

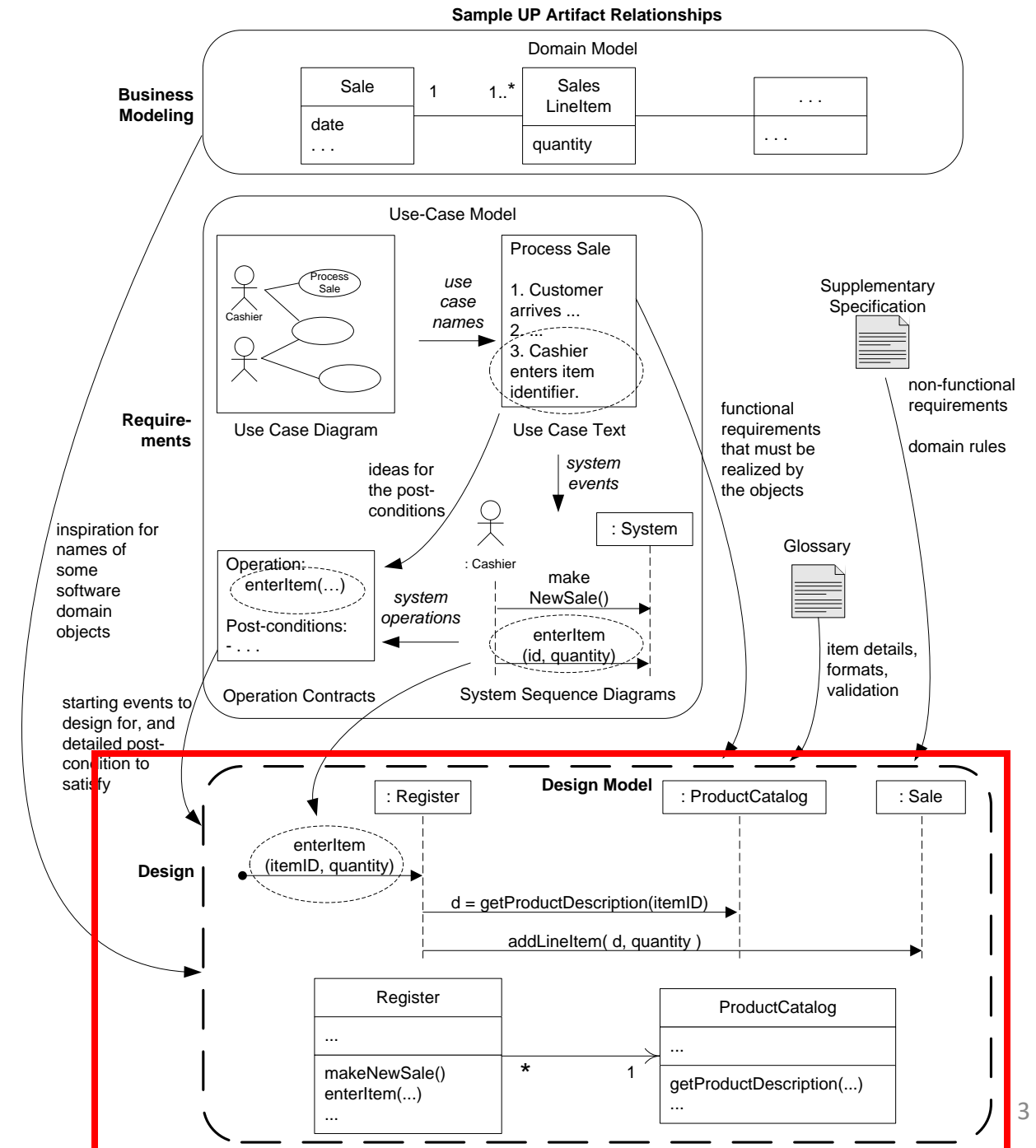
# User Scenario Realization



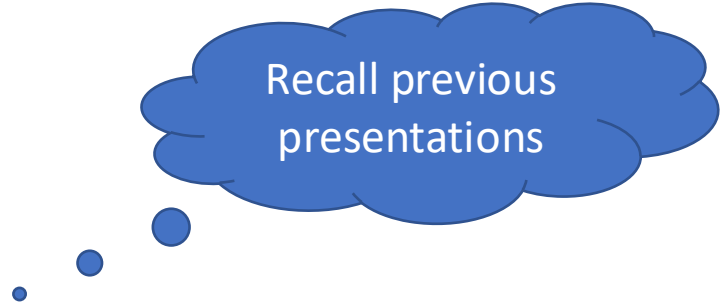
# Topics

- OO Design
- User Scenario Realization through an example
- GRASP – General Responsibility Assignment Software Patterns
  - Pure Fabrication
  - Controller
  - Creator
  - Information Expert

# Artifacts Overview



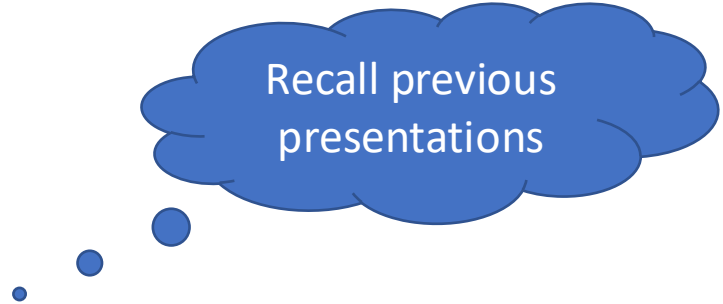
# OO Design



Recall previous presentations

- **When?**
  - After identifying the requirements and specifying the domain model
- **How? By:**
  - Promoting conceptual classes to software classes
  - Adding methods to software classes
  - Defining messages between classes/objects

# Recommended Method




Recall previous presentations

- Driven by the functional requirements, strongly supported by the user scenarios (either US or UC) and the Domain Model
- Therefore, for each US/UC the following artifacts are created:
  - **Rationale of responsibilities assignment** according to
    - **GRASP – General Responsibility Assignment Software Patterns (or Principles)**
    - SOLID
    - Other patterns (e.g. GoF) and best practices
  - **Sequence Diagram** highlighting interactions between classes/objects
  - **Partial Class Diagram**
- The **Complete Class Diagram** results from the partial CD of each user scenario realization

# User Scenario Realization

UC006 – Create a Task

# User Scenario Realization



Recall previous presentations

- A **User Scenario** is an instance of a Use Case, i.e. one path through the Use Case
- Usually, it only covers the **Main Success Scenario** (aka *happy path*) between the user and the system
  - If relevant, other possible flows (leading to success and/or handling errors) might also be considered and realized

# UC006 – Create a Task

*Platform for  
Outsourcing Tasks*

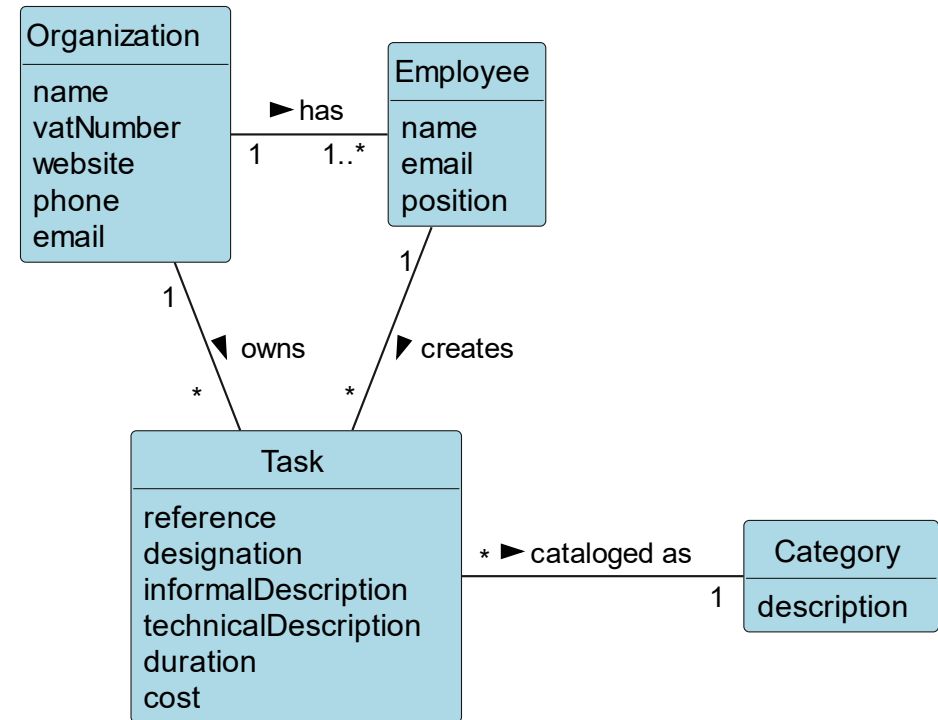
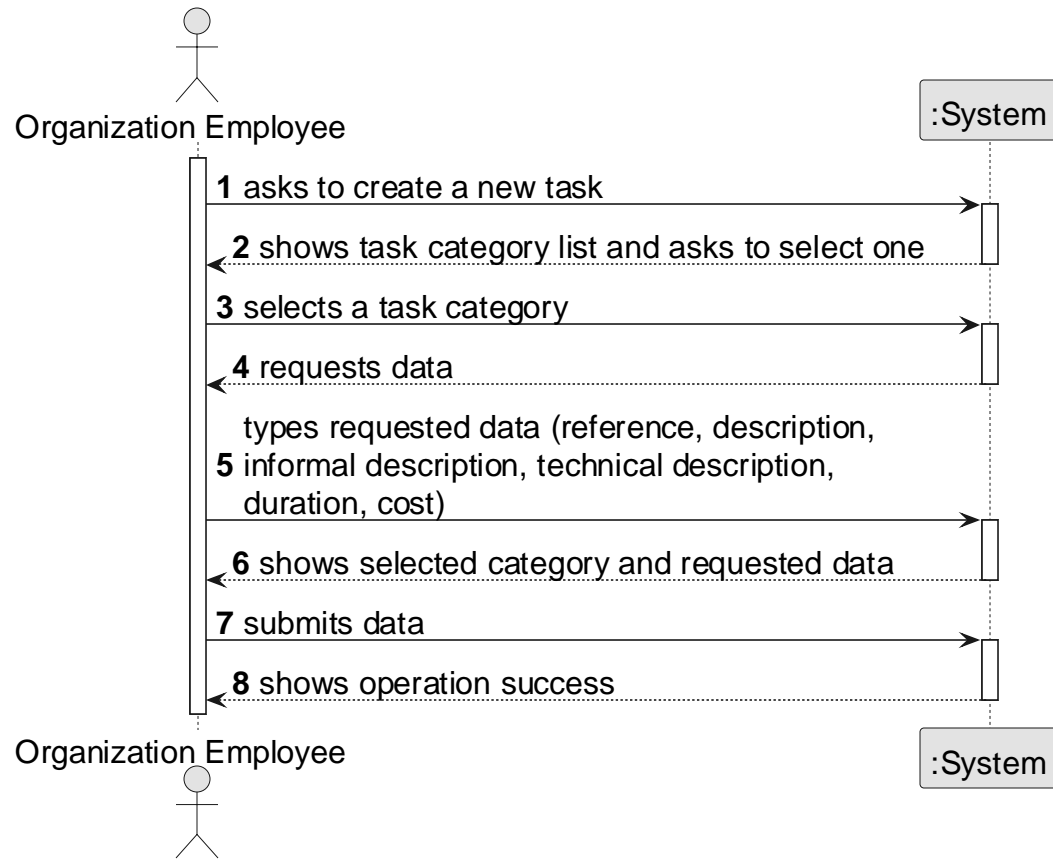
- As an Organization Employee, I want to create a new task in order to be further published.
  - AC1: All required fields must be filled in.
  - AC2: Task reference must have at least 5 alphanumeric characters.
  - AC3: When creating a task with an existing reference, the system must reject such operation ~~and the user must be able to modify the typed reference.~~
- For demonstration purposes, AC3 is partially addressed, only covering the Main Success Scenario.



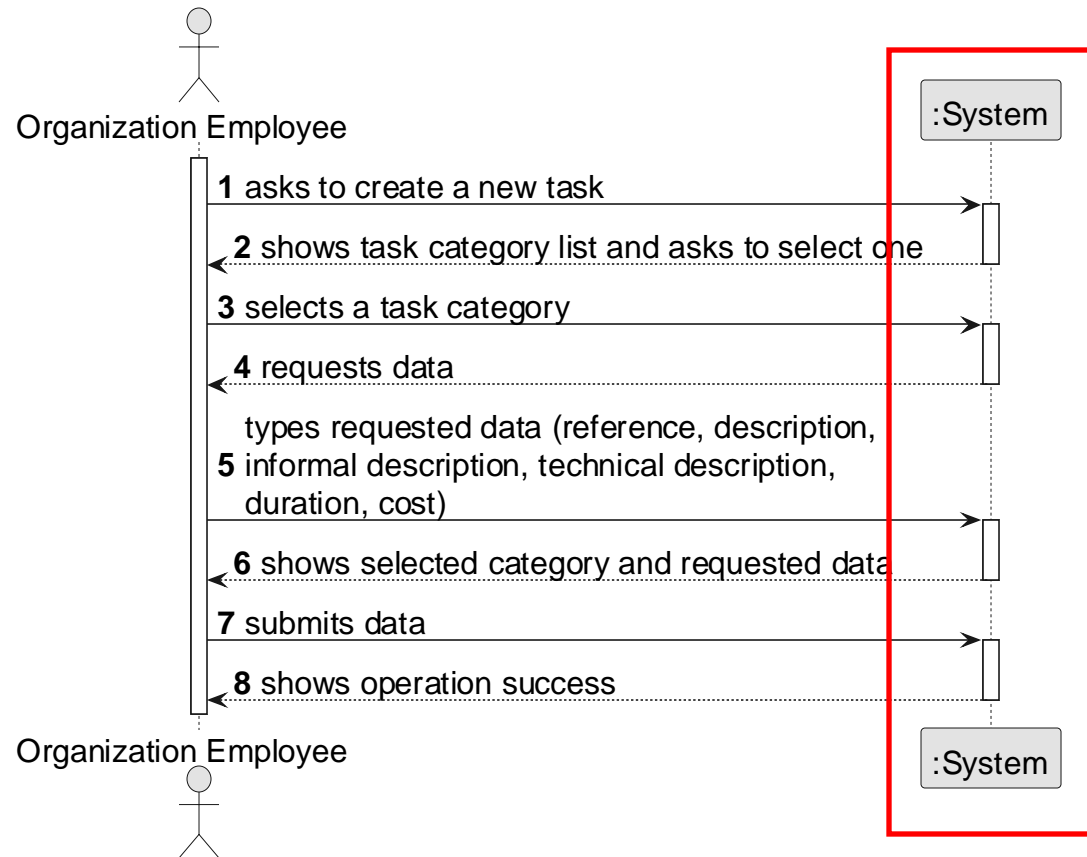
# UC006 – Create a Task (previous knowledge)

Platform for  
Outsourcing Tasks

- System Sequence Diagram (SSD)
- Relevant Domain Model excerpt

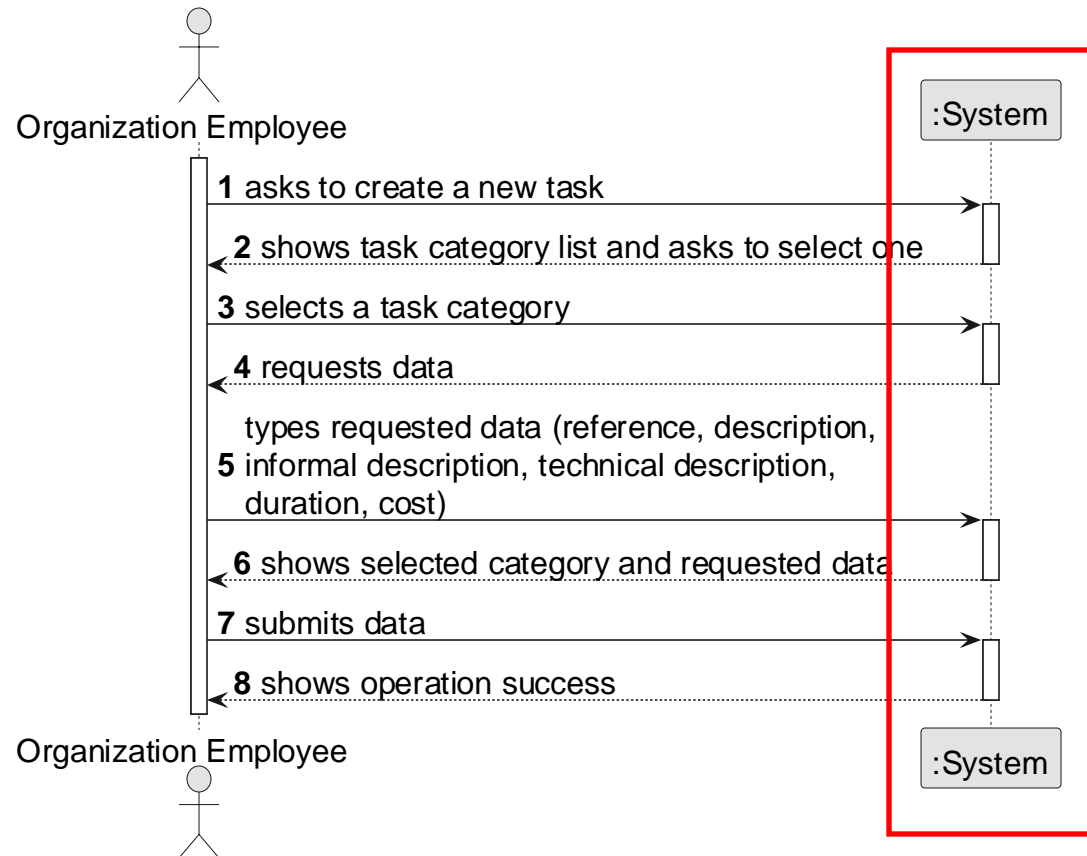


# Design from the SSD



- What happens in the **System**?
- **System** is made up of several classes/objects with different responsibilities

# Responsibility-Driven Design (RDD)



- RDD is a metaphor to help with the OO software design process
- Objects have responsibilities, obligations and behaviors depending on the role they play in the system
- Examples of responsibilities:
  - **Who** receives the actor's actions?
  - **Who** processes the actions?
  - **Who** coordinates the US/UC?
  - **Who** does "what"?

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	<b>Question(s)</b> – What does this step/message involve (creation, change, registration, query)?		
Step/Msg 2: shows task category list and asks to select one	<b>Answer(s)</b> – Specifies which classes/objects are related to the issue(s) of interest.		
Step/Msg 3: selects a task category	<b>Justification(s)</b> – Explains the reasons why the classes/objects were chosen (based on software design patterns).		
...			

# Let's take a step-by-step approach

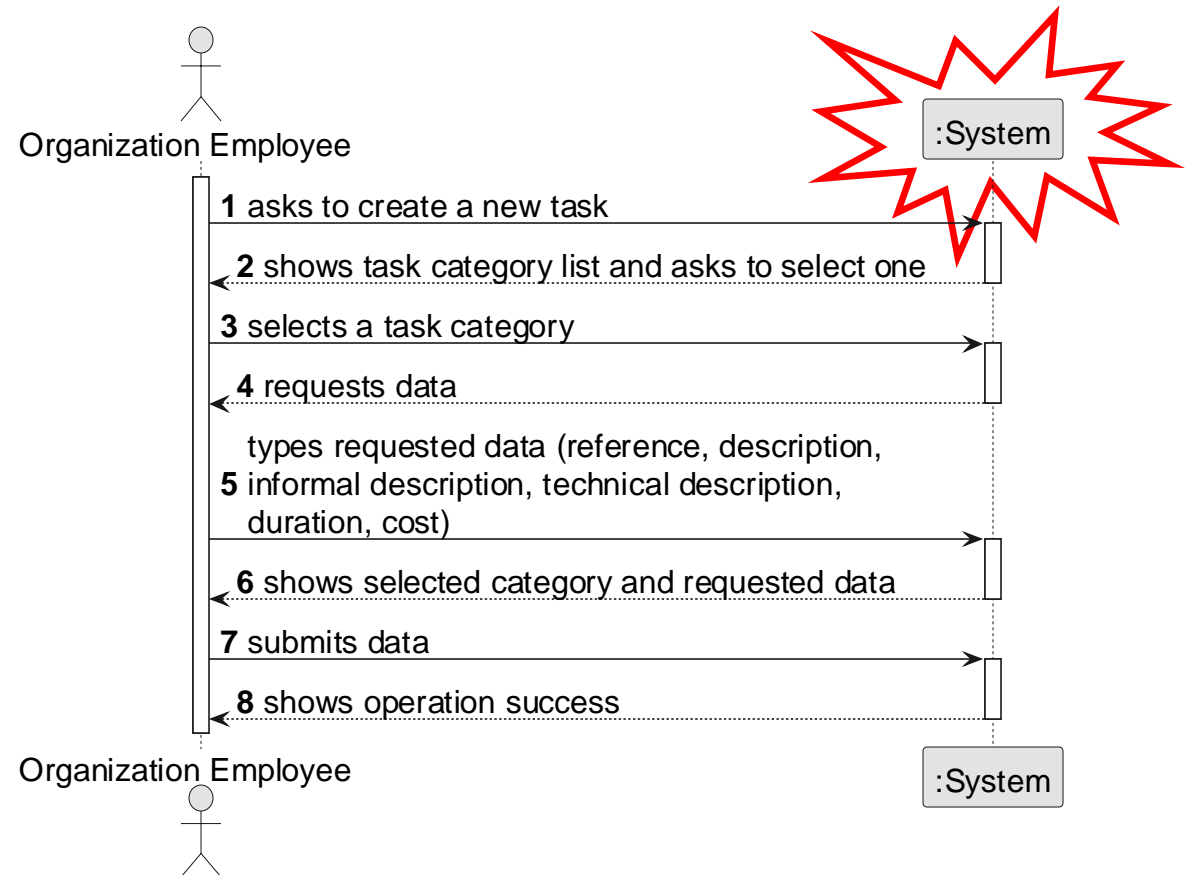
(other approaches are valid as well...)

# GRASP: Pure Fabrication

Class CreateTaskUI for handling the User Interface (UI)

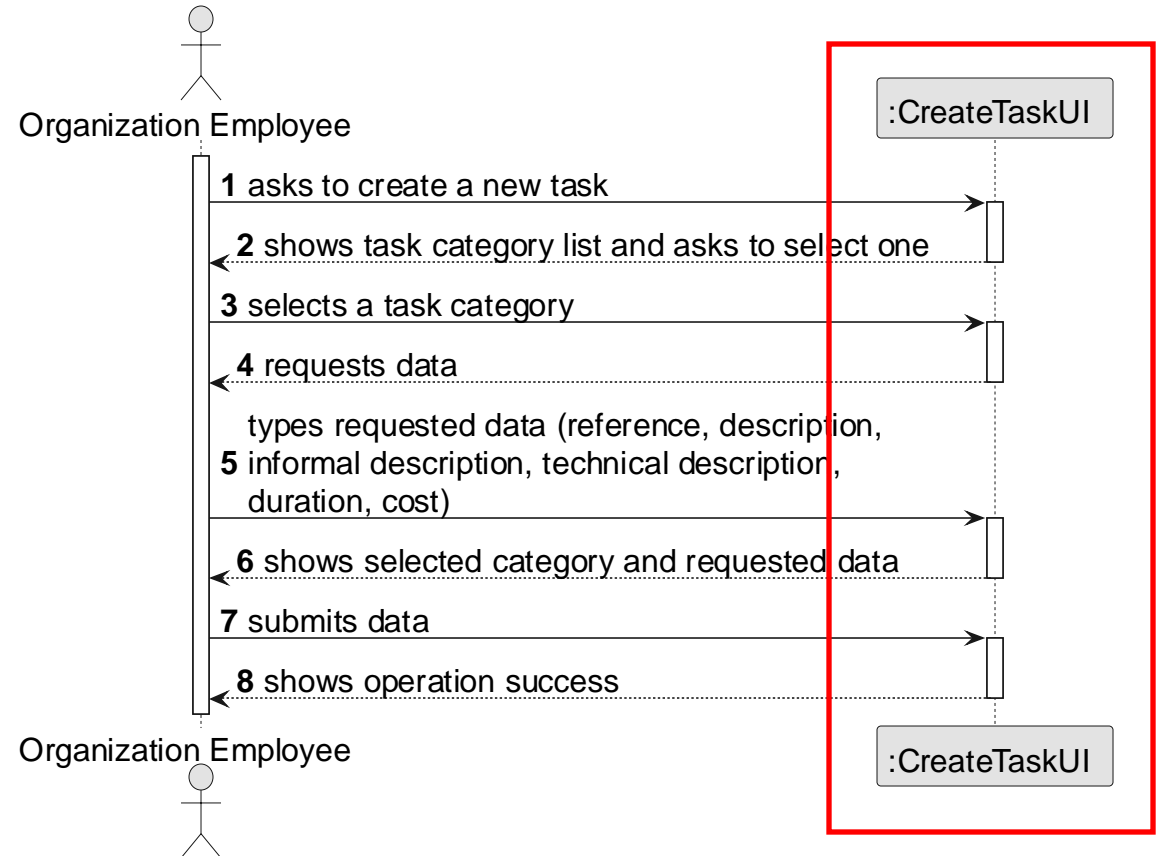
# Interacting with the Actor (1/2)

- Who is responsible for interacting with the Actor?
  - A specific UI class for each user scenario is used
  - This class is obtained by **Pure Fabrication**
  - Class named *<UCName>UI*



# Interacting with the Actor (2/2)

- Who is responsible for interacting with the Actor?
  - A specific UI class for each user scenario is used
  - This class is obtained by **Pure Fabrication**
  - Class named *<UCName>UI*





# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... instantiating the class that handles the UI?	?	?
	... obtaining the task categories list?	?	?
Step/Msg 2: shows task category list and asks to select one	... displaying the task categories?	?	?
Step/Msg 3: selects a task category	... validating selected data? ... temporarily keeping the selected task category?	?	?

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... instantiating the class that handles the UI?	CreateTaskUI	Pure Fabrication
	... obtaining the task categories list?	?	?
Step/Msg 2: shows task category list and asks to select one	... displaying the task categories?	CreateTaskUI	Pure Fabrication
Step/Msg 3: selects a task category	... validating selected data? ... temporarily keeping the selected task category?	CreateTaskUI	Pure Fabrication

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 4: requests data	... displaying the form for the actor to input data?	?	?
Step/Msg 5: types requested data	... validating input data? ... temporarily keeping input data?	?	?
Step/Msg 6: shows selected category and requested data	... displaying all the information before submitting?	?	?

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 4: requests data	... displaying the form for the actor to input data?	CreateTaskUI	Pure Fabrication
Step/Msg 5: types requested data	... validating input data? ... temporarily keeping input data?	CreateTaskUI	Pure Fabrication
Step/Msg 6: shows selected category and requested data	... displaying all the information before submitting?	CreateTaskUI	Pure Fabrication

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 7: submits data	... creating the Task object?	?	?
	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?
Step/Msg 8: shows operation success	... informing operation success?	?	?

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 7: submits data	... creating the Task object?	?	?
	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?
Step/Msg 8: shows operation success	... informing operation success?	CreateTaskUI	Pure Fabrication

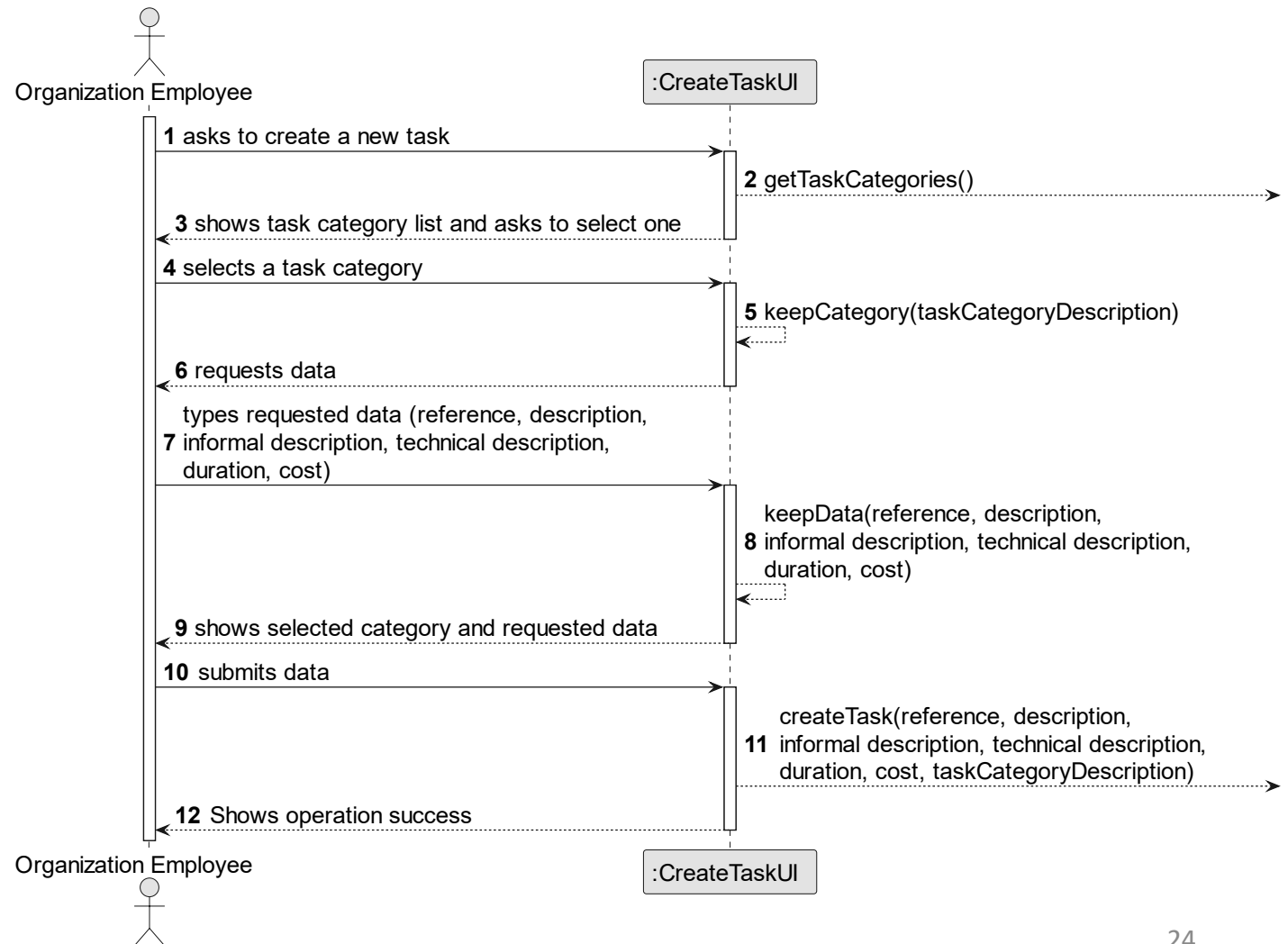
# Rationale for Responsibilities Assignment:

## UC006 – Create a Task → what is still unknown?

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... creating the Task Object?	?	?
	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?

# CreateTaskUI Responsibilities

- Everything that the CreateTaskUI can do by itself is mapped to internal methods
- Everything else, any open questions or unknowns, are forwarded to the System





# CreateTaskUI Requests

- Who is responsible for answering CreateTaskUI requests?



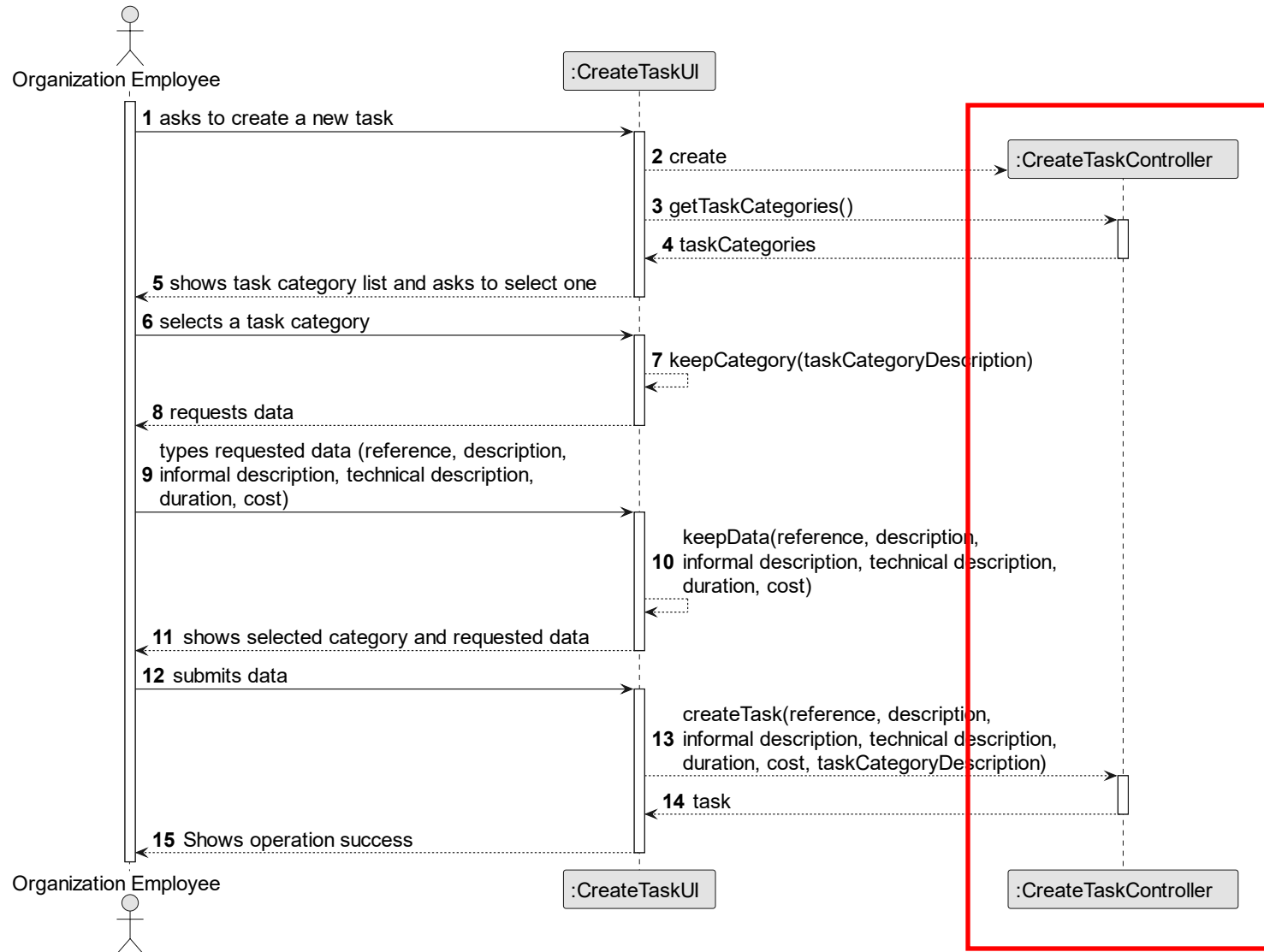
# Grasp: Controller

Class CreateTaskController for creating a boundary between the User Interface Layer and the Domain Layer

# Who coordinates the User Scenario?

- Following GRASP, the Controller pattern is adopted
- Class named *<UCName>Controller*
  - Creates a boundary between the User Interface (UI) and all other classes, orchestrating/delegating responsibilities
  - Responsible for coordinating and distributing actions performed in the UI Layer to the Domain Layer (the rest of the system)
- There are several Controller classes, one for each user scenario. E.g.:
  - CreateTaskController
  - CreateTaskCategoryController
  - ...

# UPskill Pattern: UI + Controller classes



# Controller Responsibilities

- Coordinates/controls the flow of the user scenario
  - E.g.: Ensures that “Step X” is not performed before “Step Y”
- It should not do (data) processing
  - E.g.: Calculus or validations related to data correction
- It delegates tasks/processing to domain objects
  - E.g.: Asks one (or more) domain object(s) to validate data
  - E.g.: Asks a domain object to create another object
- Serves as an intermediary between the UI layer and the Domain layer
  - **Direct communication between UI classes and domain classes should be avoided**
  - The communication must be made through the Controller class

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task → what is still unknown?

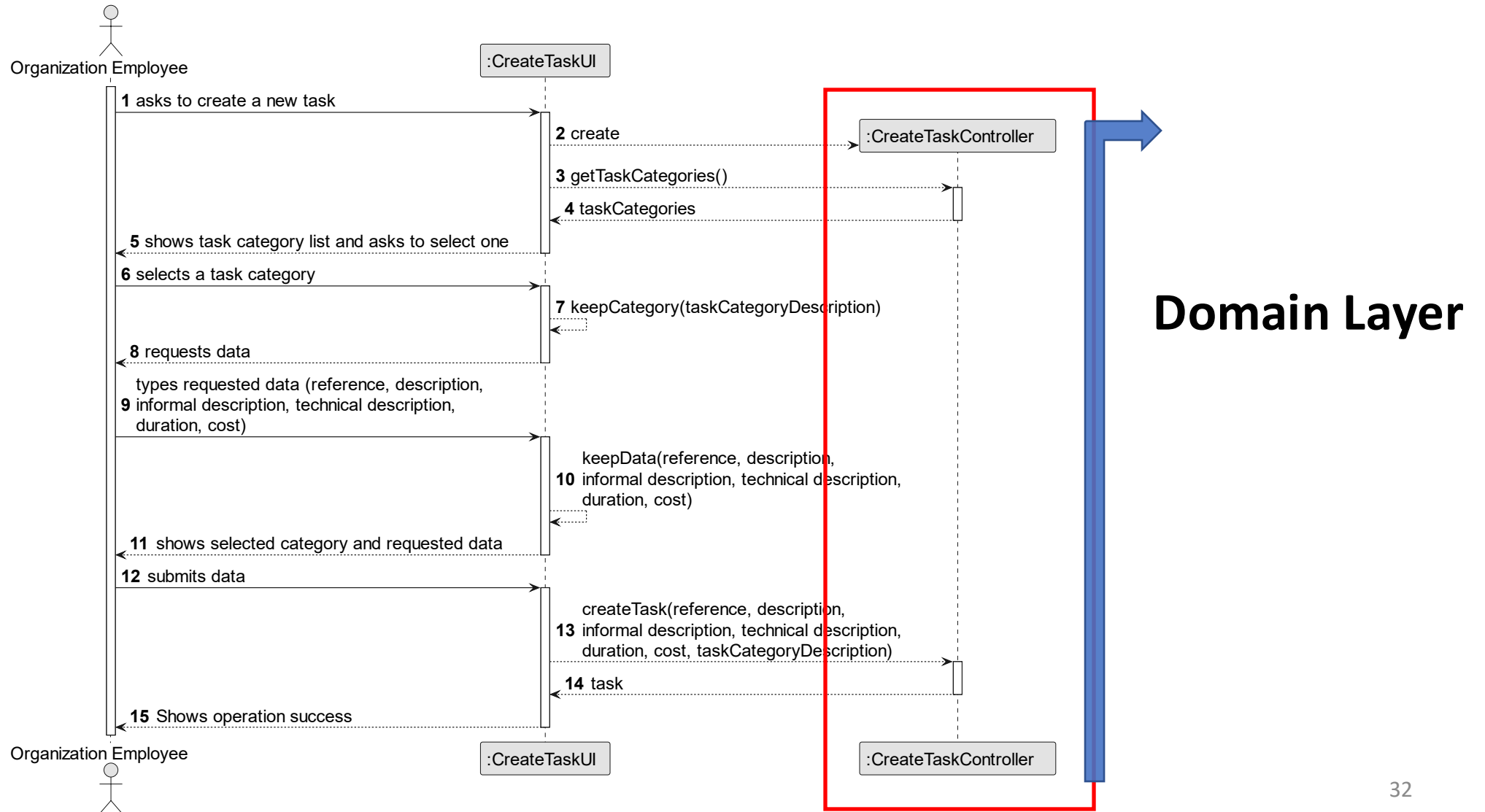
SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... creating the Task Object?	?	?
	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?

# Rationale for Responsibilities Assignment:

## UC006 – Create a Task → what is still unknown?

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining categories list?		?
Step/Msg 7: submits data	... creating		?
	... validating the data locally (mandatory data)?		?
	... adding to a collection and globally validating duplicate records?	?	?

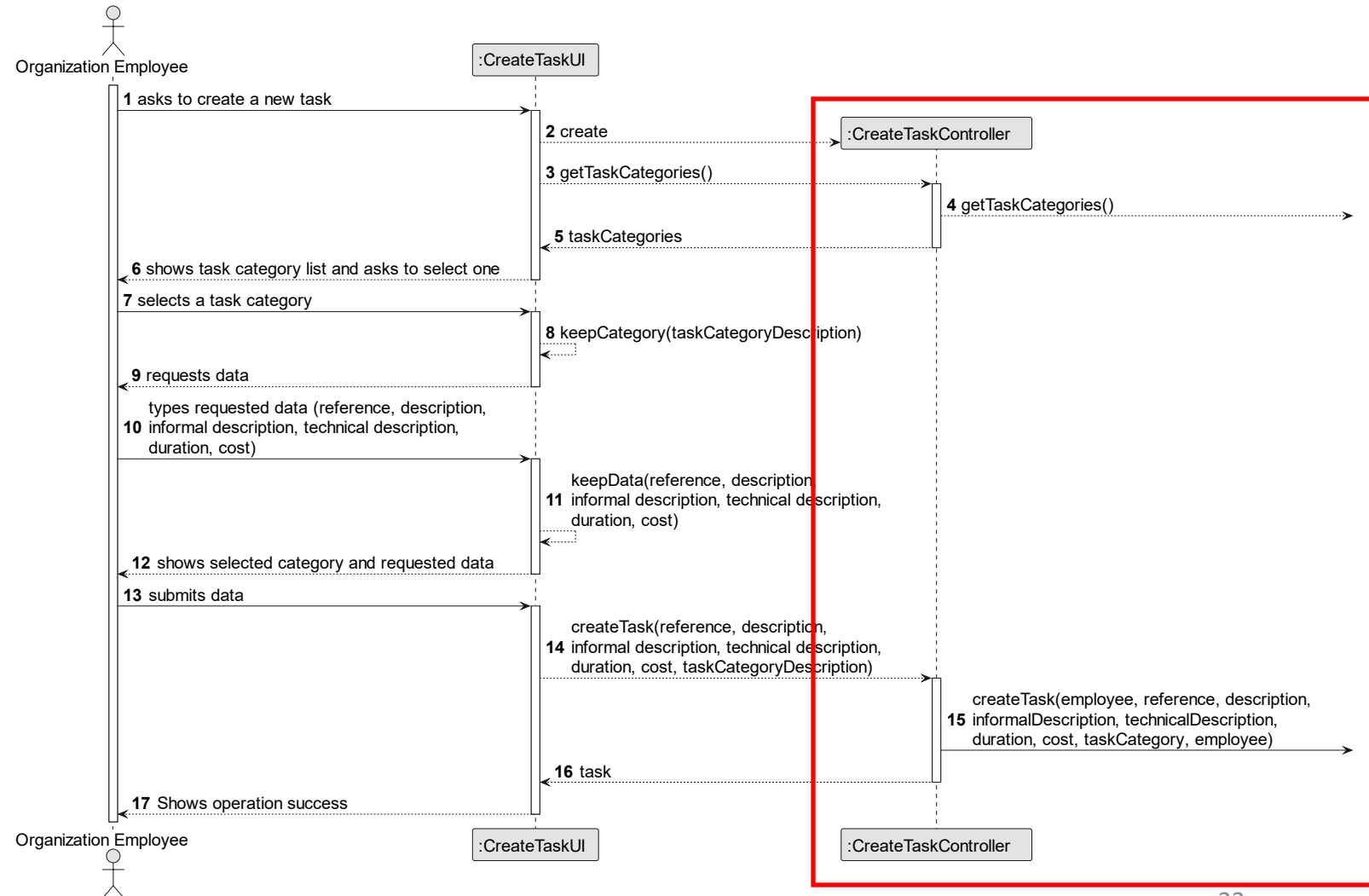
# Controller as coordinator/delegator





# CreateTaskController Responsibilities

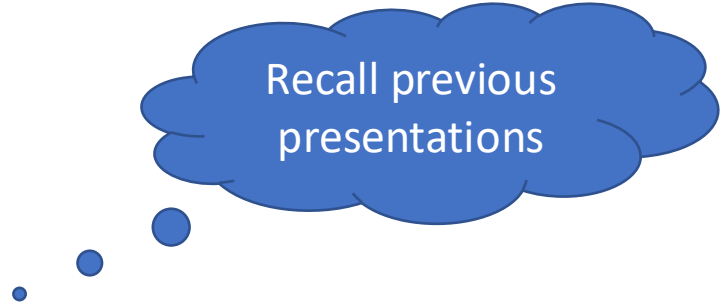
- The controller can't do anything
- The controller just coordinates the flow and delegates processing to other classes/objects



# GRASP: Creator

Class Organization as the creator for Task objects

# The Creator Pattern



Recall previous presentations

- **Problem**

- Who should be responsible for **creating a new object** of a given class?

- **Solution**

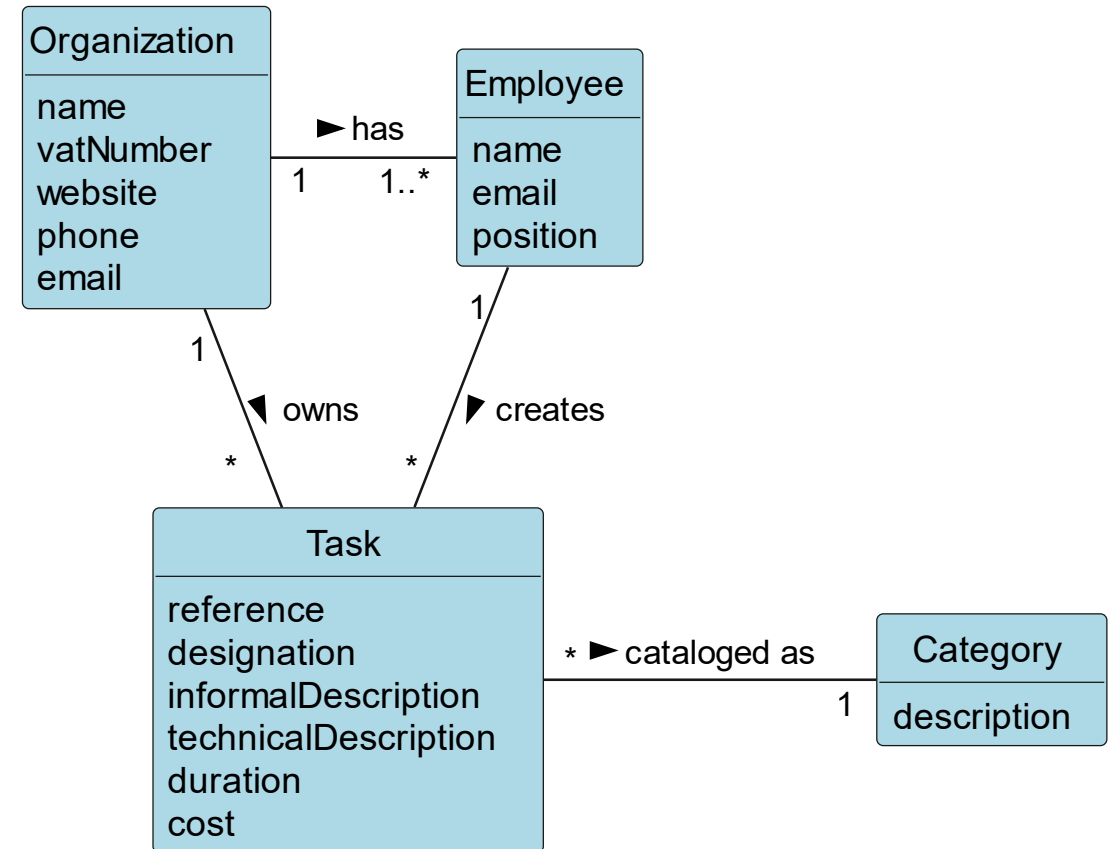
- Assign to class B the responsibility for creating instances of class A under the following conditions (in order of preference):
    - 1) B contains or aggregates instances of A
    - 2) B records instances of A
    - 3) B closely uses A
    - 4) B has the data for initializing A

# Who creates Tasks? (1/3)

- Look at the Domain Model...
- Who **contains or aggregates** Task instances?
- Which class should be responsible for creating a new Task object?
- Think about the **Creator Rationale**. Which class is more suitable?

**[Creator Rationale]** B is responsible for creating A if:

- 1) B contains or aggregates instances of A
- 2) B records instances of A
- 3) B closely uses A
- 4) B has the data for initializing A

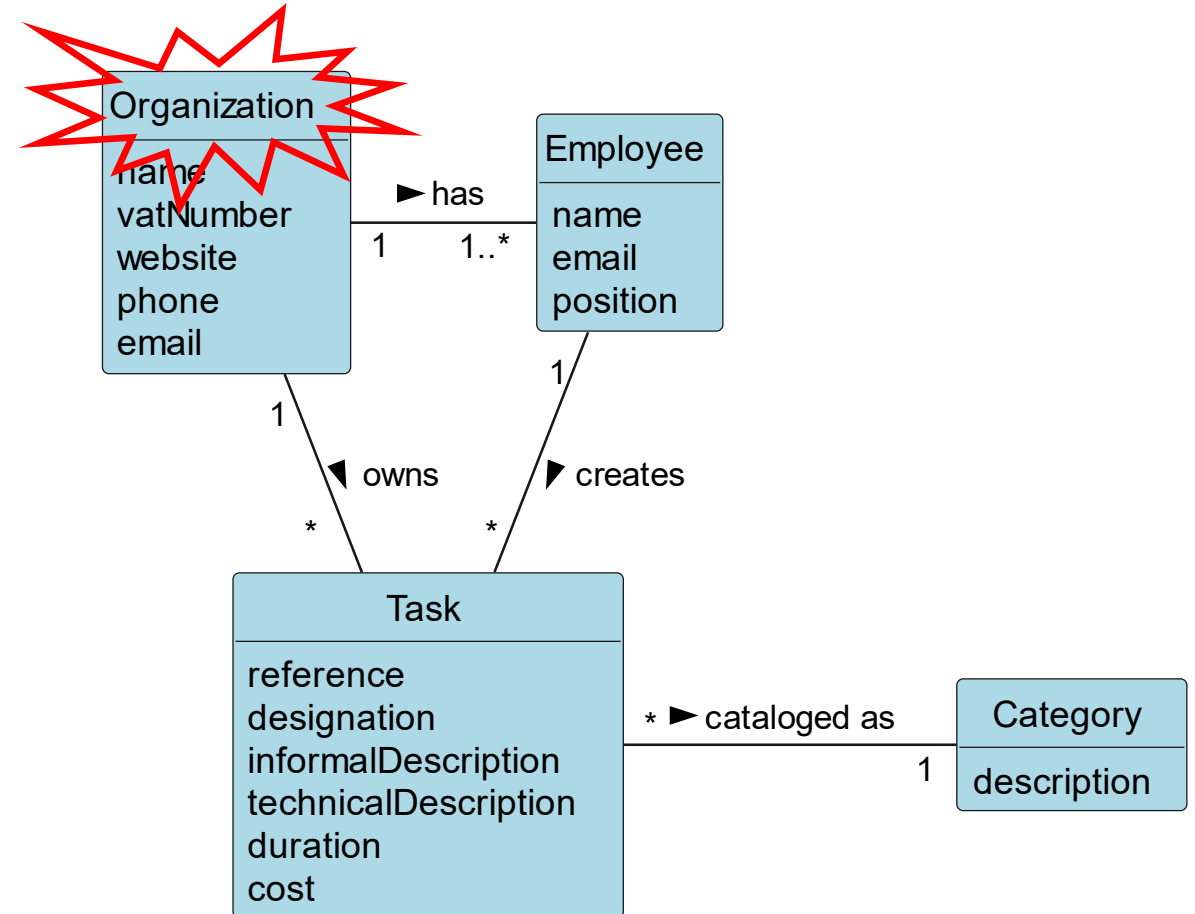


# Who creates Tasks? (2/3)

- Look at the Domain Model...
- Who **contains or aggregates** Task instances?
- Which class should be responsible for creating a new Task object?
- Think about the **Creator Rationale**. Which class is more suitable?

**[Creator Rationale]** B is responsible for creating A if:

- 1) B contains or aggregates instances of A
- 2) B records instances of A
- 3) ~~B closely uses A~~
- 4) ~~B has the data for initializing A~~



# Who creates Tasks? (3/3)

- By applying the Creator pattern, the **Organization** class is a good candidate for being responsible for creating Task instances, as it is **the class that contains or aggregates Task instances**
- Therefore, the software class **Organization** should have a method called, for example, **createTask**, to play such responsibility

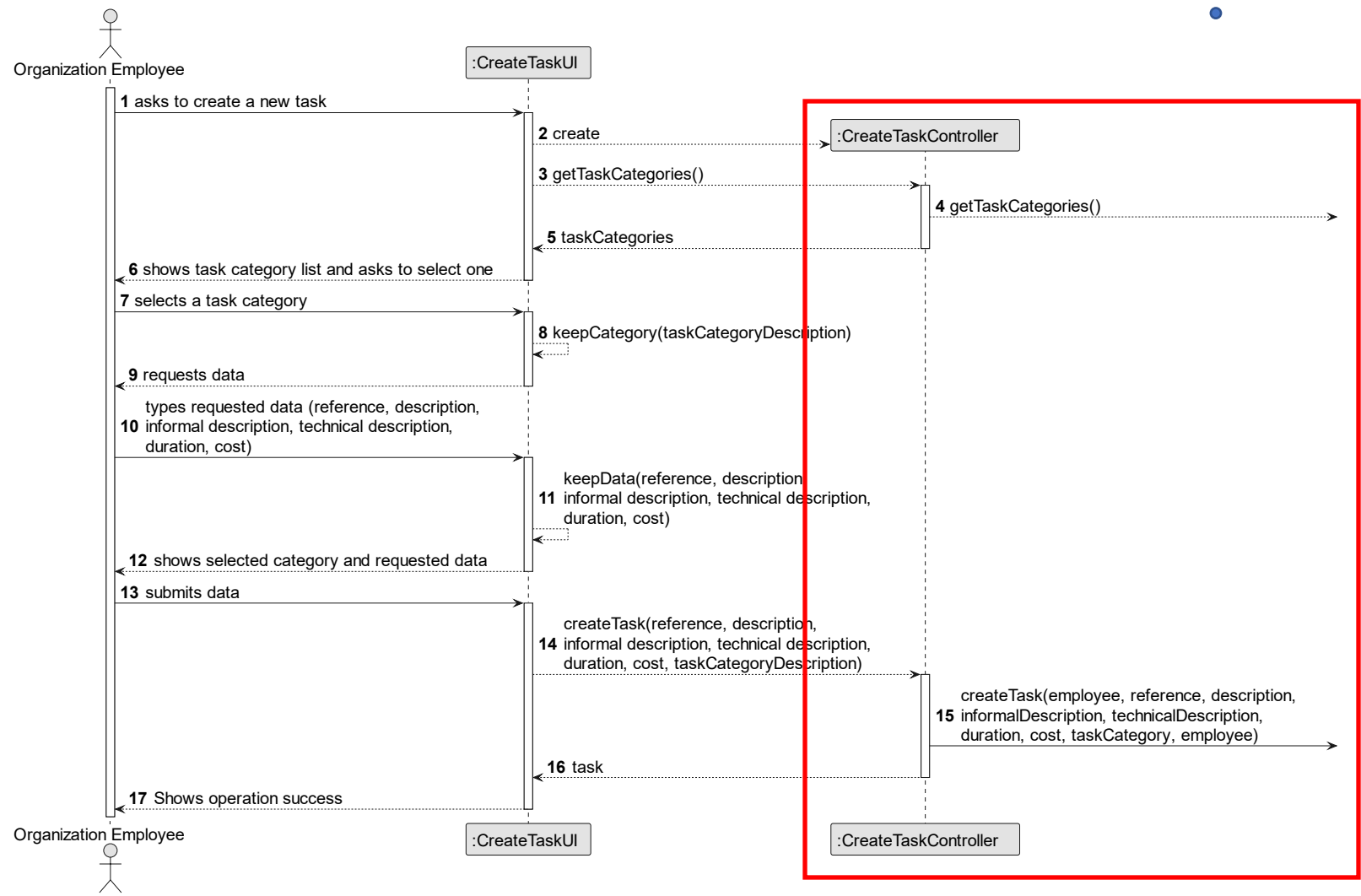
# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... creating the Task Object?	Organization	Creator R: 1, 2
	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?

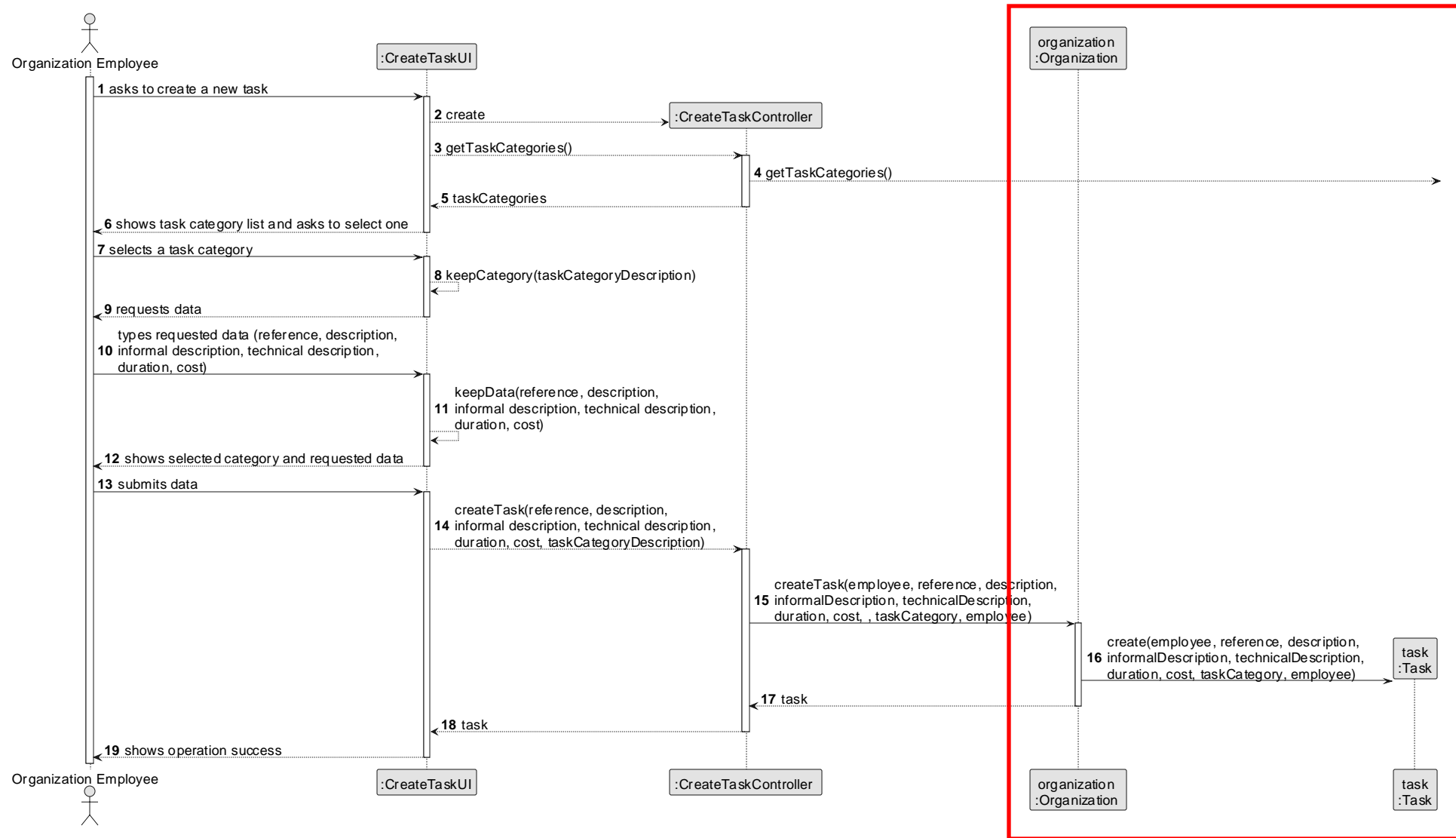
Recall previous presentations

# UPskill Pattern: UI + Controller responsibilities





# Organization responsibilities



# Rationale for Responsibilities Assignment:

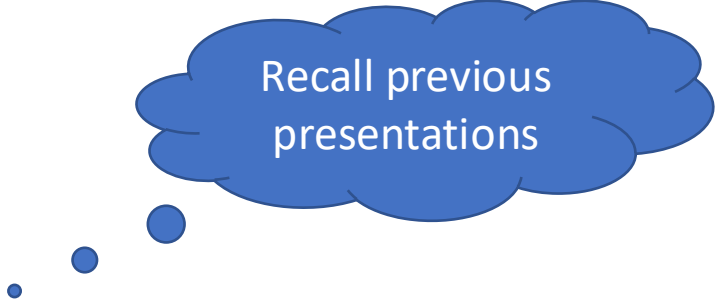
## UC006 – Create a Task → what is still unknown?

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... validating the data locally (mandatory data)?	?	?
	... adding to a collection and globally validating duplicate records?	?	?

# GRASP: Information Expert

Class Task as “local” validator

# Information Expert



Recall previous presentations

- **Problem**

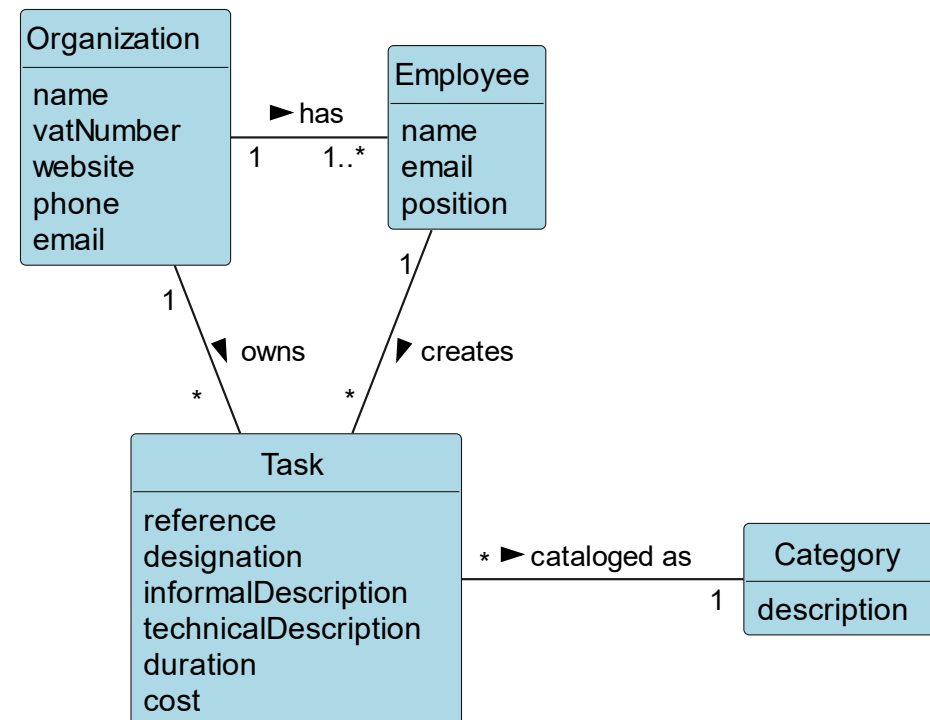
- What is the **general principle** for assigning responsibilities to objects?

- **Solution**

- Assign the responsibility to the “**information expert**”
  - I.e., assign to the class that contains the information needed to fulfill that responsibility
  - Which class?
    - Get inspired by the **Domain Model**

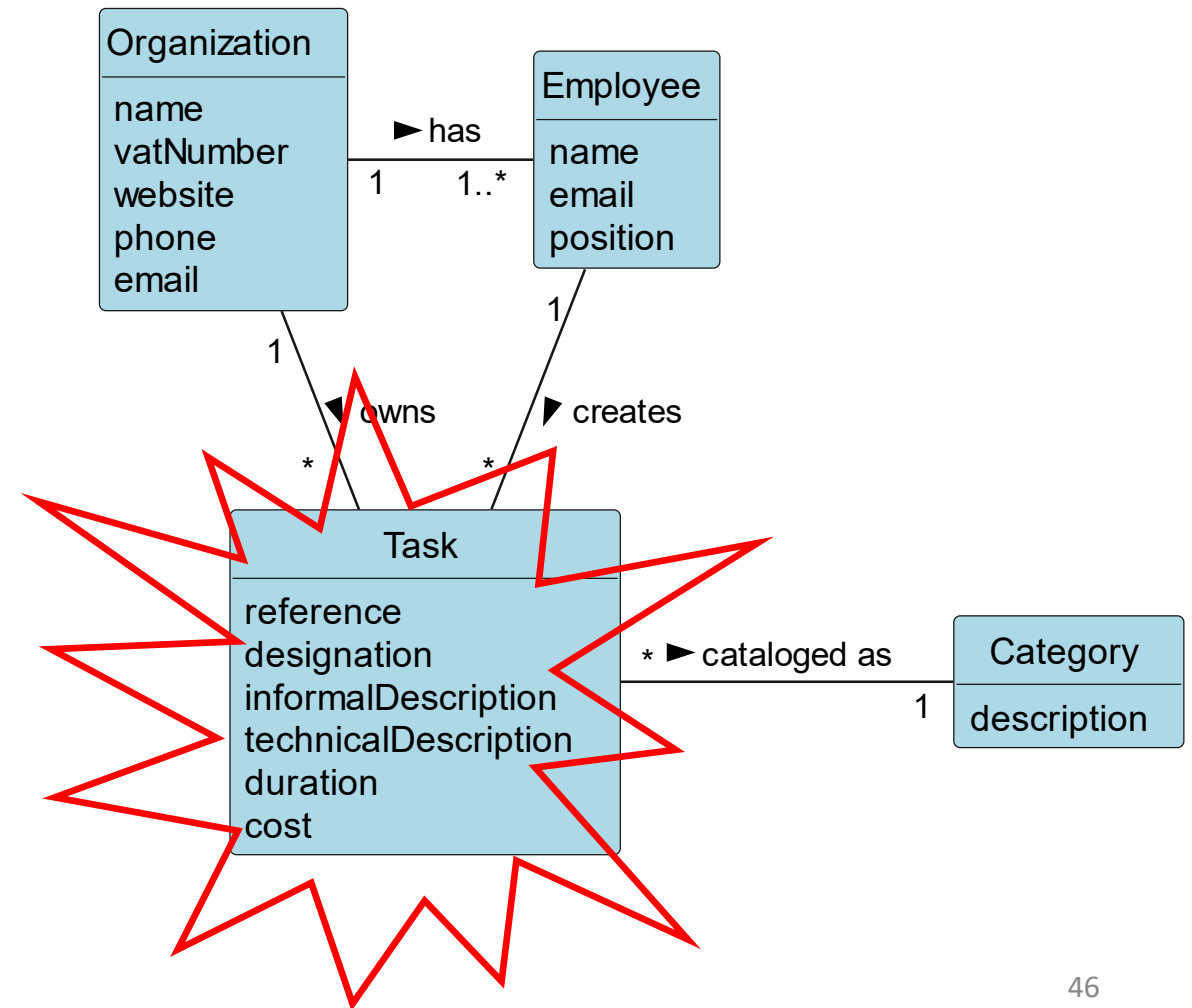
# Objects of the Domain Layer

- **Which class should be responsible for validating the data locally** (mandatory data) to comply with:
  - AC1: “All required fields must be **filled in**”; and
  - AC2: “Task **reference** must have at least 5 alphanumeric characters”
- **Get inspired by the Domain Model**
  - Promotion of conceptual classes to software classes
  - Not all concepts can/should be promoted to software classes
- New conceptual classes can still be discovered at this stage (and further stages)



# Task Object

- The task object should be **responsible for the information it holds** (e.g. reference).
- Therefore, this object should be responsible for validating its own data so that the system complies with AC1 and AC2.

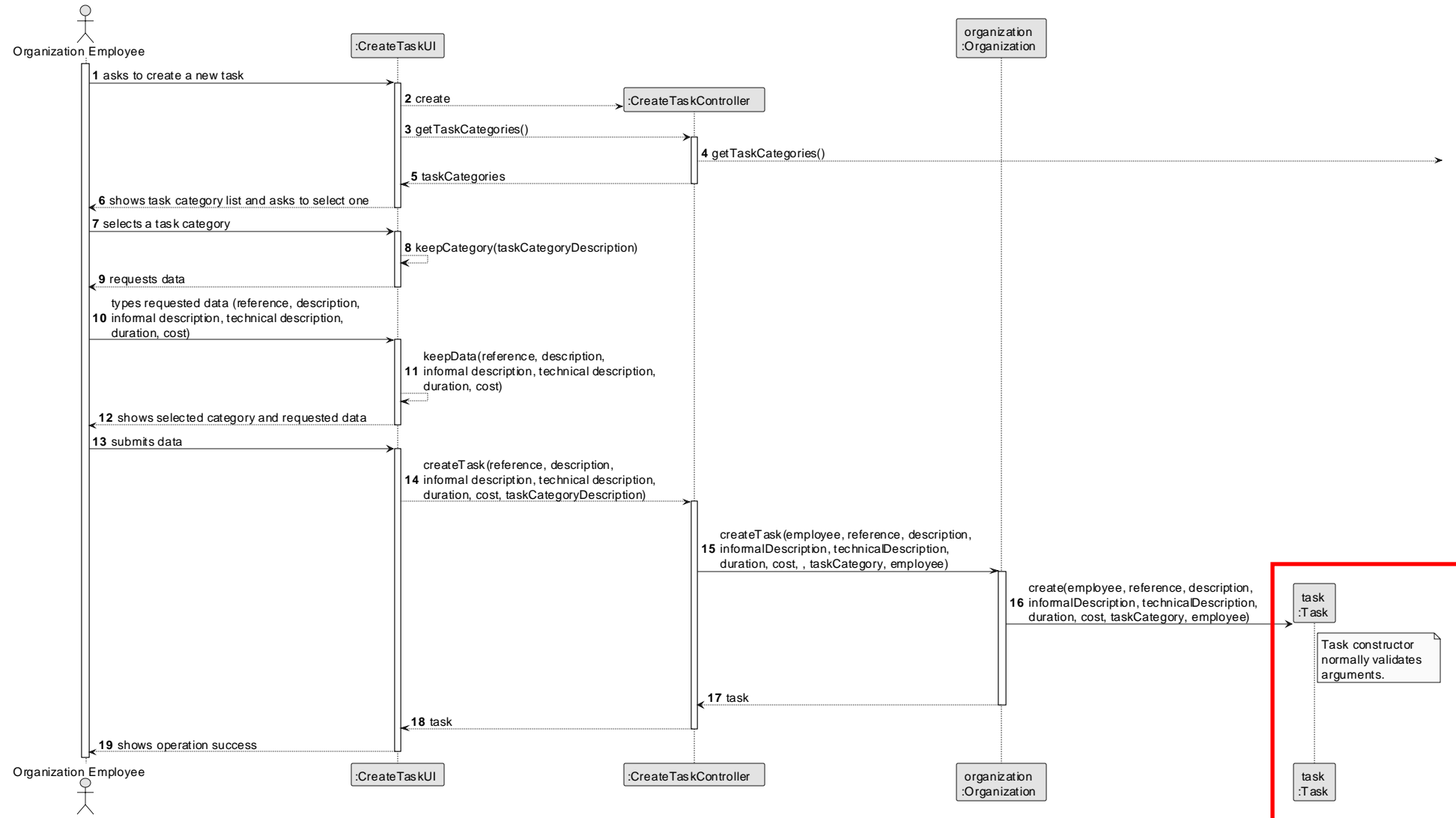


# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... validating the data locally (mandatory data)?	Task	Information Expert (the created object has its own data)
	... adding to a collection and globally validating duplicate records?	?	?

# Task responsibilities





# Rationale for Responsibilities Assignment:

## UC006 – Create a Task → what is still unknown?

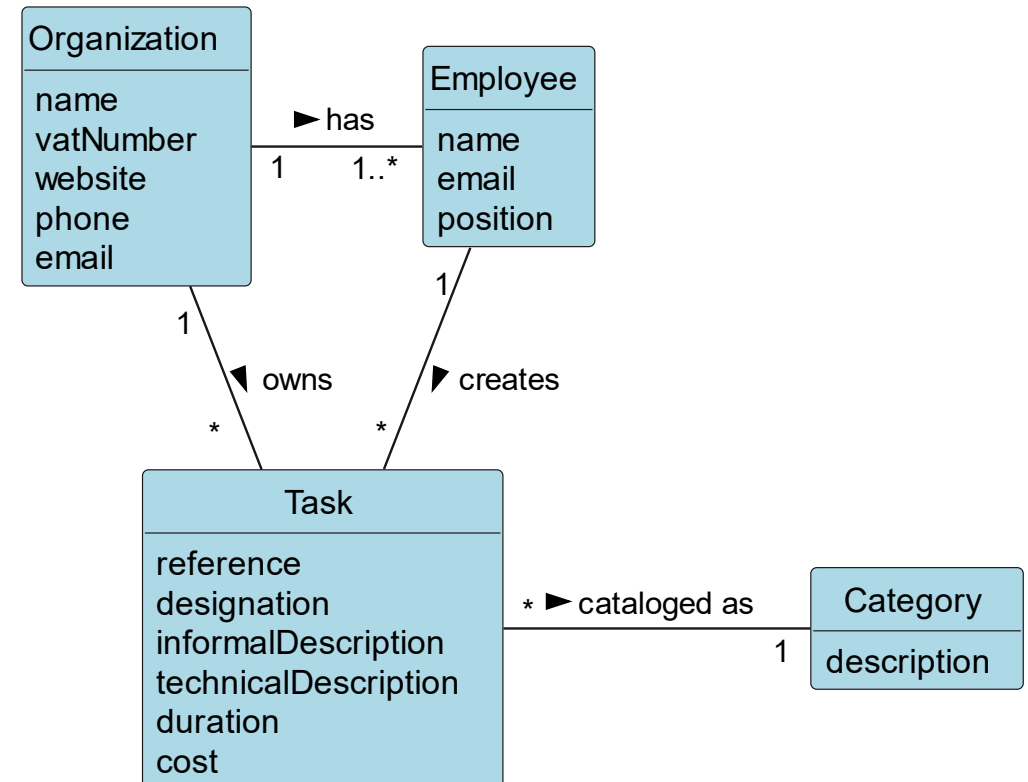
SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... adding to a collection and globally validating duplicate records?	?	?

# GRASP: Information Expert

Class Organization as “global” validator

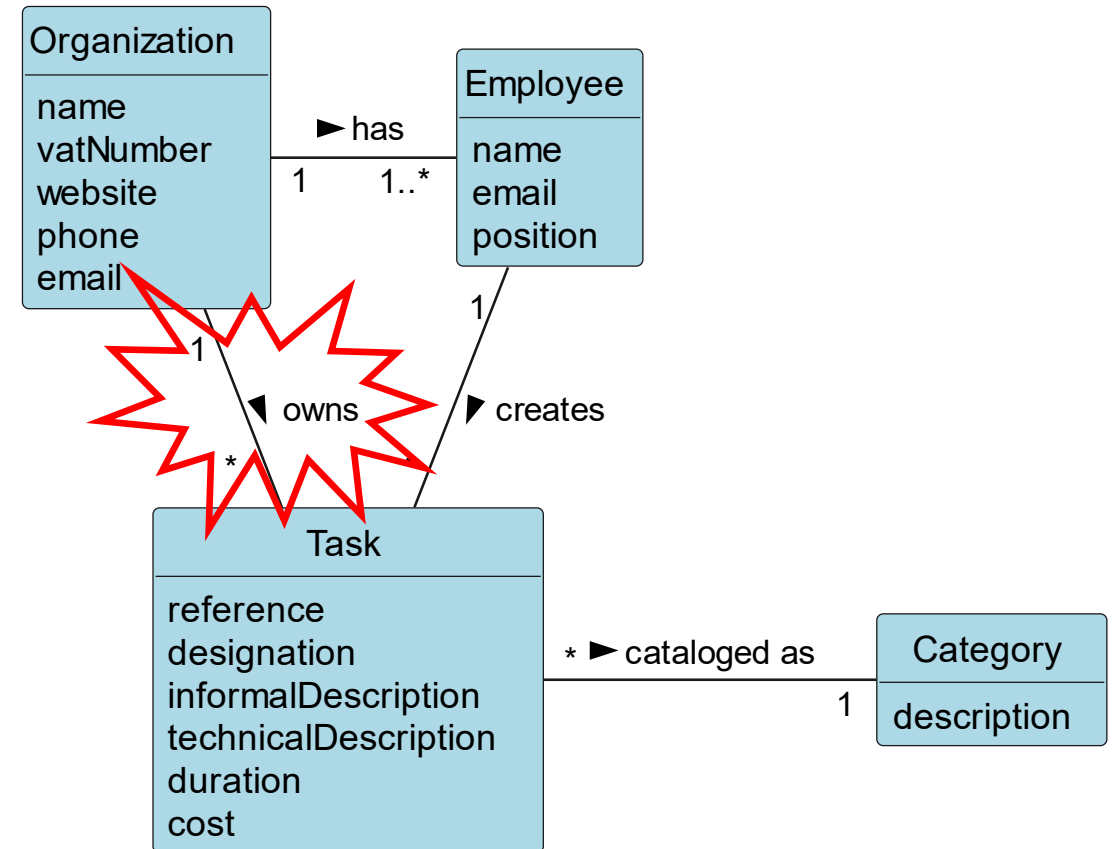
# Objects of the Domain Layer

- **Which class should be responsible for validating the data globally** (duplicate instances) to comply with:
  - AC3: “When creating a task with an existing reference, the system must reject such operation”
- **Again, get inspired by the Domain Model**



# Organization class

- The **Organization** is responsible for **containing or aggregating** instances of Task objects.
- Therefore, the Organization is also the Information Expert to answer questions related to its collection of Tasks.
- Additionally, it can validate and ensure that no duplicate Task instances are added to the collection.

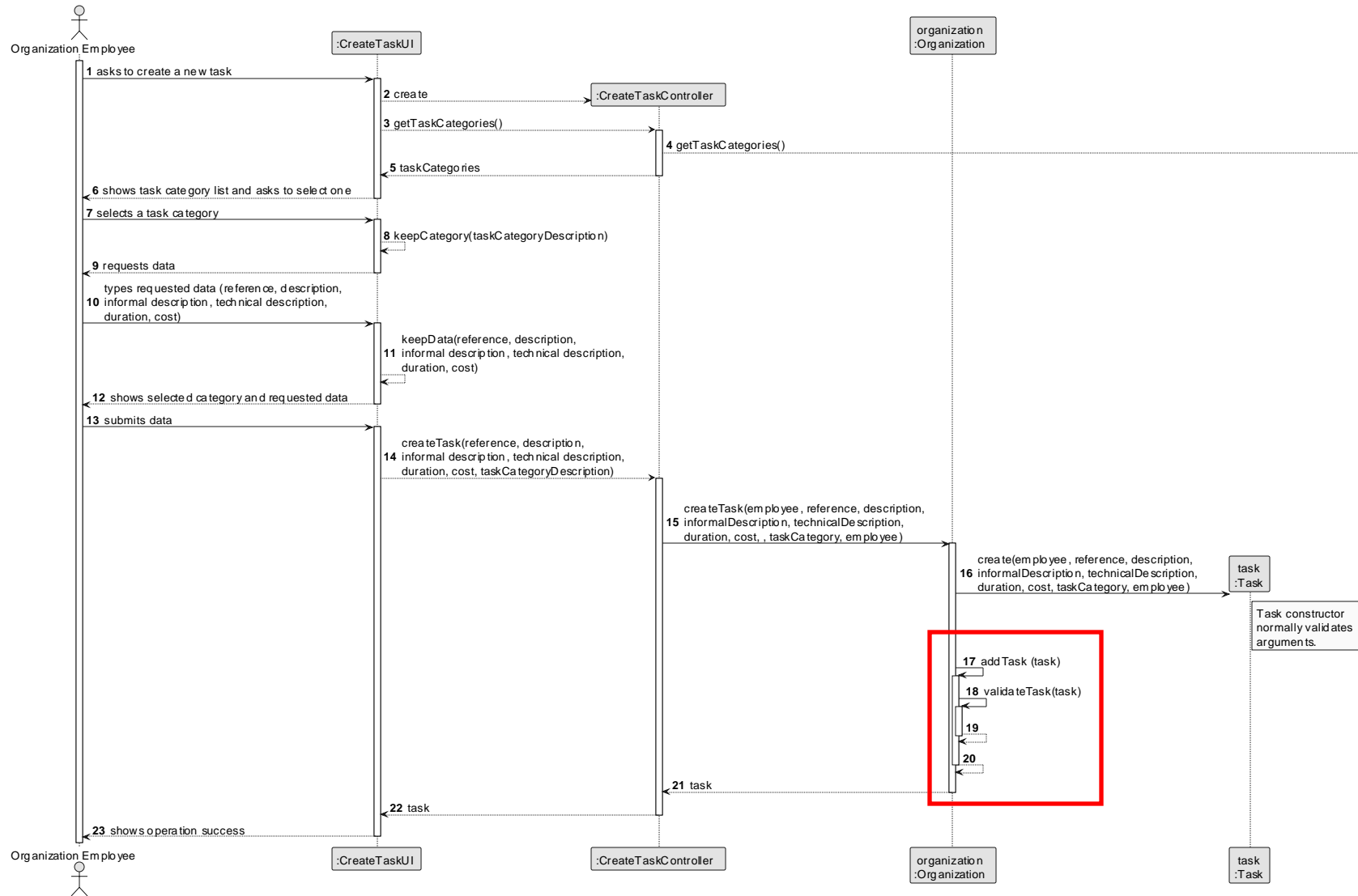


# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?
Step/Msg 7: submits data	... adding to a collection and globally validating duplicate records?	Organization	Information Expert (knows all its Task instances)

# Organization responsibilities



# Rationale for Responsibilities Assignment:

## UC006 – Create a Task → what is still unknown?

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	?	?

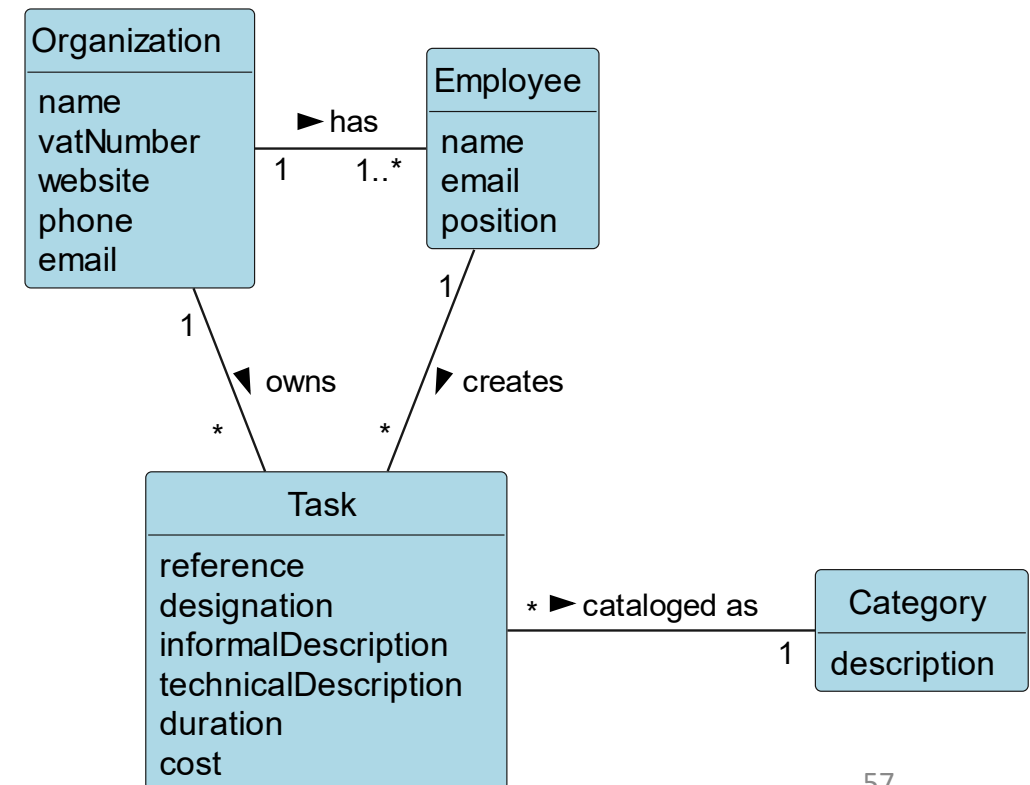
# GRASP: Information Expert

Class TaskCategoryRepository



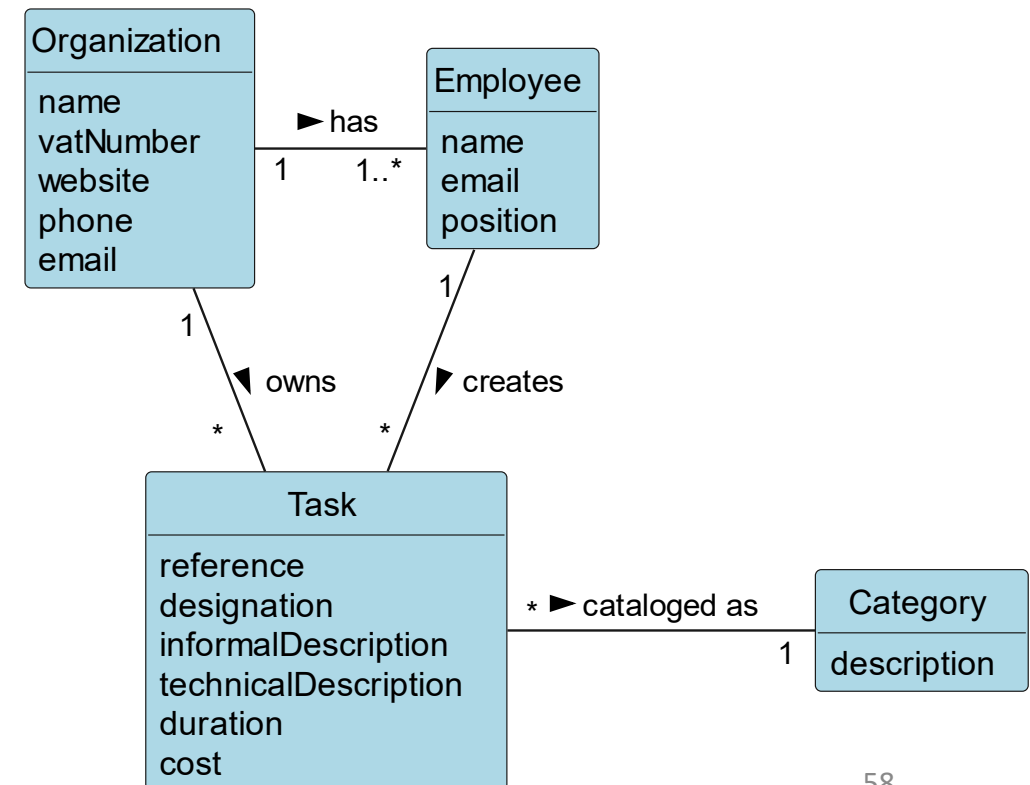
# Objects of the Domain Layer

- **Which class should be responsible for keeping instances of a class that do not “belong” to any domain object/class, such as **Task Category**?**
- **Check the Domain Model...**
- **Who keeps Task Category instances?**
  - If the categories were specific to each Organization, then it should be the Organization class.
  - **THIS IS NOT THE CASE!**
    - Organizations do not have their own categories
    - Categories are shared across organizations



# Repositories as Information Expert

- A **repository** is a special kind of class for **collection of objects that do not “belong” to any domain object/class**
- Looking at the domain model we realize that:
  - **Task Categories are kept in the system** – they form a collection of objects that do not “belong” to any domain object/class
  - Repositories are also obtained by **Pure Fabrication**

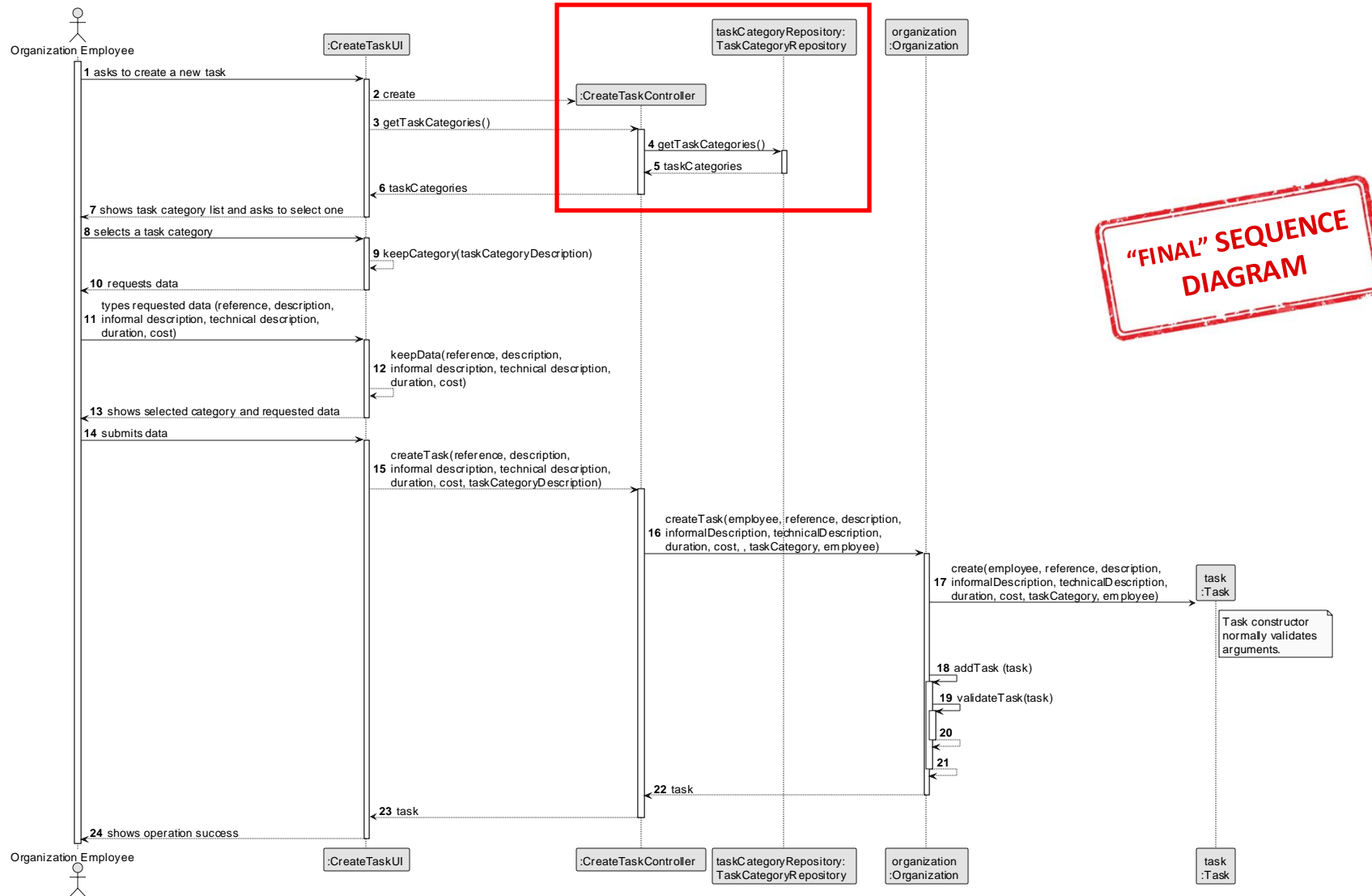


# Rationale for Responsibilities Assignment:

## UC006 – Create a Task

SSD Interaction ID	Question: Which class is responsible for...	Answer	Justification (with patterns)
Step/Msg 1: asks to create a new Task	... obtaining the task categories list?	TaskCategory Repository	Information Expert, Pure Fabrication

# TaskCategoryRepository responsibilities

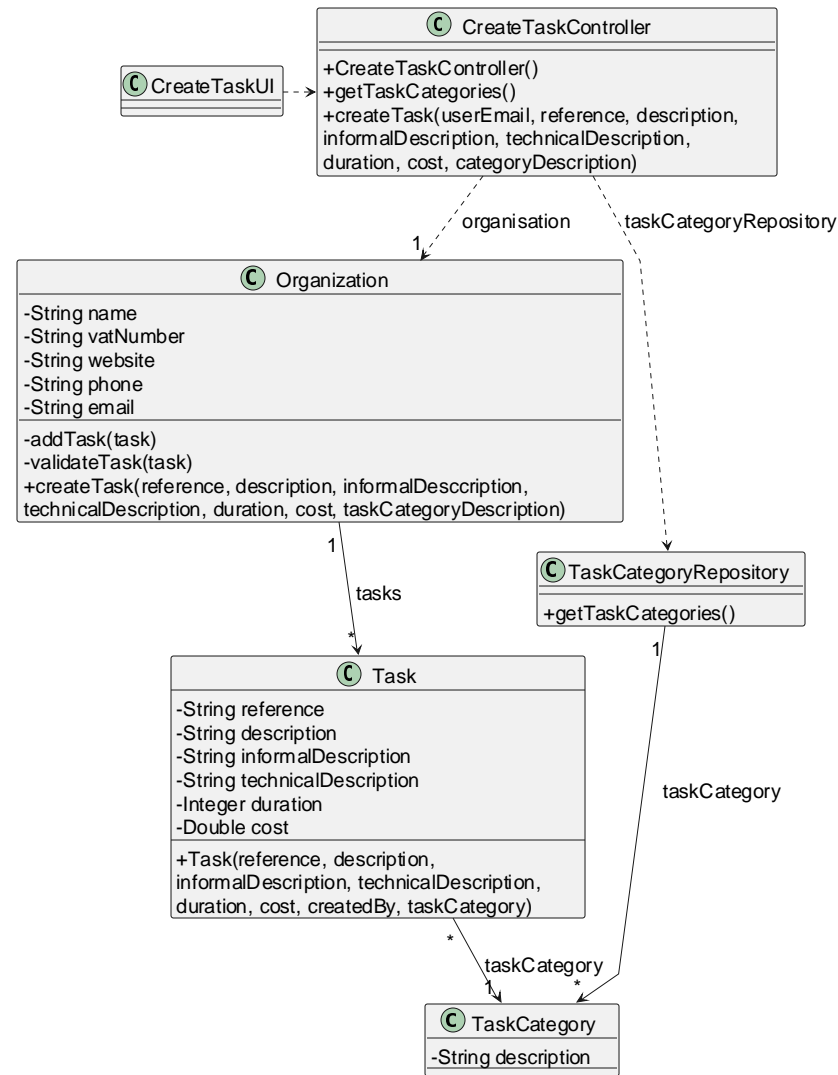


# UC006 – Create a Task

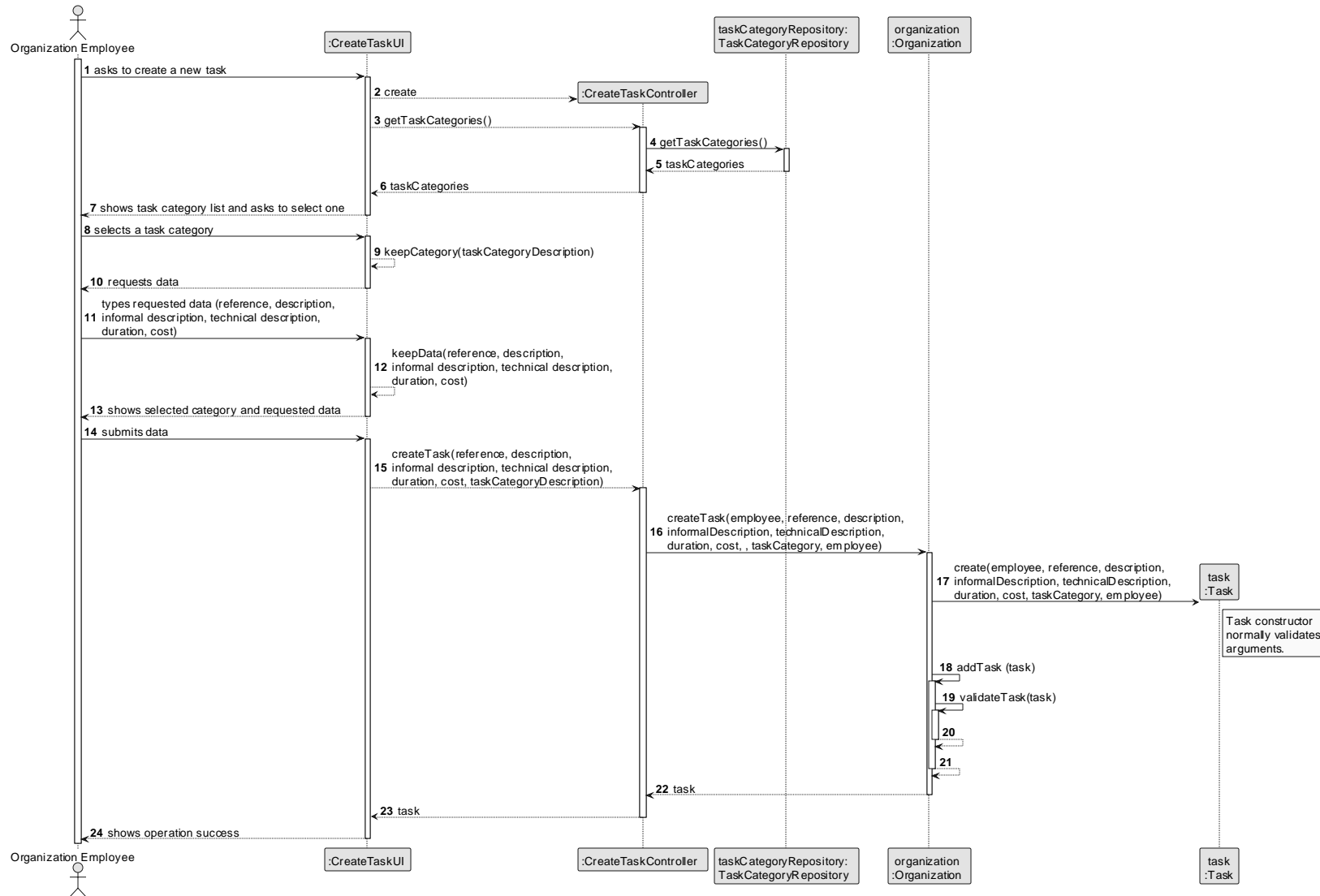
- Last check!!
  - ☒ Responsibilities assignment was performed
  - ☒ Sequence Diagram was created
  - ☒ At the same time, the Class Diagram was also created (cf. next slide)

# UC006 – Create a Task: Class Diagram

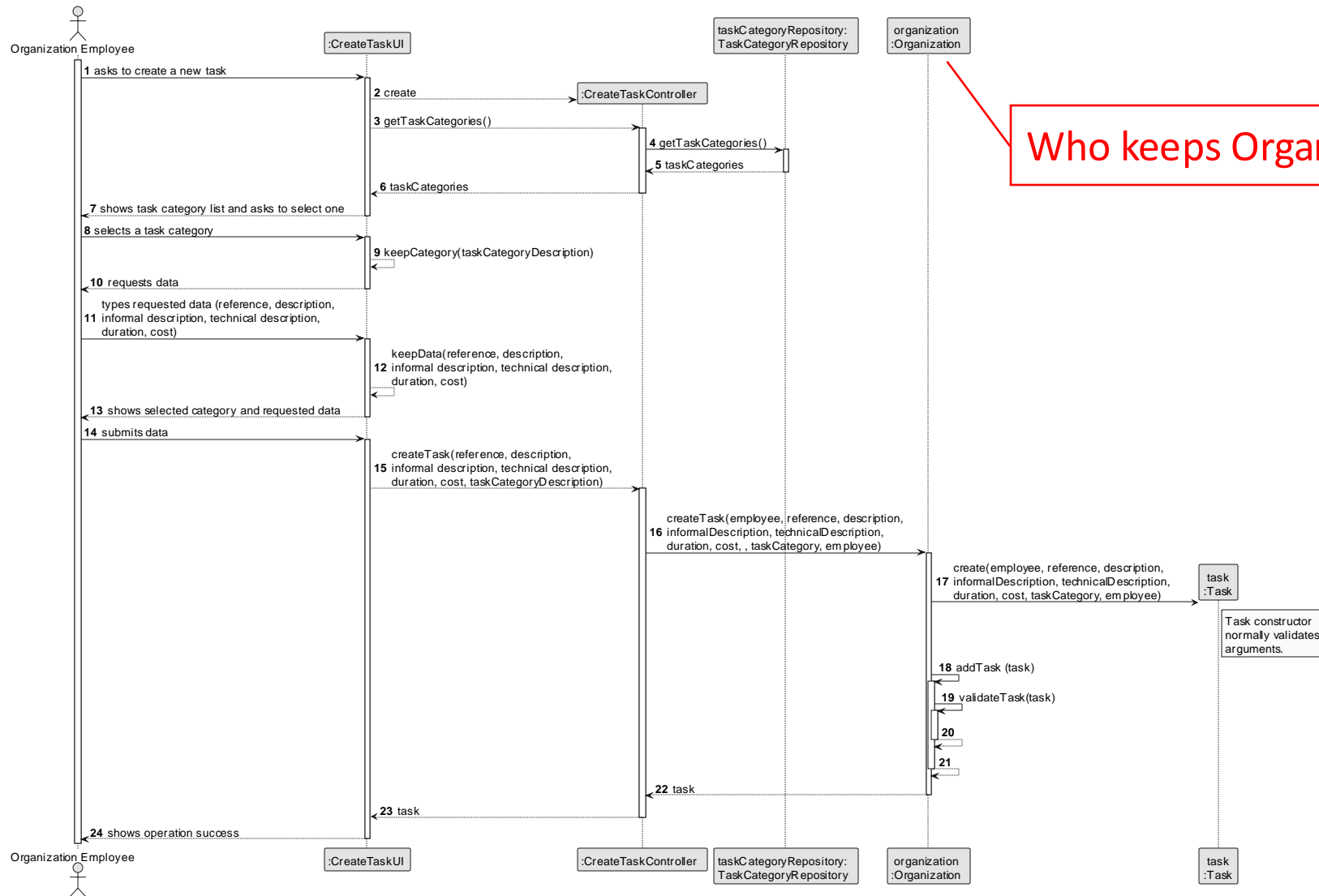
**CLASS DIAGRAM**



# Do you notice anything fishy? (1/4)

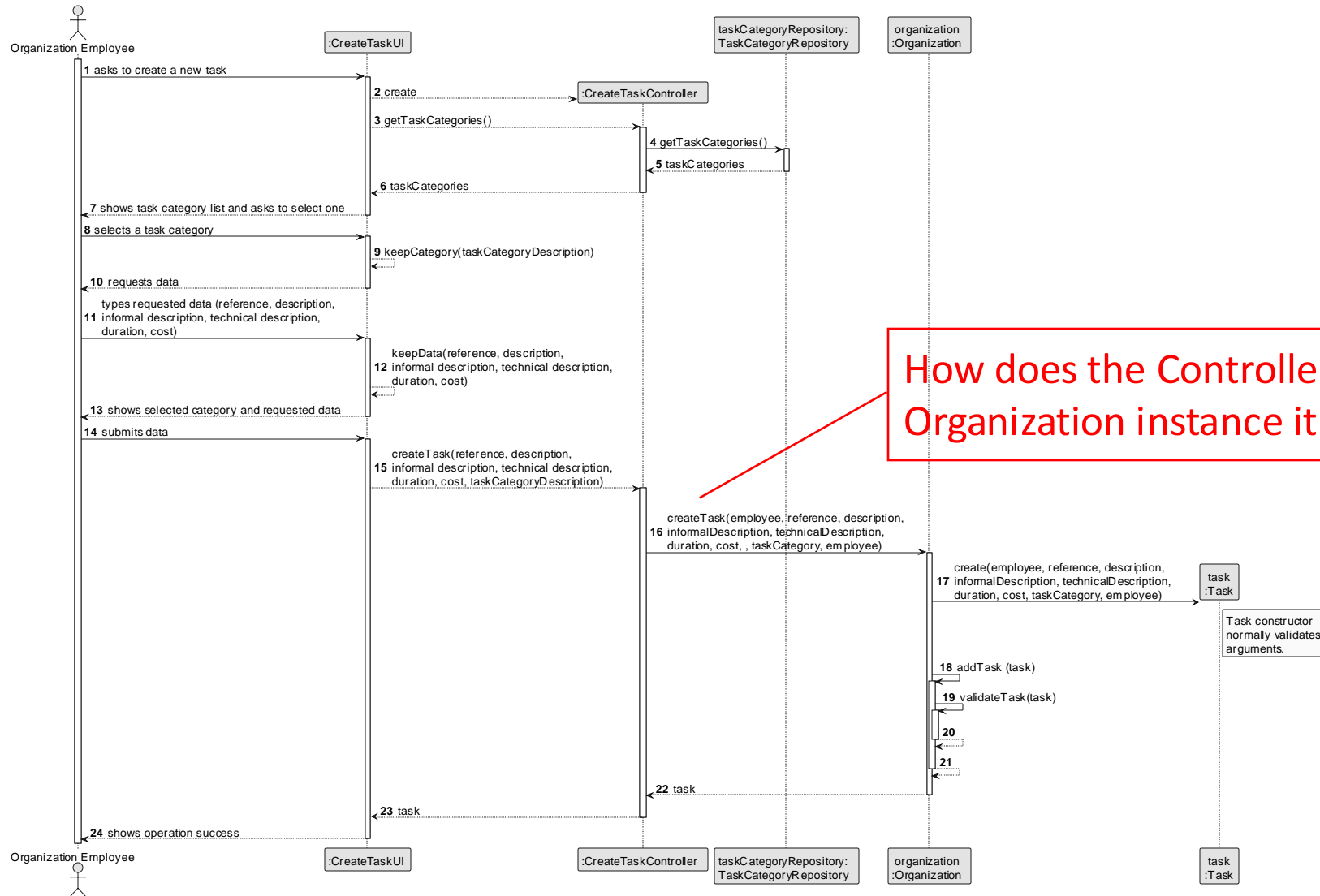


# Do you notice anything fishy? (2/4)

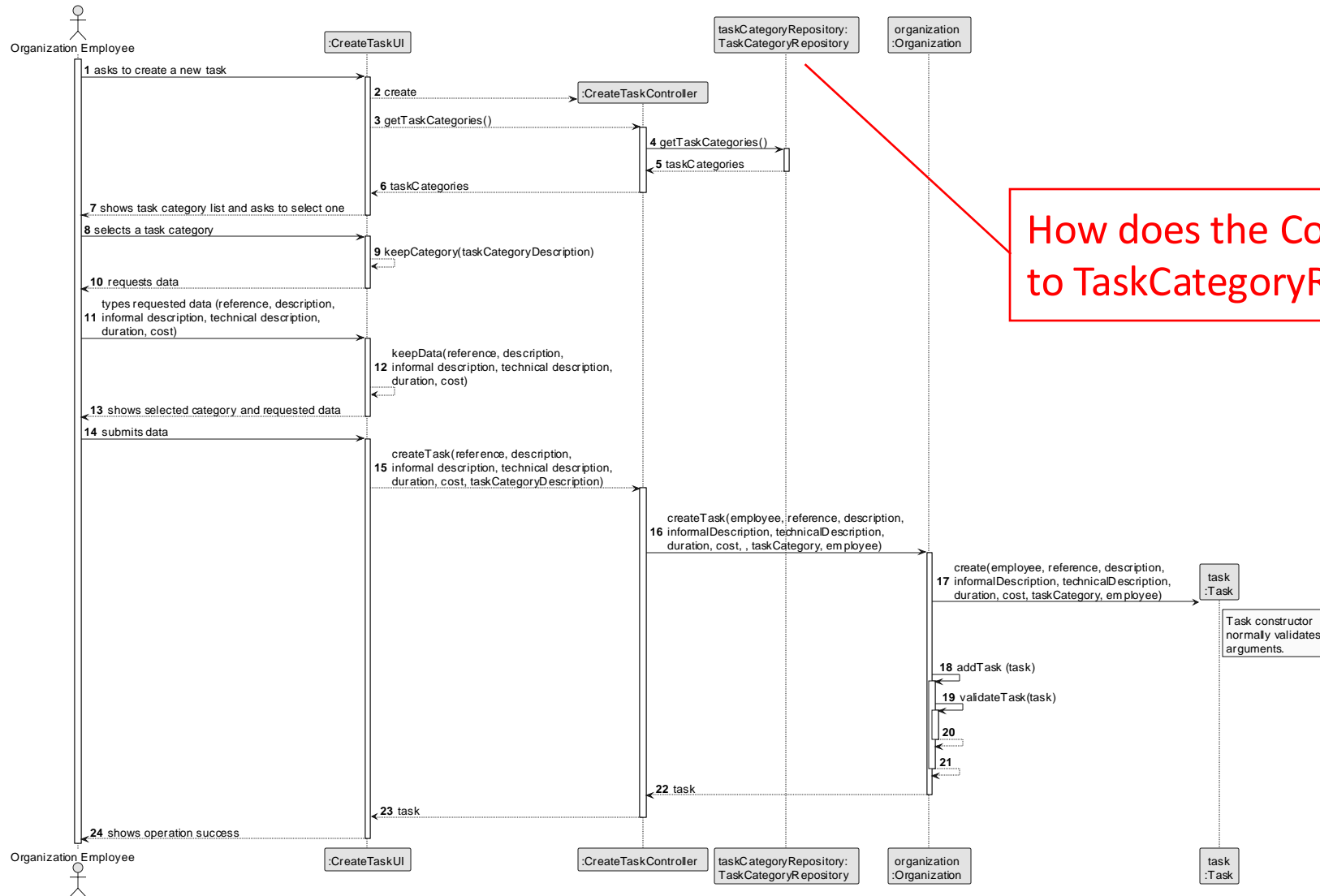




# Do you notice anything fishy? (3/4)



# Do you notice anything fishy? (4/4)



# Summary

- GRASP patterns addressed in this presentation:
  - Pure Fabrication
  - Controller
  - Creator
  - Information Expert
- Goals for the next presentations
  - Understand how the Controller can obtain access to multiple repositories
  - Understand how the Controller can get the right Organization from the logged in user

# References & Bibliography

- Larman, Craig; Applying UML and Patterns; Prentice Hall (3rd ed.); ISBN 978-0131489066
- Fowler, Martin. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional, 2004.
- Rational Unified Process: Best Practices for Software Development Teams; Rational Software White Paper; TP026B, Ver 11/01.
- <https://www.kamilgrzybek.com/blog/posts/grasp-explained>