EAPLI

Princípios de Design OO: Factories & Repositories

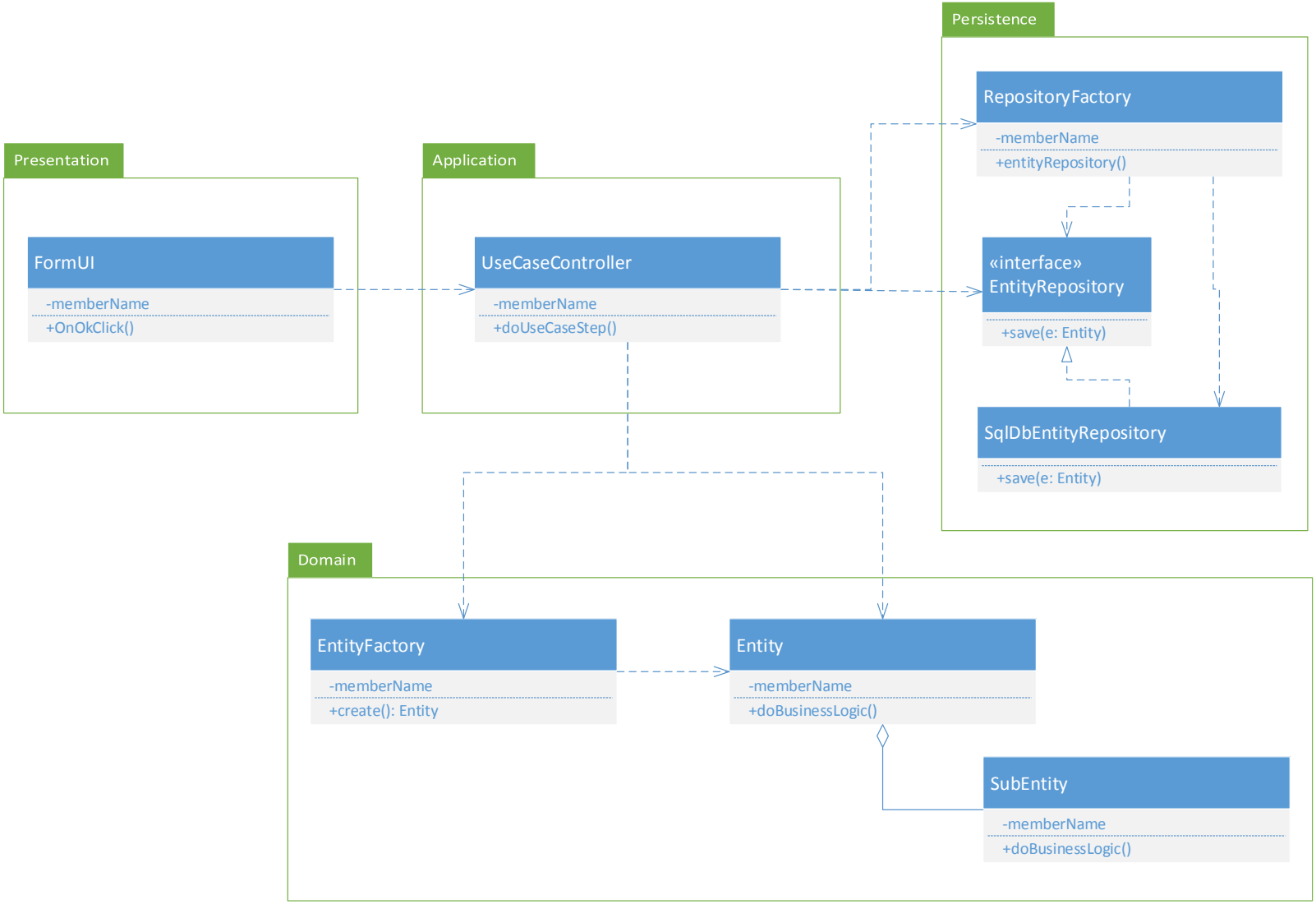
Paulo Gandra de Sousa
pag@isep.ipp.pt

Questões comuns

- A que classe atribuir uma dada responsabilidade?
- Como organizar as responsabilidades do sistema?
- Quem deve ter responsabilidade de coordenar a interação de caso de uso?
- Quem deve ter a responsabilidade de representar e implementar a lógica de negócio?
- Como gerir o ciclo de vida de um objeto?
  - Criar um objeto?
  - persistir objetos?
- Como proteger o código para modificação?

Topic	Principles and patterns
Which class should a responsibility be assigned to?	Information Expert Tell, don't ask Single Responsibility Principle Interface Segregation Principle Intention Revealing Interfaces
How to organize the system's responsibilities?	
How to model the domain?	Persistence Ignorance Entity Value Object Domain Service Aggregate Domain Event Observer
How to handle an object's lifecycle?	
How to prepare the code for modification?	

Sterotypical architecture



Como gerir o ciclo de vida de um objeto?

Quem deve ter a responsabilidade de criar objetos?
Quem deve ter a responsabilidade de persistir objetos?

Object's lifecycle

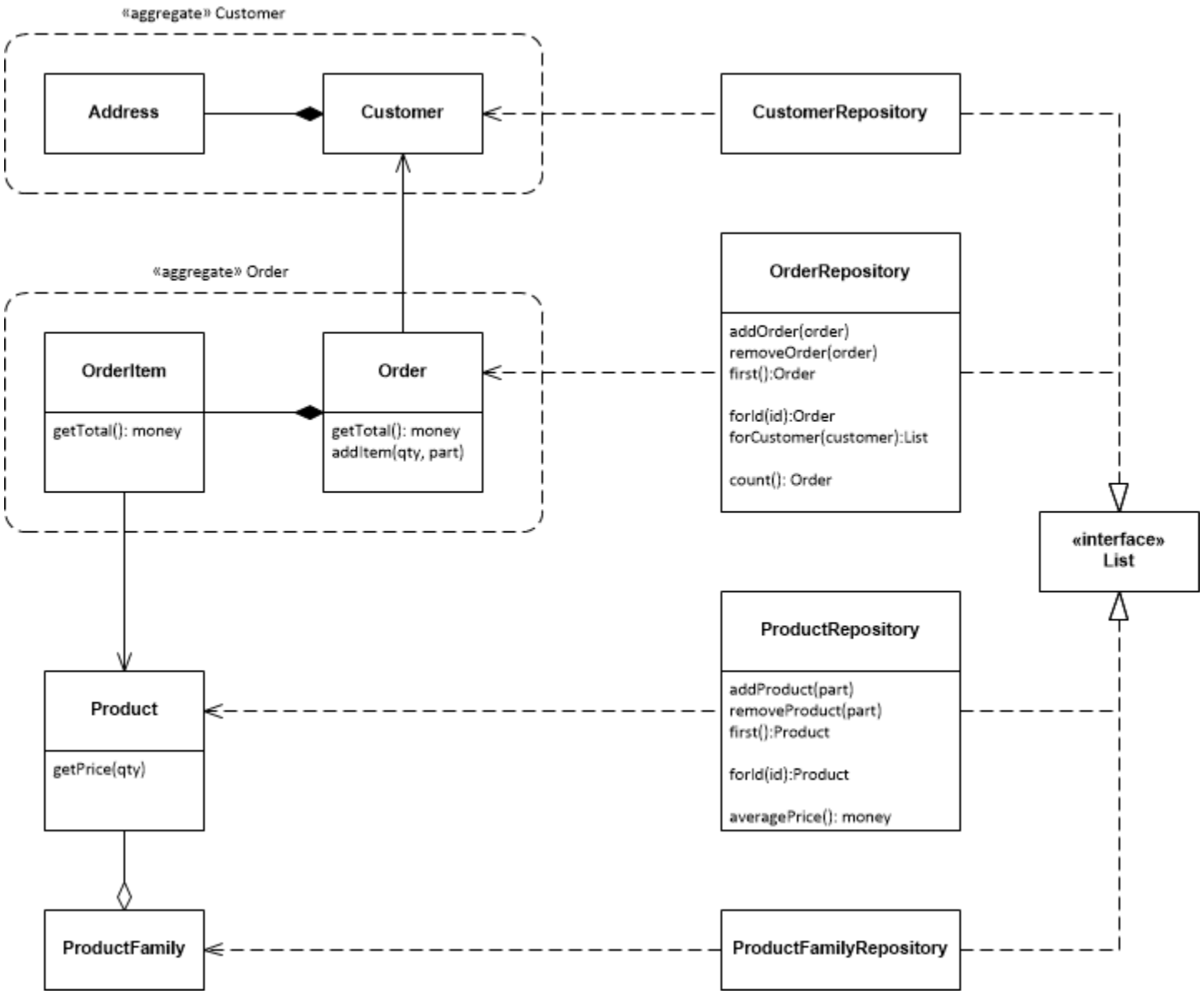
1. An object is created
2. The object is used
3. It must be persisted for later use
4. Later, the object is reconstructed from persistence
5. It is used (provably changed)
6. It is stored back again for later use
7. …

Repository

DDD

- Problem:
  - Como esconder os pormenores de persistir e reconstruir objetos do resto do código?
- Solution:
  - Abstrair a persistência numa classe Repositório que se *comporta* como uma lista
  - Criar um repositório por "agregado"

Repository example

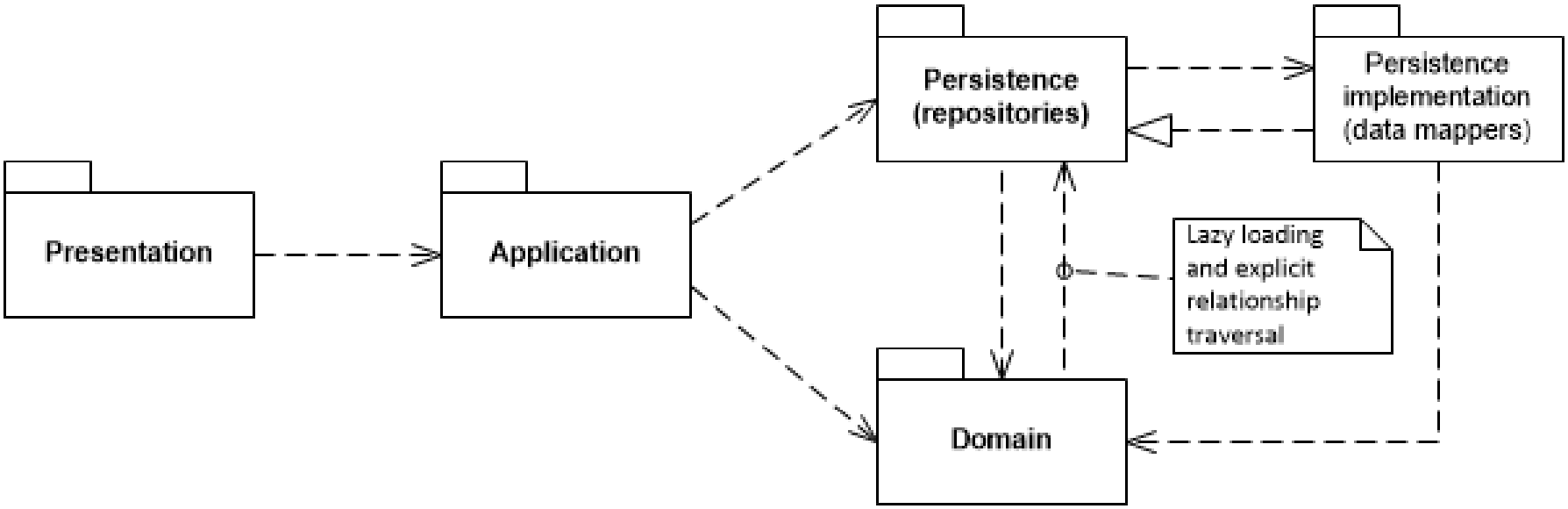


Repository's methods

- findById
- add, update, save
- delete
- all, page, iterator
- Specialized finders, e.g.,
  - ProductRepository.findByFamily
  - ExpenseRepository.findByMonth
- Aggregation functions, e.g.,
  - ProductRepository.getAverageRetailPrice
  - ExpenseRepository.getTotalExpenditure

Careful as we are actually performing business logic at the data layer

Repositories and layers



Questões comuns

- Usar um objeto vs. Criar um objecto?
  - Separate use from construction
  - Creator
  - Factory method
  - Simple Factory
  - Abstract Factory

Separate use from construction

- Criar um objeto pode ser complexo
- "criar" é uma responsabilidade diferente da de "usar"
- Tipicamente um objeto é criado num sitio e usado em muitos outros

Factory

DDD

When creation of an entire, internally consistent aggregate, or a large value object, becomes complicated or reveals too much of the internal structure, factories provide encapsulation.

Domain in vs. out

- Domain factories
  - Create domain entities, aggregates and value objects
- Other types of factories
  - E.g., Persistence

Constructors vs. Factories

- Sometimes a constructor is enough
- Does not allow manipulation of different interfaces

Creator

GRASP

- Problema:
  - Quem deve ser responsável pela criação de objetos de uma classe?
- Solução:
  - Atribuir à classe B a responsabilidade de criar instâncias da classe A nas seguintes condições:
    - B contém ou agrega objetos da classe A
    - B regista instâncias da classe A
    - B possui os dados usados para inicializar A
    - B está diretamente relacionado com A
    - Se mais de uma condição se aplicar, escolhe-se a "classe B que contém ou agrega objetos da classe A"

Factory Method

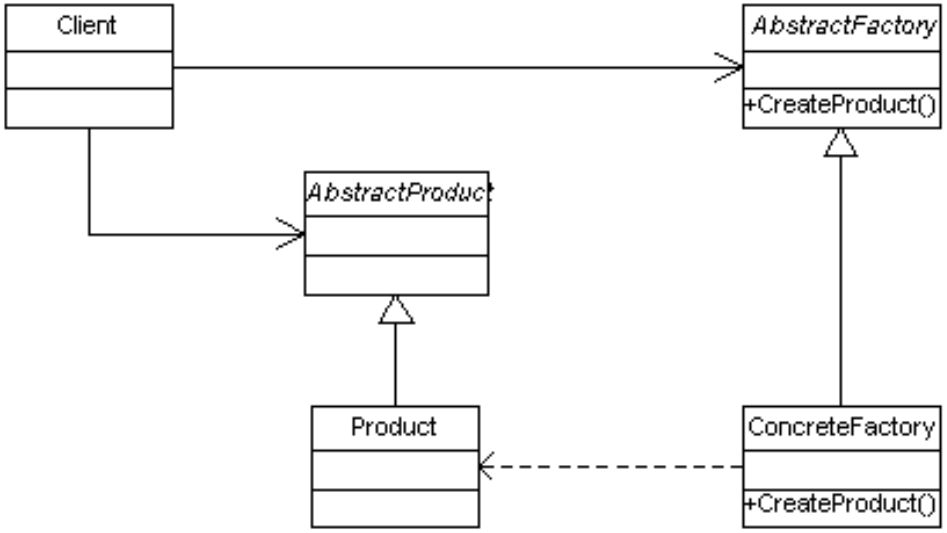
GoF

- Problema:
  - Como se simplifica a manipulação de diferentes implementações da mesma interface
- Solução:
  - Esconder a criação num método.
  - O método devolve um tipo que é mais geral que o seu tipo de facto.

Factory Method

GoF

- Um método que esconde os detalhes de criação dum objecto



- The client needs a product, but instead of creating it directly using the new operator, it asks the factory object for a new product, providing the information about the type of object it needs.
- The factory instantiates a new concrete product and then returns to the client the newly created product(casted to abstract product class).
- The client uses the products as abstract products without being aware about their concrete implementation.

fonte: Design Patterns: Elements of Reusable Object-Oriented Software

Factory pode criar objectos diferentes a partir dum Ficheiro

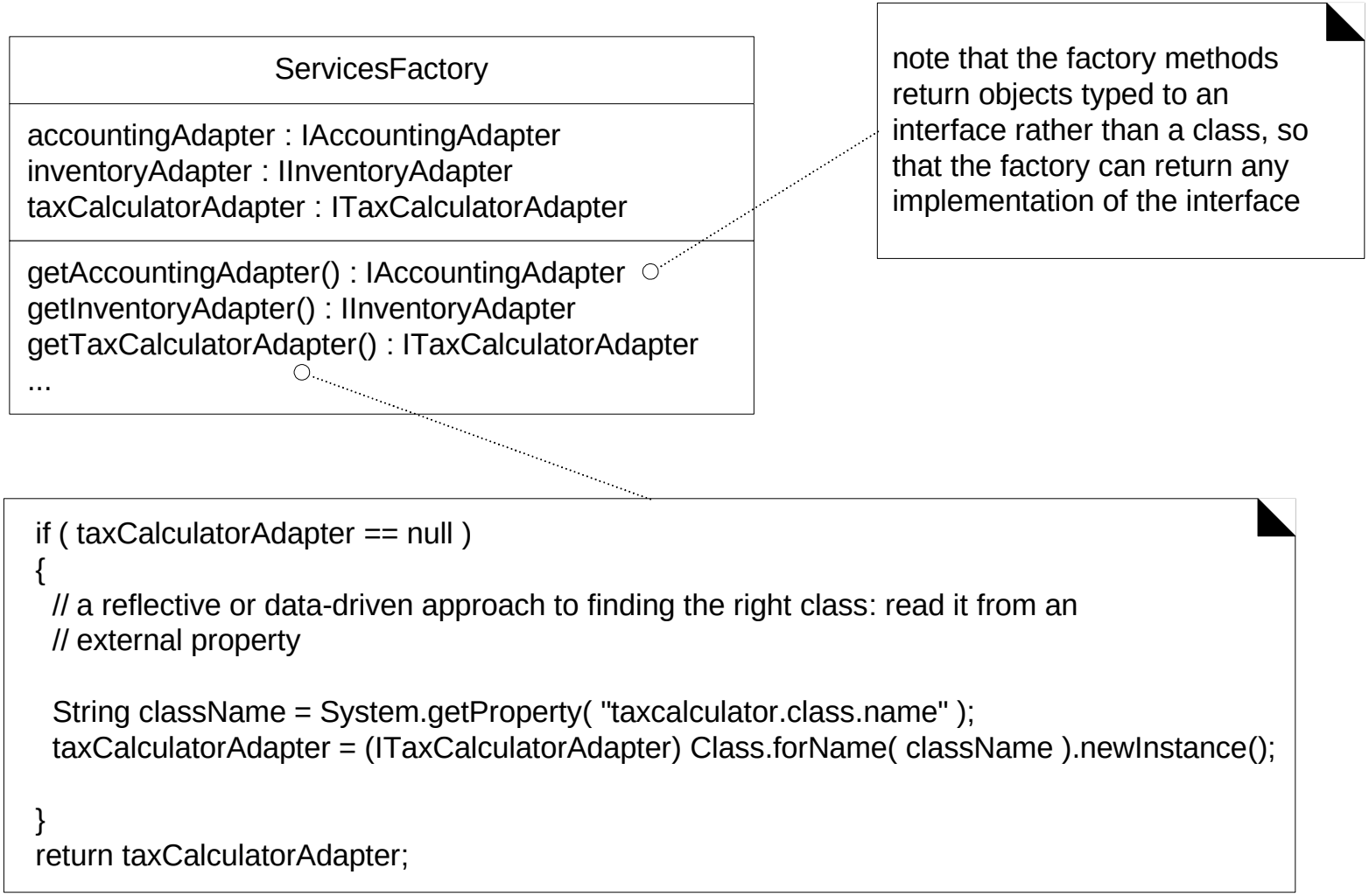
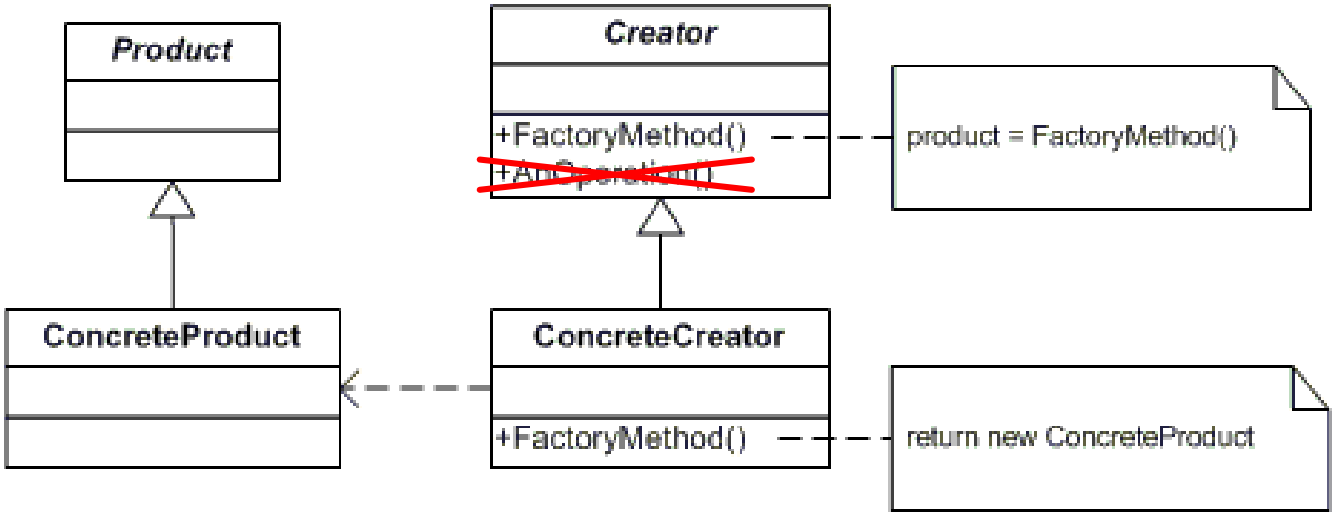


Figure 26.5

Simple Factory

GoF

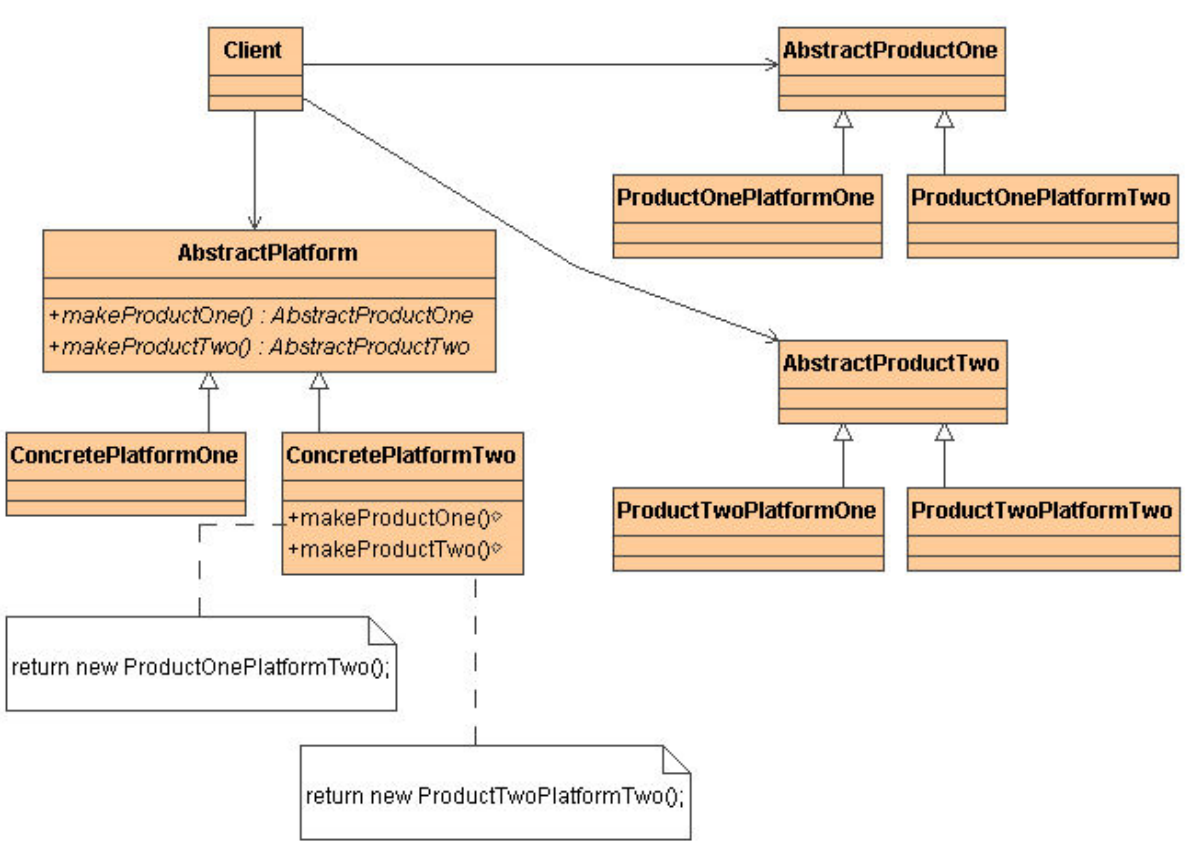
- A class with factory methods only



Abstract Factory

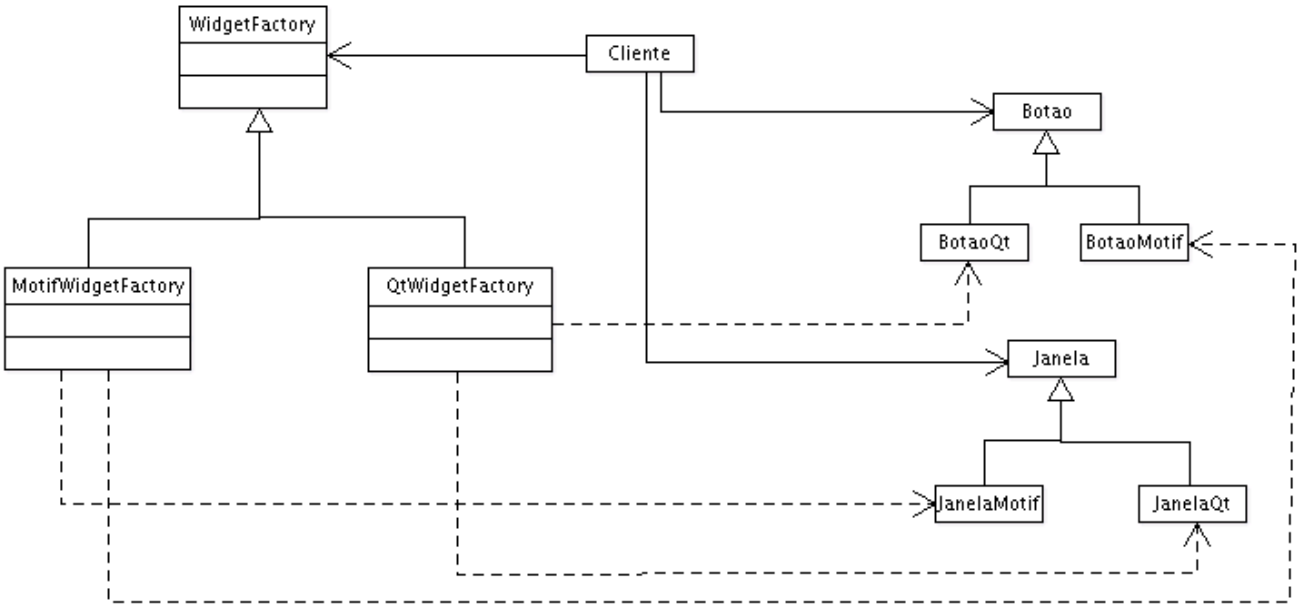
GoF

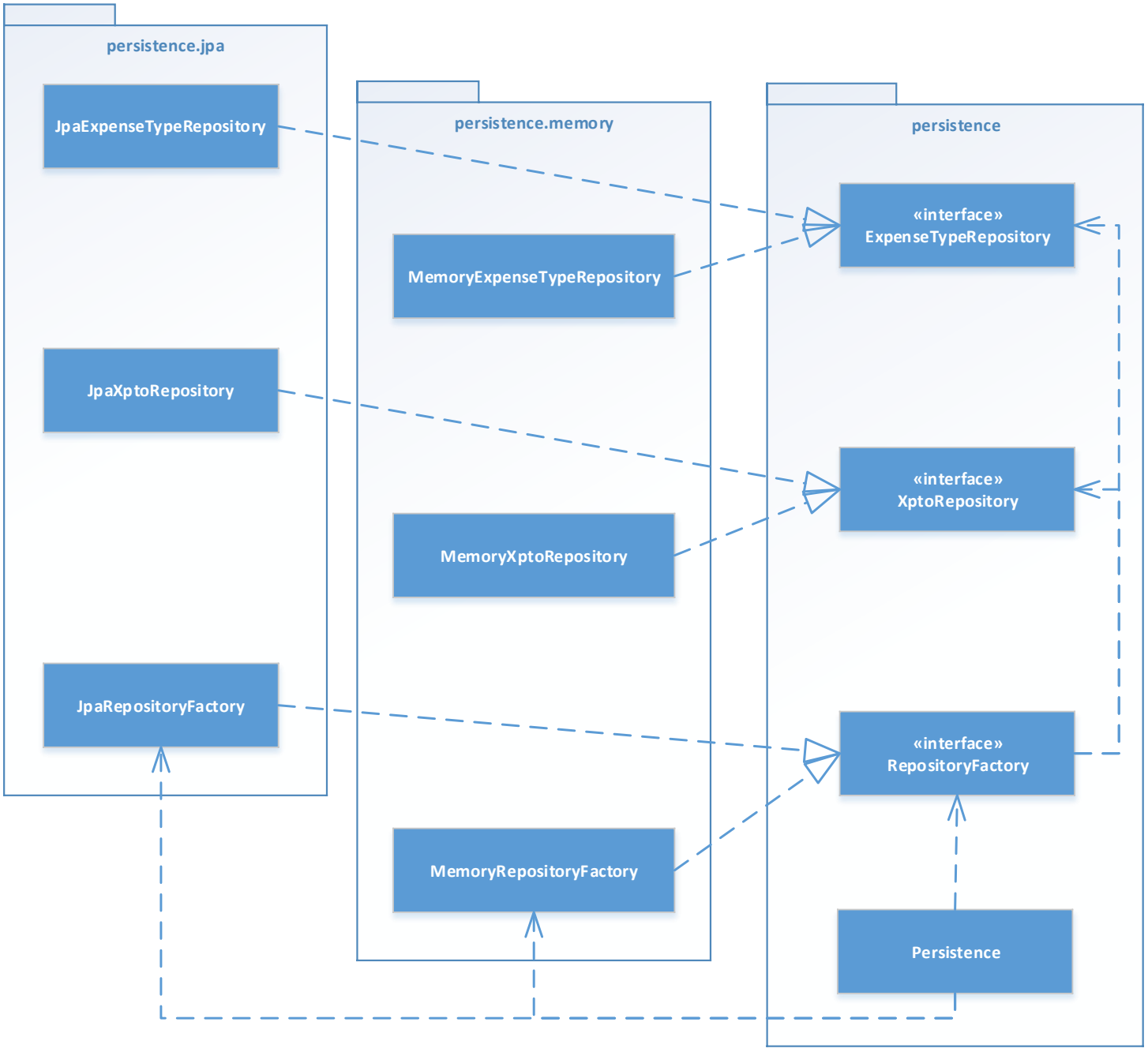
- Disponibiliza uma interface para a criação de famílias de objectos relacionados ou dependentes, sem especificar as suas classes concretas



Abstract Factory (exemplo)

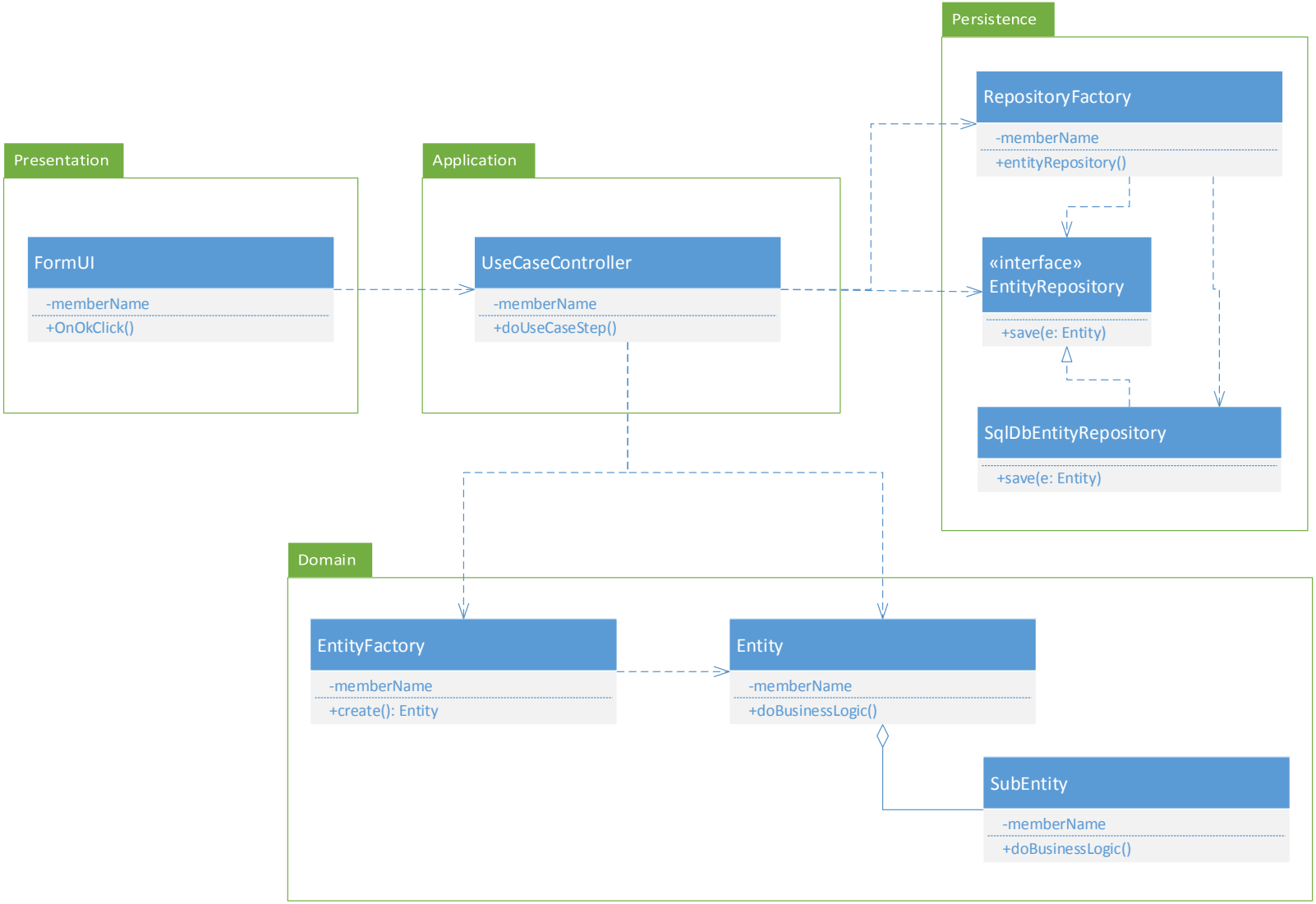
- Os objectos das interfaces gráficas são séries de produtos, que fazem sentido em conjunto (i.e. não faz sentido usar ao mesmo tempo uns controles de um estilo e outros de outros)
- Assim, uma factory deve ser capaz de construir todos os objectos do mesmo estilo/série, em vez da sua criação estar distribuída por várias fábricas




persistence.jpa
JpaExpenseTypeRepository
JpaXptoRepository
JpaRepositoryFactory
persistence.memory
MemoryExpenseTypeRepository
MemoryXptoRepository
MemoryRepositoryFactory
persistence
«interface»
ExpenseTypeRepository
«interface»
XptoRepository
«interface»
RepositoryFactory
Persistence

Topic	Principles and patterns
Which class should a responsibility be assigned to?	Information Expert Tell, don't ask Single Responsibility Principle Interface Segregation Principle Intention Revealing Interfaces
How to organize the system's responsibilities?	
How to model the domain?	Persistence Ignorance Entity Value Object Domain Service Aggregate Domain Event Observer
How to handle an object's lifecycle?	**Factories** - **Factory method** - **Simple factory** - **Abstract factory** **Repositories**
How to prepare the code for modification?	

Sterotypical architecture



Bibliografia

- Domain Driven Design. Eric Evans
- Applying UML and Patterns; Craig Larman; (2nd ed.); 2002.
- Why getters and setters are Evil. Allan Holub. http://www.javaworld.com/article/2073723/core-java/why-getter-and-setter-methods-are-evil.html
- Design Principles and Design Patterns. Robert Martin. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
- Design Patterns-Elements of Reusable Object-oriented Software, Gamma et al. (Gang of Four)