

Architectural Design

A Brief Introduction



Digital Skills & Jobs

Topics

- Architectural Design
 - UML Package Diagram
 - UML Component Diagram

Logical Organization of Classes

- Emphasizes the **logical architecture of the system**
- Provides a high-level view/perspective, i.e. a less detailed view but with a broader representation
- The high-level abstraction over classes facilitates the administration and coordination of the development process

Many Labs Example

Project Specification – *Many Labs* Example

Many Labs – Clinical Analysis Management System

Excerpt from the Project Specification Document

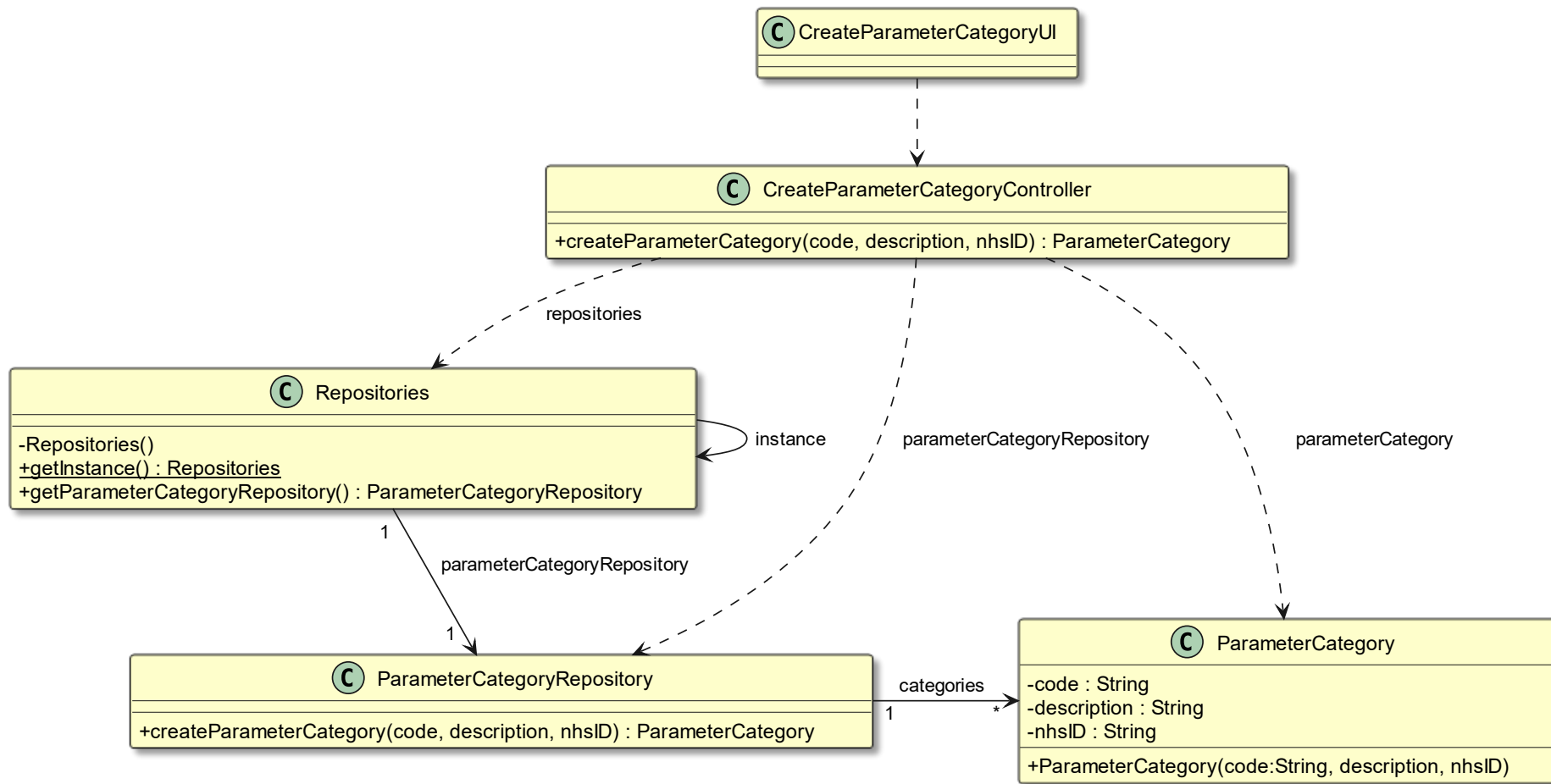
Many Labs is an English company that has a network of clinical analysis laboratories and that wants an application to manage the clinical analyses performed in its laboratories.

[...] Blood tests are frequently characterized by measuring several parameters which for presentation/reporting purposes are organized by categories. For example, parameters such as the number of Red Blood Cells (RBC), White Blood Cells (WBC) and Platelets (PLT) are usually presented under the blood count (Hemogram) category. [...] Regardless, such tests rely on measuring one or more parameters that can be grouped/organized by categories.

US 11 – As an administrator, I want to specify a new parameter category.

US11 – Class Diagram

Many Labs



UML Package Diagram

UML Package Diagram

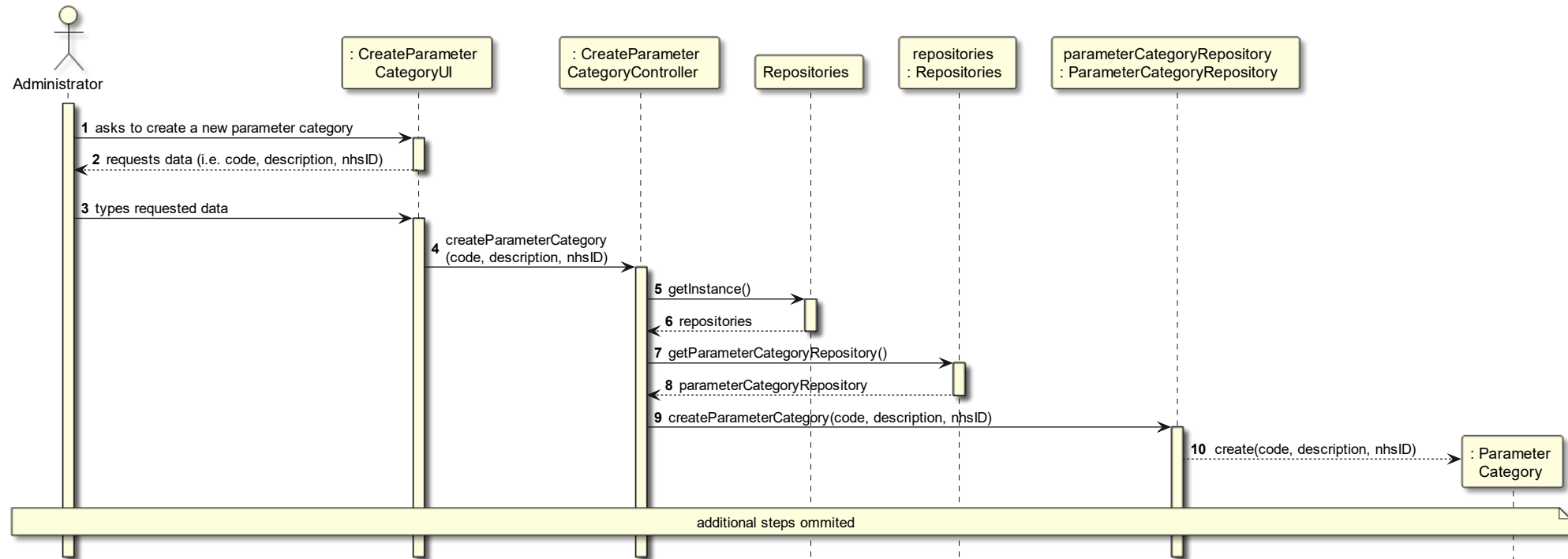
- Often **used to illustrate the logical architecture of a system**
- It's a general-purpose mechanism/notation for grouping system elements (e.g. classes, actors, use cases, other packages)
- Can be used to depict distinct perspectives
 - **Logical perspective** of the system (e.g. based on system elements)



- **Physical perspective** of the system (e.g. based on system file organization)

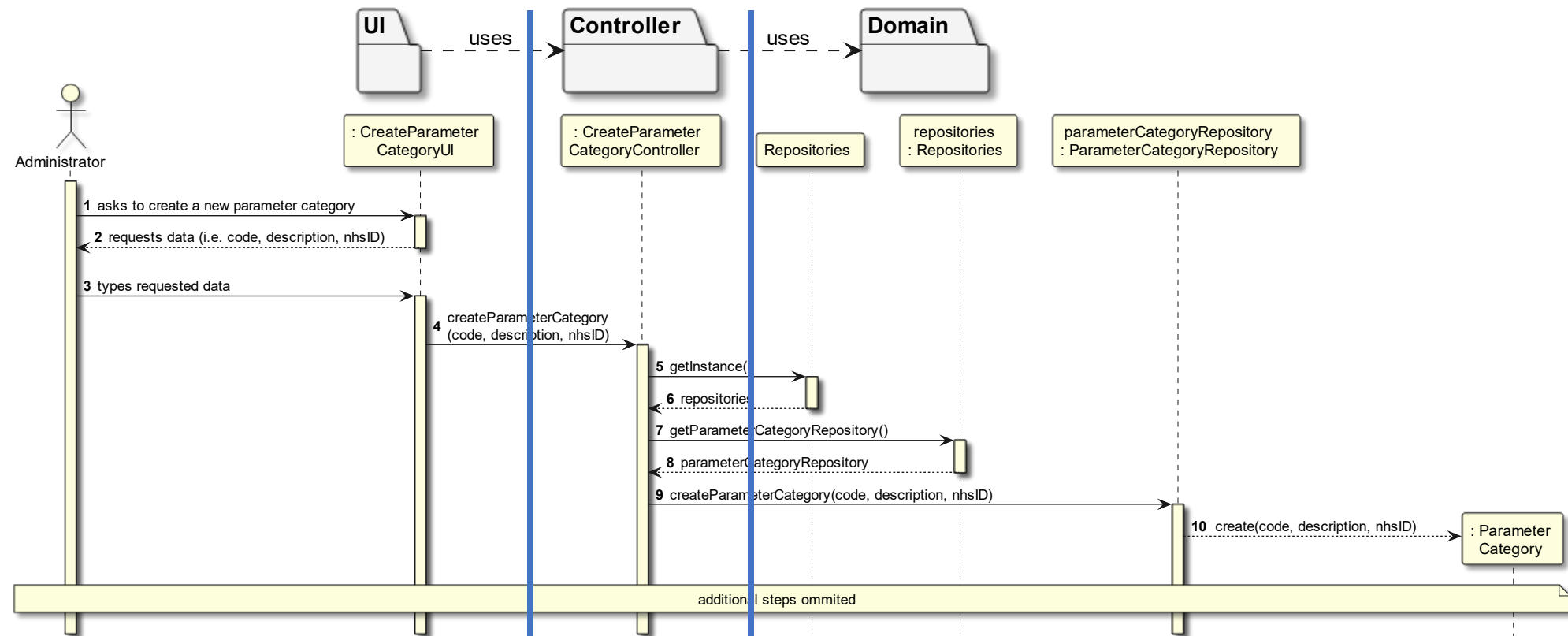
Organizing Classes by their Concerns (1/2)

- Logical perspective
 - Boundary classes between the actors and the system (**UI**) and between the system and other (sub-)systems
 - System control logic classes (**Controller**)
 - Classes of information used by the system (related to the Application **Domain**)



Organizing Classes by their Concerns (2/2)

- Logical perspective
 - Boundary classes between the actors and the system (**UI**) and between the system and other (sub-)systems
 - System control logic classes (**Controller**)
 - Classes of information used by the system (related to the Application **Domain**)



Boundary Classes

- Model the interaction between the actors and the system
- User Interface classes
 - Focus on information requested/presented to system actors
 - E.g.:
 - CreateParameterCategoryUI
 - CreateTestTypeUI
- Classes used to connect the system to other (sub-)systems
 - Focus on the necessary protocols and not on their implementation
 - Fosters adoption of High Cohesion, Low Coupling, Protected Variation, ...
 - E.g.:
 - <<interface>> PasswordGenerator (recall previous presentation)

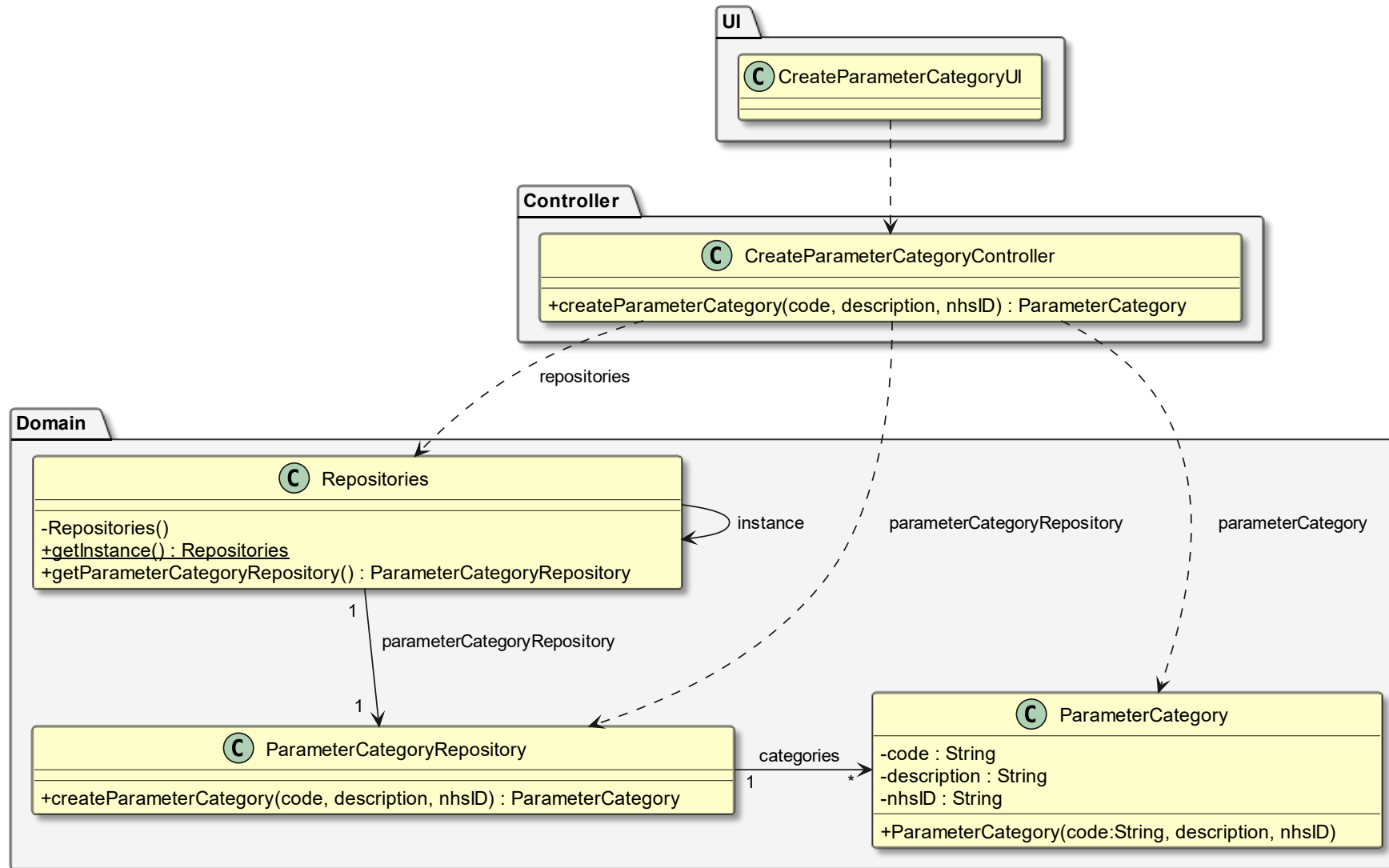
System Control Classes

- Aim to control and coordinate the behavior of the system
- Delegate tasks to other classes
 - Control classes should redirect the tasks to other classes
- Control classes decouple boundary classes from domain classes
- E.g.:
 - CreateParameterCategoryController
 - CreateTestTypeController

Domain Classes

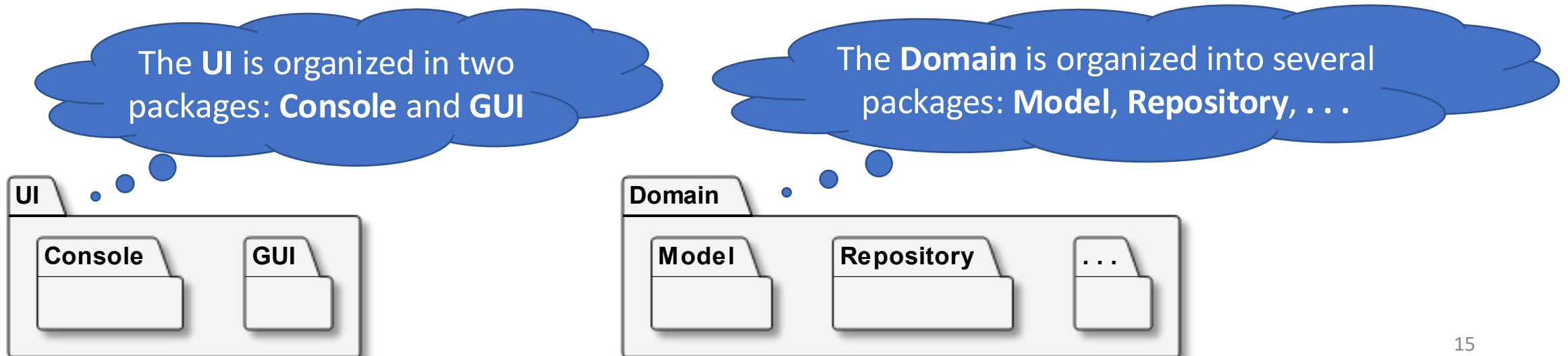
- Used to model the key concepts of the application domain
- Contains the logic to solve the problem underlying the system development
- Typically, these classes model information persistence
- Can be used in multiple contexts (i.e. multiple use cases)
- E.g.:
 - Repositories
 - ParameterCategory
 - Test
 - TestType

Applying Packages to the US11 Class Diagram

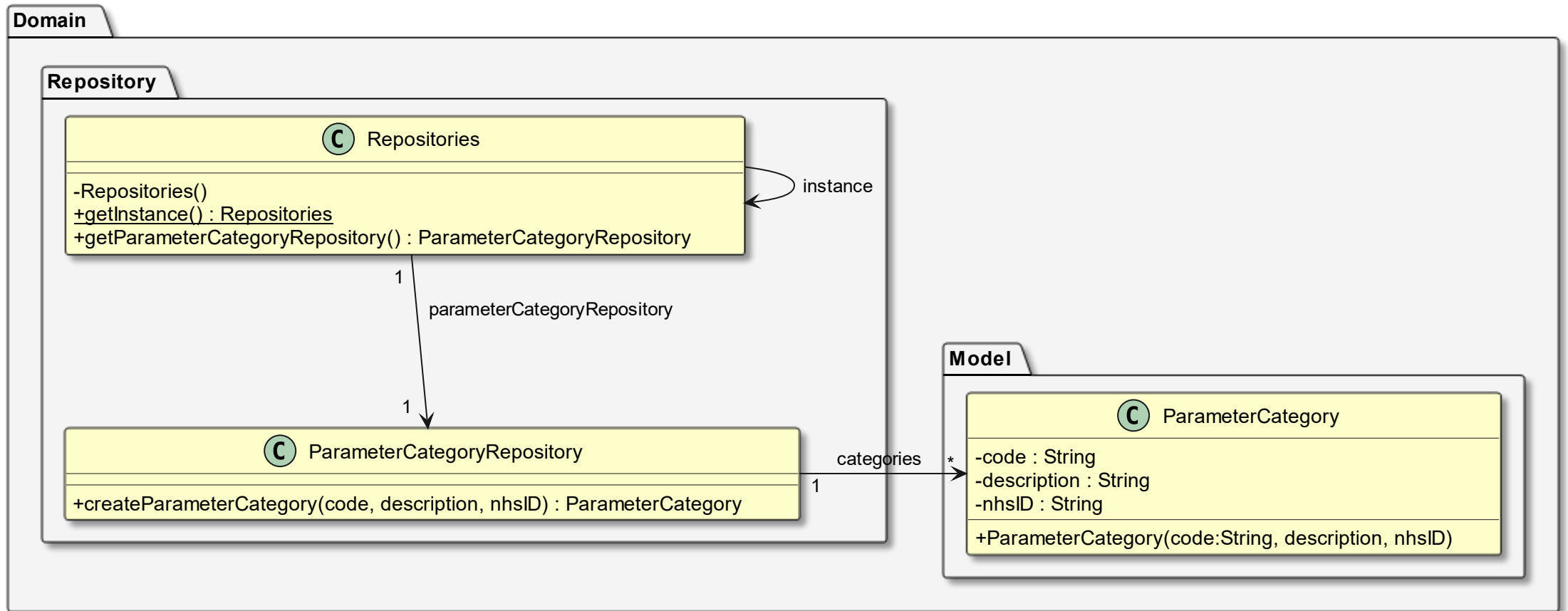


UML Package Diagram – Granularity

- In the same diagram, adopt the same level of abstraction/detail (i.e. granularity) across all depicted packages
- System granularity (UI → Controller → Domain)
- Package granularity: details regarding the internal organization of a package should be provided in another diagram



US11 – Domain Package contents



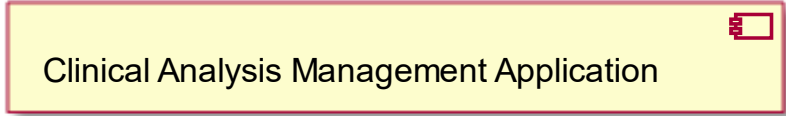
UML Component Diagram

UML Component Diagram

- General-purpose mechanism/notation **for depicting:**
 - **modular parts of a system (called components); and**
 - **their interaction in terms of provided and required interfaces.**
- Components might be either logical or physical
- Components are usually reusable
- All depicted components should have the same level of abstraction/detail (i.e. granularity)

Examples of Component Diagrams (1/2)

- System granularity
 - The system has only a single component named *"Clinical Analysis Management Application"*

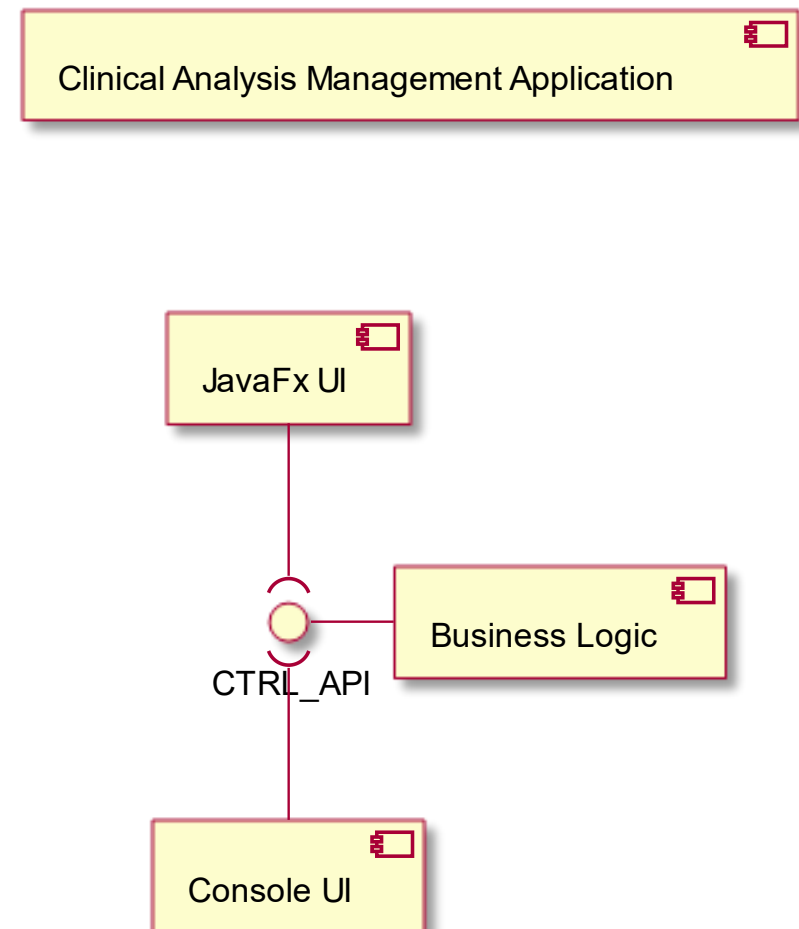


Clinical Analysis Management Application

The diagram shows a single component represented by a yellow rectangle with a red border. The text 'Clinical Analysis Management Application' is centered within the rectangle. A small red icon is visible in the top right corner of the rectangle.

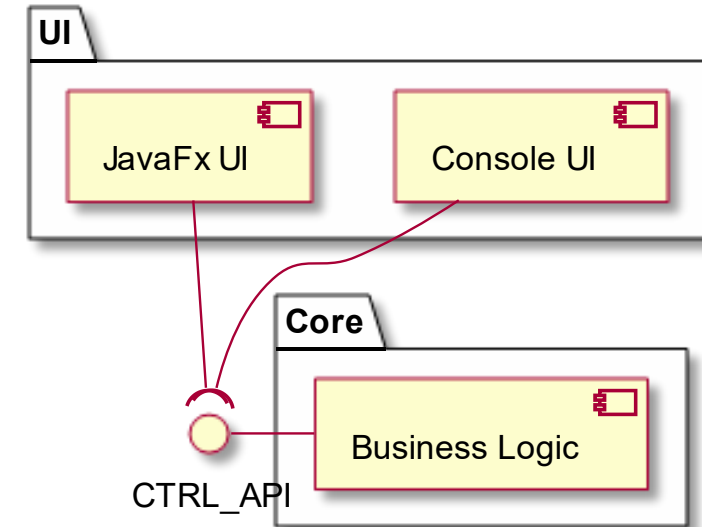
Examples of Component Diagrams (2/2)

- System granularity
 - The system has only a single component named *“Clinical Analysis Management Application”*
- Detailing the above component
 - The *“Business Logic”* component is reused by both *“Console UI”* and *“JavaFx UI”* components



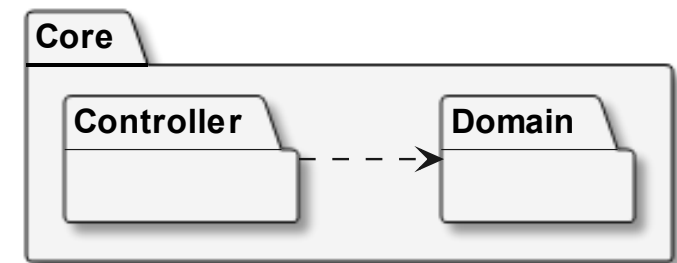
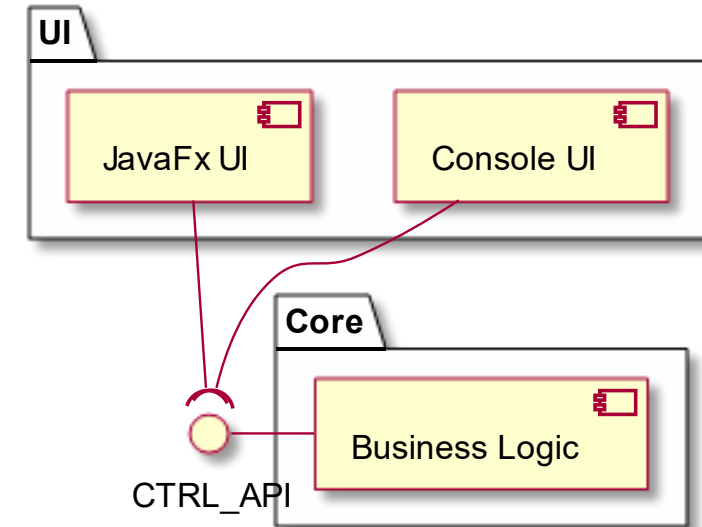
Using Packages to group Components (1/2)

- The “*UI*” package comprises two components (i.e. “*JavaFx UI*” and “*Console UI*”)
- The “*Core*” package comprises a single component (i.e. “*Business Logic*”)



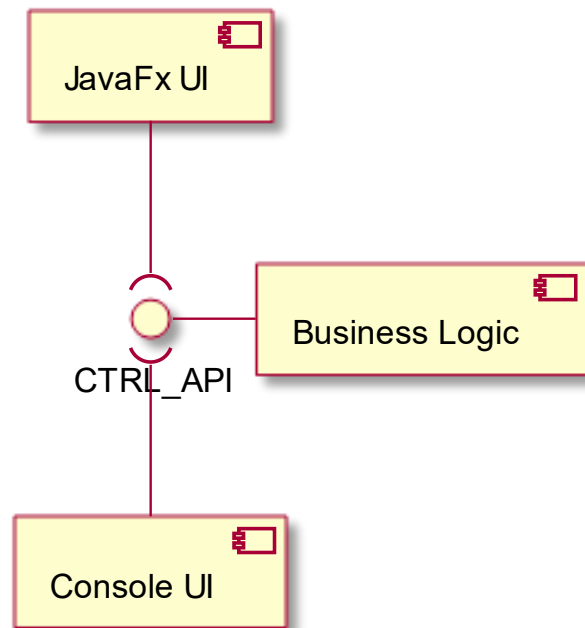
Using Packages to group Components (2/2)

- The “*UI*” package comprises two components (i.e. “*JavaFx UI*” and “*Console UI*”)
- The “*Core*” package comprises a single component (i.e. “*Business Logic*”)
- The “*Business Logic*” component is the results of assembling two packages: “*Controller*” and “*Domain*”



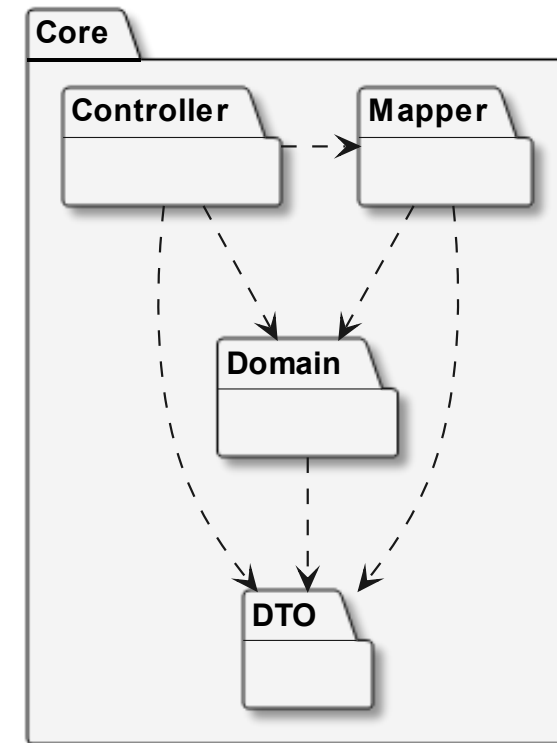
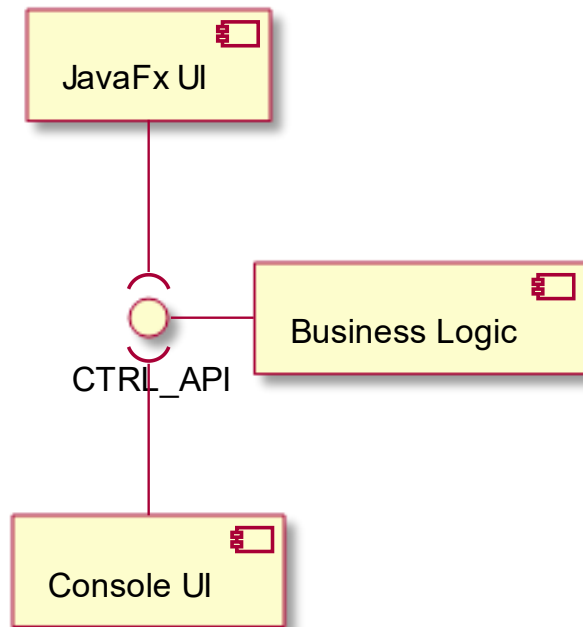
What about Mappers and DTO? (1/2)

- Should the component diagram be updated?
 - No! There is no need.



What about Mappers and DTO? (2/2)

- Should the component diagram be updated?
 - No! There is no need.
- Should the package diagram be updated?
 - Yes! Cf. the figure.



References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066