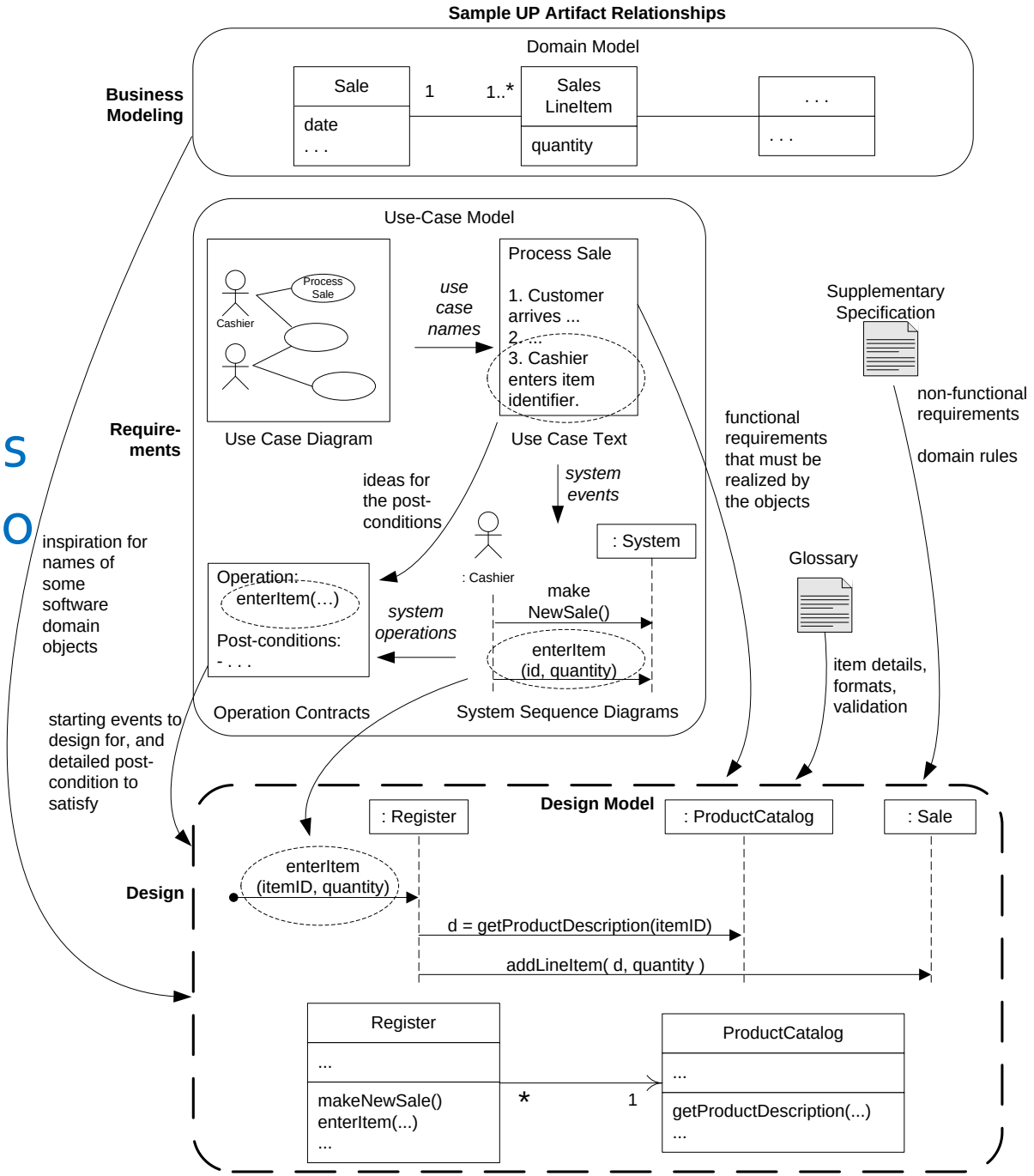
EAPLI

Princípios de Design OO: Arquitetura standard

Paulo Gandra de Sousa
pag@isep.ipp.pt

Dos requisitos para o modelo de objectos



Casos de Uso e Realização de Casos de Uso

- *Use Case Realization*
  - "A use-case realization describes how a particular use case is realized within the Design Model, in terms of collaborating objects" [RUP]
  - *Realização de um caso de uso* realça a ligação entre os requisitos, expressos por um caso de uso e o design dos objectos que garantem esses requisitos.

Object-Oriented Design
is about **behaviour**,
not structure.

Anemic Domain Model

- Plain data structures void of any behaviour
- Behaviour is concentrated in "service" and "controller" classes
  - Procedural thinking

- http://martinfowler.com/bliki/AnemicDomainModel.html

Arquitetura standard

Tarefas típicas

- Separar responsabilidades
- Interagir com o utilizador
- Controlar o fluxo da aplicação/caso de uso
- Criar entidades
- Executar lógica de negócio
- Persistir entidades
- Recuperar entidades persistidas
- Isolar partes do sistema e permitir a sua substituição

Voltaremos a cada um destes aspetos mais tarde para mais detalhes.

Organização de responsabilidades

GRASP

- Módulos (packages) com responsabilidades bem definidas
  - High cohesion
  - Low coupling
  - Information Expert

"A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle."

Brian Foote and Joseph Yoder, 1997, "Big Ball of Mud", PLoP

Layers pattern

GoF

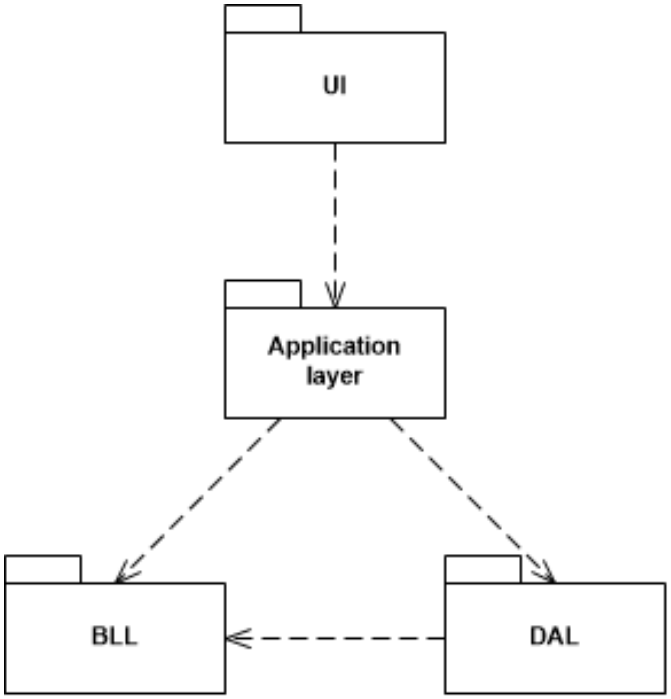
- **Problem:**
  - you are designing a system whose dominant characteristic is a mix of low- and high-level issues, where high-level operations rely on the lower-level ones.
- **Solution:**
  - Structure your system into an appropriate number of layers and place them on top of each other [each providing a specific level of abstraction].

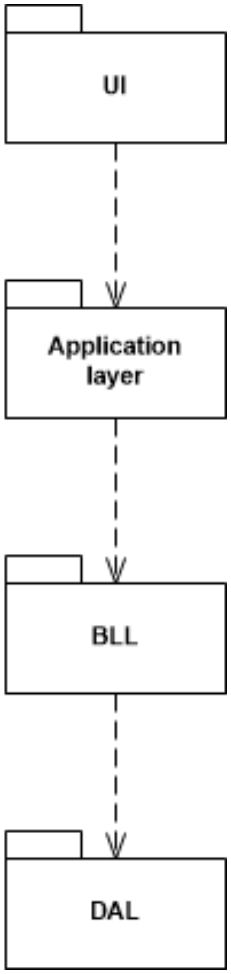
Responsabilidades típicas

- Apresentação
- Aplicação
- Domínio
- Persistência

- Reporting
- Auditing
- Authz

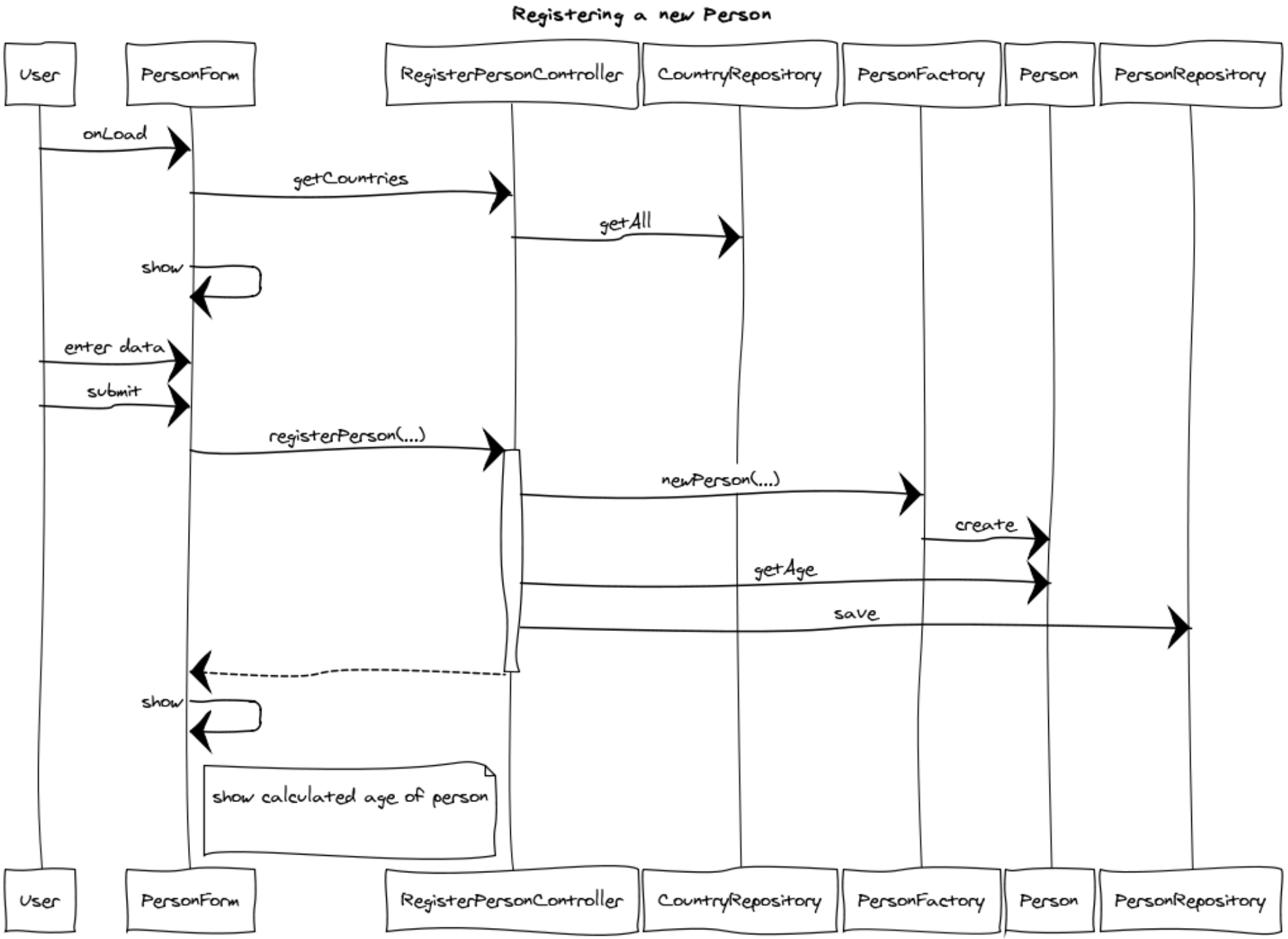
Estruturas típicas

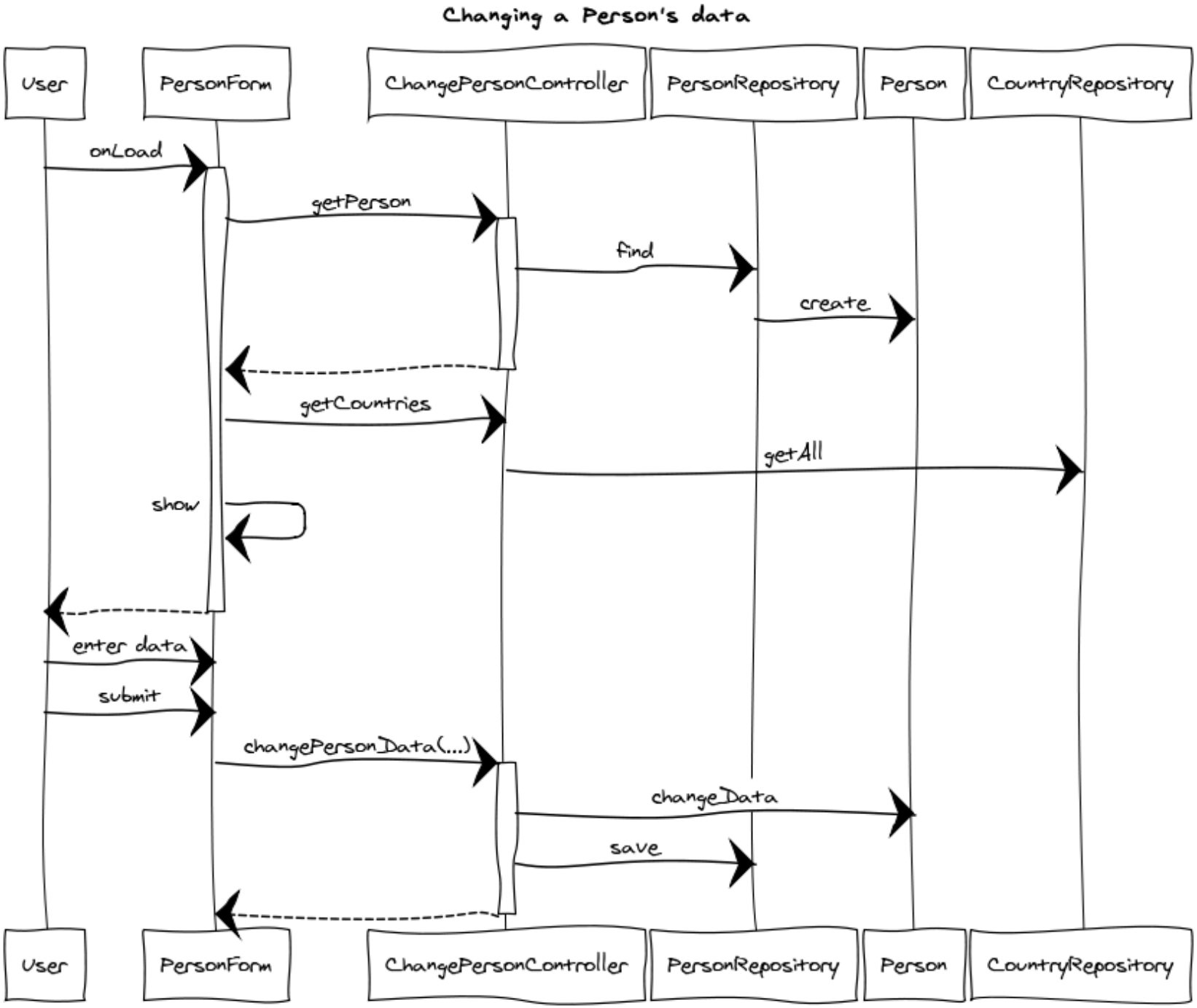




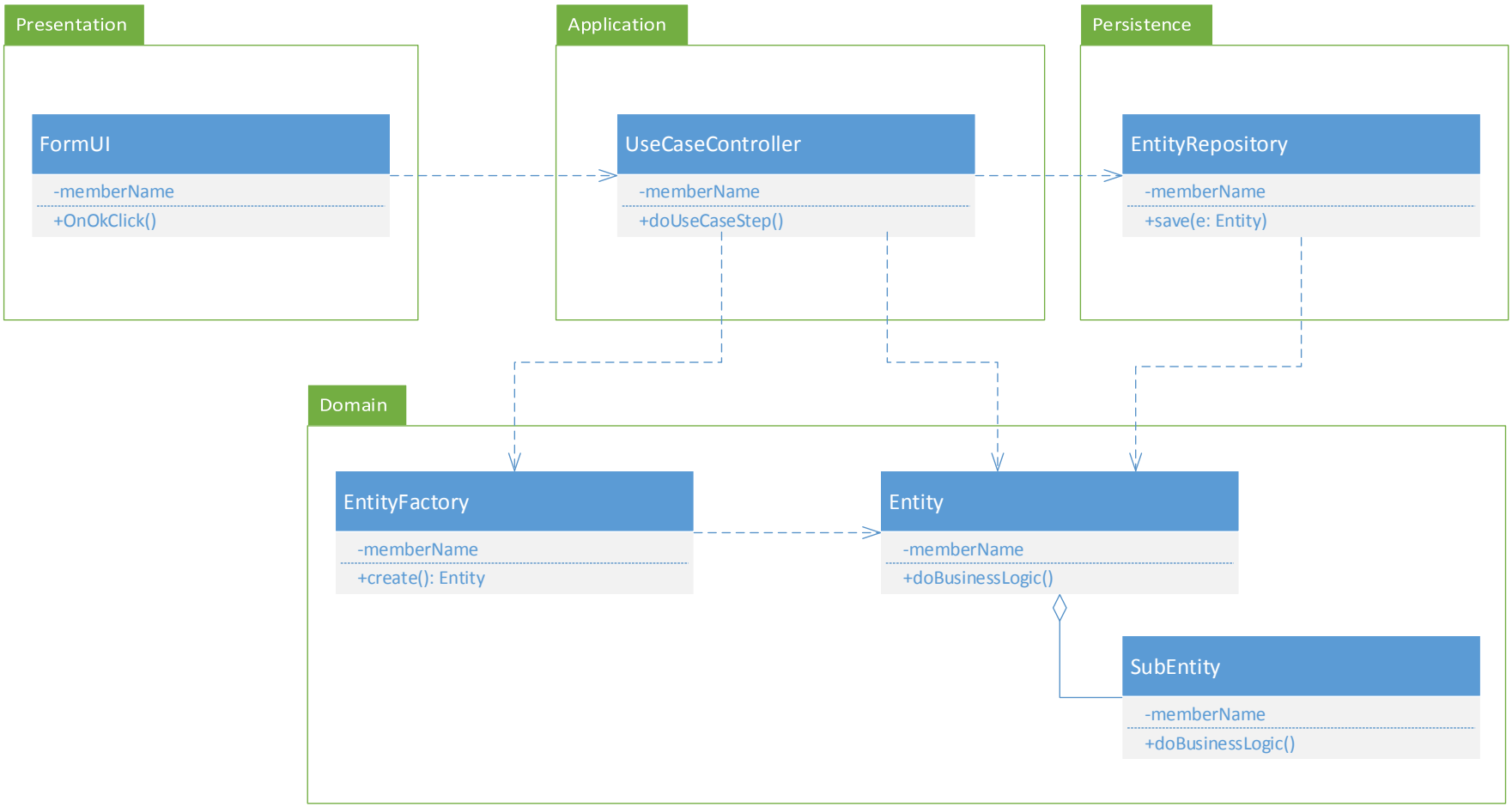
An example

- Contact information application for person's data, e.g., name, phone number, country of birth, etc.
- Use case 1: register a new contact
  - After selecting the country of birth from the list of known countries, the user enters the remainder person's data and the system saves the new person showing the age of the person
- Use case 2: change a person's data
  - After entering the person's id, the system shows the person's data to the user which may change the desired information. The system updates the person's data.





Módulos e Dependências



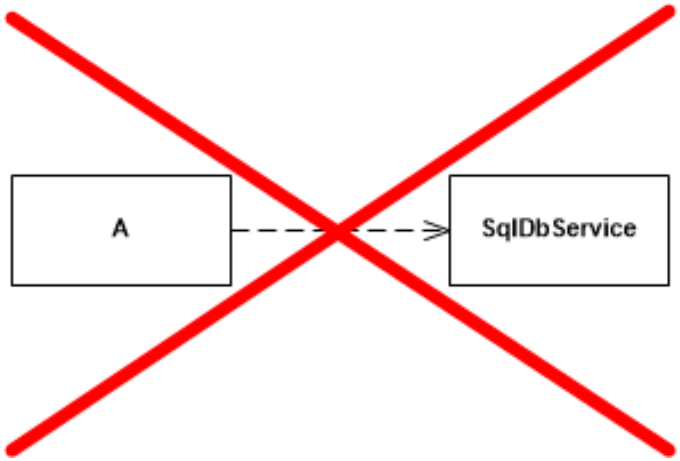
Persistence Ignorance

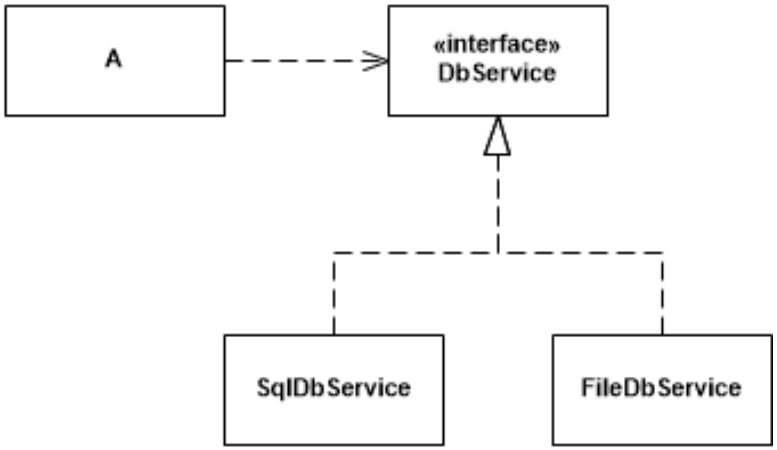
The domain should be ignorant of the actual persistence mechanism

Dependency Inversion Principle

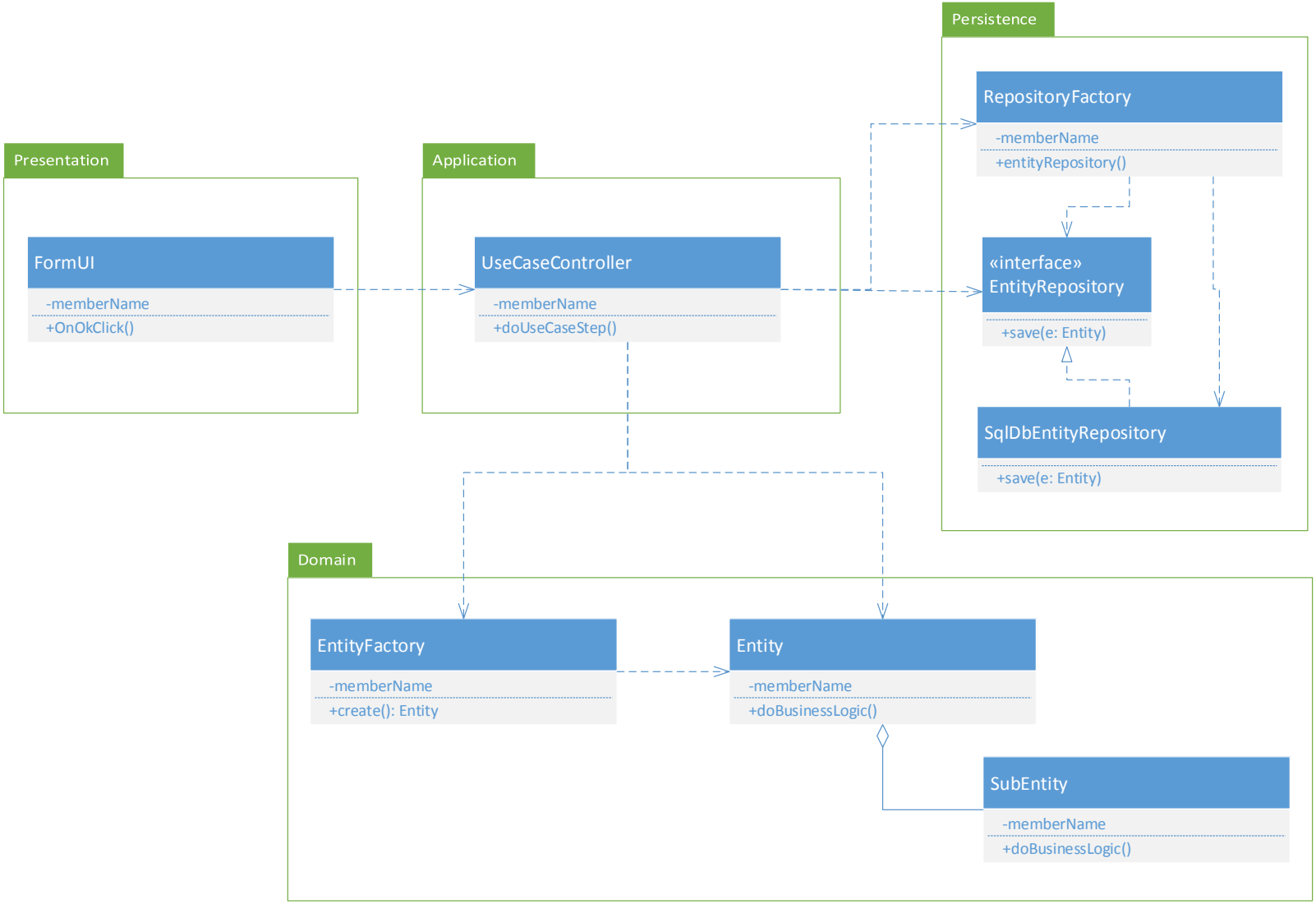
SOLID

Clients should depend on abstractions, not concretions. I.e., program to an interface not a realization.





Layers, Repositories, Factories



Muitas questões

- Como passar os dados da UI para o domínio?
- Como mostrar os dados do domínio na UI?
- Como organizar as associações entre entidades?
- Que repositórios criar?
- Que fábricas criar?
- Como obter referências para as fábricas e outros objetos "globais"?
- …

Voltaremos a cada um destes aspetos mais tarde…

Padrões
SOLID
GRASP
DDD
GoF

Resposta às Tarefas típicas

Tarefa	Solução
Separar responsabilidades	Módulos/packages Information Expert High cohesion/low coupling
Interagir com o utilizador	*Mais tarde…*
Controlar o fluxo da aplicação/caso de uso	Controller
Criar entidades	Factory Creator
Executar lógica de negócio	Classes de domínio
Persistir entidades	Repository
Recuperar entidades persistidas	Repository
Isolar partes do sistema e permitir a sua substituição	Protected Variation Dependency Inversion Principle

Bibliografia

- Anemic domain model, Martin Fowler. http://martinfowler.com/bliki/AnemicDomainModel.html
- Brian Foote and Joseph Yoder, 1997, "Big Ball of Mud", PLoP
- Design Principles and Design Patterns. Robert Martin. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf
- Domain Driven Design. Eric Evans. 2004
- Applying UML and Patterns; Craig Larman; (2nd ed.); 2002.