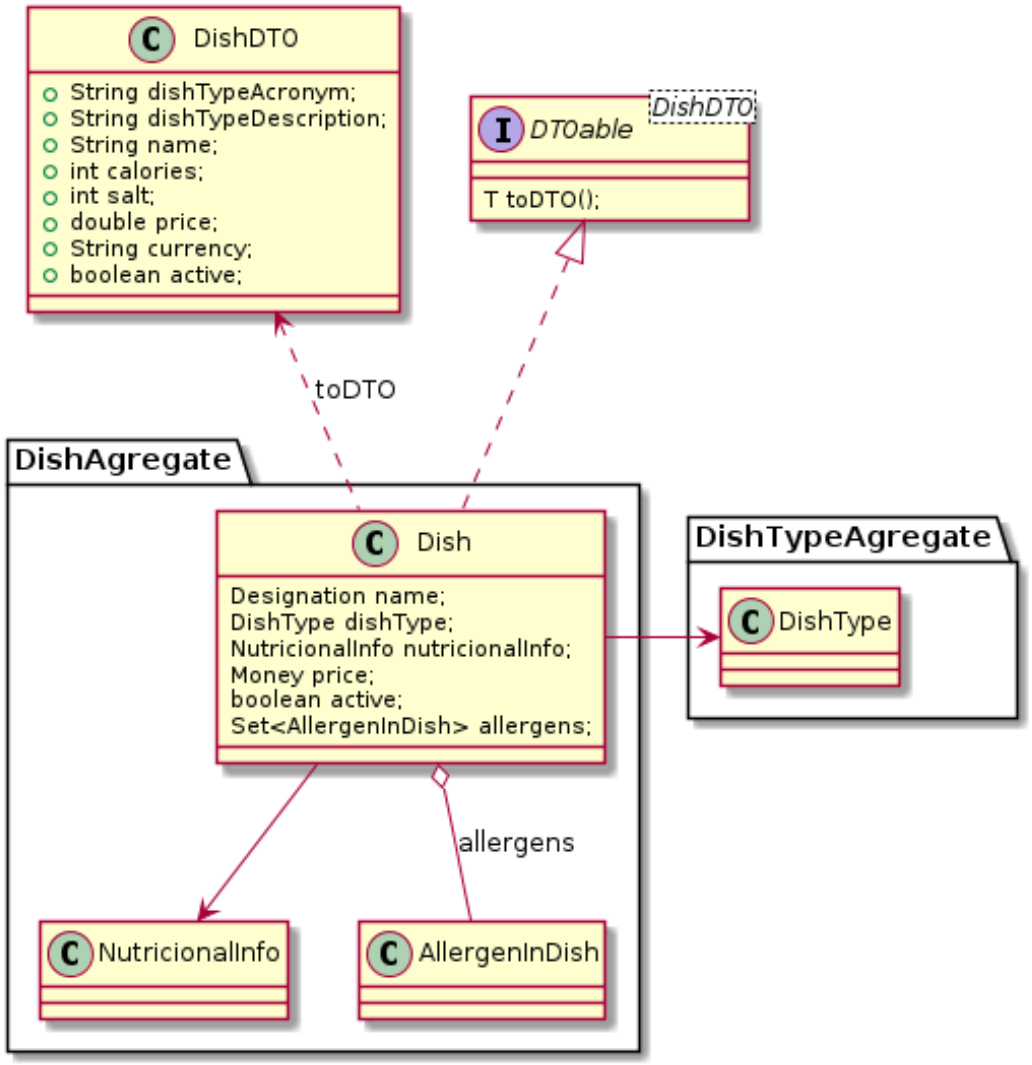
```
35  @DTO
36  public class DishDTO {
37      public DishDTO(final String dishTypeAcronym, final String dishTypeDescription,
38              final String name2,
39              final Integer calories2, final Integer salt2, final double amount,
40              final String currency2, final boolean active2) {
41          this.dishTypeAcronym = dishTypeAcronym;
42          this.dishTypeDescription = dishTypeDescription;
43          name = name2;
44          calories = calories2;
45          salt = salt2;
46          price = amount;
47          this.currency = currency2;
48          this.active = active2;
49      }
50
51      public DishDTO() {
52          // empty
53      }
54
55      public String dishTypeAcronym;
56      public String dishTypeDescription;
57      public String name;
58      public int calories;
59      public int salt;
60      public double price;
61      public String currency;
62      public boolean active;
63  }
```

Podem ser usados mappers dedicados ou declarativos para automatizar a criação de DTOs (ex. Dozer)

ListDish (c/ DTO)



Um exemplo: Dish



Com ou Sem DTOs

```
12  public class ListDishService {
13
14      private final DishRepository dishRepository = PersistenceContext.repositories().dishes();
15
16      public Iterable<Dish> allDishes() {
17          AuthorizationService.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_MENUS);
18
19          return this.dishRepository.findAll();
20      }
21  }
```

```
26      public Iterable<DishDTO> allDishes() {
27          AuthorizationService.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_MENUS);
28
29          final Iterable<Dish> dishes = this.dishRepository.findAll();
30
31          // transform for the presentation layer
32          final List<DishDTO> ret = new ArrayList<>();
33          dishes.forEach(e -> ret.add(e.toDTO()));
34          return ret;
35      }
36  }
```

Com ou sem DTOs

registerDish

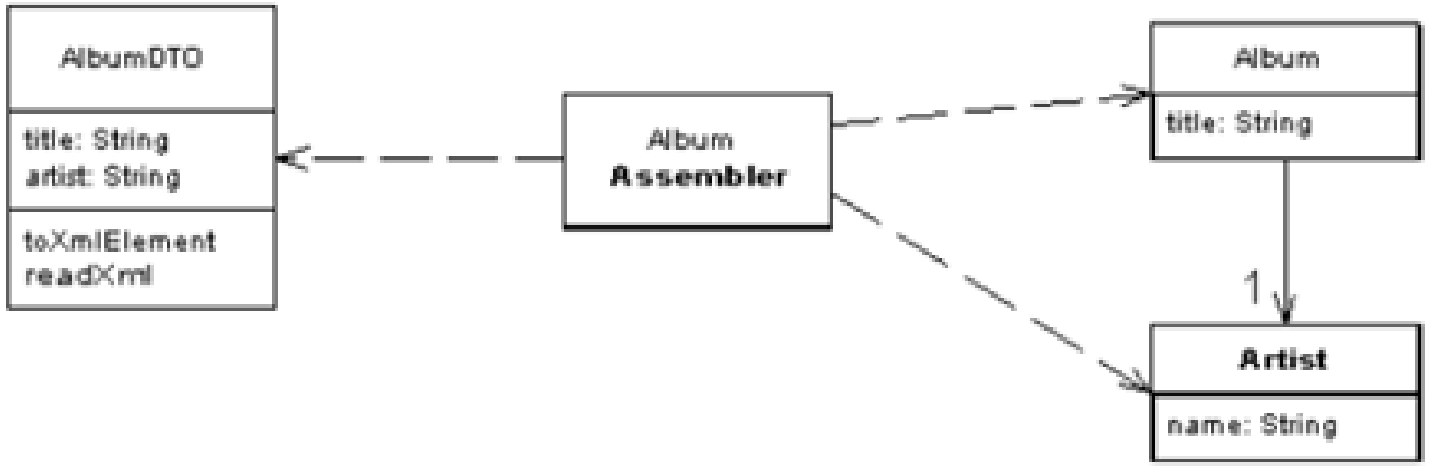
```java
public Dish registerDish(final DishType dishType, final String name,
        final Integer calories, final Integer salt,
        final double price) throws DataIntegrityViolationException, DataConcurrencyException {
    AuthorizationService.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_MENUS);

    final Dish newDish = new Dish(dishType, Designation.valueOf(name),
            new NutricionalInfo(calories, salt),
            Money.euros(price));

    return this.dishRepository.save(newDish);
}
```

```java
public void registerDish(DishDTO dto)
        throws DataIntegrityViolationException, DataConcurrencyException {
    AuthorizationService.ensurePermissionOfLoggedInUser(ActionRight.MANAGE_MENUS);

    // retrieve the dish type
    final Optional<DishType> type = dishTypeRepository.findByAcronym(dto.dishTypeAcronym);
    if (!type.isPresent()) {
        throw new IllegalArgumentException("Unknown dish type: " + dto.dishTypeAcronym);
    }

    // TODO: we are ignoring the currency and hardcoding everything is EUR
    final Dish newDish = new Dish(type.get(), Designation.valueOf(dto.name),
            new NutricionalInfo(dto.calories, dto.salt), Money.euros(dto.price));

    this.dishRepository.save(newDish);
}
```

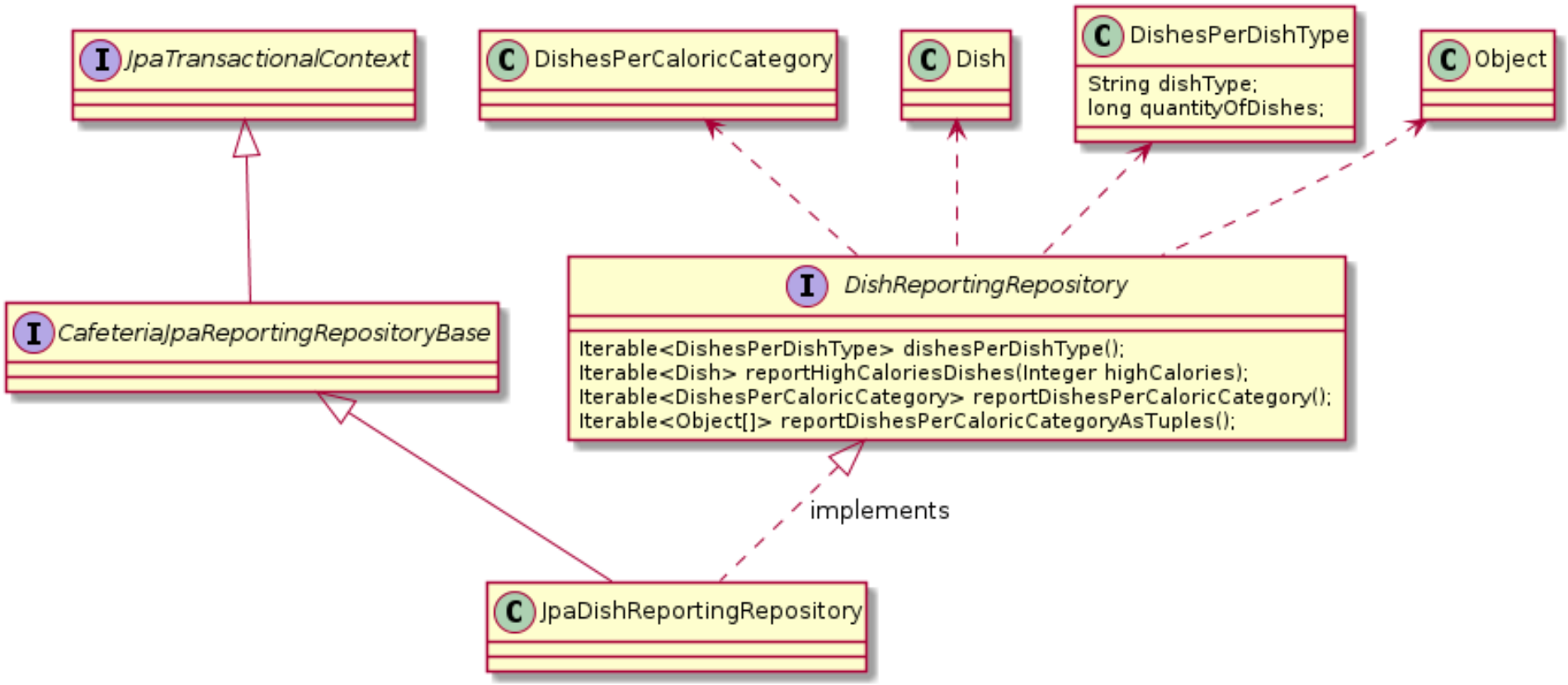
Reporting

Data Transfer Object

"An object that carries data between processes in order to reduce the number of method calls"
Martin Fowler



Reporting com DTOs



Reporting com DTOs

```
35  public class JpaDishReportingRepository extends CafeteriaJpaReportingRepositoryBase
36          implements DishReportingRepository {
37
38      JpaDishReportingRepository() { super(); }
41
42      @Override
43      public Iterable<DishesPerDishType> dishesPerDishType() {
44
45          final TypedQuery<DishesPerDishType> query = entityManager().createQuery(
46                  s: "SELECT new eapli.ecafeteria.reporting.dishes.dto.DishesPerDishType(t.acronym, COUNT(d)) " +
47                          "FROM Dish d, DishType t WHERE d.dishType = t GROUP BY d.dishType",
48                  DishesPerDishType.class);
49
50          return query.getResultList();
51      }
52
53      @Override
54      public Iterable<Dish> reportHighCaloriesDishes(final Integer highCalories) {...}
61
62      private static final String DISHES_PER_CALORIC_CATEGORY_QUERY = "SELECT caloricCategory... from *";
65
66      @{...}
68      public Iterable<DishesPerCaloricCategory> reportDishesPerCaloricCategory() {...}
73
74      @{...}
76      public Iterable<Object[]> reportDishesPerCaloricCategoryAsTuples() {...}
80  }
```

Prós e contras dos DTO's

DTO – Uma solução perfeita?

- Na perspetiva do design puro, os DTOs são uma solução próxima da perfeição. Isolam o modelo de domínio da apresentação, tendo como resultado:
  - Um baixo acoplamento
  - Transferência de dados otimizada
  - Uma maior flexibilidade no design de toda a aplicação quando existem alterações aos requisitos.

DTO – Outras vantagens

- Permite a utilização de estruturas diferentes das entidades de domínio
- Impede a exposição de informação desnecessária
- Pode evitar problemas de lazy-load na apresentação

DTO

- Os benefícios referidos teoricamente são confirmados na prática?

  Resposta: Depende.

DTO – Desvantagens

- Ter centenas de entidades no modelo de domínio é normalmente um bom indicador para se considerar alternativas.
- Se tivermos uma solução DTO 100% pura irá conduzir uma maior complexidade de classes extra.
- Uma solução DTO traz um trabalho adicional

DTO – Desvantagens

- Note-se ainda que a quantidade de DTO's necessária poderá não ser fácil de medir. Obviamente o nr. de entidades presentes no modelo de domínio entrará na fórmula, mas basta existirem chamadas diferentes de informação para termos necessidades de outros DTO's.
  - Ex. DespesasPorTipo e DespesasPorGrupo

DTO – Alternativas

- Se os DTO´s nem sempre são ideais, qual a alternativa?
  - Fazer referência ao modelo de domínio na camada de apresentação. *Ou a uma interface sobre o domínio**
  - O que vai provocar um maior acoplamento entre camadas, que poderá provocar um problema ainda maior.

Apresentação - Referenciando Entidades

- Sendo "aceitável", a camada de apresentação receber dados em formato de entidades do modelo de domínio, é uma opção à utilização dos DTO's.
- Esta opção apenas é válida se as camadas partilharem o mesmo processo.
- Por vezes a apresentação necessita de dados formatados de uma forma particular. Se não forem usados DTO's terá de se passar esta responsabilidade para a camada de apresentação.
  - Na verdade o lugar errado para formatar dados é na camada de domínio.

Interface sobre o domínio

- Esconder métodos de negócio para que a camada de apresentação não os possa invocar. Ex.

```
public interface EmployeeDTO {
    List<String> getNames();
 }
public class Employee implements EmployeDTO {
    public Money adjustSalary(Percentage adjustment) { … }
    public void promote(JobRole target) { … }
    public List<String> getNames() { … }
 }
```

- Garantir que getters nunca devolvem acesso direto aos dados da classe de dominio. Ex.

```
public List<String> getNames() {
    return Collections.unmodifiableList(this.names);
 }
```

Conclusões

Conclusões

- DTO's ou não, é uma decisão importante do projeto que afeta a comunicação da camada de apresentação com o resto do sistema
- Se usarmos DTO's, vamos ter um sistema com baixo acoplamento e aberto para diferentes tipos de clientes
- DTO's são a escolha ideal se a pudermos "pagar"

Conclusões

- DTO's adiciona uma sobrecarga significativa para qualquer sistema do mundo real
- Se somos simultaneamente fornecedores e consumidores de serviços, pode haver benefícios em a apresentação referenciar o domínio. O impacto no design e codificação é claramente muito mais suave.
- Usar ou não DTO's, não é um ponto fácil de generalizar.

Conclusões

- Para ser eficaz, a decisão final deve ser feita olhando para as **particularidades do projeto**. Poderemos acabar por ter uma abordagem mista.

"And now that you don't have to be perfect, you can be good."
—John Steinbeck, *East of Eden*

Principais referências

- Microsoft Magazine: Pros and Cons of Data Transfer Objects - Dino Esposito
- http://martinfowler.com/
- http://dozer.sourceforge.net/