

Promoting Collections to Software Classes

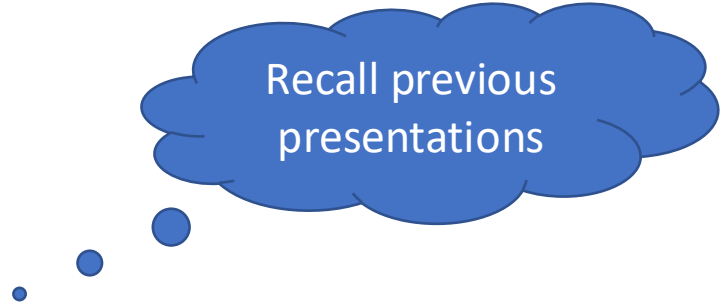
by applying High Cohesion, Low Coupling and Pure Fabrication



Topics

- Cohesion and Coupling
- GRASP
 - High Cohesion
 - Low Coupling
- Promotion of Collections to Software Classes


Cohesion and Coupling



Recall previous presentations

- **Cohesion** is a measure regarding the **coherence of the responsibilities** assigned to an element of the system
- **Coupling** is a measure of **how strongly an element** is connected to, or has knowledge of, or **is dependent on other elements** of the system
- You should adopt design alternatives favoring
 - **High Cohesion**
 - **Low Coupling**

High Cohesion



Recall previous presentations

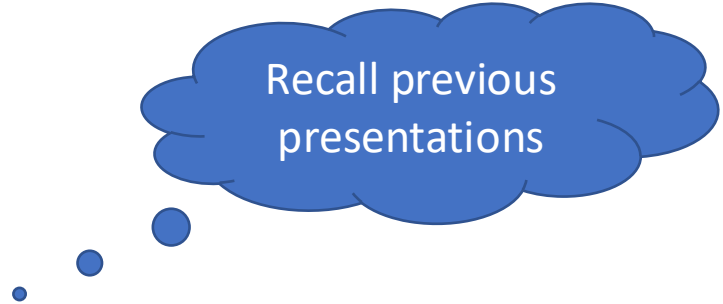
- **Problem**

- How to maintain classes/objects with coherent and easy-to-understand functionalities?

- **Solution**

- Assign a responsibility so that cohesion remains high
 - Features should be strongly related with each other
 - Prevent the same class/object from doing many different things
 - Cooperate with other classes
 - Tell other classes to do something about data they know
 - Do not ask other classes for data (avoid *getX* methods)
 - Delegate other responsibilities to other classes
- } Tell, Don't Ask Principle

Low Coupling



Recall previous presentations

- **Problem**

- How to achieve low dependency, low impact on changes and increased reuse between classes/objects?

- **Solution**

- Assign responsibilities to maintain a low coupling
- Avoid unnecessary dependencies
- Apply indirection mechanisms ([Indirection Pattern](#)) to assign the responsibility of mediation between two classes/objects to an intermediate class, thus ensuring decoupling (e.g. the controller classes play this role)

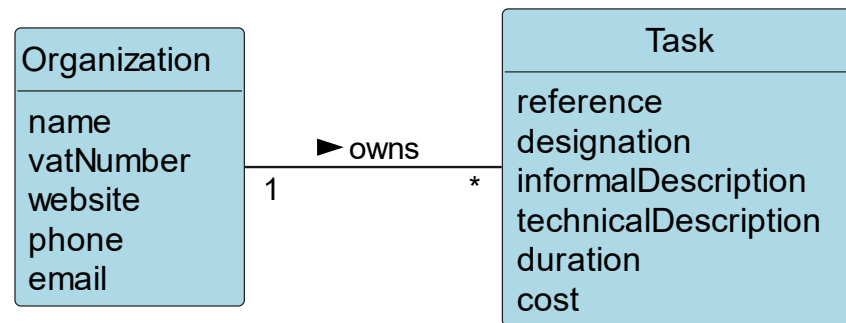
Motivating the Problem – Part I

UC010 – List Organization Tasks

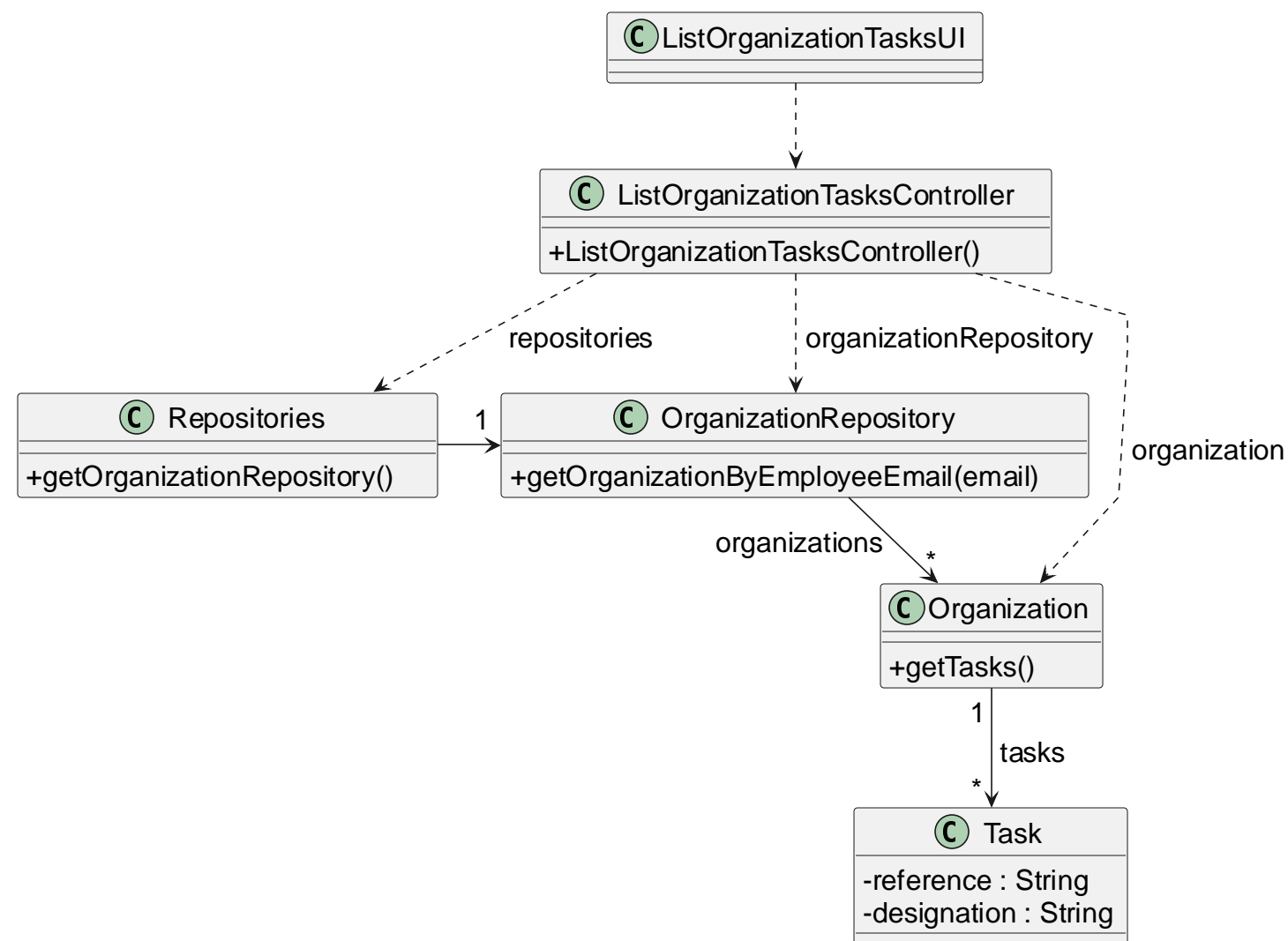
UC010 – List Organization Tasks

*Platform for
Outsourcing Tasks*

- As an Organization Employee, I want to list all tasks created by my organization.
 - AC1: The tasks must be sorted alphabetically by their designation.
- Relevant Domain Model excerpt

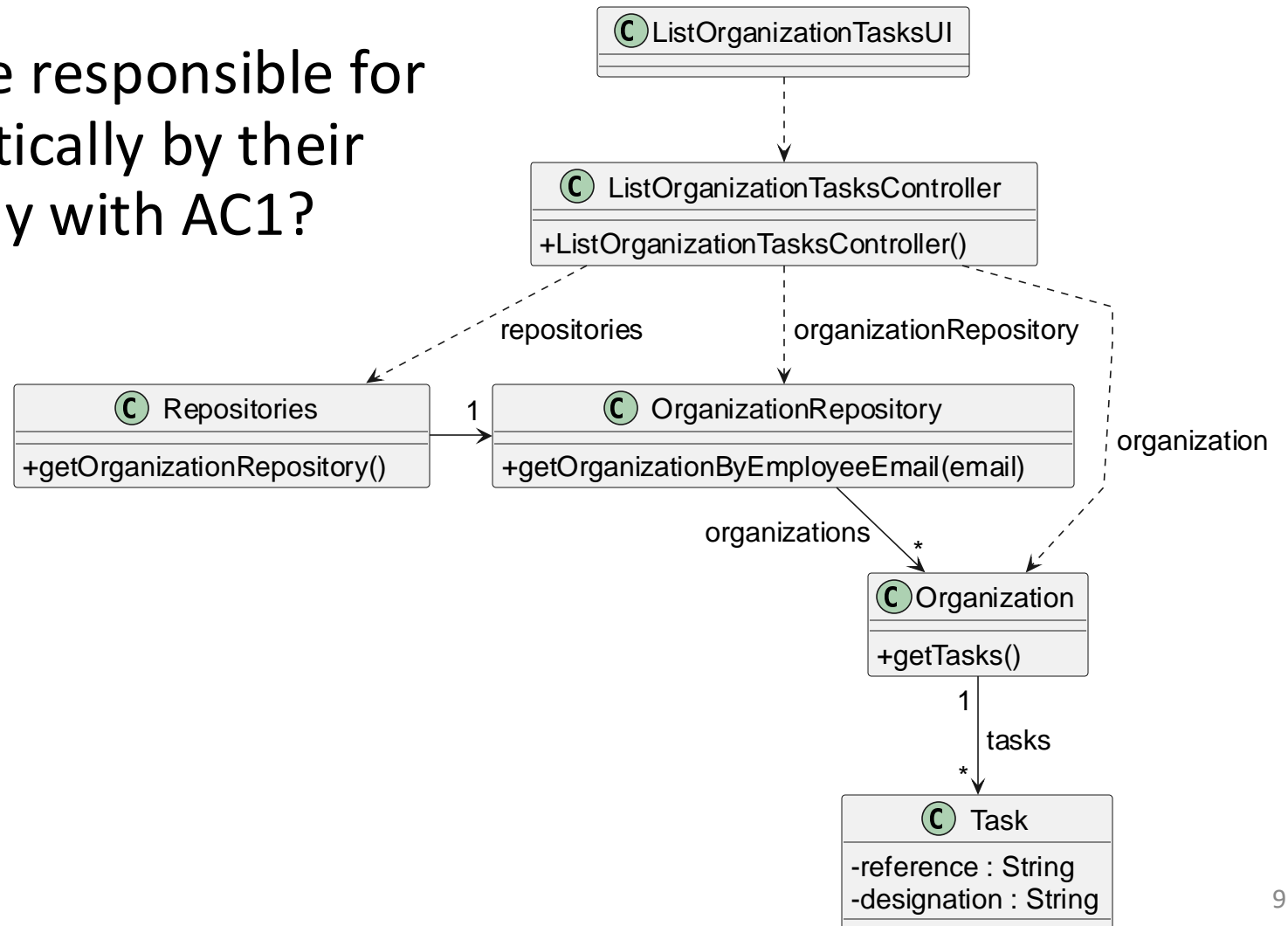


UC010 – Partial Class Diagram



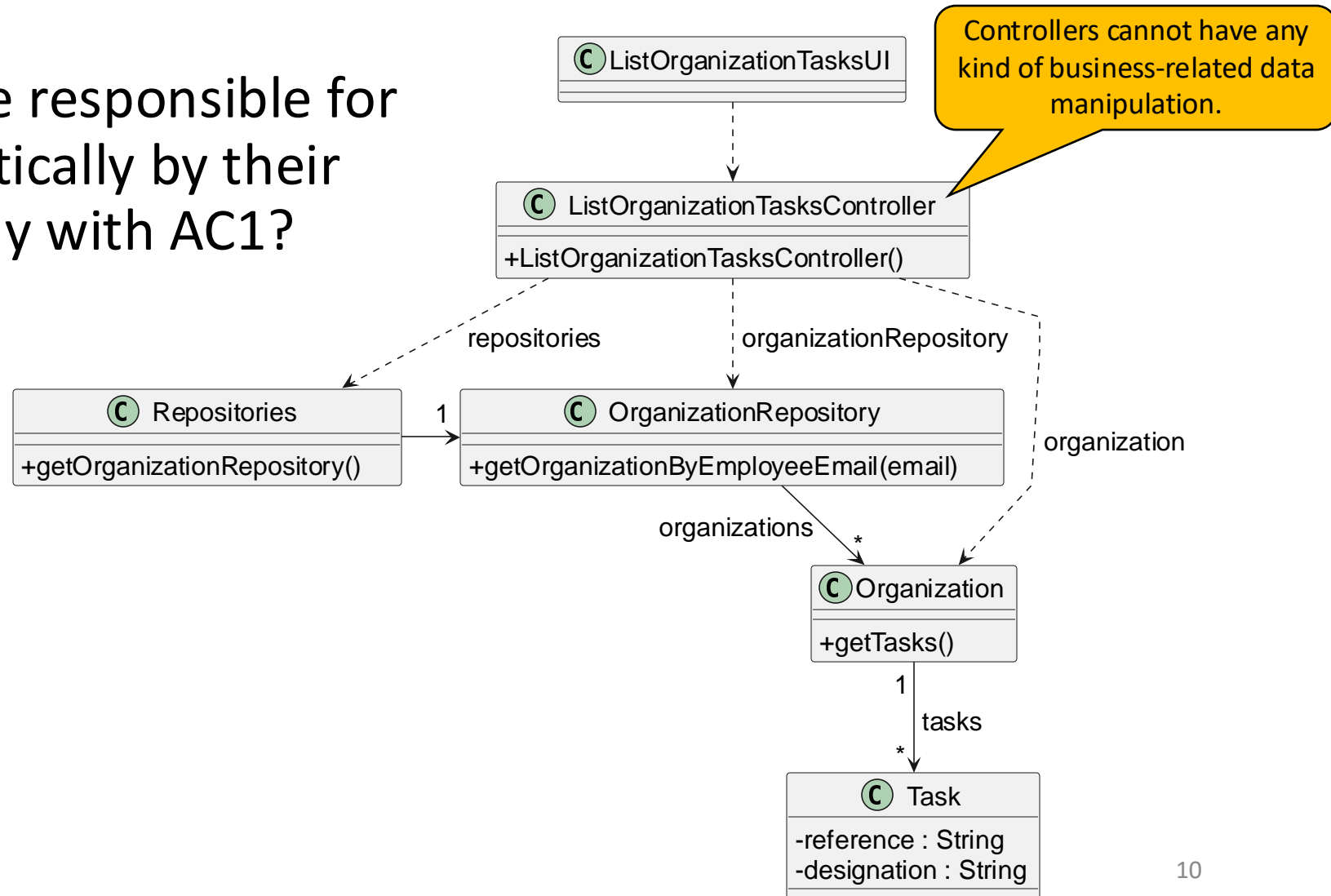
Complying with AC1: Sorting Tasks (1/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - Controller?
 - Organization?
 - Task?



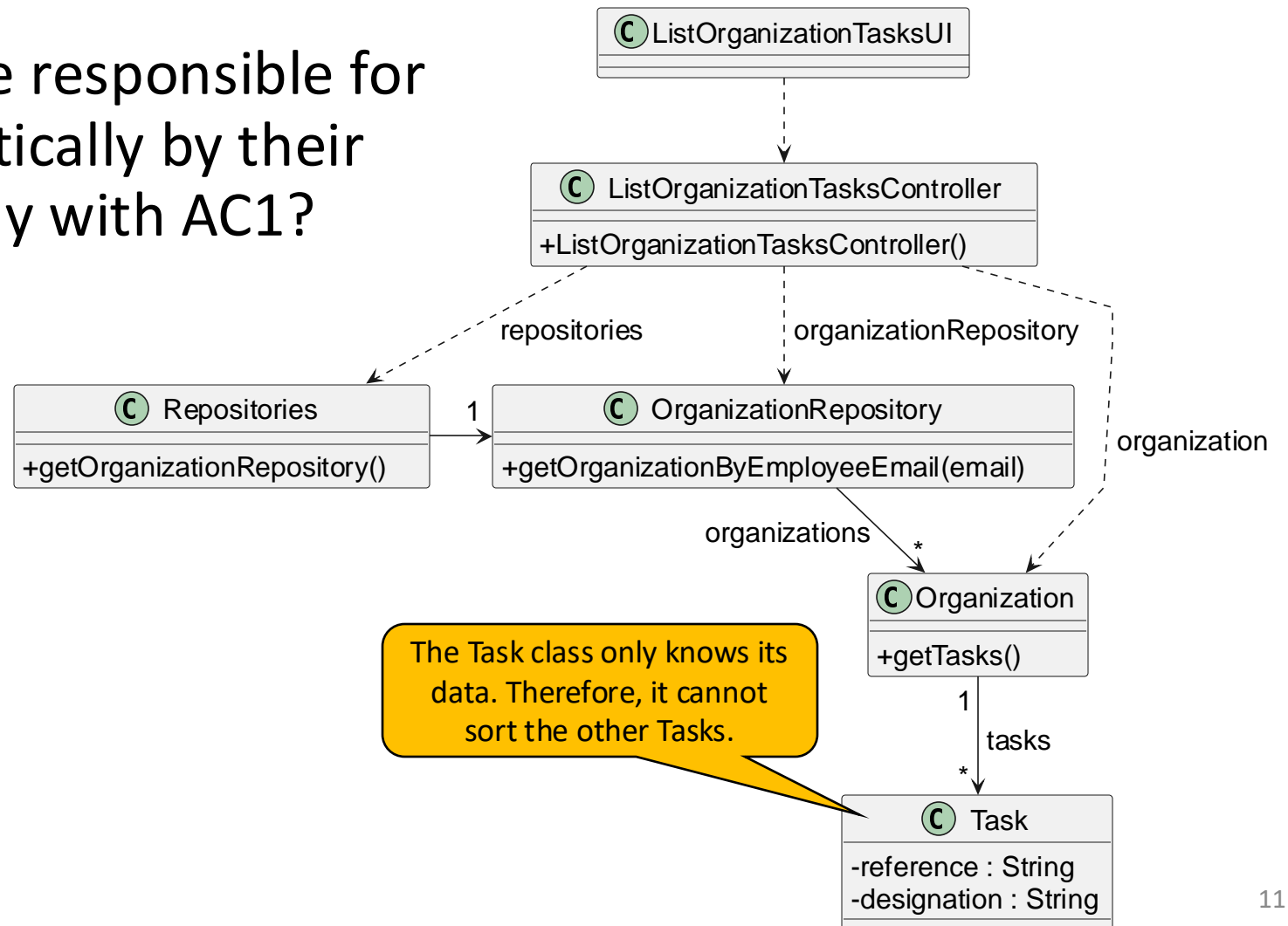
Complying with AC1: Sorting Tasks (2/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - ~~Controller?~~
 - Organization?
 - Task?



Complying with AC1: Sorting Tasks (3/5)

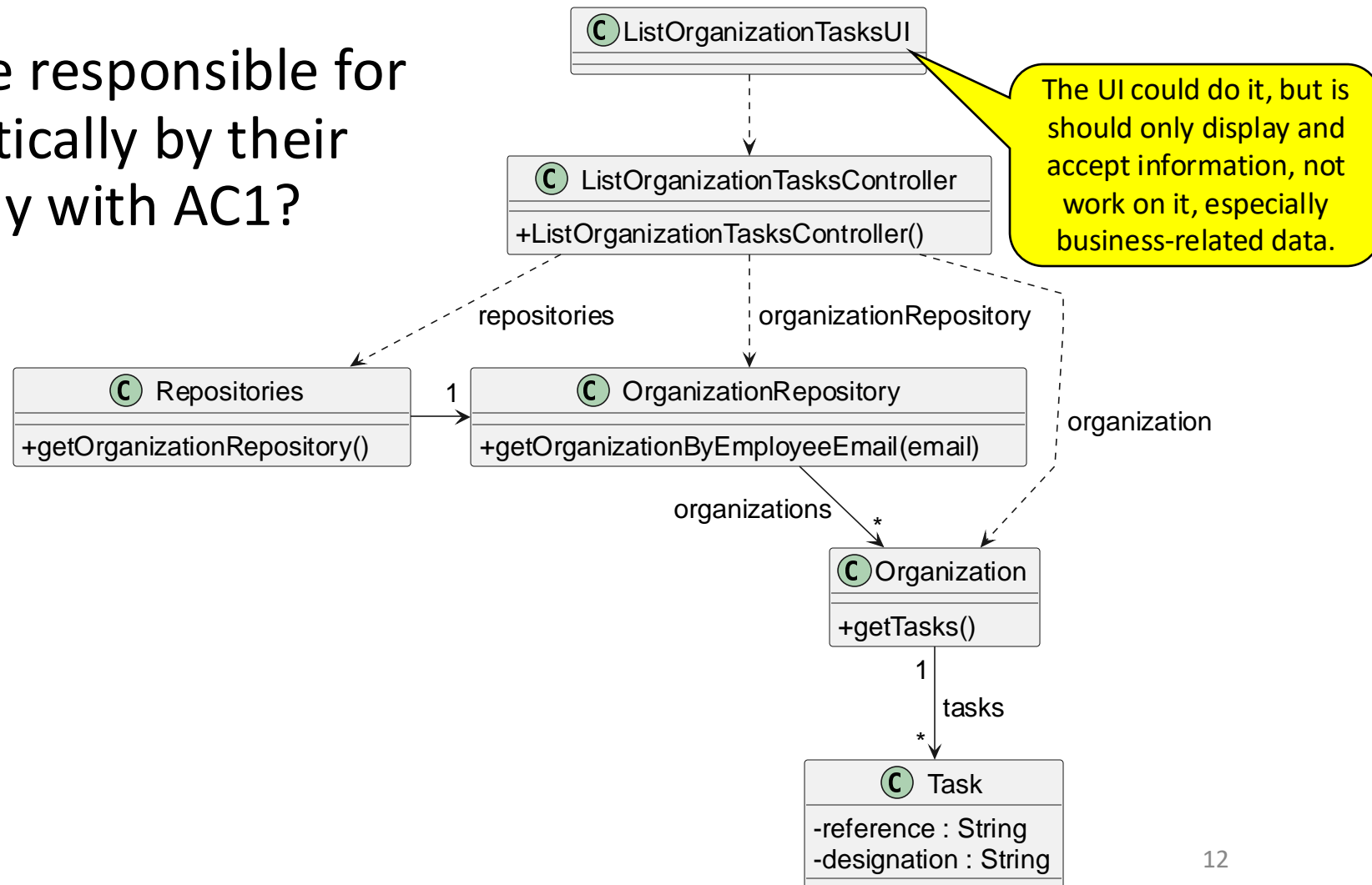
- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - ~~Controller?~~
 - Organization?
 - ~~Task?~~



Complying with AC1: Sorting Tasks (4/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?

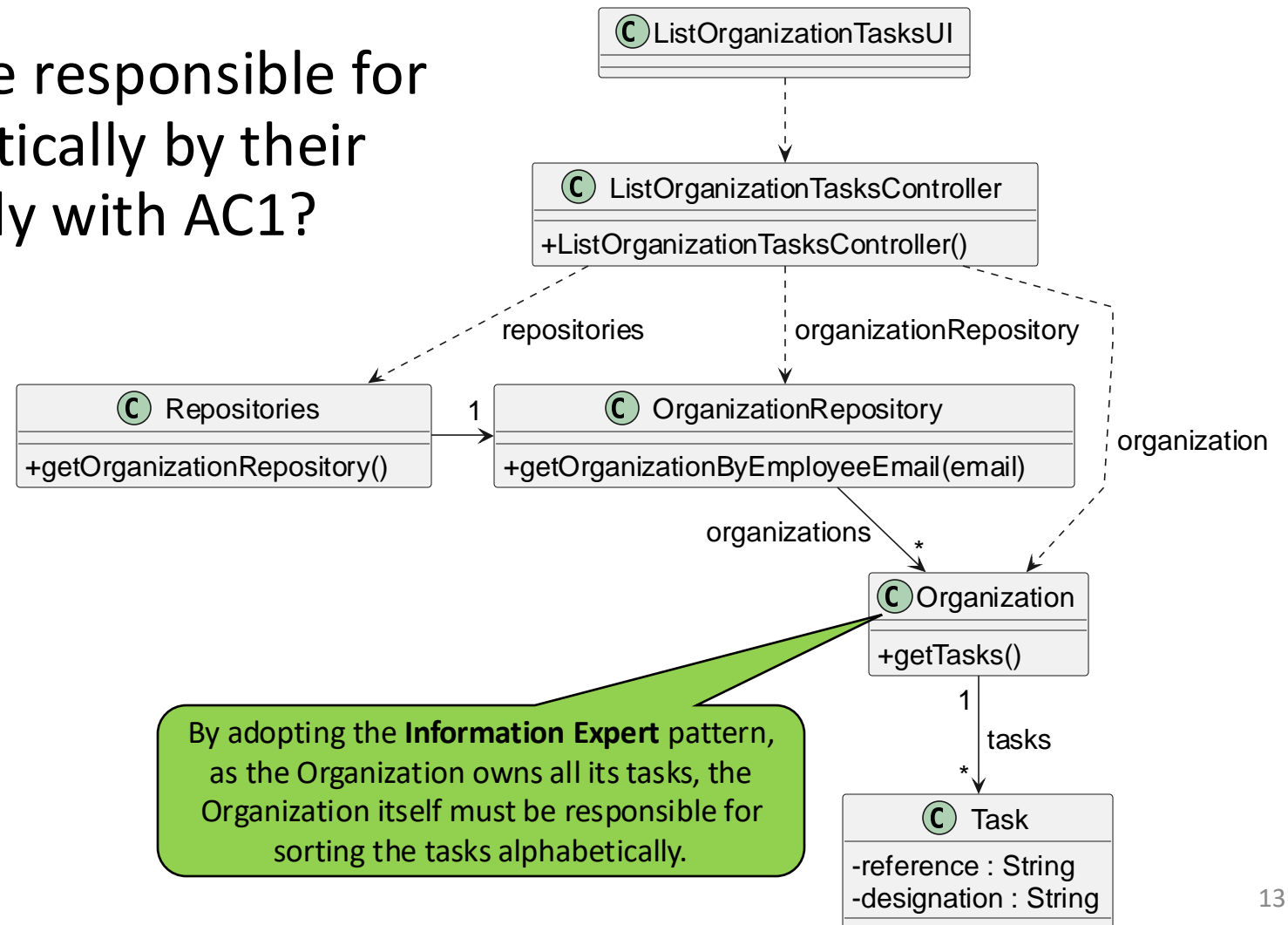
- UI?
- Controller?
- Organization?
- Task?



Complying with AC1: Sorting Tasks (5/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?

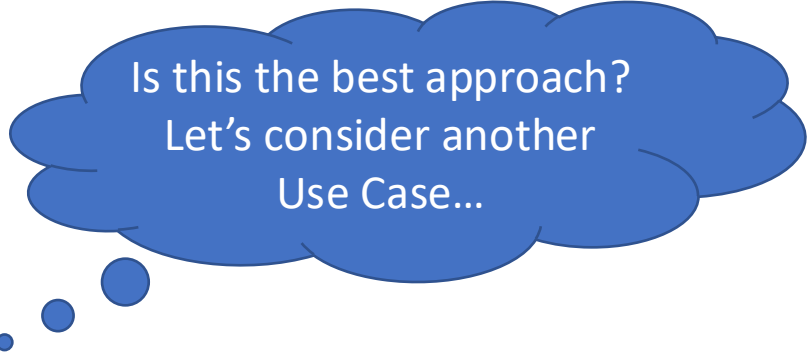
- ~~UI?~~
- ~~Controller?~~
- Organization? ✓
- ~~Task?~~



UC010 – AC1 Rationale (1/2)

- To comply with AC1 and according to Information Expert (IE), **the Organization class should be responsible for sorting the Tasks** because it holds the Tasks for the Organization
- The Organization class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Organization and Task
 - Remember the “**Tell, Don’t Ask**” principle – how can the Organization have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class?
 - using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Organization class
 - The Organization must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

UC010 – AC1 Rationale (2/2)



Is this the best approach?
Let's consider another
Use Case...

- To comply with AC1 and according to Information Expert (IE),
the Organization class should be responsible for sorting the Tasks because it holds the Tasks for the Organization
- The Organization class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Organization and Task
 - Remember the “**Tell, Don't Ask**” principle – how can the Organization have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class?
– using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Organization class
 - The Organization must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

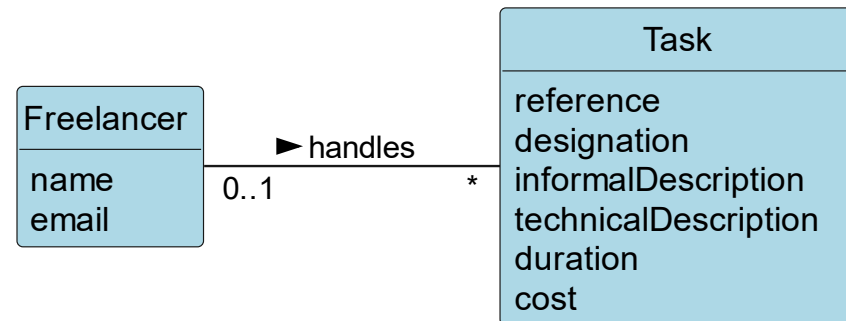
Motivating the Problem - Part II

UC011 – List Freelancer Tasks

UC011 – List Freelancer Tasks

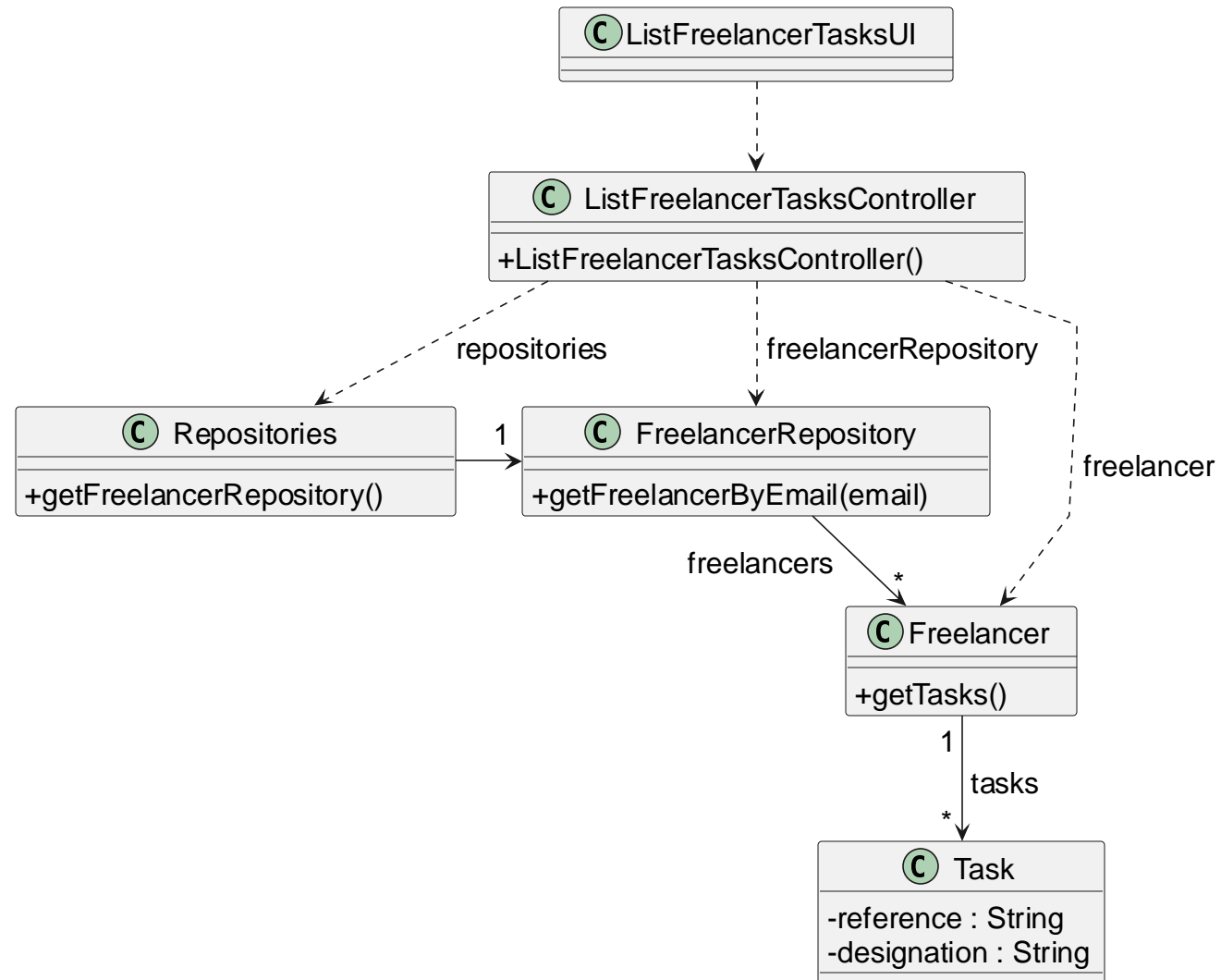
*Platform for
Outsourcing Tasks*

- As a Freelancer, I want to list all my assigned tasks.
 - AC1: The tasks must be sorted alphabetically by their designation.
- Relevant Domain Model excerpt



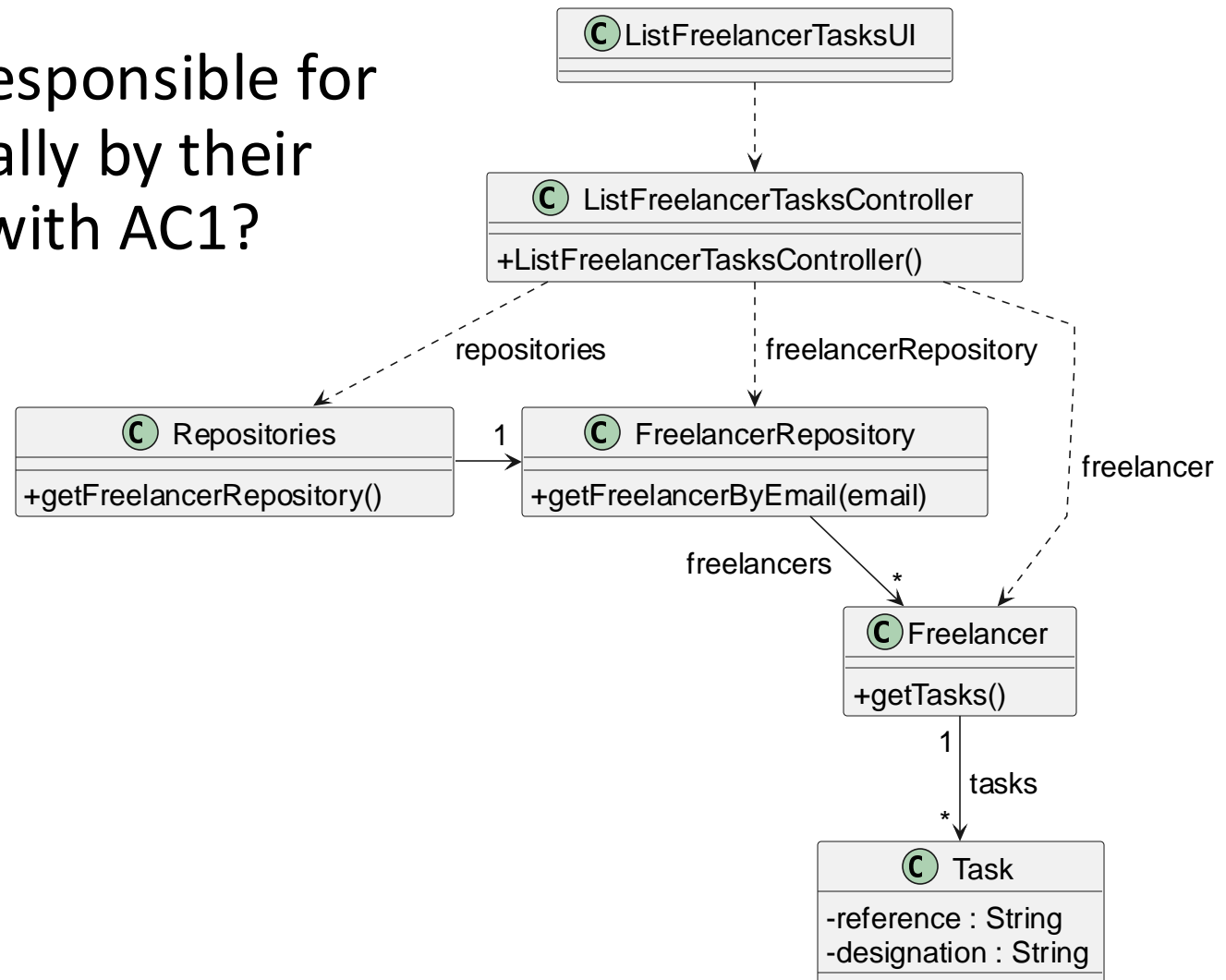
UC011 – Partial Class Diagram

Platform for
Outsourcing Tasks



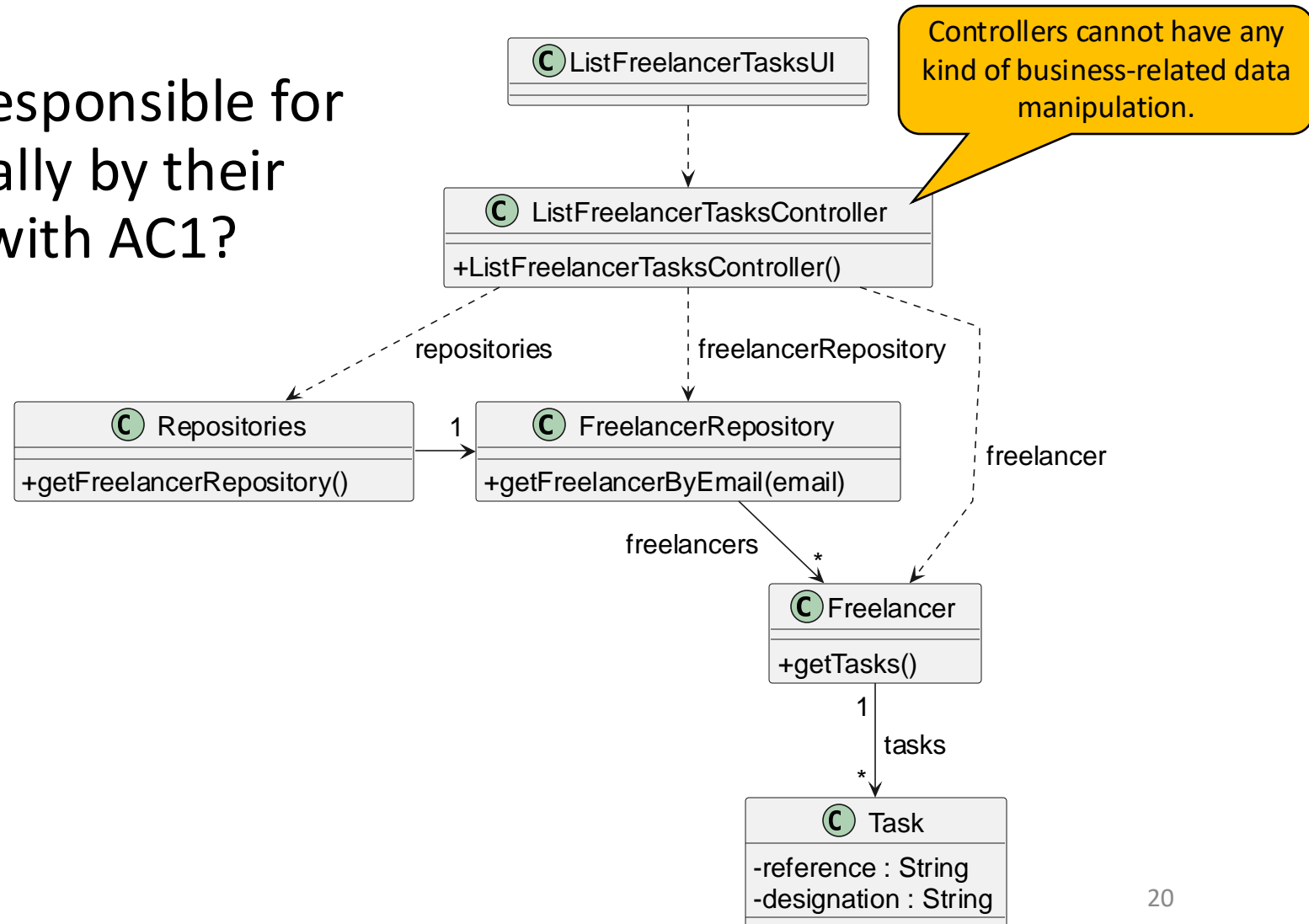
Complying with AC1: Sorting Tasks (1/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - Controller?
 - Organization?
 - Task?



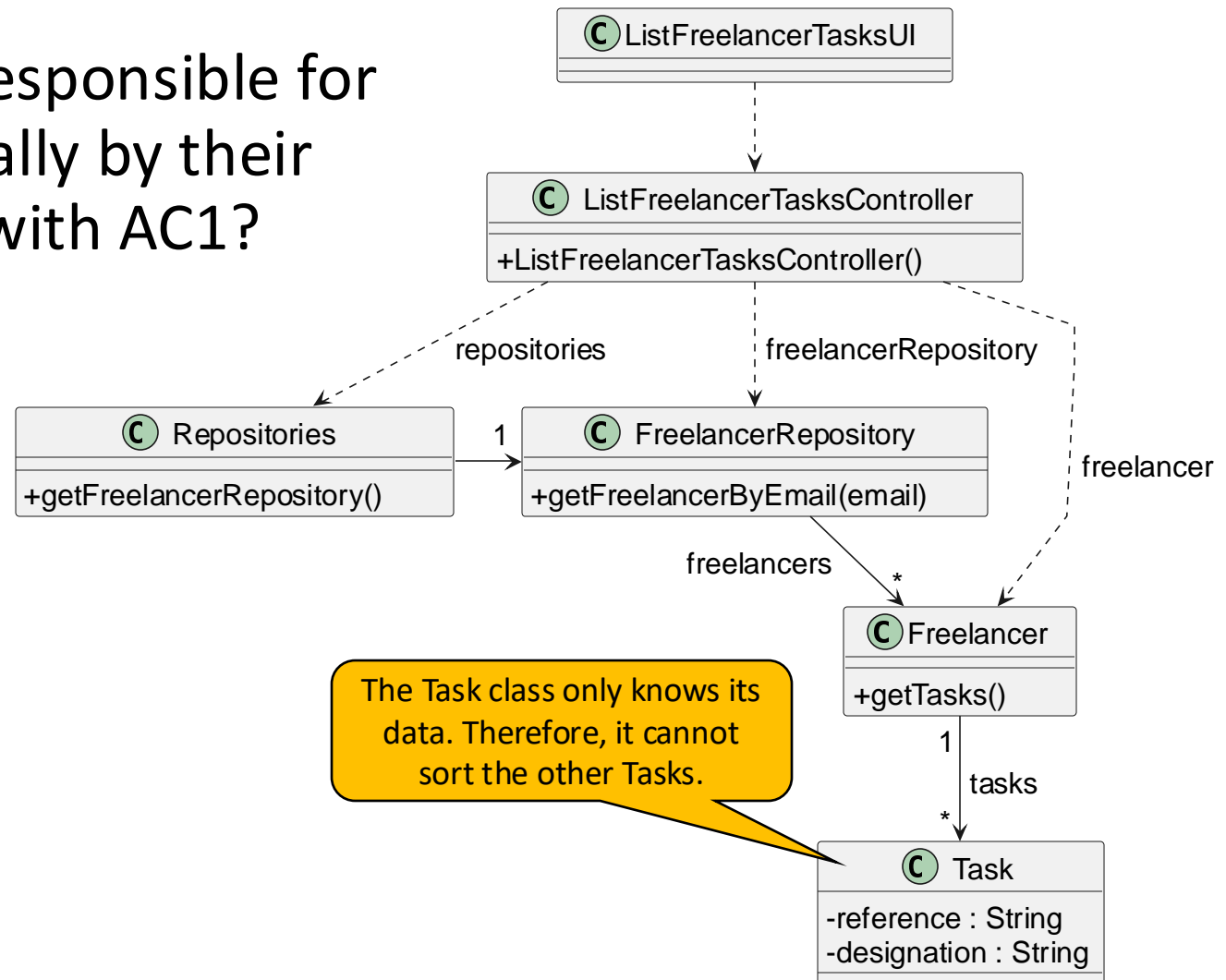
Complying with AC1: Sorting Tasks (2/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - ~~Controller?~~
 - Organization?
 - Task?



Complying with AC1: Sorting Tasks (3/5)

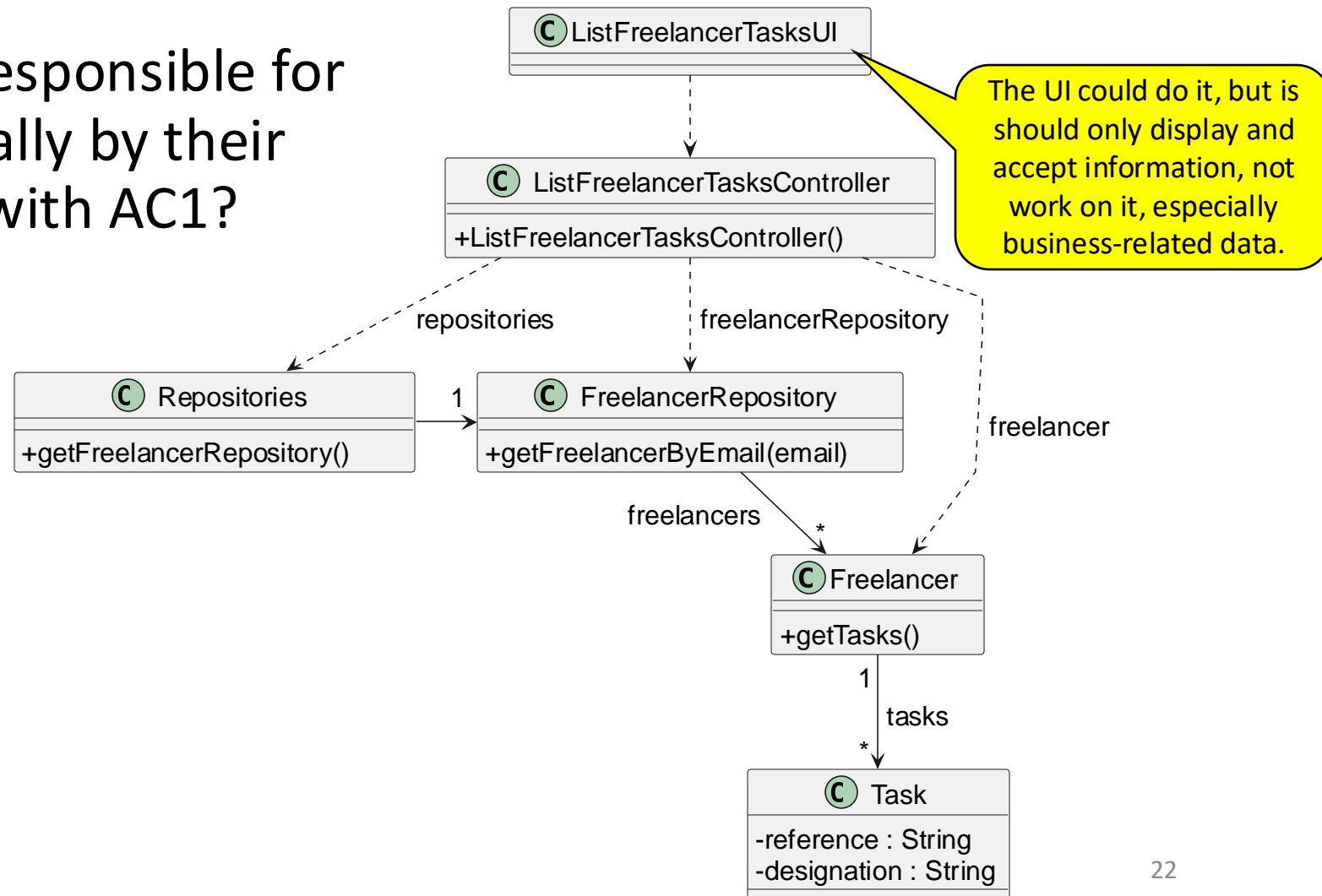
- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?
 - UI?
 - ~~Controller?~~
 - Organization?
 - ~~Task?~~



Complying with AC1: Sorting Tasks (4/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?

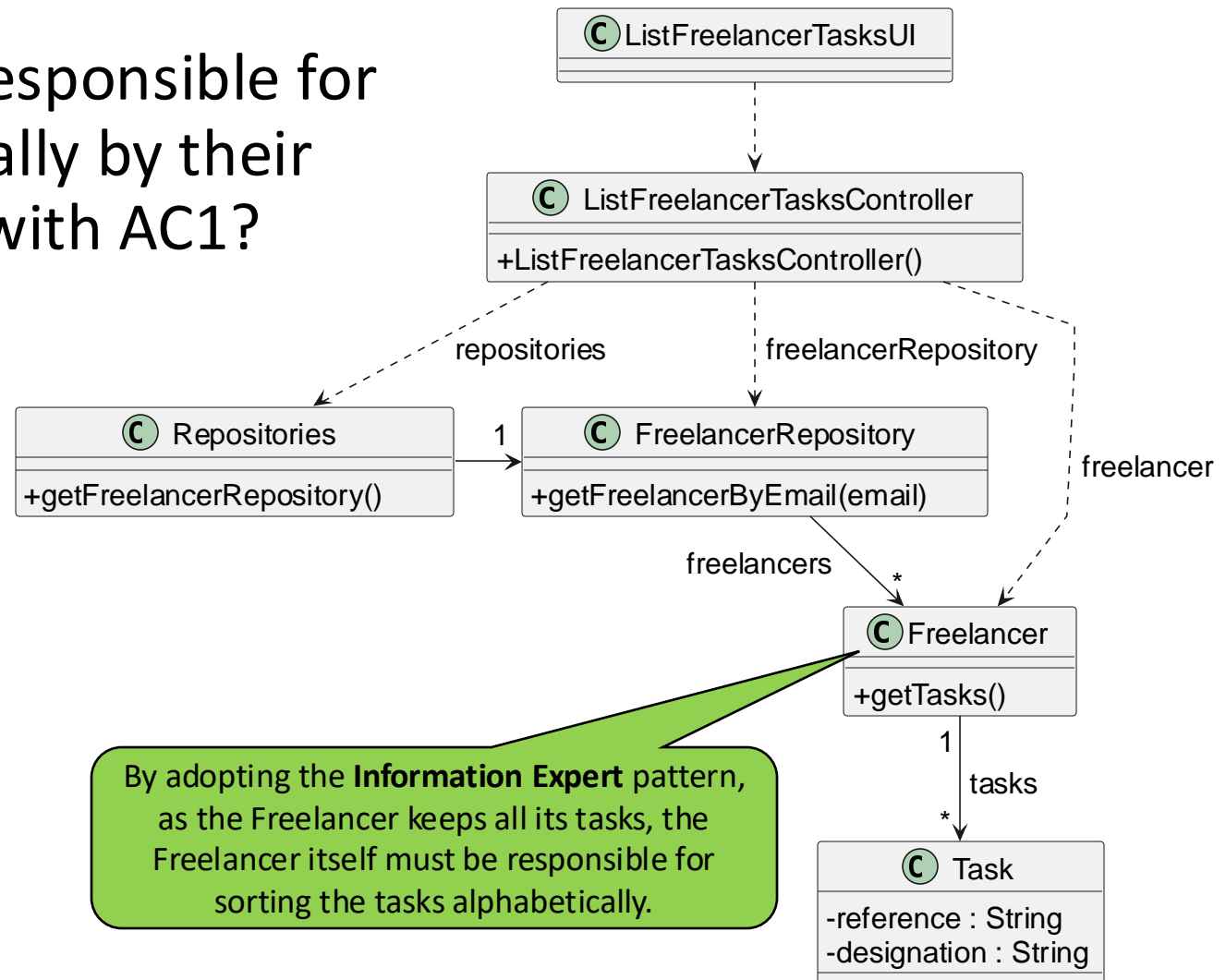
- ~~UI?~~
- ~~Controller?~~
- Organization?
- ~~Task?~~



Complying with AC1: Sorting Tasks (5/5)

- Which class should be responsible for sorting tasks alphabetically by their designation, to comply with AC1?

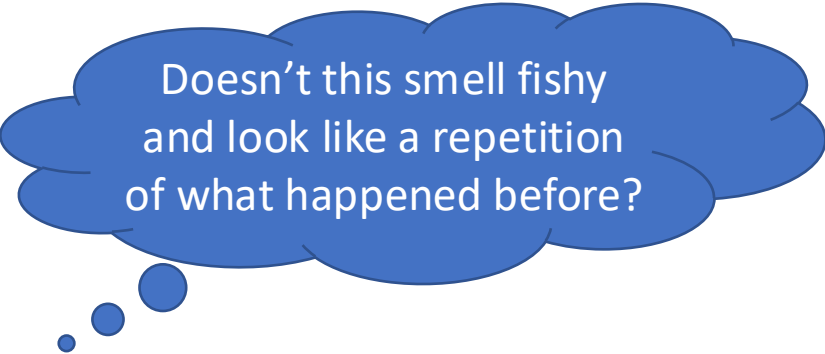
- ~~UI?~~
- ~~Controller?~~
- Organization? ✓
- ~~Task?~~



UC011 – AC1 Rationale (1/5)

- To comply with AC1 and according to Information Expert (IE), **the Freelancer class should be responsible for sorting the Tasks** because it holds the Tasks assigned to the Freelancer
- The Freelancer class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Freelancer and Task
 - Remember the “**Tell, Don’t Ask**” principle – how can the Freelancer have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class?
 - using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Freelancer class
 - The Freelancer must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

UC011 – AC1 Rationale (2/5)



Doesn't this smell fishy and look like a repetition of what happened before?

- To comply with AC1 and according to Information Expert (IE), **the Freelancer class should be responsible for sorting the Tasks** because it holds the Tasks assigned to the Freelancer
- The Freelancer class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Freelancer and Task
 - Remember the “**Tell, Don't Ask**” principle – how can the Freelancer have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class?
 - using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Freelancer class
 - The Freelancer must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

UC011 – AC1 Rationale (3/5)

Doesn't this smell fishy and look like a repetition of what happened before?

Haven't we increased, once again, the coupling of the Task class?

- To comply with AC1 and according to Information Exposure, **the Freelancer class should be responsible for sorting** the Tasks assigned to the Freelancer
- The Freelancer class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Freelancer and Task
 - Remember the “**Tell, Don't Ask**” principle – how can the Freelancer have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class?
 - using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Freelancer class
 - The Freelancer must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

UC011 – AC1 Rationale (4/5)

Doesn't this smell fishy and look like a repetition of what happened before?

- To comply with AC1 and according to Information Expectation, **the Freelancer class should be responsible for sorting the Tasks** holds the Tasks assigned to the Freelancer
- The Freelancer class should implement the **getSortedTasks()** method
- While this makes sense, as a side effect, this approach:
 - **Increases Coupling** between Freelancer and Task
 - Remember the “**Tell, Don't Ask**” principle – how can the Freelancer have access to the **designation** attribute for comparison without a **getDesignation()** method in the Task class? – using Comparable/Comparator interfaces
 - **Reduces Cohesion** of the Freelancer class
 - The Freelancer must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

Haven't we increased, once again, the coupling of the Task class?

How to avoid code repetition on different classes?

UC011 – AC1 Rationale (5/5)

Doesn't this smell fishy and look like a repetition of what happened before?

Haven't we increased, once again, the coupling of the Task class?

How to avoid code repetition on different classes?

- To comply with AC1 and according to Information Expert, **the Freelancer class should be responsible for sorting** the Tasks assigned to the Freelancer
- The Freelancer class should implement the **getSortedTasks()** method
- While this makes sense, it leads to the following problems:
 - **Increases Coupling** between classes
 - Remember the “Tell, Don't Ask” principle
 - **designator** attribute of the Task class – using Comparable/Comparable interface
 - **Reduces Cohesion** of the Freelancer class
 - The Freelancer must not only hold the created Tasks, but also have the responsibility for sorting those Tasks

HOW TO SOLVE THIS PROBLEM?

Promotion of Collections to Software Classes

How to solve this problem? (1/2)

- Why not have both Organization and Freelancer classes **delegate the sorting responsibility to another class?**
- What responsibilities should that class have?
 - That class should be responsible for handling operations over a particular collection
 - In this case, the class would be handling responsibilities related to the tasks collection

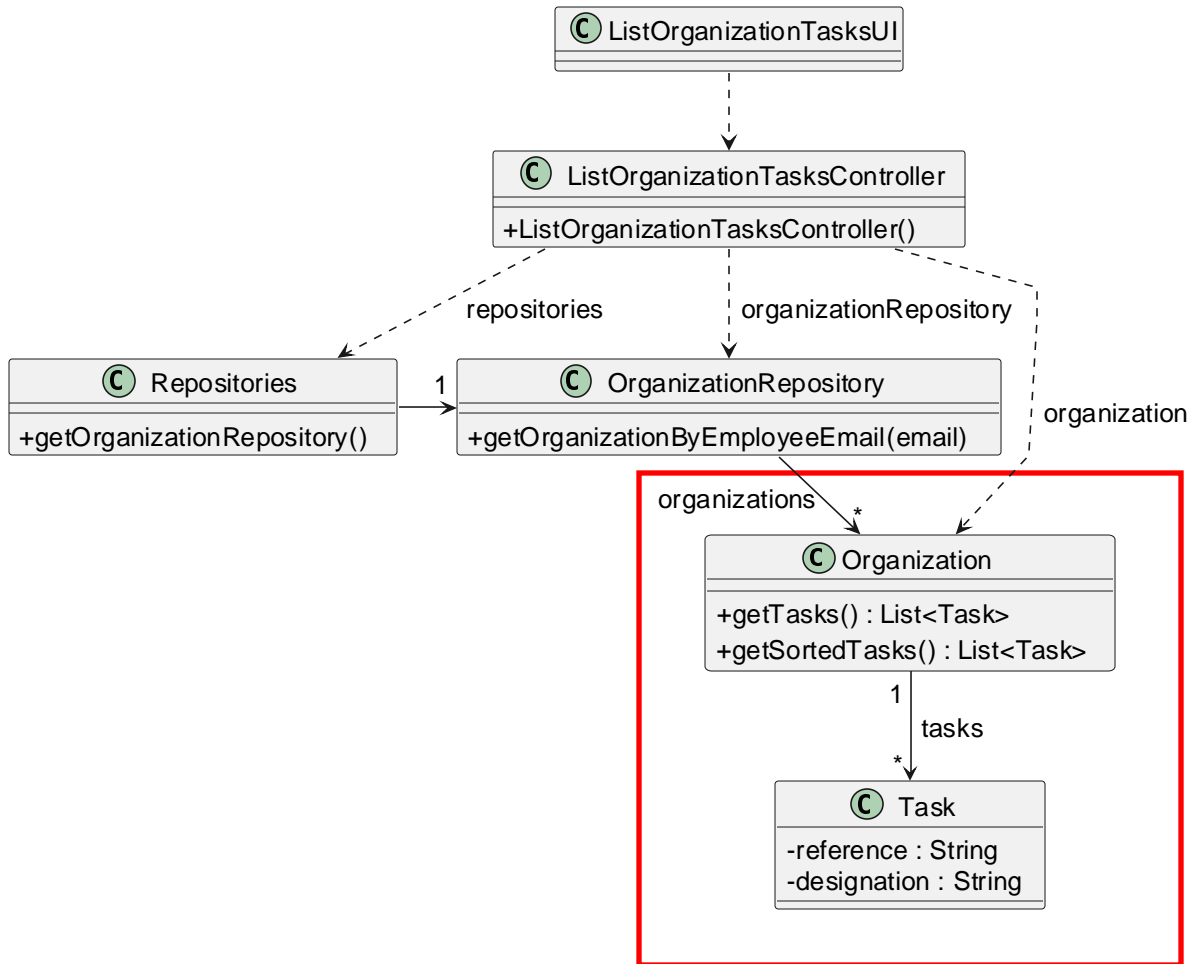
How to solve this problem? (2/2)

- **Keep using a Collection** (e.g. Java ArrayList) **if the association requires typical collection methods only** (e.g. add, get, set, delete, iterator)
- **Promote the Collection to a New Class if**, in addition to the collection methods, **specific methods are required**
 - E.g.: a sorting method like `getSortedTasks()`

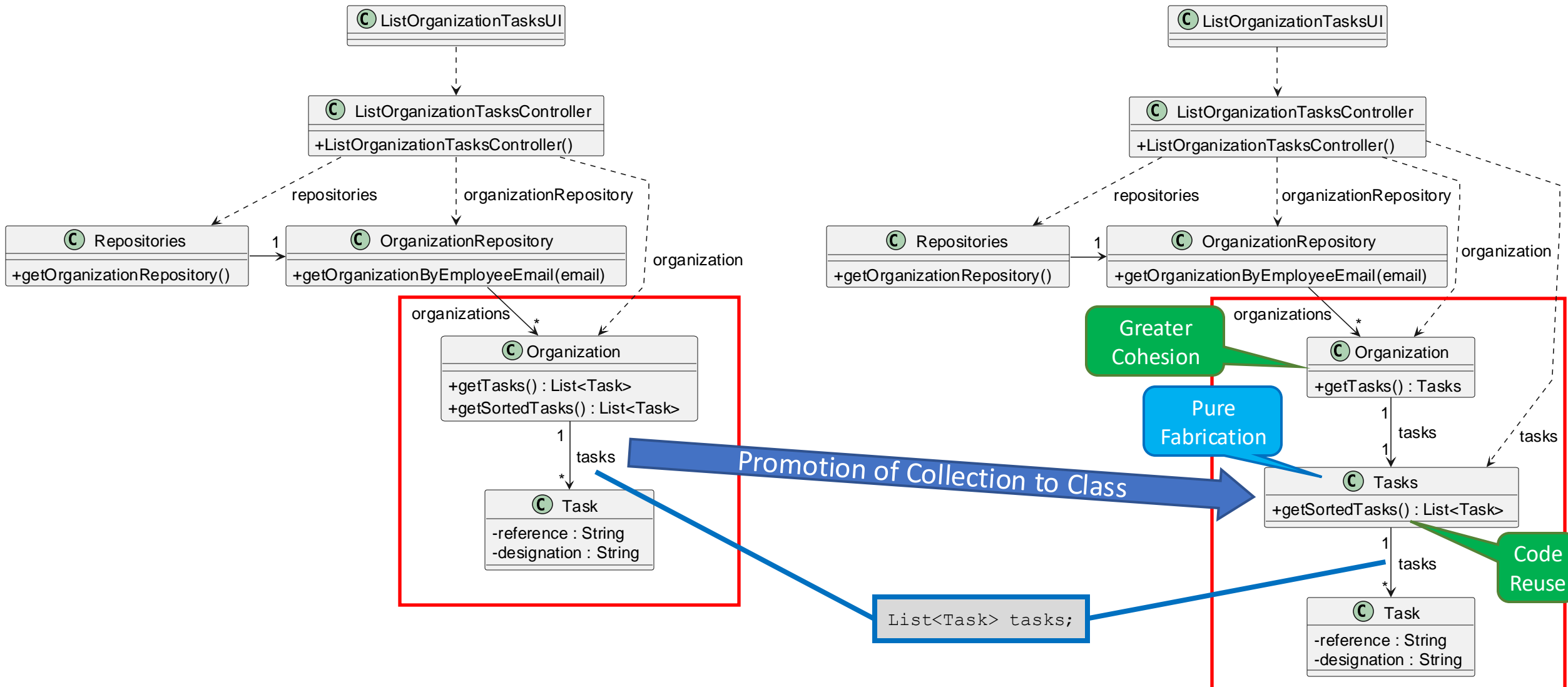
How to name the new software class?

- When the New Class has a more **“global” scope** in the system, it might be named using as a suffix like, for example: Store, Container or **Repository**
 - E.g.: OrganizationStore, OrganizationContainer, **OrganizationRepository**
- When the New Class has a more **“local” scope** (i.e. restricted to a given instance), it might be named using a suffix like, for example: Collection, List (when of type List) or simply the name of the class **in plural**
 - E.g.: TaskCollection, TaskList, **Tasks**

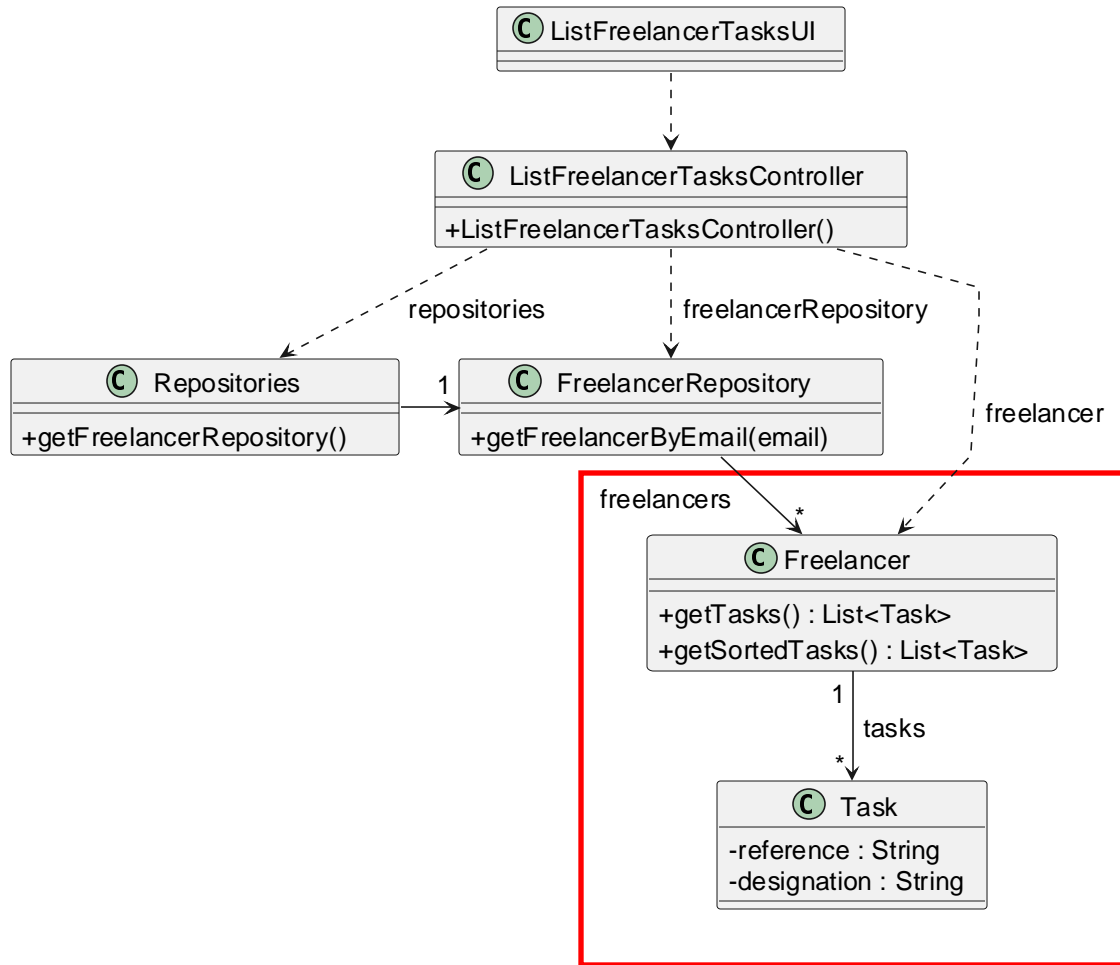
Changes to the UC010 design (1/2)



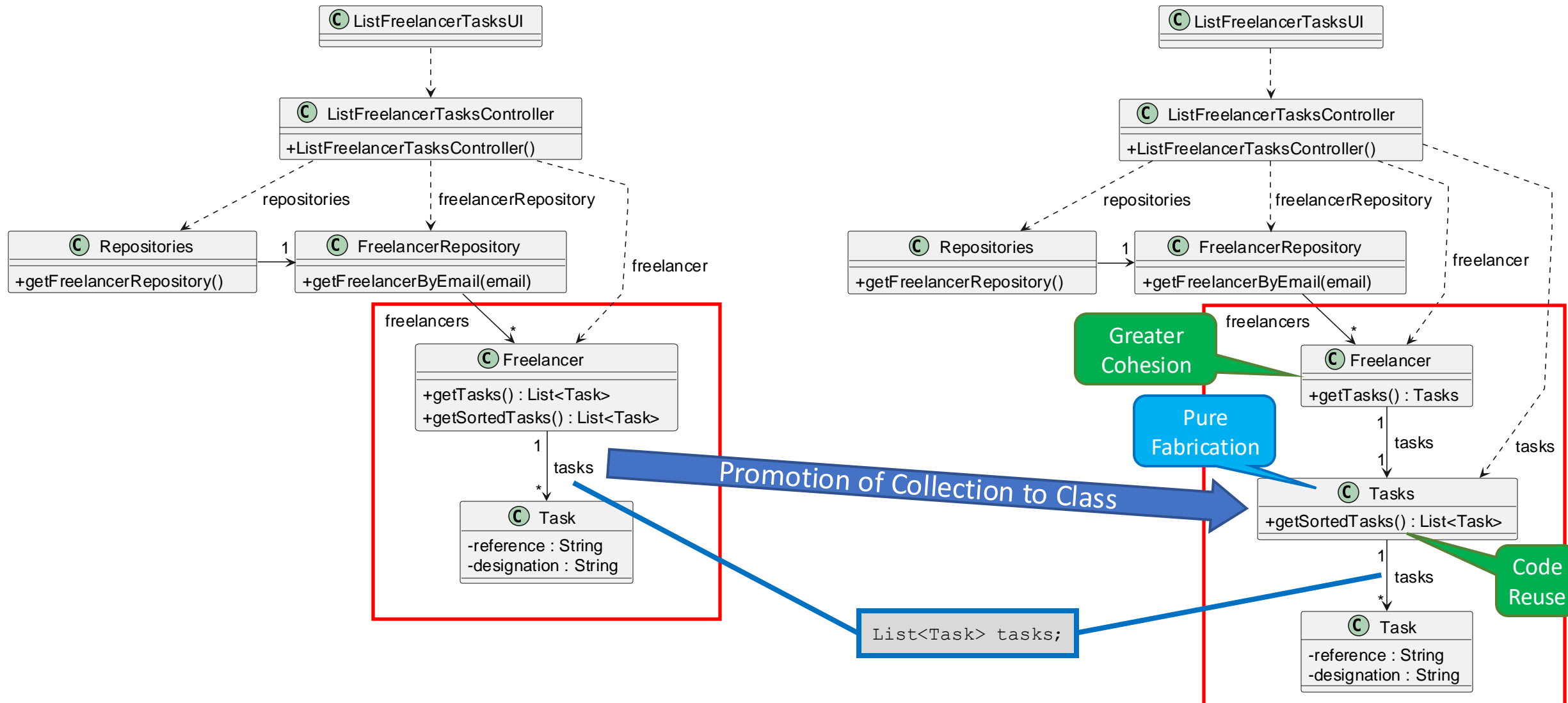
Changes to the UC010 design (2/2)



Changes to the UC011 design (1/2)



Changes to the UC011 design (2/2)



Summary

- High Cohesion and Low Coupling must be considered while designing
 - Not only to promote collections to software classes
 - Pure Fabrication
 - But also, on other scenarios (e.g.: filter/sort a collection by some criteria) to evaluate plausible alternatives
 - Tell, Don't Ask
 - Information Expert
- The Domain Model is used to inspire the creation of software classes (the Design Model), but the opposite is not true

References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- Booch, G. 1994. Object-Oriented Analysis and Design. Redwood City, CA.: Benjamin/Cummings.